Tamper-resistant IoT-based Video Surveillance with Blockchains

Elfat Esati

Zurich, Switzerland

Student ID: 15-702-103

Supervisor: Eryk Schiller

Date of Submission: January 30, 2021

University of Zurich

Department of Informatics (IFI)

Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

The Internet of Things (IoT), artificial intelligence (AI), and blockchain are one of the most disruptive technologies that scholars have begun evaluating. In one hand blockchain technology as a distributed decentralized peer to peer network promises to keep records of the various transactions that ever happened in a P2P network. On the other hand IoT allows virtual and physical objects to be connected together. However IoT alone comes with challenges which are centralized architecture, security, transparency, data integrity, issues with analysis of big data and vulnerability to attacks. AI plays a significant role as a strong analytic tool and delivers a scalable and accurate analysis of data in real-time. Similarly AI faces a number of challenges: centralized architecture, security and resource limitations. Alone any of the three domains would have the potential to alter business, leisure and society. But together the synergy they produce will be unprecedented. In this paper we show that the convergence of BC, AI, and IoT will close the gaps of each of these technologies to come up with a scalable and highly secured intellectual functioning. The aim of this paper is not only to explore the synthesis of the three domains but it will also present the design and development of an IoT architecture with Blockchain (BC) and AI to support access control with facial detection and recognition. We designed and implemented an IoT-based surveillance system incorporating all the three domains that jointly work together for authentication of people allowed to enter a building. By incorporating the most state of the art technologies and algorithms we proved that three domains complement each other's weaknesses.

Acknowledgments

I'm grateful for being part of the Communication Systems Group at University of Zurich and for the full support in encouraging me to start this work, preserve with it and finally publish it. It is my pleasure to express my deep sense of thankfulness to my mentor and guide Dr. Eryk Schiller for advising and sharing with me his immense knowledge and motivating me throughout this journey. He constantly helped me steering towards the right direction by advising me with the most state of the art technologies and perspectives to look at the thesis.

I owe a deep sense of gratitude to Prof. Stiller, the head of Communication Systems Group for his prompt inspirations and scholarly advice who fully trusted me and enabled me to complete the thesis at his department. Many thanks to the Student Financial Aid Office of the University of Zurich for financial support throughout my study period.

I'm extremely thankful to my parents and family for the encouragement and spiritual support they gave during all the years of my study.

Contents

Chapter 1

Introduction and Motivation

Internet of Things (IoT) is transforming the world of things into an autonomous world. It has a high impact in many industries such as manufacturing, transportation, automotive, consumer goods, and even healthcare will never be the same once IoT is applied. Thanks to the advances of the underlying technologies, IoT devices are being equipped with more powerful processors and are able to effectively distribute the processing load to other cores. This offers the opportunity to run more complex tasks in the IoT devices. However IoT comes with a number of challenges or gaps that still need to be improved such as: security issues, centralization and the vulnerability to attacks. On the other hand blockchains as a decentralized network is able to hold to records and transactions in blocks secured by cryptography. We can see that the advantages of blockchain converge with the disadvantages of IoT which can eventually transform how information is processes and stored. On the other hand Artificial Intelligence (AI) has a significant role as an accurate analysis of data in real time. However with the design and development of an efficient data analysis tool using AI comes with challenges: centralized architecture, security issues and the decisions made from AI are not transparent and recorded. Therefore integrating the blockchain with AI has the ability to produce a robust technology to resolve the several AI issues. AI as a black box lacks transparency, therefore with the transparency of blockchain by sharing the data in many nodes in a subsequent order provides a clear way for tracking back the data to the AI decision process.

Therefore the convergence of the three domains will bring a successful synergy that will transform how data is being processed, analyzed and stored. In this study we aim to explore the three mentioned emerging paradigms that will influence the future. Therefore the main goal of this research is the design and development of a Blockchain-enabled Intelligent IoT Architecture with Artificial Intelligence that will provide an efficient implementation of a use case while incorporating the three domains with the most state of the art technologies and applications. There are a number of attempts that shed light on the benefits of converging IoT, BC and AI [1] but most of them are either reviews or explorations but far away from a concrete implementation of a use case [2]. A more close research in this matter is an attempt [3] which proposes a BlockIoTIntelligence architecture as they call it that converges the blockchain, AI for IoT whose aim is to achieve the goal of big data analysis, security and centralization issues of IoT applications such as smart city, healthcare and smart transportation. With their findings they claim that BlockIoTIntelligence can mitigate the existing challenges and obtain high accuracy with a reasonable latency and security in the decentralized way. If narrowing down the scope to blockchain and IoT we see a slightly more research which attempts to close the gaps of IoT by removing the centralized control with the help of blockchain [4]. In another study [5] they attempted to find out the security and privacy in IoT gaps that could be filled with the help of the blockchain to ensure the reliability and availability of the data.

Having had a closer look on the research that shed light on the synthesis of the three domains we see a number of proposals and architecture but yet we do not see a real use case where the three technologies can complement each other. Therefore the use case we are designing and implementing provides an efficient way of converging blockchain, AI and IoT with the most current state-of-the-art approaches. We are entering an age where surveillance is becoming a norm and facial recognition technologies are hunting the streets. The roll out of facial recognition technologies is expected to become more prominent and an important step in improved security. On the other side, major data breaching has become common place and people worry about the type of personal information held by organizations. In the midst of the COVID-19 pandemic, attention have turned to facial recognition technology as a way to combat the spread of the virus.

Before introducing our own architecture an interesting use case that resembles our architecture is an implementation of a camera based sensor for monitoring room occupancy.

Hence to combat the many issues we have discussed and found in the above use case we will be using a low powered IoT device named ESP EYE which with the help of AI running on it will detect and recognize the individuals and grant or reject access. In the due time the ESP EYE will capture images of individuals and forwards them to the blockchain where it will be stored. The proposed architecture and implementation paves the way towards a strong architecture that fills the many gaps of the three domains. Therefore the main goal of this project is to design and implement an IoT-based surveillance system with blockchain and AI to support access control with facial detection and recognition. The face detection and recognition is done in the ESP EYE itself with the help of two machine learning algorithms. For the face detection we employee the Multi-task Cascaded Convolutional Neural Networks (MTCNN). With the help of Human Face Recognition Model (FRMN) the person will be recognized.

1.1 Description of Work

First of all this study begins with an introduction to an IoT architecture using the Esp Eye device which runs a Convolutional Neural Networks (CNN) for face detection and recognition. In addition to that the architecture also shows how the ESP EYE is enabled to issue transactions to a blockchain. The main contributions of the thesis is not only the design but also the implementation of a use case with the above mentioned technologies. We have proposed an IoT surveillance system which is based on blockchain that jointly work together for authentication of people allowed to enter a building. By employing a face recognition IoT device we can do the verification and identification of allowed people to enter an institution without any physical interaction required. There are three main milestones of this research. First, we will give a high level approach of the architecture while specifying the technologies and algorithms to be used and the reasons behind choosing such technologies. Furthermore, we will introduce the technologies and their architecture and will evaluate against other similar technologies. After that we will evaluate the Convolutional Neural Networks for resource constraints devices which are applied for face detection and recognition. In addition we will also have a closer look at the Hyperledger Fabric as a design choice in this architecture. Hyperledger Fabric will be evaluated and we will compare and contrast with other blockchain technologies. The main section of the thesis will be about the design and implementation details of the Blockchain-enabled Intelligent IoT surveillance system. The Esp Eye will run the CNN a Deep Learning algorithm for face detection and recognition. Besides, we will prove how the Esp Eye is able to issue transactions to Hyperledger Fabric.

