Integration of IoT and Cloud Computing: Development of an Intelligent Face-recognition System

Asif Ahmed
Student Id: 011 141 068
Md. Younus Bipul
Student Id: 011 141 075
Syad Md. Imran
Student Id: 011 141 086
Niger Sultana Tahniat
Student Id: 011 141 088

A thesis in the Department of Computer Science and Engineering presented
in partial fulfillment of the requirements for the Degree of
Bachelor of Science in Computer Science and Engineering

United International University
Dhaka, Bangladesh
November, 2018

Declaration

We, Asif Ahmed, Md. Younus Bipul, Niger Sultana Tahniat and Syad Md Imran, declare that this thesis titled, “**Integration of IoT and Cloud Computing: Development of an Intelligent Face-recognition System**” and the work presented in it are our own. We confirm that:

- This work was done wholly or mainly while in candidature for a BSc degree at United International University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at United International University or any other institution, this has been clearly stated.
- Where we have consulted the published work of others, this is always clearly attributed.
- Where we have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely our own work.
- We have acknowledged all main sources of help.

____

Name: Asif Ahmed, ID: 011 141 068, Dept: CSE

____

Name: Md Younus Bipul , ID: 011 141 075, Dept: CSE

____

Name: Syad Md Imran, ID: 011 141 086, Dept: CSE

____

Name: Niger Sultana Tahniat, ID: 011 141 088, Dept: CSE

Certificate

I do hereby declare that the research works embodied in this thesis entitled "**Integration of IoT and Cloud Computing: Development of an Intelligent Face-recognition System**" is the outcome of an original work carried out by Asif Ahmed, Md. Younus Bipul, Niger Sultana Tahniat and Syad Md Imran under my supervision.

I further certify that the dissertation meets the requirements and the standard for the degree of BSc in Computer Science and Engineering.

Salekul Islam
Professor and Head, CSE
United International University

Abstract

IoT has seen steady growth over recent years with smart home appliances, smart personal gear, personal assistants, industrial assistance and many more. Devices used in the Internet of Things (IoT) are often low-powered with limited computational resources. Whereas, the computation part is done in the backend Cloud server. In this thesis, we compare how the scenario changes when computation is done in edge Cloud, near to the data source and thus reducing the distance of network hop and size of data for IoT scope. We developed a face recognition framework as an IoT application with computational server in two different infrastructures: a local, near to the client as edge Cloud, and also in commercial Cloud platform. Also implementing a part of computation in edge node or gateway can decrease the number of data packets in a huge amount and therefore, reduces network latency. In our thesis, the processing time of our developed system and network latency have been measured and compared. The results demonstrate that using edge Cloud, rather than core Cloud is comparably faster in terms of network latency. Moreover, decreasing the size of the transmitted data by computing in client side, reduces network latency and congestion.

Acknowledgement

We would like to thank our supervisor Dr. Salekul Islam for exposing us to this real world application and supporting us, not only to make a thesis document, but to learn a great length on programming and development in IoT, Cloud computing and networking sectors.

Also we would like to acknowledge the developers of the open-source libraries that we have used in our application.

And in last, a huge appreciation for our family and friends for their support and advice.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Motivation

Internet of Things (IoT) and Cloud computing, among many terms in community of information technology, are the most hyped and popular topics of this era. Modern day computer science, engineering research and development is basically running on these trends. Cloud computing has been proven to be one of the most reliable method for its scalability, pay-per-use, workload resilience and flexibility. A lot of applications use Cloud computing. Reliability of connection, availability of high bandwidth of Internet, elasticity and pay-as-you-go business style are some of the reasons of depending on a great scale on Cloud computing. The uprising of large scale IoT application and appliances rely on the Cloud paradigm. Both data-intensive and computation-intensive applications are developed and deployed based on Cloud framework. As the rise of Cloud era uplifted the IoT world, lots of research is undergoing on various short-hands and challenges based on connectivity and security for Cloud computing.

IoT applications are created based on collecting real world and real time data, putting through, in some cases, vast network of algorithms and getting desired result. Thus, in most scenarios, IoT appliances require low response time, least network latency, and also, less pressure on bandwidth holds a great deal of advantage. The data traffic to move the huge number of tiny data packets from devices, through the core network to the Cloud, occupying the Internet, increases network traffic congestion and decrease network bandwidth. It is a huge bottleneck. Rather than transferring data to Cloud, servers which are fitted with thousands of stack of resources, far away from data, edge computing comes in as a solution.

1.1.1 Cloud in the Internet

The Internet is often called “the network of networks”, because it consists multiple small heterogeneous and autonomous network, where IoT takes or will take a major amount of ground. At present, the Internet can be roughly architectured into three different layers: core, edge and access (figure 1) [1]. The small networks are managed by Internet Service Providers (ISP) and they are connected to the core network. The core network interconnects all the small networks with high speed and high capacity routers and switches to interconnect information. The small networks are known as edge network. These small networks also consist multiple number of router and switches nearer to the end user. The end users connect to the Internet through the access network.

Following the definition of core and edge network in the Internet architecture, the Cloud architecture can be classified as Core Cloud, and Edge Cloud. In our thesis, we name the public Cloud which have been built around the world by the commercial vendors, as Core Cloud. Due to the limited number of service point availability of core Cloud, the end user or in our case, the ‘thing’ has to pass data through the core of the Internet, hopping multiple routers and switches.

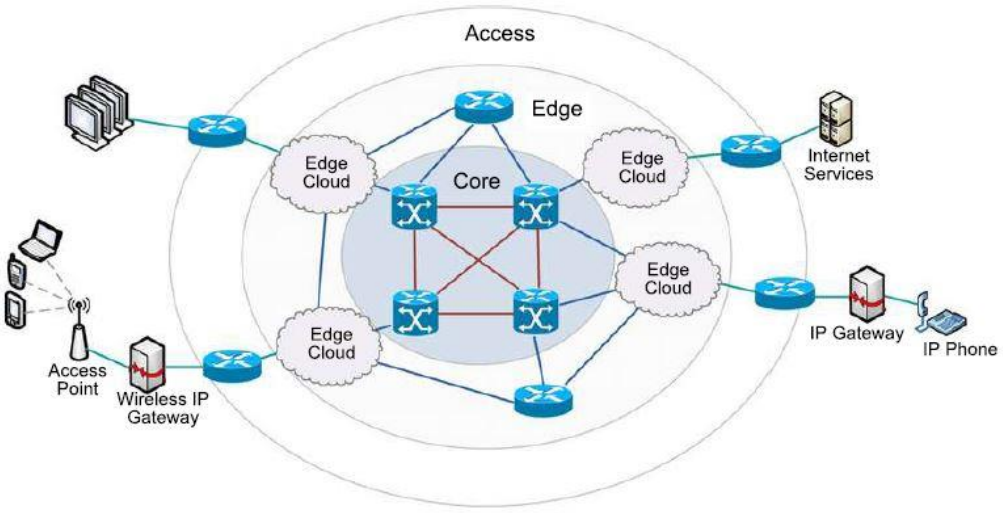


Figure 1: Edge Cloud-based next-generation Internet architecture [2]

On the other hand, the edge Cloud, which is at the edge of the network, consists of multiple smaller, generic clouds [2], nearer to end users (figure 1). It proceeds with the benefit bringing computation closer to the client allowing for scalability, lower latency and smaller client-side footprint (e.g. [3]). The IoT applications deployed in the edge Cloud will have lower latency, and will decrease network congestion.

1.2 Objective

In our thesis, we are demonstrating Edge Cloud computing vs core Cloud computing, implementing real time face recognition as a data-intensive IoT application, both on an edge Cloud and on a core Cloud, and studying the difference of performance for our facility. This thesis gives the experimental results of Face Recognition as an IoT application deployed in edge Cloud (i.e., UIU data center) and a core Cloud (Microsoft Azure).

For all types of IoT applications, raw data is sent to Cloud for computation and getting desired result back to device. In most developments of Face Recognition system (eg. [4]), raw data, as in frames or image with human face is sent out to classify identity from a dataset. For which, the network of that system costs much more data packets and bandwidth allocation. The processing time increases with the number of faces in that very image [4]. In our implementation, we have pre-processed the data in client that will reduce the size of the data to be transferred. And also, following the pre-processing, each face is separated and sent as individual data, if there are multiple face in one frame. Thus, proposing normalizing and pre-processing raw data can reduce the size of data to be sent, can save huge amount of network allocation.

1.3 Organization

The remainder of this thesis is organized as follows. In Section 2, we derived a study on the key concepts from previous literature – Internet of Things (IoT), Cloud Computing and Edge Computing. Description of the architecture and implementation methodologies of our case study, Real Time Face Recognition as an IoT application, is addressed in section 3. Section 4 gives a comparison of the performance of our deployed application, between edge Cloud and core Cloud. Finally, in Section 5 concludes our thesis with definite results and conclusion.

Chapter 2

Background and Literature Review

2.1 Internet of Things (IoT)

Internet of Things (IoT) means network of things, physical devices. It is a novel paradigm that is rapidly gaining ground in the scenario of modern technology. It provides the ability to transfer data over a network without iterating human to human or human to computer and works via objects or devices. In IoT, data comes in via devices or things. These could be sensors, mobile phone, electronic devices, vehicles, watch, air conditioner, car, camera and various other 'things' - which, through unique addressing schemes, are able to interact with each other and cooperate with their neighbors to reach common goals [5].

Internet of Things (IoT), the term was first introduced in 1999 for supply chain management [6], and then the concept of "making a computer sense information without the aid of human intervention" was widely adapted to other fields such as healthcare, home, security, environment, and transports [7], [8].

In future various information and things will be connected to network and we expect it will be. People can live more convenient and comfortable lives with everyday things and their information coordinated together. Various information and things are inter-connected to a network is referred to as the IoT.

It should not be surprising that IoT is included by the US National Intelligence Council in the list of six ''Disruptive Civil Technologies" with potential impacts on US national power. NIC predicts that ''by 2025 Internet nodes may reside in everyday things – food packages, furniture, paper documents, and more". It highlights future opportunities that will arise, starting from the idea that ''popular demand combined with technology advances could drive widespread diffusion of an Internet of Things (IoT) that could, like the present Internet, contribute invaluably to economic development" [5].

2.2 Cloud Computing

Cloud computing is a paradigm shift that provides computing over the Internet. It consists of highly optimized virtual data centers that provide various hardware, software and information resources for use when needed. Organization can simply connect to the Cloud, use the available resources on a pay per use basis from in Cloud.

It gives consumer ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or provider-consumer interaction. [9] This Cloud model is composed of -
Five essential characteristics: *On-demand self-service, Broad network*, *Resource pooling, Rapid elasticity and Measured service.*
Three service models: *Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS).*
And four deployment models: *Private Cloud*, *Community Cloud*, *Public Cloud and Hybrid Cloud*.

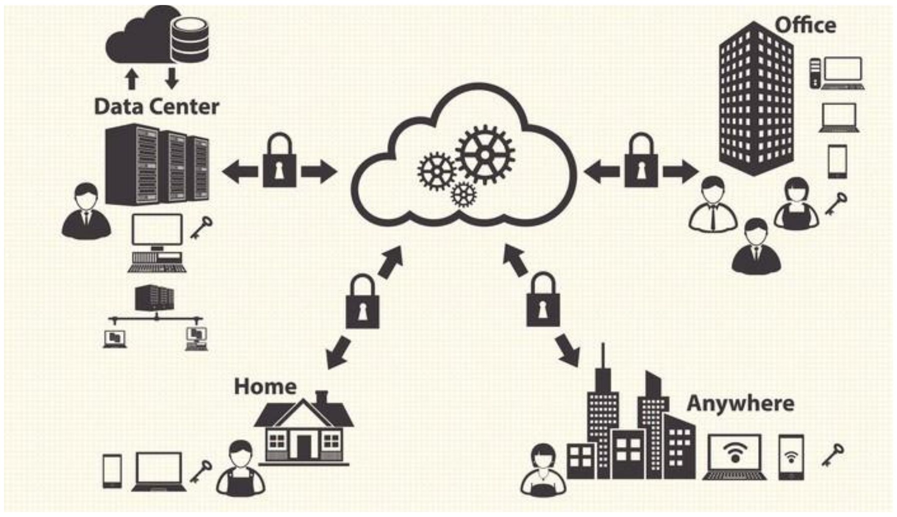


Figure 2: Architecture of Cloud Computing
(https://www.fool.com/knowledge-center/what-is-Cloud-computing.aspx)

However, a related term “grid computing,” from the high-performance computing community, suggests protocols to offer shared computation and storage over long distances, but those protocols did not lead to a software environment that grew beyond its community [10].

Up till this year, there are various popular Cloud vendors such as AWS (Amazon Web Service), Microsoft Azure, Google Cloud Platform, Salesforce etc., which provides various service and virtual resources on pay per use terms.

2.3 Edge Computing

Edge computing refers to the enabling technologies allowing computation to be performed at the edge of the network, on downstream data on behalf of Cloud services and upstream data on behalf of IoT services. Here we define “edge” as any computing and network resources along the path between data sources and core Cloud data centers or core clouds [11].

We named Cloud in edge network as Edge Cloud. The detailed implementation of the edge Cloud is beyond the scope of this thesis. Figure 3 shows different stacks of services of the Edge Cloud [2].

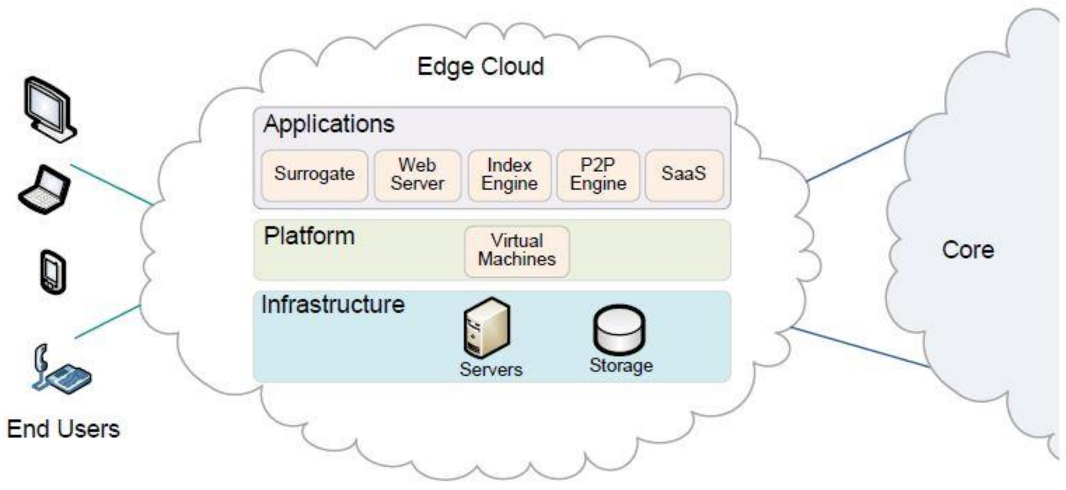


Figure 3: Inside the Edge Cloud [2]

Data is increasingly produced near the edge of the network, therefore, it would be more efficient to also process the data at the edge of the network. Previous work such as micro datacenter [12], [13], cloudlet [14], and fog computing [15] has been introduced because Cloud computing is not always efficient for data processing when the data is produced far from it, in the edge of the network.

2.3.1 Edge Computing over Cloud Computing on IoT

Transferring all data to a Cloud for computing has been an efficient way to process data because there's more computing resource in the Cloud than in the devices at the network edge or IoT enabled devices. However, although data-processing speeds have risen rapidly, the bandwidth of the networks that carry data to and from the Cloud has not increased appreciably. Thus, with edge devices generating more data, taking more network hops, the network is becoming Cloud computing's bottleneck.

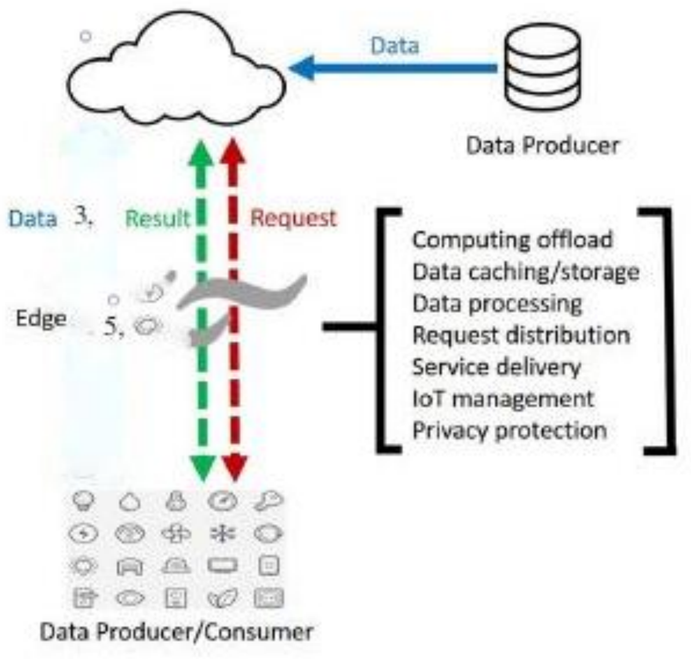


Figure 4: Edge Computing Paradigm
(IEEE INTERNET OF THINGS JOURNAL, VOL. 3, No. 5, October 2016)

Let us consider an autonomous vehicle as example of this scenario. One Gigabyte data will be generated by the car every second and it requires real-time processing for the vehicle to make correct decisions [16]. If all the data is sent to the Cloud for processing, the response time would be too long. Not to mention that current network bandwidth and reliability would be challenged for its supporting a large number of vehicles within one area.

Almost all kinds of electrical devices will become part of IoT, and they will play the role of data producers as well as consumers, such as air quality sensors, LED bars, streetlights and even an Internet-connected microwave oven. It is safe to infer that the number of things at the edge of the network will develop to more than billions in a few years. Thus, raw data produced by them will be enormous, making conventional Cloud computing not efficient enough to handle all these data. This means most of the data produced by IoT will never be transmitted to the Cloud, instead it will be consumed at the edge of the network.

To demonstrate edge Cloud over core Cloud, various research and study can be discussed.
Specially for multimedia application, results show huge difference in performance, based on network parameters. Several case studies, ranging from Cloud offloading to smart home and city, as well as collaborative edge to materialize the concept of edge computing is studied and several challenges and opportunities in the field of edge computing has been presented to gain attention on researching more on edge computing over Cloud computing [11].

Salekul Islam and Jean-Charles Grégoire have proposed a study of deploying a prototype that transcodes audio/video stream inside edge Cloud and core Cloud to monitor performance of the prototype by analyzing the inter-arrival jitter [17]. With their SaaS model offers a new role for the ISP—or an extension of his role as CDN provider, as well as an extended contract with the user which can go to guaranteed performance and support for mobility, rather than using commercial Cloud vendors.

Nasif et al. [4] have proposed to transfer the face recognition computation from the smartphone to the edge Cloud to get faster results using a server’s powerful processing capability and big storage facility. And monitor differences regarding performance with smartphone. This application can identify faces from an image and then compare the uploaded face with these extracted faces. The experimental results demonstrate that face recognition is performed comprehensively faster at the edge Cloud than on the smartphone. To extend the study and demonstrate the proposition of edge Cloud over core Cloud, this application can be extended to perform more efficiently as for IoT use case and measure network parameter differences between edge Cloud and core Cloud.

Chapter 3

Real Time Face Recognition

In our thesis, we have constructed an IoT infrastructure with a raspberry pi as client device, a middleware and deployed the computational process in both edge Cloud (United International University Data Center) and core Cloud (Microsoft Azure).

The main component of our system, as we need a huge computational cost to review and continuous data to run around, is a real time face recognition system from IP camera used in the facility.

3.1 Face Recognition

Face Recognition is a system where given a picture or frame of person’s face, the application recognizes the identity of that person, computing data from multiple known face data. In general face recognition is a service consisting two different computational process: face detection and recognition. In our developed system, in the face detection phase, human face or faces are detected from video frame fetched from IP camera’s video feed. Then the captured face is embedded into a feature vector via trained neural network model. In the recognition phase, using that feature vector, computing through a machine learning algorithm and classification from a trained model of face dataset identification of that face is acknowledged. Point to note, no motion detection or converting images to grayscale is done.

3.1.1 Detection and feature extraction

An overview of the detection process is shown in the figure 5. Basically, the process in Raspberry Pi catches every frame from connected camera and detect human faces from it using Histograms of Oriented Gradients (HOG) model (a unified model for face detection, pose estimation, and landmark estimation in real-world, cluttered images [18])

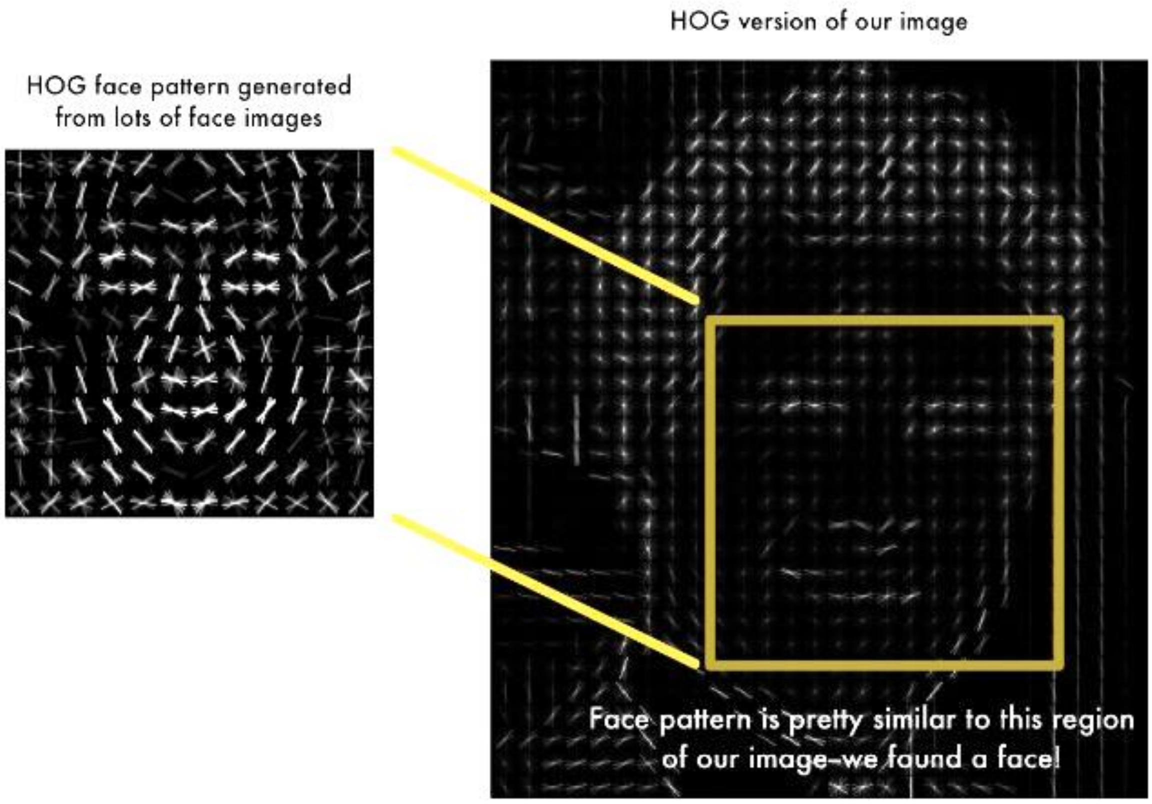

HOG version of our image
HOG face pattern generated from lots of face images
Face pattern is pretty similar to this region of our image–we found a face!