1.2 High Level Overview

We know that AI and Blockchain are the leading innovations we witness today. Being the most disrupting technologies, we often talk about their advantages but yet we do not see a real use case. Hence in this section we will provide a high level description of the proposed surveillance system using facial recognition for access control. The system employees a real time face detection and recognition of authorized individuals to grant access to an institution. This means that access is granted and the door opens automatically for him or her without the need to touch anything. Every time an individual needs access a picture of him or her is taken and is then stored in the blockchain. Access is revoked when an individual is not registered in the system but yet a picture of the individual is taken and stored in the blockchain. Once a stranger is detected continuously a warning is raised in the form of an email to inform the administrators for the issue. Simultaneously, the number of people entering and leaving an institution or a room can be tracked. Tracking the number of people entering or leaving the room can be very beneficial in many situations, such as the number of students taking an exam or their presence in a classroom. The use case diagram can be seen in the Figure 1.1.

The assumptions is that the authorized individuals are first registered with the system. If an individual is not registered with the system then access is revoked. The process involves an individual approaching an entry point of an institution or a room, where the Esp Eye will detect and recognize the person and grant or reject access. The face recognition is done with the help of ESP WHO [6] platform and the recognized images of individuals are forwarded to the blockchain where it will be stored. Also the cases are considered when a person appears for a continued presence in front of the device.

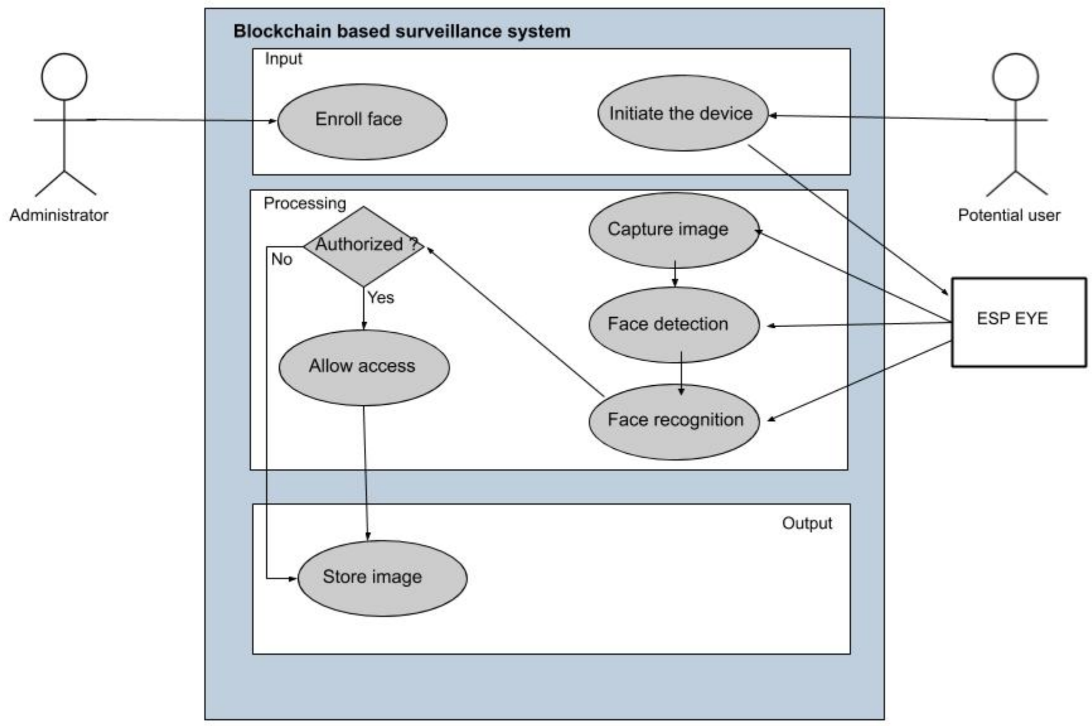


Figure 1.1: High Level Overview

1.3 Thesis Outline

The rest of the thesis is enumerated in the following order: The second chapter introduces the relevant research and use cases about the inclusion of the three domains and their synergy. In Chapter 3 we will introduce and discuss the face detection and recognition algorithms using MTMN which refers to Multi-task Cascaded Convolutional Neural Networks (MTCNN) and Mobile Nets (MN) for face detection and Human Face Recognition Model (FRMN) for face recognition.

With Hyperledger Fabric we are going to add security with the decentralised system by storing all the images of detected individuals. Therefore in order to see the details, its components and the advantages we have allocated Chapter 4 for this. In chapter 5, we will dig deeper in the design and implementation by showing the details of all involved components, the interactions between the components and analyzing the whole architecture.

The second-last chapter examine the evaluation of the work by looking at the performance, various quality measures and energy efficiency. Finally, the last chapter encapsulates the whole work by writing the last remarks.

Chapter 2

Related Work

In this section we are going to provide and introduce the most recent projects and research that shed light on the integration of the three domains. Additionaly we are going to see some similar use cases of real time face detection and recognition for access control. Convergence of the three domains is not by chance therefore in this section we explore the ideas and benefits that bring the three domains together.

2.1 Synergy of the trio: Internet of Things, Artificial Intelligence and Blockchain

In order to give a better understanding of Internet of Things (IoT), Artificial Intelligence (AI) and Blockchain (BC), it mirrors to think of them as an interconnected biological process. IoT is like our brain, which senses billions of connected devices in our planet while producing a universe of new data. AI being the rationale part of the brain, it will be thinking while analyzing data and make a decision based on that. On the other hand BC resembles memory which stores information in the growing list of blocks.

2.1.1 An overview of Internet of Things

IoT is transforming the world of things around us into a world of sensors that speak about the things. Practically almost anything can be equipped with a sensor and make the things smart. In terms of IoT the sensors are not just used to detect and measure but they are also used to respond to changes thanks to the proliferation of internet-enabled devices which are embedded with computational power.

The number of IoT devices took over the population worldwide since 2008. Based on [7] the number of IoT devices is expected to increase and reach around 31 billion by the end of 2020. However, this number is not expected to ever stop but it may double by the end of 2025 [8] as depicted in Figure 2.1 .

Most IoT devices run on microcontrollers or microprocessors with very minimal processing power which makes them less powered than a smartphone. The reason behind is not that such devices cannot be equipped with more processing power but to be energy efficient and run with minimal power on battery and reducing environmental impacts of the energy use. With such growing number of IoT devices in the near future it could eventually pose a challenge even without increasing the processing power of such devices. Eventually there could be a high demand of developing a green communication across the network [9]. This at some point may affect the developers of IoT devices who have to find more lightweight programs to run AI algorithms. As a result different algorithmic approaches are taking place and more efficient algorithms are being developed [10].

We would like to mention a number of common characteristics that most IoT devices share:

- Connectivity: with the help of a number of protocols they can communicate with each other.
- Heterogeneity: a variety of devices and objects including communication protocols
- Unique identity: a unique identifier for each device
- Big data: IoT as number one source of big data