Figure 5: Histograms of Oriented Gradients (HOG) to detect face (https://medium.com/@ageitgey)

Firstly, each face is detected in the frame. Then from each face frame, we get an array of 128 floating numbers (implemented with `dlib`, pre-trained CNN model - `resnet_model_v1`, ResNet by Kaiming He), which is called face encoding from that captured face frame, using standard techniques as FaceNet embedding as feature vectors. It actually achieves state-of-the-art face recognition performance using only 128 floating number per face. Widely used Labeled Faces in the Wild (LFW) dataset, this system achieves accuracy of 99.38%. [19] On YouTube Faces DB it achieves 95.12%. A slight abstraction of this feature extraction is shown in figure 4. A sample face encoding is enlisted in the appendix section in our sample face data.

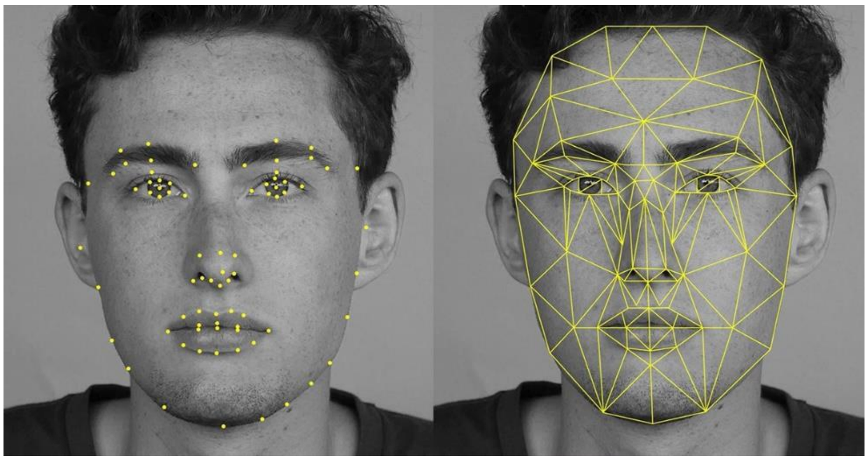


Figure 6: Generating face encodings from facial image (https://hackernoon.com)

In most studies and development, video feed is fitted with a motion detection system, and whenever something moves in the frame, it takes a snap of the frame and send that frame to recognition face, where it detects human face. We proposed that, every frame will go through processing and whenever there is a human face, it will only capture the frame of that face rather than capturing the whole frame. Then, from that face frame, feature vector is acquired. This part is done in the client side of the architecture. The feature vector wrapped in a json structure, sizes only 3.9kb per face, is a feasible data to send to server, compared to large image data depending on resolution and full frame size of camera. A sample of the face data is included in the appendix section.

3.1.2 Recognition

The face encoding is used to classify with our pre-trained knn model of face encodings from face dataset and get user's identity, the primary key (ID, name) of that person's data in the database. We have used knn for classifying data, as it is a simple approach based on feature's distance evaluation [20] and classifies with good accuracy. With that primary key, recognized person's credential is fetched. The result then is then saved with that timestamp, location-in-frame, and camera id in database as a log, and also sent to the client node (Pi) as a notification wrapped in a json string.

This part is done in the server side. We have used 1000 numbers of human face from UTK face dataset [21] as our test dataset. The computational time on the server side depends on the size of the dataset.

3.2 Generic Architecture

Here is a flowchart (figure 6) demonstrating our IoT application's architecture. Where in the client side, each and every frame is captured. If there is one or multiple face in the frame, it is computed into separated face data (face encodings, location of the face in frame, timestamp) for each face. Then the data is sent to the recognition application in the server side of the architecture, both in core Cloud and edge Cloud, via the middleware. After recognition, the results are sent, also via the middleware to the client side.

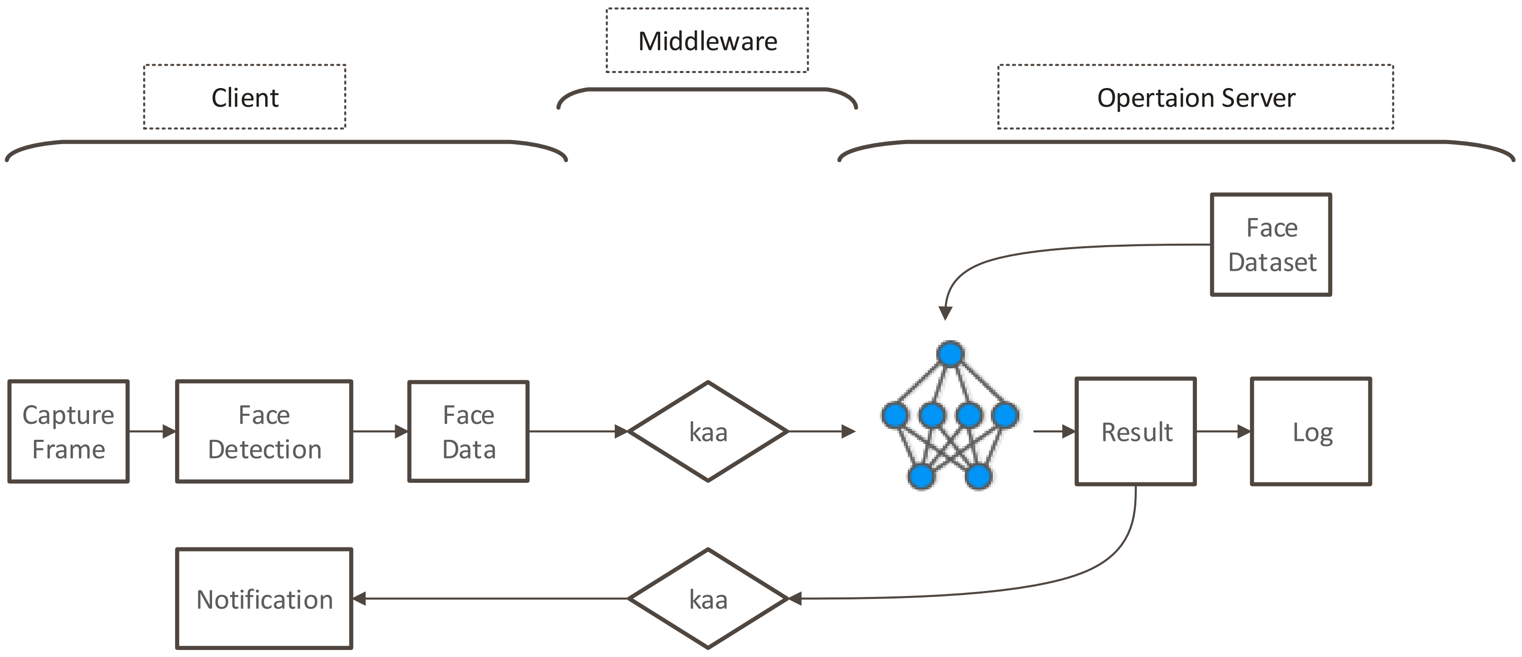


Figure 6: Flowchart of proposed architecture of Face Recognition System in IoT

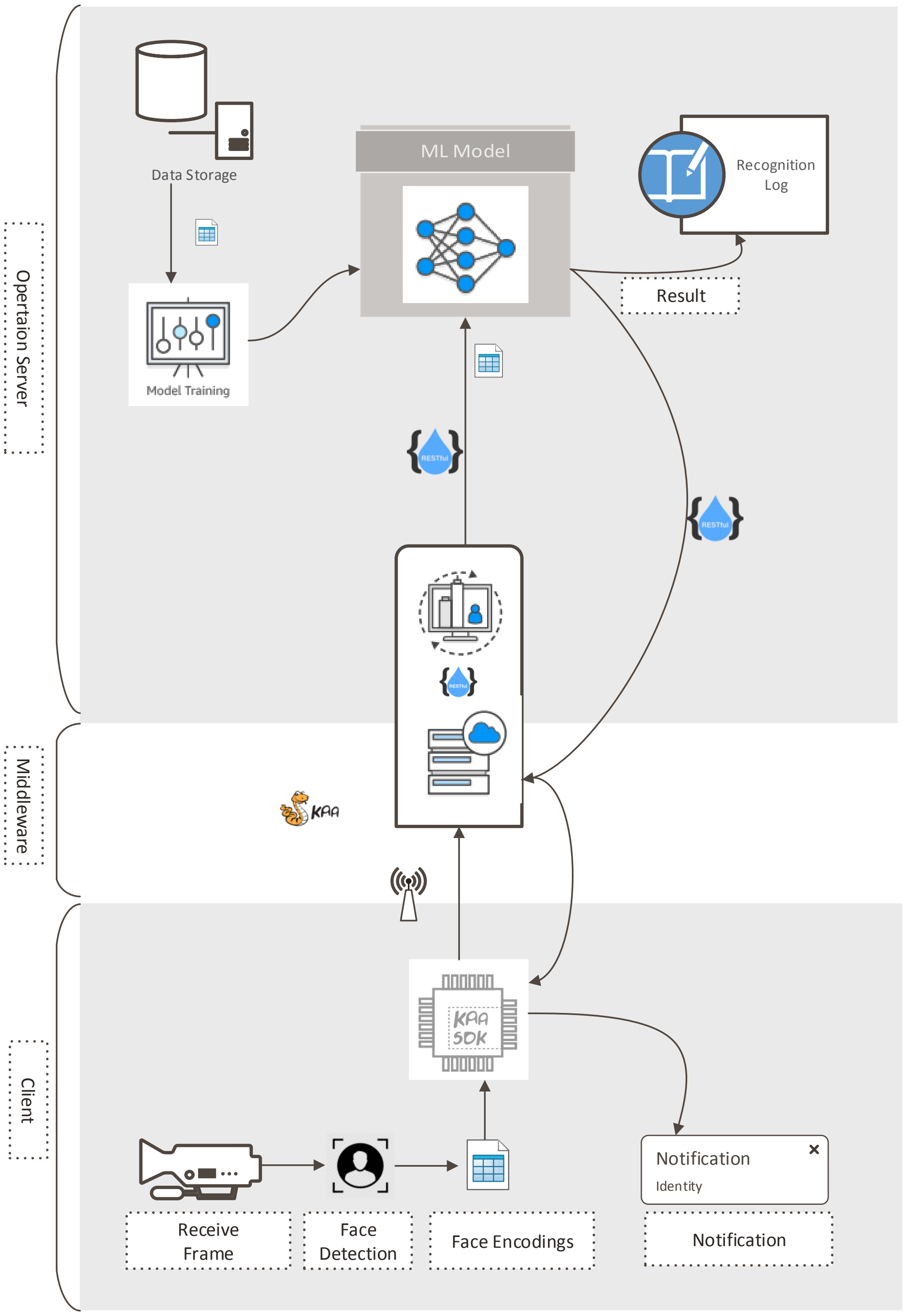


Figure 7: Architecture of Face Recognition System in IoT

3.3 Implementation

Our real time face detection with IP camera has four different modules/components. These components together make the system work.

1. IP camera: An Internet Protocol camera, or IP camera, is a type of digital video camera commonly employed for surveillance, and it is used send and receive data via a computer network and the Internet. Although most cameras that do this are webcams, the term IP camera is usually applied only to those used for surveillance that can be directly accessed over a network connection.
   To develop the real-time face recognition with IP camera, we used Dahua HD Mini IR Bullet Camera. It is a 5 MP Pro Series camera that offers high-resolution video. We attached it with client hardware (raspberry pi/laptop).

2. Client hardware: A client is a computer or a program that, as part of its operation, relies on sending a request to another program or a computer hardware or software that accesses a service made available by a server (which may or may not be located on another computer).

   We have used both laptop and raspberry pi 3 model B as client to evaluate performance of different scenarios. Configuration of client:

Table 1: Configuration of hardwires used

	Raspberry pi	Laptop
OS	Rasbian (linux)	Ubuntu (linux)
RAM	1 GB	8 GB
Processing speed	1.2 GHz	2.4 GHz
Number of CPU	4	4

3. Middleware: The middleware is a software layer or a set of sub-layers set between the technological and the application levels. Its feature of hiding the details of different technologies, setting the complexity of transportation through network protocols hidden in an API, for the development of the specific application enabled by the IoT infrastructures.

   For implementing an IoT scenario, we have used Kaa IoT Platform [22], an enterprise IoT middleware (PaaS). All the IoT protocols, regulations and security measurement is ensured with this service. In our development process, we have used the open source version of Kaa platform running on a ubuntu virtual machine in server. In Kaa, we created an application with administration and developer credentials that receives face data of a dedicated format. No other garbage data can pass through the middleware. To transfer data to Kaa, dedicated SDK had to be used, which is implemented by the very installed middleware.

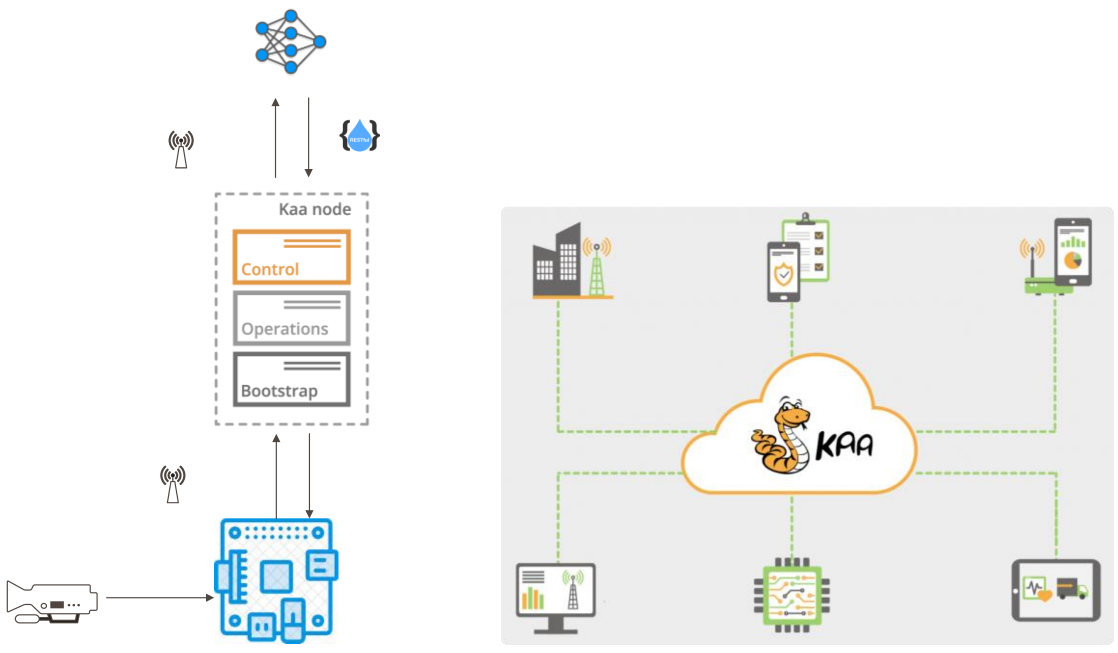


Figure 7: Middleware Kaa

So, the program running in Raspberry Pi is designed with that SDK and an operational data transfer module. The result notification is also received via that SDK and an operational notification receiver module.

4. Operation Server: In computing, a server is a computer program or a device that provides functionality for other programs or devices, called "clients". This architecture is called the client–server model, and a single overall computation is distributed across multiple processes or devices. Servers can provide various functionalities, often called "services", such as sharing data or resources among multiple clients, or performing computation for a client. A single server can serve multiple clients, and a single client can use multiple servers.

   We took a VM with 64 bit Ubuntu 14.04 and 8 GB RAM as edge server in our local network. For Cloud server, we chose Microsoft Azure’s Standard B2ms (2 vcpus, 8 GB memory) Ubuntu VM.

3.4 Development

In our development process, firstly, we deployed middleware kaa, both in edge Cloud and core Cloud to demonstrate an IoT framework, which gives us a web server, maintains security issues and transport protocols for both clouds.

3.4.1 Configuring IoT middleware platform, Kaa

As mentioned earlier, we are using Kaa IoT platform as middleware. First we set up kaa environment in server. The pre-requisite for setting up a kaa environment are:

1. Ubuntu or Debian system
2. 64 bit OS
3. 8 GB RAM
4. Oracle JDK 8
5. PostgreSQL 9.4
6. MariaDB 5.5
7. MongoDB
8. Zookeeper 3.4.5

First of all, we set up environment and install all the pre-requisites that are listed above. After that we downloaded the community addition pre-built packages of kaa from the official website and installed it. Finally, we configured the network interface for the operations and Bootstrap services by specifying a host name and ip address that will be visible to the devices. We did the same thing in Edge Cloud and Core Cloud. This is how we had kaa server up and running.

Then we added a tenant developer under tenant admin. Tenant admin can add application and developer can configure the application as per requirement. We added an application named “face recognition”. With kaa and client side SDK, device can be configured for important parameters such as sample period of uploading logs, what data types and how those will be transferred by creating application common type library, from the deployed kaa webserver. We configured sampling period and types of data to be received with one configuration scheme, a data collection scheme and a notification scheme. The 3 schemas are mentioned below in the appendix section in json format. With these schemas, kaa webserver provides generic control over connected devices.

We are using a backend application to analyze the data so we used the “Rest Log Appender” feature that sends the data to particular host via RESTful call. In our case, we are using the same server for analysis. So we set the host to be localhost with the port 9000.
After configuration, we can download kaa SDK. We have used java to implement data transfer using kaa’s java SDK.

3.4.2 Client side application

As client, we used Raspberry pi and laptop with IP camera. We used linux environment to develop client applications. To set up client environment we installed Oracle JDK 8, `openCV` [23], `dlib`[24] and `face_recognition`[25] for python. Client side application is a combination of 3 different modules running in parallel.

1. Detection and Encode Module: It detects face using HOG model from the video stream of ip camera. It also encodes the face into 128 floating point feature vector and saves it to a json file, along with timestamp, camera id and location of the face in the frame. It is implemented with python. Opens source libraries like `openCV`, `dlib`, `face_recognition` are used in this process. This is the feature extraction phase. The json file sizes 3.9kb per face regardless the size of image or frame or resolution. Of course better resolution would increase the accuracy of recognition.
   By doing this pre-processing of data at this side of the network, we are basically reducing the size of data to be transmitted. Transportation is becoming the bottleneck of Cloud based computing system. So, by reducing the size of data to be sent, we can save bandwidth. Less bandwidth and less data means faster and efficient transportation. The data to be sent to server is in json format. A sample data is attached in the appendix section.

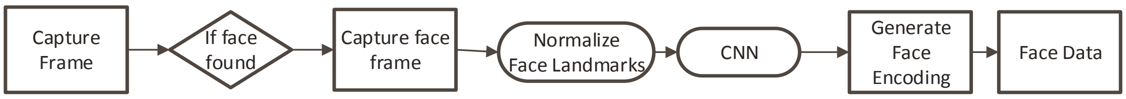


Figure 8: Flow diagram of Detection and Encode Module

2. Data Transfer Module: It follows the java program architecture defined by kaa. It uses the kaa SDK to send data from client to kaa server. It sends the face encoding data as string saved in the json file and a timestamp as long data type. Data upload sampling period is defined from configuration schema. For logging data to kaa, we used open source java library Simple Logging Façade for Java (slf4j) [26]. It serves as an abstraction for various logging frameworks.

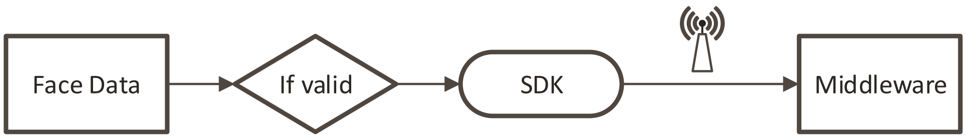


Figure 9: Flow diagram of Data Transfer Module

3. Notification Module: This java program catches the notification sent from the server. It is also compiled with kaa SDK. It is programmed to calculate the client processing time, client to server latency, prediction time, server to client latency and total time.