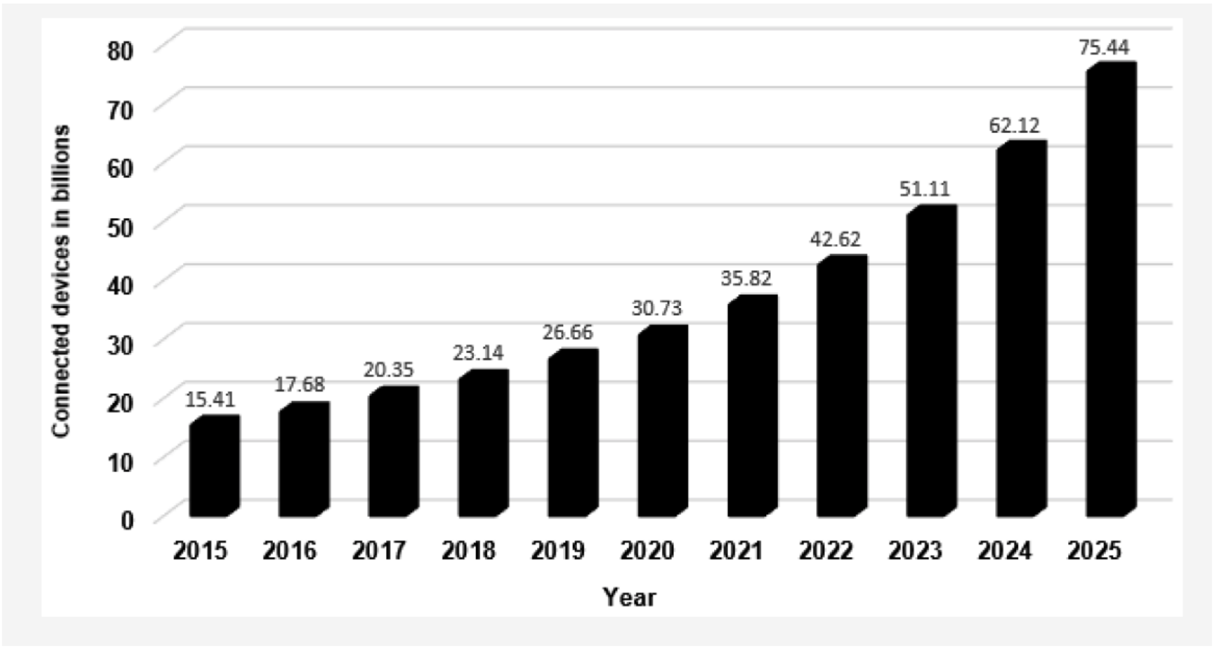


Figure 2.1: The growth of IoT devices from 2015 to 2025 [1]

However IoT suffers from its typical architecture design which is the centralized architecture. Servers either on the cloud or on premise manage and deal with all requests coming from various nodes. This architecture comes with multiple challenges: scalability issues, security and privacy challenges and issues with the analysis of big data [11].

2.1.2 Enabling Deep Learning in Internet of Things

AI has become a buzzword and one of the topics that has a significant impact on many different domains. Figure 2.2 depicts the relationship between Artificial Intelligence, Machine Learning and Deep Learning. Machine learning is the technique used to allow computer programs to access data and use it to automatically learn and improve from experience. Whereas neural network involves a large number of processors operating in parallel arranged in tiers in the same way neurons work in our brain. Deep learning employees neural networks but with many layers of neural networks.

It is known that AI can solve and make sense of the immense amount of data produced from IoT devices. However, the main question for research appears: How to deploy neural networks directly on these tiny devices. Therefore, there is an opportunity and challenge at

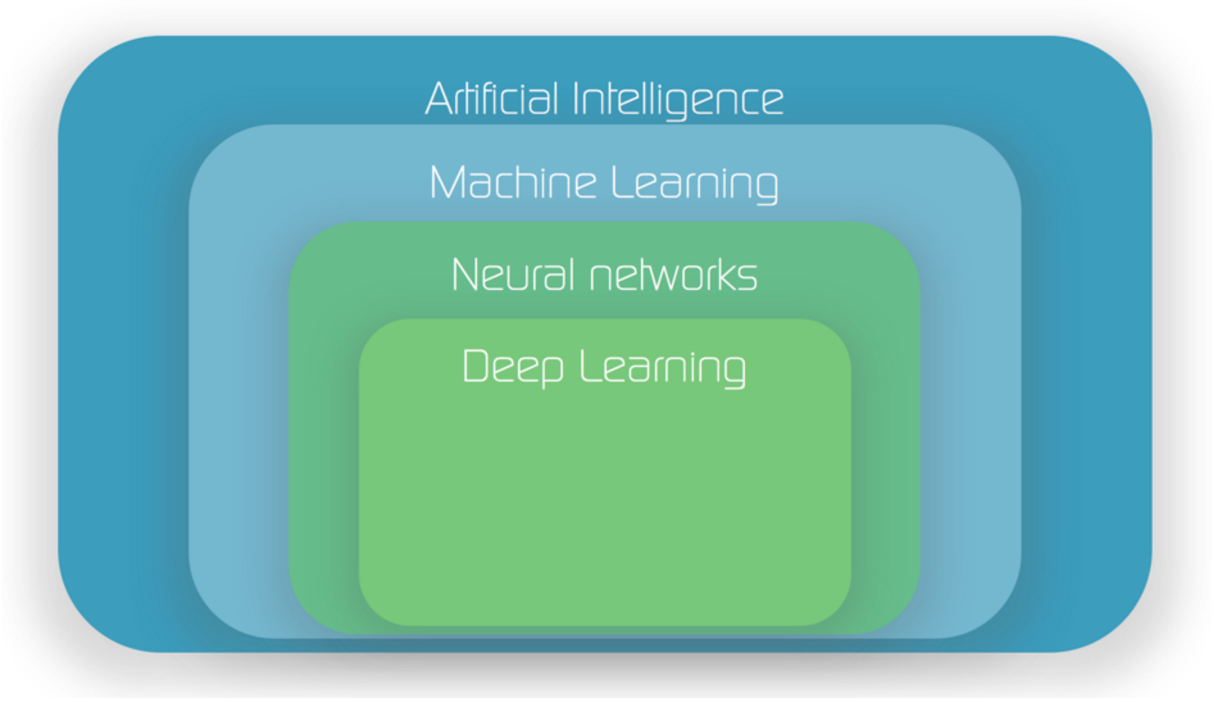


Figure 2.2: Relationship of AI terms [12].

the same time to run machine learning on such tiny IoT devices based on microcontrollers. By running machine learning on these tiny devices, we can directly perform real time data analytics in the device itself, therefore there is no need to store all the data generated from the IoT devices. Typically commercial use cases of smart IoT devices often offload AI part to the cloud. One of the most recent study [10] that worked on this direction tested two approaches with deep learning:

- Offloading deep learning platforms to the cloud;
- Migrating deep learning in an IoT device;

The two approaches were tested and look from the perspective of whether offloading machine learning can reduce energy efficiency and satisfy the real time requirements of object recognition. In the first approach they used convolutional neural networks on the cloud, while the Jetson TX1 was responsible to just take images and forward them to the cloud. The results show that executing machine learning in the Jetson TX1 consumed more energy compared to offloading it on the cloud. However, offloading AI to the cloud also comes with drawbacks which has lead to a latency starting from 2 seconds that goes up to 5 seconds which is far higher compare to the execution of AI in the Jetson TX1 itself. This infers that the variability of response time makes it quite unreliable and not useful for real time AI processing.

Furthermore MIT researchers [13] have implemented a system called MCUNet, which has a high potential to bring deep learning in much smaller devices like tiny computer chips despite their limited memory and computational power.

2.1.3 Intersection of Blockchain and Artificial Intelligence

There is a high research on BC and AI but analysed in isolation and in various domains and applications. Some research [14] focus on the application of AI in the BC for making BC more efficient for instance, consensus mechanisms and better governance. However there seem to be more research and applications of BC in AI. Similar to IoT, AI domain also suffers from security, its centralized architecture and resource limitations. This is exactly what blockchain is looking to solve. There is a lot of discussion and research in this area, however most of them are reviews and solutions that do not come up with a use case or implement such solutions. In order to have a global image the Table 2.1 describes the features of the two technologies and the benefits of integrating such features.

There is another interesting contribution [15] that came up with an AI blockchain platform to fight the propagation of fake news. This platform allows publishers to setup a distribution platform in BC while AI monitors the actors who publish news to BC. Expectation is that BC serves as the “factual database” to trace back the news. So the news which cannot be traced back to the blockchain is considered fake news.

2.1.4 Integration of Blockchain with Internet of Things

We have already listed a number of issues that IoT world is facing, which mainly are the centraized architecture, security and privacy. For instance [16] attempted to discover the security gaps that could be mitigated with the help of blockchain to ensure the reliability and availability of the data.

Blockchain	**AI**	**Benefits of blockchain**
Decentralized	Centralized	Enhanced Data Security
Immutable	Probabilistic	Collective Decision making
Data Integrity	Volatile	Decentralized Intelligence
Resilient to attacks	Data, knowledge and decision centric	High Efficiency
Deterministic	Changing	Improved trust on robotics decision

Table 2.1: Benefits of integrating blockchain with AI.

Other studies [17] attempted to integrate crypto based blockchains such as Ethereum in their approaches. We argue here that most of the crypto based blockchains are not efficient in storing data coming from IoT devices. First users have to pay fees for each transaction and the there is a limitation in the number of transactions it can process. Although we can escape from the centralization still there is the risk of bottleneck. Therefore in our study we will be using Hyperledger Fabric a non crypto based blockchain aimed for storing data.

2.2 Use cases on Artificial Intelligence, Internet of Things and Blockchain

We have already described the intersections on how BC, AI and IoT accommodate each other in pair, however there is a very high potential for the usage of all the three domains in one use case. With such a high number of IoT devices there is a potential to take the advantage of AI and BC at the same time. In another article (Kurian 2020)[18], stated that every institution which takes the advantage of exploiting these technologies has the chance to radically enhance their existing processes and create entirely new business models.

An interesting work [19] presents how the collaboration of the three domains can build a sustainable smart city. Given the many issues people in urban areas face, the concept of the sustainable smart city brings new opportunities for the application. With their new approach they aim to have a transparent monitoring system for measuring pollution which in turn helps to raise awareness to the population.

In another use case [20] the authors propose an IoT based home automation and surveillance system. In this setup they have employed a raspberry pi, a camera attached to it and a DC motor for controlling the door. The video surveillance is not detecting or recognizing people, so the burden falls into the owner who through the help of an application is able to open the door. Compared to our solution there is no AI and BC integrated in their architecture.

A use case that pretty much is matching with our use case is the design and implementation of a camera based sensor for room capacity monitoring. The aim of it is to count the number of people present in a room with the help of a raspberry pi and a camera. In Figure 2.3 we can see the architectural overview.

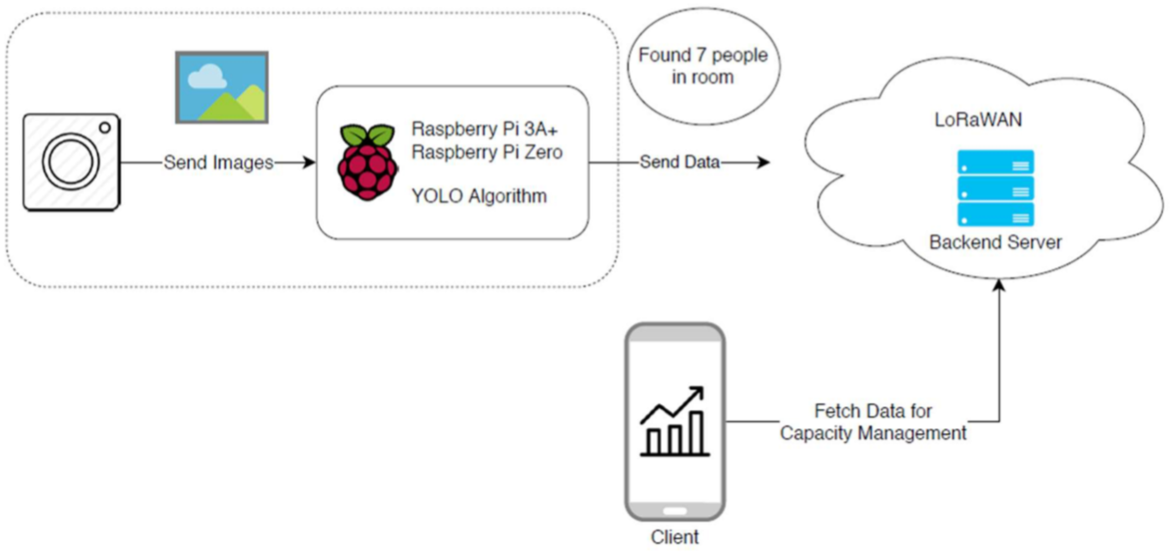


Figure 2.3: Architectural Overview [21].

Their architecture employees AI and IoT. The use case was designed and implemented by a group of students at FHNWS [1] who employed a raspberry pi equipped with camera and lora gateway. The role of camera is to take pictures and after that a machine learning algorithm analyzes the image and finds the number of people in the room. Once done the

[1]Fachhochschule Nordwestschweiz

data is then send to a LoraWan Server. To achieve that, they have attached a LoraWan antenna to the RPI and eventually with a web application they can monitor the room. In this case face detection happens in the RPI, the algorithm counts the number of people in the image and sends it to the Lorawan Backend Server. From here we can conclude that the issue of centralisation is still open, data is being stored in a database, security issues are not tackled enough. Besides there is also the need for a RPI to be placed in a room as the camera is attached to it and that needs power to run.

With our proposed architecture we are not only automating the IoT-based surveillance system but provide a robust solution that will take care of the leaks that AI and BC can neutralize.

Chapter 3

Face Detection and Recognition in Esp Eye

3.1 Background of real time face recognition in low powered IoT devices

Monitoring and tracking people and their activities with the current approaches of the surveillance system normally generates enormous amount of data coming from IoT devices. This leads to a number of issues that need to be treated well for example data migration from camera over a limited bandwidth for face detection and recognition leads to high latency and the camera need to be placed near the electrical plug as they require fairly high power. Therefore this approach leads to generating a lot of data while transferring it to a different source for face detection and recognition. Besides many public places use surveillance camera for video capture based on motion detection, which means that once someone approaches near the camera it then starts recording and throughout the day high volume of data can be generated. This chapter provides a guidance to the existing algorithms for face detection and recognition and then explores the algorithms that are implemented for the resource constraint devices. Hence we also discuss the reasons behind choosing to deploy the face detection and recognition algorithm in the Esp Eye itself. In addition MTCNN and FRMN for face detection and recognition are described in detail.

3.1.1 The generic framework for face detection and recognition

Face detection is normally the first step towards face recognition or verification as it can be seen in Figure 3.1. Basically the framework follows a two step process: face image detection and face recognizer. In the first process an image is taken from the camera then the algorithm localizes the face from the background image. This process happens continuously, which means if there is no face detected then it captures the next image and it does the same. When no face is detected the image is deleted from memory. The next process assumes the image is taken and a face has been detected then the face recognition starts. In this phase another algorithm takes place in order to determine who are the individuals in that image. During both processes algorithms extract features or patterns, which is the key step of the algorithm. Feature extraction is essential for localizing facial components such as mouth, nose, eye and others.

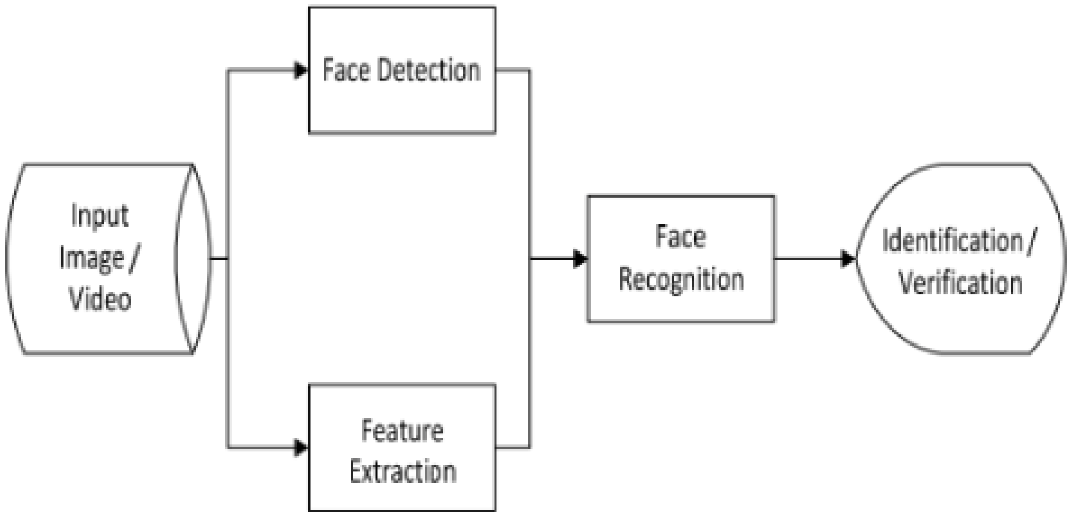


Figure 3.1: Face detection and recognition.