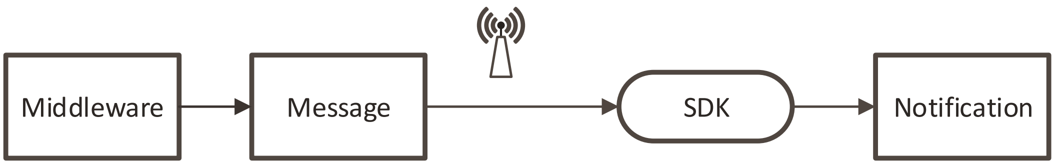
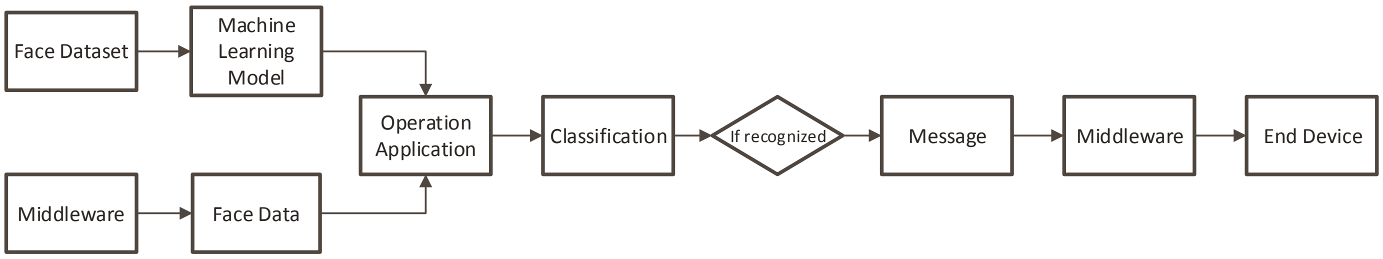


Figure 10: Flow diagram of Notification Module

3.4.3 Server side application

The application is developed in python. It uses python machine learning framework scikit-learn [27] for classifying face from face data model. First it trains a knn model using dataset of 1000 pictures. We have set the distance threshold to 6 for the classifier to classify accurately. It listens to a particular port for collecting data sent by kaa. Then it uses the knn model to predict the class label of the collected data, creates a notification text in json format and sends a notification to kaa using REST API. Then kaa sends the notification to client device. We attached the schema of the notification.json in the



appendix section.

Figure 11: Flow diagram of server side application

Chapter 4

Results and Analysis

As detailed in development section, we have used Raspberry Pi 3 as our client IoT enabled device. We also took a survey running the client side application in a pc.
The following graph (figure 12) shows the time difference between raspberry pi 3 and pc for pre-processing the face image and feature extraction from IP camera connected and iterating ten times. Average time of this process is given in table 2.

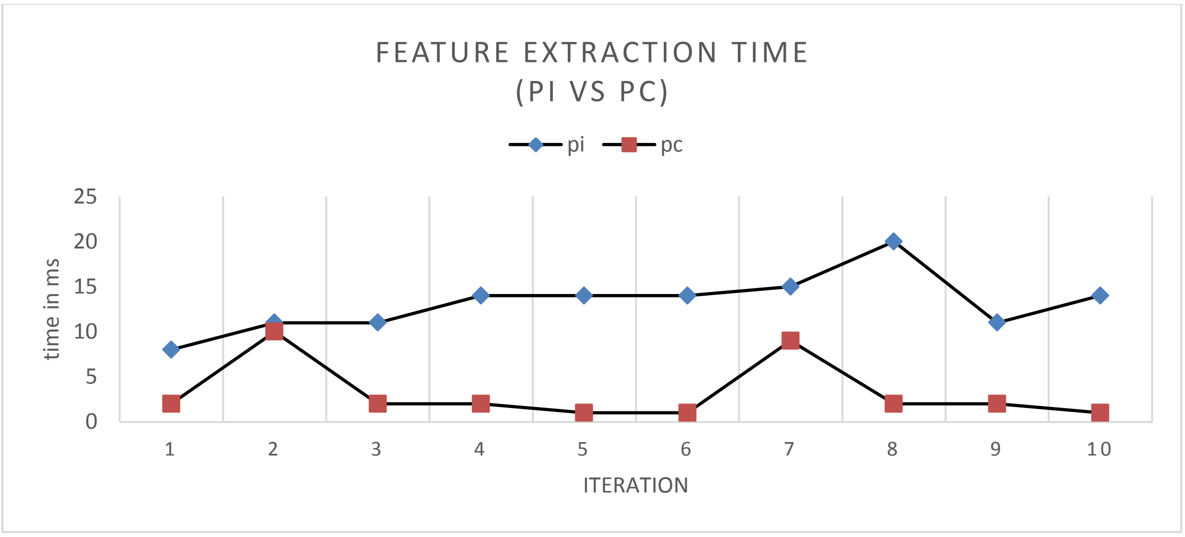


Figure 12: Feature extraction time difference in raspberry pi and pc

This graph of time signature fluctuates on basis of application's memory and processor's usage in the device. The time figure rises when there it finds multiple faces in one frame and has to separately compute them.

Table 2: Average time of feature extraction

Device	Time (ms)
pi	13.2
pc	3.2

For the same iterations, client to server (figure 10) and server to client (figure 11) network latency is monitored. This graph differentiates edge Cloud and core Cloud, in terms of time a json data transfer. Table 3 and table 4 correspondingly gives the average network latency for both direction.

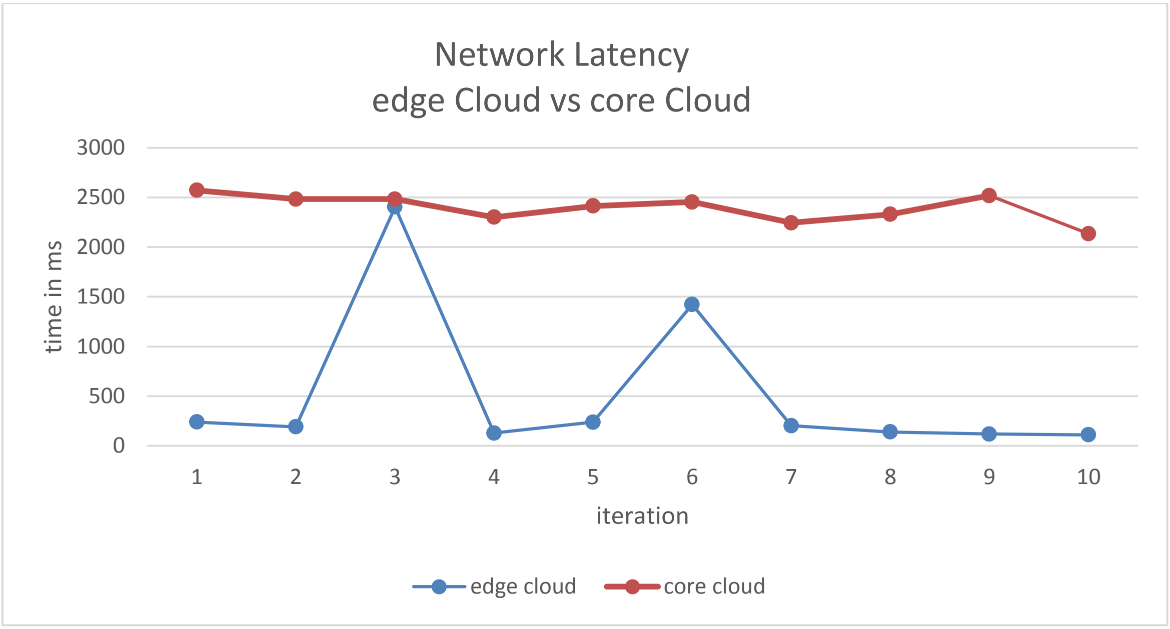


Figure 13: Network latency for edge Cloud and core Cloud (Client to Server)

Table 3: Average network latency for edge Cloud and core Cloud (Client to Server)

Server	Time in ms
Edge Cloud	520
Core Cloud	2393

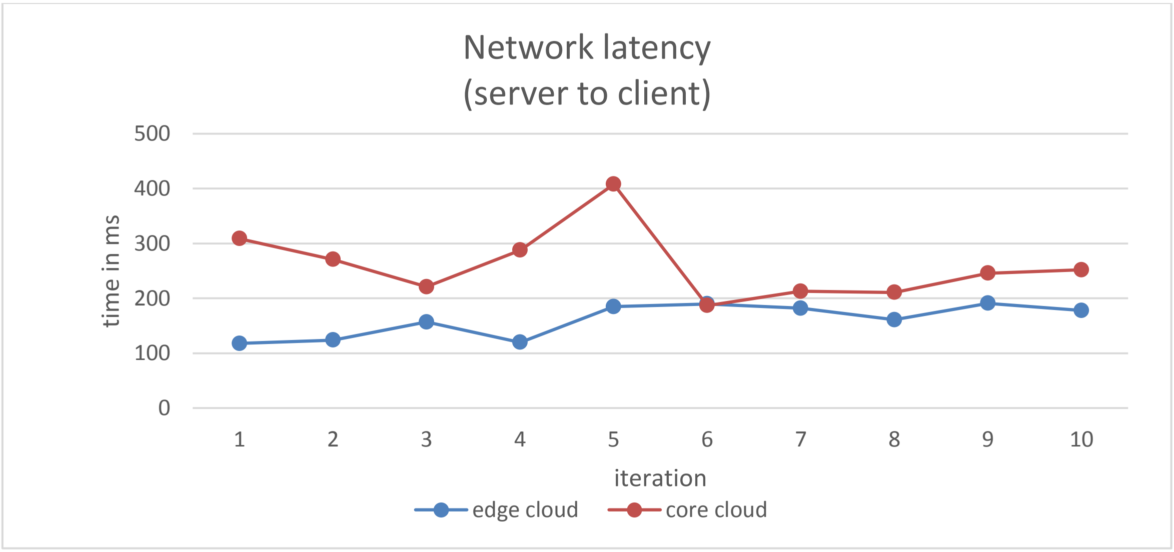


Figure 14: Network latency for notification (Server to Client)

Table 4: Average network latency for notification (Server to Client)

Server	Time in ms
Edge Cloud	160
Core Cloud	260

These graphs fluctuate according to the usage of the edge network usage in the facility.

In to the server side application, classification time is measured both in edge server and Cloud server, between 1000 faces (table 5).

Table 5: Classification time in server (Dataset – 1000 face image)

Server	Time (ms)
Edge Cloud	40
Core Cloud	33

Between the differentiation of edge and core Cloud, the whole face recognition process takes ~600ms in edge server and ~3000ms in Cloud server, regarding our facility’s Internet connection.

By looking at the overall performance of our developed system, response time in edge Cloud give less response time than core Cloud. In our thesis, we can say that, in case of IoT, keeping all the other study issues such as security, data storage and scalability issues aside, edge computing suits better than Cloud computing.

For extraction on more performance measure, figure 15 shows the time the server application in our edge Cloud facility takes to classify face for different size of dataset. We took 1000 face images for our initial testing. Furthermore, the results below are shown based on 2000, 3000, 4000 and 5000 face images in face dataset model. Table 6 gives the average time of the process for these different datasets.