3.1.2 Existing algorithms for face detection and recognition

There are a number of algorithms that have been developed and now used to tackle the face detection and recognition issues. A number of successful algorithms which are widely used are listed below:

- **Principle Component Analysis (PCA)**: The PCA is a statistical approach and is known as one of the simplest face recognition systems. In this approach images with detected faces are transformed into eigenfaces. Eigenface is a method that determines the variance of faces from a collection of images. The main idea behind is to linearly project images onto a lower dimensional images.

- **Linear Discriminant Analysis (LDA)**: The LDA is a method used in pattern recognition and machine learning by finding a linear combination of features. It is used for dimensionality reduction and performs very well in face recognition. It finds a projection transformation by maximizing the between-class distance and minimizing the within-class distance.

- **Skin colour based algorithm**: As the name says, it uses skin color as a feature for face detection and it is fast, self adaptive algorithm. A bounding box is drawn to extract the face from the image.

- **Wavelet based algorithm**: In this method the image is cut up into a subset of frequency components and to each component is applied a mathematical function. One of the widely known algorithm is the Gabor wavelet.

- **Artificial neural networks based algorithm**: Typically face recognition is achieved with the help of deep learning more specifically with Convolutional Neural Network. Through a multi-layered network it performs a specific task using classification.

3.1.3 Hardware requirements for running real time face recognition

It is important to mention that a typical computer or a laptop possess an AMD processor, while a raspberry pi and smart phones and some other devices like watches are equipped with an ARM processor. Obviously ARM architecture offers lower performance compared to AMD, but there is a reason behind, the ARM processors consume less power than AMD processors. Performing real time recognition in both AMD and ARM is not an issue, both of them can handle but with different approaches. However both processors need power either plugged or if on battery they can last max one day long. In contrast the low powered IoT devices can run for years with a typical voltage of 3.6 volts. Therefore to our knowledge so far it is not possible to perform real time recognition with a processor other then the above mentioned.

Esp Eye is the first device to perform real time face recognition. However, this does not mean that the same face detection and recognition algorithms for computers have been used here. Esp Eye is one of its kind that comes with ESP WHO [22] platform which supports both face detection and face recognition. The ESP EYE [23] is equipped with Tensilica LX6 dual core processor. To our knowledge this is the only device that can perform real time face recognition in a microprocessor that lies out of the two classes AMD and ARM processors.

3.2 Face detection with deep learning using MTNN

MTMN refers to both MTCNN (Multi-task Cascaded Convolutional Networks) and MobileNets. There are a number of deep learning methods that have been implemented and paved the way for face detection but MTCNN is a framework which integrates both face detection and alignment. With the help of MobileNets it builds lightweight deep neural networks which uses depth-wise separable convolutions for face detection.

3.2.1 MTCNN a three layer CNN model

MTCNN is one of the state of the art approaches which is described and published in the paper "Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks" [24] in 2016. MTCNN is popular because it has achieved 95 percent accuracy on a range of banchmark datasets. MTCNN is a novel approach because it performs a lightweight Convolutional Neural Network based framework which actually performs simultaneously face detection and alignment whereas in other CNN-based frameworks face detection and alignment are two distinct processes. Due to its lightweight CNN architecture it can perform in real time which is possible to deploy it in an ESP32 chip.

The process consists of three stages of Convolutional Neural Networks that are able to detect faces and landmark location such as eyes, nose, and mouth:

- **P-net**
- **R-net**
- **O-net**

The Figure 3.2 adapted from their original paper depicts a summary of the three layers with the output of each layer on the right side. The three mentioned models perform independent of each other, the output of each is used as an input of another. First of all once the image is captured it is then scaled into multiple different sizes based on different scaling ratios, which forms a collection of images called “Image Pyramid” similarly as in Figure 3.3. This allows the model to learn different image scales effectively. This type of image processing is used extensively because it makes it easier to detect faces no matter how far or close they stand in the image.

Proposal Network or P-net

The first stage of Proposal network (P-Net) as depicted in Figure 3.4 is a full convolutional neural network which extracts face candidate regions from images at various scales and bounding box regression vectors. Bounding-box regression is a technique for refining or localizing boxes in objects. In other words it is a rectangle, which is drawn over the image to emphasize the face. For each of the scaled image that reside in the image pyramid it runs a 12x12 kernel which starts in the top left corner or at these coordinates (0,0) and (12,12). This portion of the image 12x12x3 is the input for P-Net as it can be seen in Figure 3.4. Furthermore, another kernel of size 3x3 is applied to this image and then it produces feature map or smaller images depending on the stride. For example in the first filtering it produces feature maps of size 5x5. The stride is the number of jumps or pixels