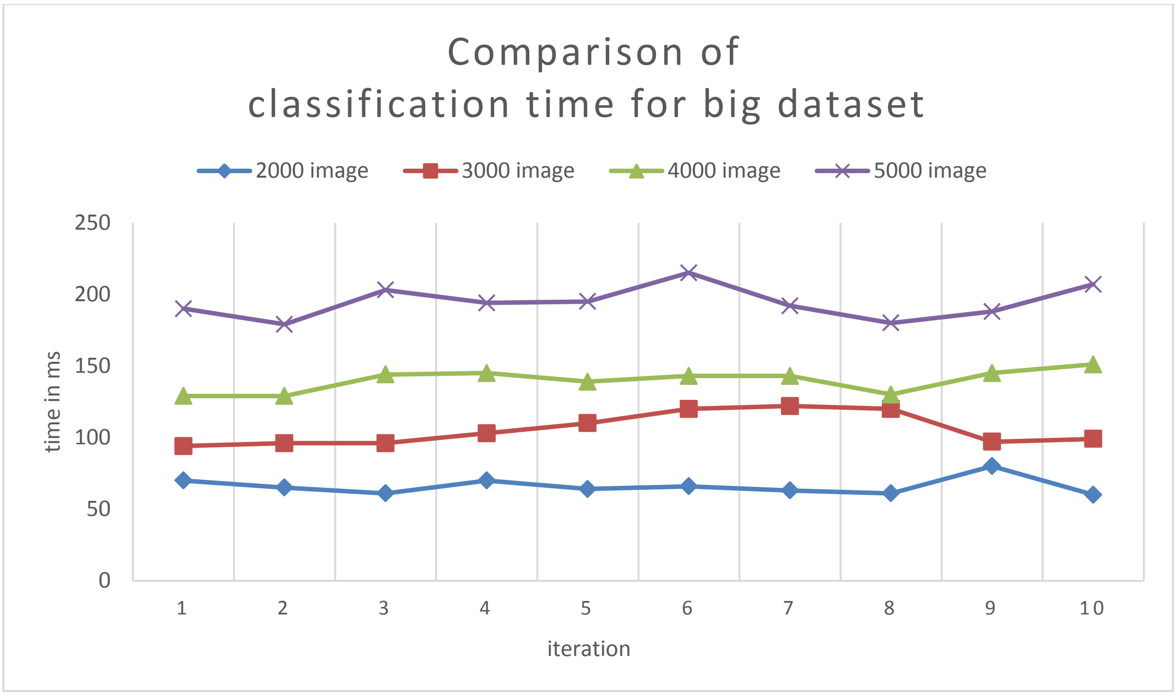


Figure 15: Classification time for different scale of dataset

Table 6: Average classification time for different size of dataset

Number of faces in dataset	Time in ms
2000	66
3000	106
4000	138
5000	194

All the machine learning and classification is done using python's scikit-learn library.

Table 7 shows the difference in processing time of our whole face recognition system using edge Cloud and core Cloud using raspberry pi as client device.

Table 7: Average total time of Face Recognition process

Server	Time (ms)
Edge Cloud	733
Core Cloud	2700

We can see that, the deployed process in core Cloud shows nearly 4 times processing time compared to edge Cloud.

All the network data are collected using university facility Internet connection.

Chapter 5

Conclusion

The main contribution of this thesis are:

1) Design and develop a real-time face-recognition system by integrating IoT platform and Cloud computing by using open-source tools and libraries.
2) Monitor and analyze the network and hardware performance of the system.
3) Compare the performance between edge Cloud and core Cloud for this IoT application.

In edge computing we want to put the computing at the proximity of data sources. This have several benefits compared to traditional Cloud-based computing paradigm. Here, we can see that, we can reduce network client to server network latency by ~73% (2700ms to700ms), regarding our facilities Internet connection, by positioning the server into an edge Cloud rather than core Cloud (MS Azure). Also implementing feature extraction process in client side application reduces the size of the data to be transferred, thus, taking low bandwidth allocation. Referring to our research, we can say commercial Cloud platforms are not well suited for IoT, which leads to the emergence of edge computing for better performance.

References

[1] Bound, J and Perkins, CE, “Evolution of the Internet core and edge: Ip wireless networking”, Proceedings of USENIX Annual Technical Conference, Boston, MA, USA, 2001.

[2] S. Islam and J.-Ch. Grégoire, “Network Edge Intelligence for the Emerging Next-Generation Internet,” Future Internet, vol. 2, no. 4, pp. 603–623, 2010.

[3] H. Chang, A. Hari, S. Mukherjee, and T. Lakshman, “Bringing the Cloud to the edge,” in Proc. IEEE Conf. on Computer Communications Workshops (INFOCOM WKSHPS), Toronto, ON, Canada, May 2014.

[4] N. Muslim and S. Islam, “Face recognition in the Edge Cloud,” Proc. Int. Conf. Imaging, Signal Process. Commun. - ICISPC 2017, pp. 5–9, 2017.

[5] L. Atzori, A. Iera, and G. Morabito, “The Internet of Things: A survey,” Comput. Networks, vol. 54, no. 15, pp. 2787–2805, 2010.

[6] K. Ashton, “That Internet of Things thing,” RFiD J., vol. 22, no. 7, pp. 97–114, 2009.

[7] H. Sundmaeker, P. Guillemin, P. Friess, S. Woelfflé, “Vision and challenges for realising the Internet of Things,” Cluster of European Research Projects on the Internet of Things—CERP IoT, 2010.

[8] J. Gubbi, R. Buyya, M. Palaniswami, and S. Marusic, “Internet of Things (IoT): A vision, architectural elements, and future directions,” Futur. Gener. Comput. Syst., vol. 29, no. 7, pp. 1645--1660, 2013.

[9] P. Mell and T. Grance, “Draft NIST working definition of Cloud computing,” Referenced on June. 3rd, 2009 Online at http://csrc.nist.gov/groups/SNS/Cloud-computing/index.html, 2009.

[10] M. Armbrust et al., “A view of Cloud computing,” Commun. ACM, vol. 53, no. 4, pp. 50–58, 2010.

[11] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge Computing: Vision and Challenges,” IEEE Internet Things J., vol. 3, no. 5, pp. 637–646, 2016.

[12] A. Greenberg et al., “The Cost of a Cloud: Research Problems in Data Center Networks,” ACM SIGCOMM Computer Commun. Review, vol. 39, no. 1, pp. 68–73, Jan. 2009.

[13] E. Cuervo et al., “MAUI: Making Smartphones Last Longer with Code Offload,” Proc. 8th ACM MobiSys, 2010.

[14] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, “The case for VM-based cloudlets in mobile computing,” IEEE Pervasive Comput., vol. 8, no. 4, pp. 14–23, Oct./Dec. 2009

[15] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the Internet of things,” in workshop on Mobile Cloud computing. ACM, 2012.

[16] V. Rijmenam, M., “Self-driving cars will create 2 petabytes of data, what are the big data opportunities for the car industry,” 2017.

[17] S. Islam and J.-Ch. Grégoire, “Giving users an edge: A flexible Cloud model and its application for multimedia,” Future Generation Computer System, vol. 28, no. 6, pp. 823–832, 2012.

[18] X. Zhu and D. Ramanan, “Face Detection, Pose Estimation, and Landmark Localization in the Wild,” Comput. Vis. Pattern Recognit. (CVPR), 2012 IEEE Conf., pp. 2879–2886, 2012.

[19] F. Schroff, D. Kalenichenko, and J. Philbin, “Facenet: A unified embedding for face recognition and clustering,” In Proc. CVPR, 2015.

[20] J. Stallkamp, H. K. Ekenel, R. Stiefelhagen, “Video-based Face Recognition on Real-World Data,” ICCV, 2007.

[21] Zhang, Zhifei, Song, Yang, and Qi, Hairong, “Age Progression/Regression by Conditional Adversarial Autoencoder”, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[22] Kaa iot platform. https://www.kaaproject.org

[23] G. Bradski, “The OpenCV Library,” Dr. Dobb's Journal of Software Tools, 2000.

[24] King D. E., “Dlib-ml: A Machine Learning Toolkit,” Journal of Machine Learning Research, vol. 10, pp. 1755-1758, 2009.

[25] Open source face recognition library for python. https://github.com/ageitgey/face_recognition

[26] Simple Logging Facade for Java (SLF4J), java logging api. https://www.slf4j.org

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, “Scikit-learn: Machine Learning in Python,” Journal of Machine Learning Research, vol. 12, pp. 2825-2830, 2011.

Appendix A

Kaa Scheamas

1. Configuration schema: this defines the default sampling period of data upload from device.

```
{
"type":"record",
"name":"Configuration",
"namespace":"org.kaaproject.kaa.schema.sample",
"fields":[{
        "name":"samplePeriod",
        "type":"int","by_default":1
        },
        {
        "name":"__uuid",
        "type":[{
                "type":"fixed",
                "name":"uuidT",
                "namespace":"org.kaaproject.configuration",
                "size":16
                },
                "null"],
        "displayName":"Record Id","fieldAccess":"read_only"
        }
],
"version":1,
"displayName":"Configuration"
}
```

With the configuration schema, end device’s sample period can be generically configured from kaa webserver. To identify end device multiple client, uuid is fetched.

2. Data collection schema: this defines the structure of the data to be received.

   ```
   {
     "type" : "record",
     "name" : "TDataCollection",
     "namespace" : "org.kaaproject.kaa.schema.sample",
     "fields" : [ {
             "name" : "face",
             "type" : {
                     "type" : "string",
                     "avro.java.string" : "String"
                     }
             }, {
             "name" : "timeStamp",
             "type" : "long"
     } ],
     "version" : 1,
     "dependencies" : [ ],
     "displayName" : "TDataCollection"
   }
   ```

   This schema defines the data type of logging collected data from device, which are, ‘face’ which consists json structured data with face encoding, timestamp and location in the frame, and timestamp – the timestamp of data upload from device.

3. Notification schema: this defines the structure of the notification to be sent from kaa.

   ```
   {
     "type" : "record",
     "name" : "Notification",
     "namespace" : "org.kaaproject.kaa.schema.example",
     "fields" : [ {
             "name" : "message",
             "type" : {
             "type" : "string",
             "avro.java.string" : "String"
             }
     } ],
     "version" : 1,
     "dependencies" : [ ]
   }
   ```

   Notification to be sent after recognition phase, which is named ‘message’ and data type string.