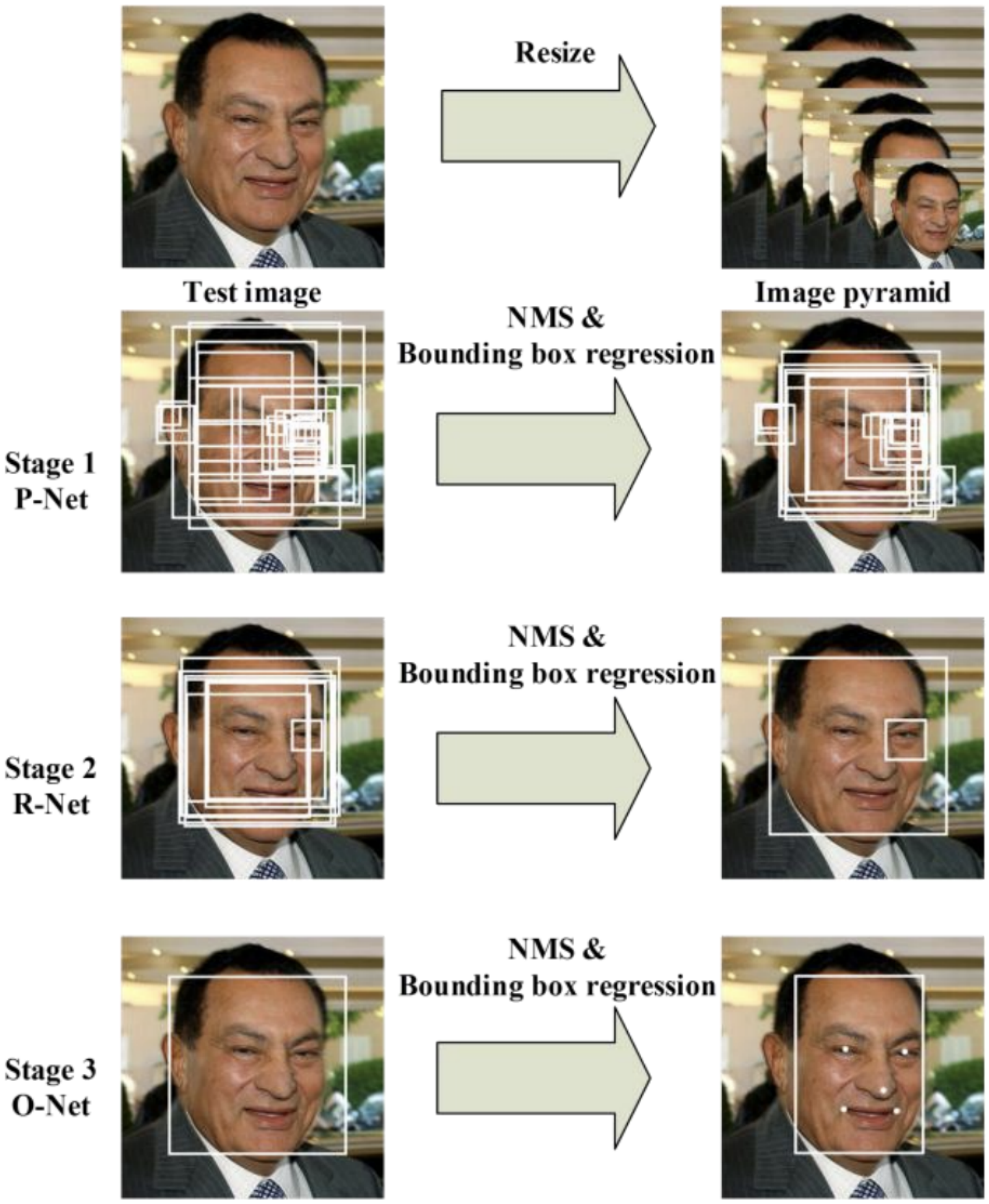


Figure 3.2: Pipeline of MTCNN that includes three-stage multi-task deep convolutional networks [24].

that the kernel moves to apply a filter. In order to deliver a 5x5x10 from 12x12x3 with a kernel 3x3 the stride is 2. Therefore stride plays a role in time complexity of the algorithm, the lower the stride the higher the time complexity. From this 12x12 portion of a scaled

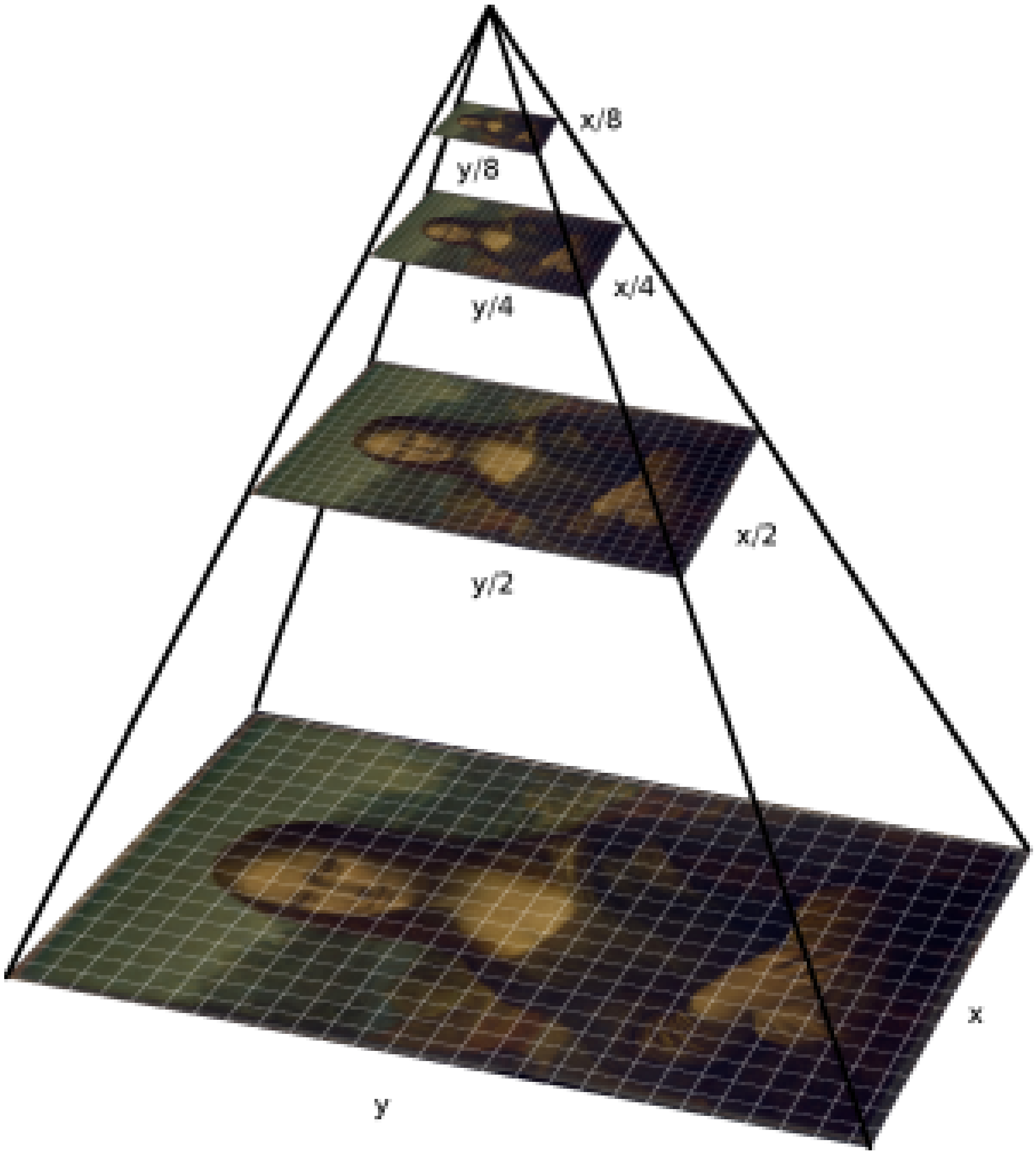


Figure 3.3: Image rescaled in a form of a pyramid [25]

image from pyramid the P-net outputs the coordinates of a bounding box in case there is a face. Before releasing the last bounding box coordinates a number of similar boxes are filtered out with lower confidence or score and only the ones with higher confidence are kept. The bounding boxes left may still overlap as in Figure 3.5, therefore we need the one that perfectly covers the face. To do that the non-max suppression technique is applied which reduces the number of boxes by taking the best fit and not the one the network is more confident in.

The aim of the this network is to check whether there is a face in the input and if there is it outputs the face frame or box with the four coordinates. In the face classification the network outputs the probability of being a face and not being a face and both values add up to 1. The bounding box holds the exact position of the box with coordinates.

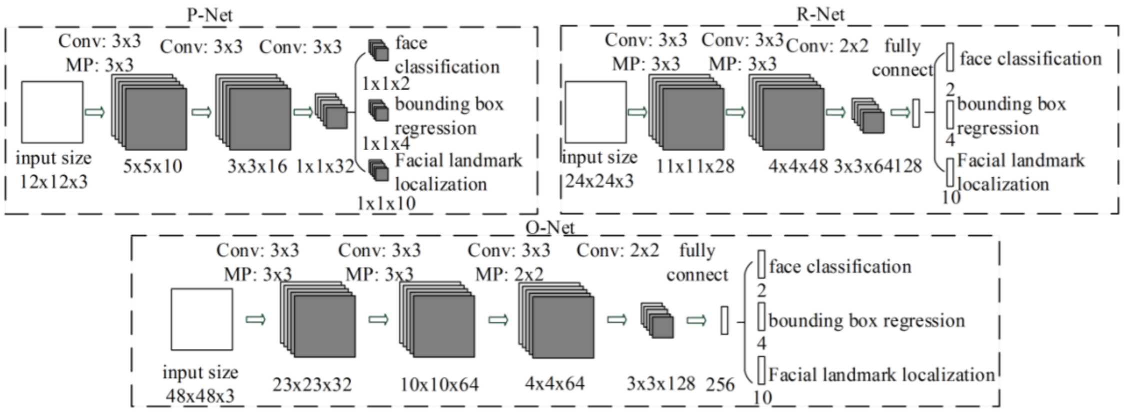


Figure 3.4: MTCCN: P-Net, R-Net and O-Net structure [24].

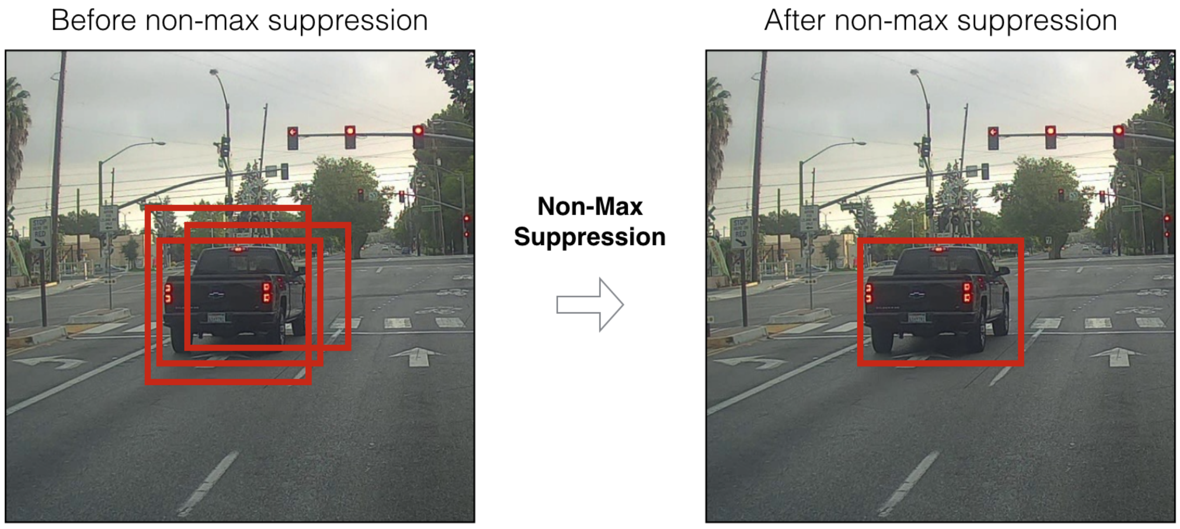


Figure 3.5: Non-Maximum Suppression [26].

Refine Network or R-net

The structure of R-net can be seen in the Figure 3.4 which looks pretty much the same as R-net. R-net is a rough prediction of faces therefore there are a number of false positives so the aim of R-net is refinement of the bounding boxes and reduce the number of false positives. The input of this stage is the bounding box generated by P-net which can be of different sizes. From the Figure 3.4 we can see that the input size is of dimensions 24x24x3. So regardless of the size of the bounding box it still needs to be scaled to the expected input size before sending to R-Net. Therefore the bounding boxes is scaled to 24 x 24 pixels and can be fed to R-Net. Once again the same as in P-net , it filters out boxes with low confidence and applies the NMS on every survived box. The output is the same as of that of P-net which also consists of three parts: probability of being a face or not, square frames and the position of landmarks. This stage can help to recover the weaknesses of P-net and improve accuracy by reducing the number of false positives.

Output Network or O-Net

In the Output Network almost the same happens as in R-Net with some changes in middle layer, where additional channels are added and accuracy is increasing. The accuracy rate is expected to be higher which is proportional to the increased time complexity. In this case the output of R-Net serves as the input of this stage but scaled to 48x48x3. Now we can already see the reason behind using three neural networks and not one. Since the final decision is received from O-Net we can directly use it and avoid the previous two networks. But the speed will be very slow since O-Net has to do a lot of operation on all candidate windows which can be very high. With the help of P-net and R-Net a lot of non relevant windows will be filtered out from the start. From P-Net to R-Net a lot filtering happens, so the advantage here is to perform fine grained face detection only on highly probable candidate windows. A summary of the whole face detection process is as follows:

P-Net:

1. Image is captured by Esp Eye.
2. Create image pyramid with multiple scaled copies of the image.
3. Pass each image of pyramid to P-net.
4. Candidate windows as output.
5. Filter out bounding boxes with low confidence.

6. Adjust the current 12 x 12 kernel coordinates to “un-scaled image” from pyramid.
7. Apply Non-Maximum Suppression for all kernels.
8. Adjust the bounding box coordinates to “un-scaled image”.
9. Candidate windows as output.

R-Net:

1. Pass the bounding boxes.
2. Pass each image of pyramid to P-net.
3. Candidate windows as output.
4. Repeat step 5 to 9 of P-net.

O-Net: Repeat the same steps as in R-Net.

3.2.2 MobileNetsv2 for lightweight CNN

MobileNet is a CNN architecture developed by researchers at Google which is proven to build light-weight deep neural networks which uses less computational power to run. This architecture allows to run face detection and recognition for IoT devices, Mobile devices and computers with low computational efficiency without compromising the accuracy of the results. MobilNet version 2 is the recent version which comes with a research paper published in late 2018 as a refinement of the MobileNet version 1 which make it even more powerful.

In order to understand why this model is being used and at which step in face detection is used we would like to first give a quick recall on how convolutional neural network works. CNN is composed of neural networks as in Figure 3.4 which in our case receives an image and transforms it through a series of hidden layers. Each hidden layer is made up of a number of neurons which are nothing else but mathematical functions and in terminology it is called activation function which calculates the weighted sum of multiple pixels and decides whether or not to send a signal to the next neuron. The activation function takes the weighted sum of all inputs plus the bias.

Before reaching the next neuron, a convolution or a filter is applied to the input image and a feature map is created from it. We can see in P-net in the first layer there are 10 feature maps created with a size of 5x5. That means a convolutional filter is applied 10 times to the input image and the size of the kernel in our case is 3x3. But this does not mean that the same filter is applied but different ones as in Figure 3.7. This kernel slides across all the pixels of the input image by covering all of them and at each step it computes the weighted sum and puts in the feature map as in Figure 3.6. Depending on the size of the kernel and input image the new feature map may have a smaller size. Here is where pooling comes into play, with pooling we can upsample or downsample or change the spatial dimensions but in fact only max-pooling is performed because there it needs to detect outliers which is the moment the network can detect features. So the deeper it goes in the image the better to detect features. When the output of the P-net is generated which is an image or candidate window it has to be upsampled because R-Net only receives image of dimensions 24x24x3. This means it zooms in the image more and more until the last network is performed. Besides it can be seen that the number of filters is increasing and in the last layer of O-Net it generates a feature map of dimensions 3x3x128 which means 128 filters have been applied.

From here we can understand the reason behind the staging. Performing the convolution to the image directly from the last stage would need a lot of computations because filtering takes processing time. For instance if the person to be detected is standing far away and in the image that person may cover for example only 5x5 pixels out of an image of 360x240. So applying a high number of filters in each extracted image with 5x5 pixels is not efficient. Hence it is better to first apply filtering in larger parts of the image so that we can get rid of candidate windows which do not include faces.

The staging approach is quite efficient in face detection while MobileNets is essential in the way convolution is performed in applying the filters and creating feature maps.

Normal Convolution

The main idea behind MobileNetv2 is to get high accuracy with less computational power. In the regular convolution, the convolution kernel or the filter is applied on all the channels