Sample Face Data

Face data in json structure, generated from client side application with 128d face encoding.

```
{
   "timestamp":"2018-09-12 22:28:04.854874",
   "face_encoding":[
      -0.03991749510169029,
      0.10491644591093063,
      0.0908227413892746,
      -0.008847326040267944,
      -0.07960712909698486,
      -0.042521778494119644,
      -0.053688082844018936,
      0.00014458224177360535,
      0.12723200023174286,
      -0.03247608616948128,
      0.23210051655769348,
      -0.014697907492518425,
      -0.2446957230567932,
      -0.028651222586631775,
      -0.003507627174258232,
      0.09977242350578308,
      -0.14234806597232819,
      -0.06628001481294632,
      -0.1056591123342514,
      -0.1179349422454834,
      0.044296085834503174,
      0.011001733131706715,
      0.0131568294018507,
      0.005964099895209074,
      -0.09541729837656021,
      -0.2940107583999634,
      -0.07901164889335632,
      -0.12461303174495697,
      0.0903143510222435,
      -0.18279217183589935,
      0.010395172983407974,
      0.02964000217616558,
      -0.11968293786048889,
      -0.03763097897171974,
      -0.018499260768294334,
      0.02241109125316143,
      0.006105402484536171,
      -0.09956780076026917,
      0.15293844044208527,
      -0.03480614721775055,
-0.151466965675354,
-0.056745123118162155,
0.0028462838381528854,
0.2556203007698059,
0.18022345006465912,
0.04067671298980713,
0.019054628908634186,
-0.053532302379608154,
0.059580810368061066,
-0.2805930972099304,
0.027582716196775436,
0.1558370143175125,
0.01873459480702877,
0.11697729676961899,
0.07859192043542862,
-0.1667841076850891,
0.04074300825595856,
0.09162883460521698,
-0.1402018666267395,
0.0398016981780529,
0.05118858441710472,
-0.09438028186559677,
-0.0348590612411499,
-0.11478354781866074,
0.21047230064868927,
0.06895994395017624,
-0.08207542449235916,
-0.12454008311033249,
0.11185845732688904,
-0.1558324247598648,
-0.036294132471084595,
0.10316243767738342,
-0.11435894668102264,
-0.16498178243637085,
-0.22588521242141724,
0.09368528425693512,
0.36699917912483215,
0.19930100440979004,
-0.13919590413570404,
0.03991914913058281,
-0.1053084135055542,
-0.014978972263634205,
0.04010719433426857,
0.022071311250329018,
-0.06790884584188461,
0.0029971925541758537,
-0.09384240210056305,
0.0677608773112297,
0.1655023992061615,
-0.08613958954811096,
        0.03404010087251663,
        0.1837223768234253,
        -0.046676866710186005,
        0.03188410773873329,
        -0.001710508018732071,
        0.018096420913934708,
        -0.08234531432390213,
        0.050373777747154236,
        -0.06457515805959702,
        0.019401784986257553,
        0.13476905226707458,
        -0.13644984364509583,
        0.029162578284740448,
        0.06979145109653473,
        -0.17401345074176788,
        0.09501060843467712,
        -0.012585272081196308,
        -0.031326793134212494,
        0.043887048959732056,
        0.005560955032706261,
        -0.07175079733133316,
        -0.03547824174165726,
        0.2355225831270218,
        -0.25204572081565857,
        0.23794953525066376,
        0.24037323892116547,
        0.038678523153066635,
        0.13249173760414124,
        0.038171716034412384,
        0.1466071605682373,
        -0.044453300535678864,
        -0.06796195358037949,
        -0.1310708373785019,
        -0.027809495106339455,
        0.020307786762714386,
        -0.013669928535819054,
        -0.054414622485637665,
        -0.000889856368303299
    ],
    "location_in_frame":"[3676L, 2108L, 4960L, 3392L]"
}
```

Programming Code

Client Side Application

detection_encode.py
Note: This program requires some packages to be installed:
openCV for python, scikit-learn dlib, face_recognition

Source code:

```
#AsifAhmed011141068
import face_recognition
import cv2
import datetime
import numpy as np
import json
import threading



# Get a reference to webcam #0 (the default one)
#webcam
#video_capture = cv2.VideoCapture(0)
#ipcam
video_capture = 
cv2.VideoCapture('rtsp://admin:admin123@10.10.5.117:554/cam/realmonitor?channel=1
&subtype=0')



def make_480p():
    video_capture.set(3, 640)
    video_capture.set(4, 480)

def change_res(width, height):
    video_capture.set(3, width)
    video_capture.set(4, height)

def face_encode_gen(face_locations):
    print (" thread start")
    if len(face_locations) !=0:
        face_encoding = []
        face_encoding = face_recognition.face_encodings(rgb_small_frame, face_locations)

        for top, right, bottom, left in face_locations:
```

```
        # Scale back up face locations since the frame we detected in was scaled to 1/4 size
and add paddding 15px
            location_in_frame = []
            top *= 4
            right *= 4
            bottom *= 4
            left *= 4
            location_in_frame = [left, top, right, bottom]


            face_encoding = np.array(face_encoding)

            face_encoding = face_encoding.tolist()


            face_data = {
                    "timestamp" : str(timestamp),
                    "location_in_frame" : str(location_in_frame),
                    "face_encoding" : face_encoding[0]
                    }

            #-----------------save json file for java to fetch-------------------------#
            with open("test/face_"+ timestamp +"_data.json", 'w') as outfile:
                    json.dump(face_data, outfile, encoding='utf-8', separators=(',', ':'),
                    sort_keys=False, indent=4)
            print(" - created - : " + "face_" + timestamp +"_data.json")
            #-------------------------------------------------------------------------#



make_480p()


process_this_frame = True
fps = 1
video_capture.set(cv2.CAP_PROP_FPS, fps)

while True:

    # Grab a single frame of video
    ret, frame = video_capture.read()

    # Resize frame of video to 1/4 size for faster face recognition processing
    small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)
    #half_frame = cv2.resize(frame, (0, 0), fx=0.8, fy=0.8)


    # Convert the image from BGR color (which OpenCV uses) to RGB color (which
face_recognition uses)
    rgb_small_frame = small_frame[:, :, ::-1]
    # Only process every other frame of video to save time
    if process_this_frame:
        timestamp = str(datetime.datetime.now())
        face_locations = face_recognition.face_locations(rgb_small_frame)
        #print(face_locations)
        try:
            if len(face_locations) !=0:
                threading.Thread(target=face_encode_gen, args = (face_locations,)).start()
            else:
                print("no face")
        except:
            print("q")
    process_this_frame = not process_this_frame


    #Display the resulting image
    #cv2.imshow('Video', small_frame)

    # Hit 'q' on the keyboard to quit
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release handle to the webcam
video_capture.release()
cv2.destroyAllWindows()
```

DataTransfer.java

Note: To run this application, build the source code with dedicated Kaa SDK and slf4j.jar.

Source code:

```
//AsifAhmed011141068

import org.kaaproject.kaa.client.DesktopKaaPlatformContext;
import org.kaaproject.kaa.client.Kaa;
import org.kaaproject.kaa.client.KaaClient;
import org.kaaproject.kaa.client.SimpleKaaClientStateListener;
import org.kaaproject.kaa.client.configuration.base.ConfigurationListener;
import org.kaaproject.kaa.client.configuration.base.SimpleConfigurationStorage;
import org.kaaproject.kaa.client.logging.strategies.RecordCountLogUploadStrategy;
import org.kaaproject.kaa.schema.sample.Configuration;
import org.kaaproject.kaa.schema.sample.TDataCollection;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.io.IOException;
import java.util.Random;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.TimeUnit;

import java.nio.file.FileSystems;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardWatchEventKinds;
import java.nio.file.WatchEvent;
import java.nio.file.WatchKey;
import java.nio.file.WatchService;
import java.nio.file.Files;
import java.io.*;
import java.nio.*;

/**
 * Class implement functionality for First Kaa application. Application send face
encodings data
 * from the Kaa endpoint with required configured sampling period
 */
public class DataTransfer {

    private static final long DEFAULT_START_DELAY = 1000L;

    private static final Logger LOG = LoggerFactory.getLogger(FirstKaaDemo.class);

    private static KaaClient kaaClient;
```

```
   private static ScheduledFuture<?> scheduledFuture;
   private static ScheduledExecutorService scheduledExecutorService;

   public static long startTime;
   public static long endTime;
   public static int flag=1;

   public static void main(String[] args) {
      LOG.info(FirstKaaDemo.class.getSimpleName() + " app starting!");

      scheduledExecutorService = Executors.newScheduledThreadPool(2);

      //Create the Kaa desktop context for the application.
      DesktopKaaPlatformContext desktopKaaPlatformContext = new
DesktopKaaPlatformContext();

      /*
       * Create a Kaa client and add a listener which displays the Kaa client
       * configuration as soon as the Kaa client is started.
       */
      kaaClient = Kaa.newClient(desktopKaaPlatformContext, new
FirstKaaClientStateListener(), true);

      /*
       *  Used by log collector on each adding of the new log record in order to check
whether to send logs to server.
       *  Start log upload when there is at least one record in storage.
       */
      RecordCountLogUploadStrategy strategy = new
RecordCountLogUploadStrategy(1);
      strategy.setMaxParallelUploads(1);
      kaaClient.setLogUploadStrategy(strategy);

      /*
       * Persist configuration in a local storage to avoid downloading it each
       * time the Kaa client is started.
       */
      kaaClient.setConfigurationStorage(new
SimpleConfigurationStorage(desktopKaaPlatformContext, "saved_config.cfg"));

      kaaClient.addConfigurationListener(new ConfigurationListener() {
         @Override
         public void onConfigurationUpdate(Configuration configuration) {
            LOG.info("Received configuration data. New sample period: {}",
configuration.getSamplePeriod());

onChangedConfiguration(TimeUnit.SECONDS.toMillis(configuration.getSamplePeriod(
)));
         }
```

```
        });

        //Start the Kaa client and connect it to the Kaa server.
        kaaClient.start();

        LOG.info("--= Press any key to exit =--");
        try {
            System.in.read();
        } catch (IOException e) {
            LOG.error("IOException has occurred: {}", e.getMessage());
        }
        LOG.info("Stopping...");
        scheduledExecutorService.shutdown();
        kaaClient.stop();
    }

    /*----------------------------------------------------------------------------------*/
    // reads whole json file as a string

    private static String readEncodings(String path) throws Exception{

        String data = "";
        //data = new String(Files.readAllBytes(Paths.get(path)));
        data = new String(Files.readAllBytes(Paths.get(path)));
        for(int i=0; i<5;i++)
                        {
                            LOG.info("-");

                        }
        System.out.println("Data to send : " + path);
        for(int i=0; i<5;i++)
                        {
                            LOG.info("-");

                        }
        return data;
    }

    // gets just created face_ json file
        private static String getEncodings() throws Exception{

                Path testFolder = Paths.get("test");
                WatchService watchService =
FileSystems.getDefault().newWatchService();
                testFolder.register(watchService,
StandardWatchEventKinds.ENTRY_CREATE);

                boolean valid = true;
                do {
                        WatchKey watchKey = watchService.take();
                        for (WatchEvent event : watchKey.pollEvents()) {
                              WatchEvent.Kind kind = event.kind();
                              if
(StandardWatchEventKinds.ENTRY_CREATE.equals(event.kind())) {
                                    // reads only when a json file is created
                                      try{
                    String fileName = event.context().toString();

                    if(fileName.endsWith("_data.json") && fileName.startsWith("face_")){
                      for(int i=0; i<5;i++)
                      {
                        LOG.info("-");

                      }
                      System.out.println("Captured:" + fileName);
                      for(int i=0; i<5;i++)
                      {
                        LOG.info("-");

                      }
                      return readEncodings("test/"+fileName);  // sends created file path
and returns json formatted string
                    }
                  } catch(Exception e){
                    System.out.println("error");
                  }
                              }
                        }
                        valid = watchKey.reset();

                  } while (valid);
      return ("");
    }


    private static void onKaaStarted(long time) {
      if (time <= 0) {
        LOG.error("Wrong time is used. Please, check your configuration!");
        kaaClient.stop();
        System.exit(0);
      }

      scheduledFuture = scheduledExecutorService.scheduleAtFixedRate(
          new Runnable() {
            @Override
            public void run() {

                        try{
```

```
                        String face = "";
                       face = getEncodings();


                        if(face != ""){

                            startTime= System.currentTimeMillis();
                            kaaClient.addLogRecord(new TDataCollection(face, startTime));

                            LOG.info("Sampled data: {}",face);

                        }
                            }
                            catch(Exception e)
                            {
                                System.out.println("error");
                            }



                                                                        flag=0;
                                                                        System.out.println("-------");
                                                              //}
                }
            }, 0, time, TimeUnit.MILLISECONDS);
}

private static void onChangedConfiguration(long time) {
    if (time == 0) {
        time = DEFAULT_START_DELAY;
    }
    scheduledFuture.cancel(false);

    scheduledFuture = scheduledExecutorService.scheduleAtFixedRate(
            new Runnable() {
                @Override
                public void run() {

                            try{

                        String face = "";
                        face = getEncodings();


                        if(face != ""){

                            startTime= System.currentTimeMillis();
                            kaaClient.addLogRecord(new TDataCollection(face, startTime));
                            for(int i=0; i<5;i++)
```

```
                                LOG.info("Sampled Data: {}",face);

                              }

                            }
                                catch(Exception e)
                                {
                                   System.out.println("error");
                                }

                                                                 flag=0;
                                                                 System.out.println("---0---");
                                                           }
                //}
              }, 0, time, TimeUnit.MILLISECONDS);
    }

    private static class FirstKaaClientStateListener extends SimpleKaaClientStateListener
  {

        @Override
        public void onStarted() {
            super.onStarted();
            LOG.info("Kaa client started");
            Configuration configuration = kaaClient.getConfiguration();
            LOG.info("Default sample period: {}", configuration.getSamplePeriod());
            onKaaStarted(TimeUnit.SECONDS.toMillis(configuration.getSamplePeriod()));
        }

        @Override
        public void onStopped() {
            super.onStopped();
            LOG.info("Kaa client stopped");
        }
    }
}
```

NotificationSystem.java

Note: To run this application, build the source code with Kaa SDK and slf4j.jar

Source code:

```
//AsifAhmed011141068

import java.util.List;

import org.kaaproject.kaa.client.DesktopKaaPlatformContext;
import org.kaaproject.kaa.client.Kaa;
import org.kaaproject.kaa.client.KaaClient;
import org.kaaproject.kaa.client.SimpleKaaClientStateListener;
import org.kaaproject.kaa.client.notification.NotificationListener;
import org.kaaproject.kaa.client.notification.NotificationTopicListListener;
import org.kaaproject.kaa.common.endpoint.gen.Topic;
import org.kaaproject.kaa.schema.example.Notification;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.util.concurrent.TimeUnit;

public class NotificationSystem {

   private static final Logger LOG =
LoggerFactory.getLogger(NotificationSystemTestApp.class);

   public static void main(String[] args) {
      new NotificationSystemTestApp().launch();
   }

   private void launch() {
      // Create client for Kaa SDK
      KaaClient kaaClient = Kaa.newClient(new DesktopKaaPlatformContext(),
            new SimpleKaaClientStateListener() {
               @Override
               public void onStarted() {
                  LOG.info("Kaa SDK client started!");
               }
            },true);

      // Registering listener for topic updates
      kaaClient.addTopicListListener(new NotificationTopicListListener() {
         @Override
         public void onListUpdated(List<Topic> topicList) {
            LOG.info("Topic list updated!");
            for (Topic topic : topicList) {
               LOG.info("Received topic with id {} and name {}", topic.getId(),
topic.getName());
```

```
                }
            }
        });

        // Registering listener for notifications
        kaaClient.addNotificationListener(new NotificationListener() {
            @Override
            public void onNotification(long topicId, Notification notification) {

          int i;
                    for(i=0; i<=5;i++)
                    {
                            LOG.info("--");

                    }
        LOG.info("Received notification {} for topic with id {}", notification, topicId);
              long end= System.currentTimeMillis();
        String n= notification.toString();
        int len=n.length();

        i= n.indexOf("=");
        i++;
        String sub= n.substring(i,len-2);
        LOG.info("start time {}",sub);
        long start= Long.parseLong(sub);
        LOG.info("Total time required {} millis",end-start);
                    for( i=0; i<=5;i++)
                    {
                            LOG.info("--");

                    }
}
        });

        // Starts Kaa SDK client
        kaaClient.start();
    }
}
```

Server Side Application

This application is developed using the k-nearest-neighbors (KNN) algorithm for face recognition.

Algorithm Description:
The knn classifier is first trained on a set of labeled (known) faces and can then predict the person in an unknown image by finding the k most similar faces (images with closest face-features under eucledian distance, threshold = 0.6) in its training set, and performing a majority vote (possibly weighted) on their label.

For example, if k=3, and the three closest face images to the given image in the training set.

This implementation uses a weighted vote, such that the votes of closer-neighbors are weighted more heavily.

Usage:

1. Prepare a set of images of the known people you want to recognize. Organize the dataset images in a single directory with a sub-directory for each known person.

2. Then, call the 'train' function with the appropriate parameters. Make sure to pass in the 'model_save_path' if you want to save the model to disk so you can re-use the model without having to re-train it.

3. Catches face encodings from Kaa rest log appender and classifies data. Creates a message text and using REST api, makes middleware to send that message to client device as a notification, via notification module.

NOTE: This program requires some packages to be installed:
install pip, scikit-learn, face_recognition

Source code:

```
#AsifAhmed011141068
import datetime
import socket
import sys
import time as tt
import math
from sklearn import neighbors
import os
import os.path
import pickle
from PIL import Image, ImageDraw
import face_recognition
import numpy as np
from face_recognition.face_recognition_cli import image_files_in_folder
import logging as log
import json


log.basicConfig(filename='webcam.log',level=log.INFO)

ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg'}

#cmd= "curl -v -S -u [id]:[pass] -F 'notification={\"applicationId\": \"1\", \"schemaId\":
\"32769\", \"topicId\" : \"32769\" , \"type\" :\"USER\"  };type=application/json' -F
file=@notification.json
\"http://137.116.141.76:8080/kaaAdmin/rest/api/sendNotification\" | python -mjson.tool"

encodings = []

time=[]

def train(train_dir, model_save_path, n_neighbors=None, knn_algo='ball_tree',
verbose=False):

    if model_save_path == "trained_knn_model.clf":
        knn_clf = neighbors.KNeighborsClassifier(n_neighbors=n_neighbors,
algorithm=knn_algo, weights='distance')

    X = []
    y = []

    # Loop through each person in the training set
    for class_dir in os.listdir(train_dir):
        if not os.path.isdir(os.path.join(train_dir, class_dir)):
            continue

        # Loop through each training image for the current person
        for img_path in image_files_in_folder(os.path.join(train_dir, class_dir)):
            image = face_recognition.load_image_file(img_path)

            face_bounding_boxes = face_recognition.face_locations(image)

            if len(face_bounding_boxes) != 1:
                # If there are no people (or too many people) in a training image, skip the
image.
                if verbose:
                    print("Image {} not suitable for training: {}".format(img_path, "Didn't find a
face" if len(face_bounding_boxes) < 1 else "Found more than one face"))
            else:
                # Add face encoding for current image to the training set
                X.append(face_recognition.face_encodings(image,
known_face_locations=face_bounding_boxes)[0])

                y.append(class_dir)

    # Determine how many neighbors to use for weighting in the KNN classifier
    if n_neighbors is None:
        n_neighbors = int(round(math.sqrt(len(X))))
        if verbose:
            print("Chose n_neighbors automatically:", n_neighbors)

    #------------Create and train the KNN classifier------------------------------

    #Save the trained KNN classifier
    if model_save_path == "trained_knn_model.clf":
        knn_clf.fit(X, y)
        with open(model_save_path, 'wb') as f:
            pickle.dump(knn_clf, f)
        return knn_clf



def predict(face_encoding_array, clf=None, model_path=None, distance_threshold=0.6):

    if clf is None and model_path is None:
        raise Exception("Must supply classifier either thourgh clf or model_path")

    # Load a trained KNN model (if one was passed in)
    if clf is None:
        with open(model_path, 'rb') as f:
            clf = pickle.load(f)


    #if len(face_encoding_array) == 0:
    #    get_data_from_client()

    # Use the KNN model to find the best matches for the test face
    if model_path == "trained_knn_model.clf" :
        closest_distances = clf.kneighbors(face_encoding_array, n_neighbors=3,
return_distance=True)
        #print closest_distances

        are_matches = [closest_distances[0][i][0] <= distance_threshold for i in
range(len(face_encoding_array))]
        #print are_matches
        #result = clf.predict(face_encoding_array)
        #print "knn predict result:"
        #print (result)
        # Predict classes and remove classifications that aren't within the threshold
        return [(pred) if rec else ("unknown") for pred, rec in
zip(clf.predict(face_encoding_array), are_matches)]




def show_prediction(predictions, timestamp, location_in_frame, startTime, received_at,
predicted_at):

    #log and send notification to client
    for name in predictions:
        #if name != "unknown" :
        #name = name.encode("UTF-8")
        notification = str("Found - " + name + " - timestamp : " + timestamp + " -
location_in_frame : " + location_in_frame + " - times =" +
str(startTime)+","+str(received_at)+","+str(predicted_at))
        log.info(notification)
        print(notification)
        #create notification.json
        f=open("notification.json", "w+")
        f.write("{\"message\" :  \""+notification+"\"}")
        f.close()
        #send anotification.json
        os.system(cmd)



def get_data_from_client():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    print ('Socket created')
    #Bind socket to local host and port
    s.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)

    try:
        s.bind(("localhost", 9000)) #server and port number
    except socket.error as msg:
        print ('Bind failed. Error Code : ' + str(msg[0]) + ' Message ' + msg[1])
        s.close()
        sys.exit()
    print ('Socket bind complete')
    #Start listening on socket
    s.listen(5)
    print ('Socket now listening')

    #now keep talking with the client
    #wait to accept a connection - blocking call
    conn, addr = s.accept()
    print ('Connected with ' + addr[0] + ':' + str(addr[1]))

    data=conn.recv(1024*5)
    received_at = int(round((tt.time()+49.0866834)*1000))

    print data
    #print type(data)
    s.close()
    conn.close()

    jdata=data[data.find("{"):]
    try:
        #jdata=data[data.find("{"):]
        face_data_json=json.loads(jdata)
        #print(face_data_json)
        #print type(face_data_json)
        face = json.loads(face_data_json['event']['face'])
        #print(face)
        #print type(face)

        startTime = long(face_data_json['event']['timeStamp'])
        return face, startTime, received_at
    except ValueError, e:
        return "x",0,received_at

def face_recog():

    while True:

    # face_encoding


        face, startTime, received_at = get_data_from_client()
        #print face, startTime
        if(face != "x"):
            #get_data_from_client()

        # fetch face_encoding_str from face json / from camera app
            face_encoding = face['face_encoding']

            face_encoding = np.array(face_encoding)
            face_encoding_array = []
            face_encoding_array.append(face_encoding)

            timestamp = face['timestamp']
            #print timestamp

            #get location in frames from face json/ from camera app
            location_in_frame = face['location_in_frame']


            knn_predictions = predict(face_encoding_array,
model_path="trained_knn_model.clf")
            predicted_at=int(round((tt.time()+49.0866834) * 1000))
            print knn_predictions
            print "knn:"
            show_prediction(knn_predictions, timestamp, location_in_frame, startTime,
received_at, predicted_at)


if __name__ == "__main__":


    if os.path.isfile("trained_knn_model.clf"):
        print("KNN Already trained!")
    else :
        print("Training KNN classifiers...")
        knn_classifier = train("train", model_save_path="trained_knn_model.clf",
n_neighbors=1)
        print("Training complete!")

    face_recog()
```