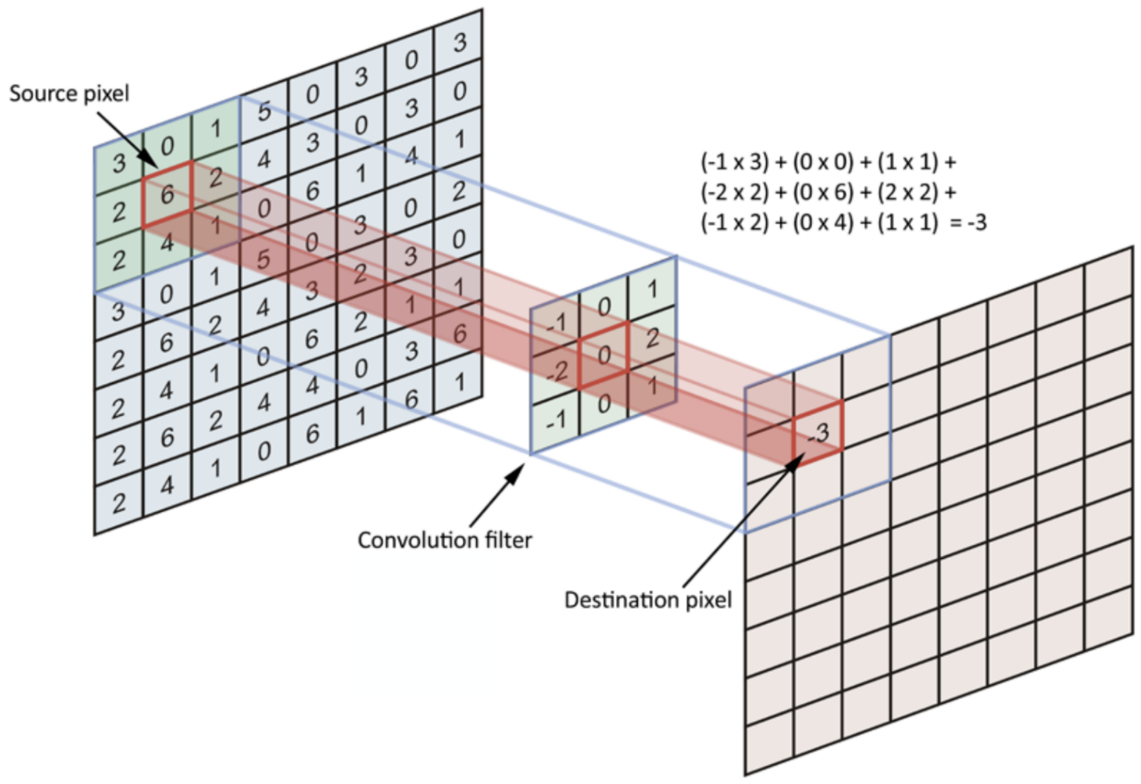
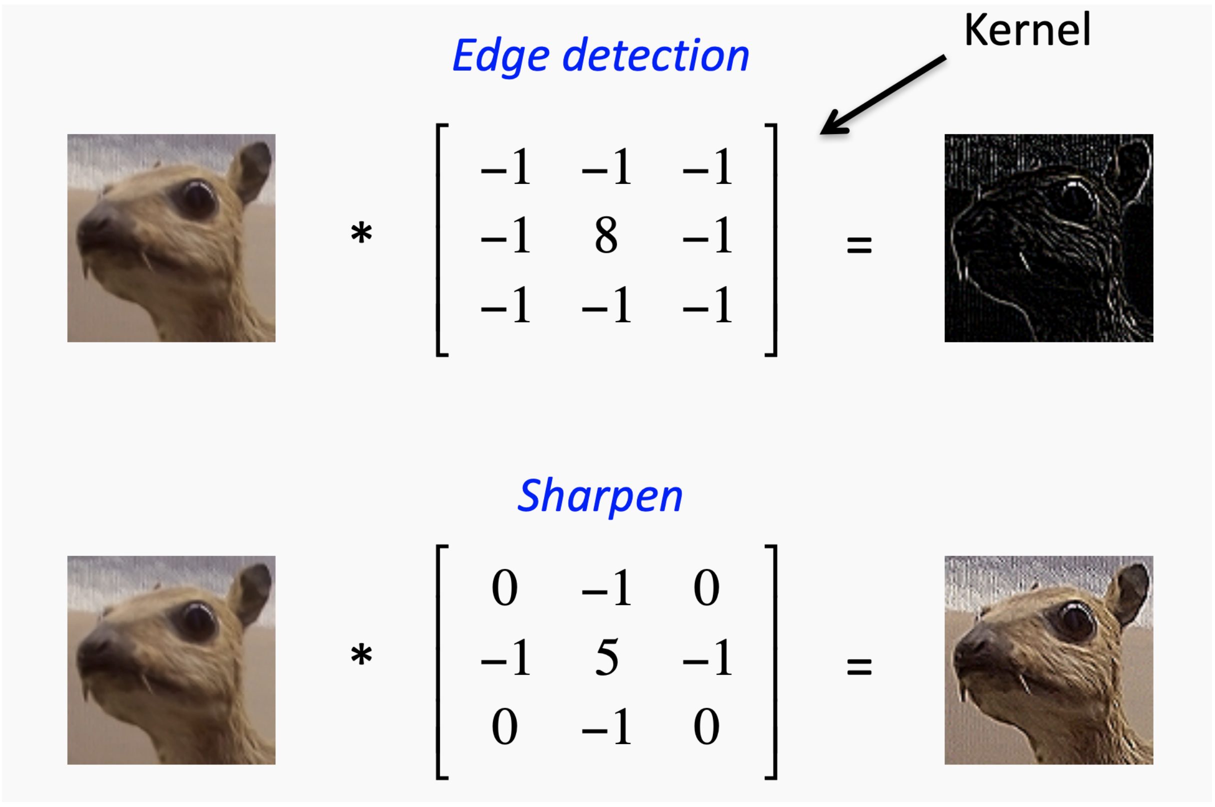

Source pixel
Convolution filter
Destination pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3

Figure 3.6: The convolution operation [27].

of the input image. The input image normally has 3 channels and for each color RGB there is one channel. Let's assume we have an image of 12x12x3 pixels as in Figure 3.8 and run convolution with kernel of size 5x5 and a stride 1. When applying the kernel it multiplies each pixels with its counterpart and at the end we get a feature map of 8x8x1. No matter how many channels the image has it only generates one pixel with only one channel. However since there are 3 channels we need to also have the convolutional kernel with 3 channels. So instead of doing 25 computations the algorithm has to do 5x5x3=75 multiplications with quadratic time complexity. And at the end we receive a feature map with one channel.

Therefore if we want to have a feature map with more dimensions then we need to apply more kernels. If we want a feature map with 256 channels then we have to apply the 5x5x3 kernel 256 times. Normally this is how convolution works but this is not the case with the MobileNets architecture. MobileNets is using this approach just once at the very first layer. In all other layers it employees the depthwise separable convolution with two parts: depthwise convolution and pointwise convolution.



$$\text{Edge detection:}\quad * \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} =$$

$$\text{Sharpen:}\quad * \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} =$$

Figure 3.7: Examples of filters [27].

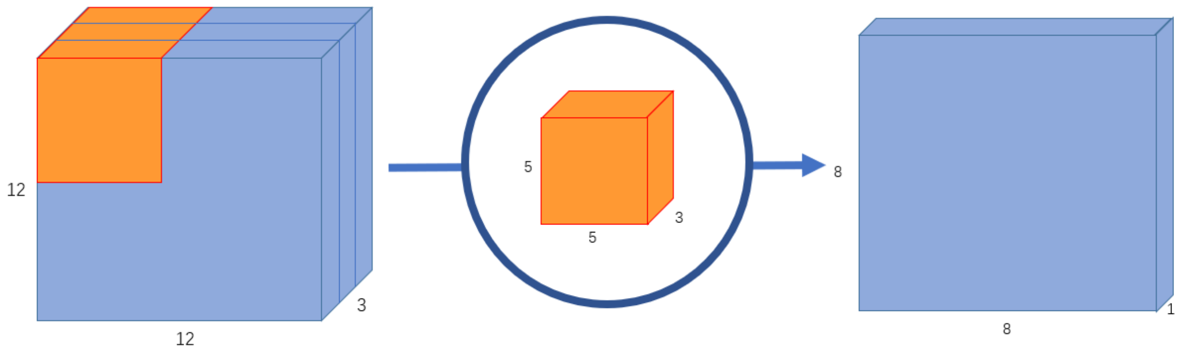


Figure 3.8: Normal convolution [28].

Depthwise convolution

With depthwise convolution instead of applying one kernel with 3 channels it applies 3 kernels with one channel. Here we do not combine the three channels but perform it separately. This makes it easier to create a feature map with multiple channels. With normal convolution to achieve this it has to apply the kernel with 3 channels 3 times, but here the number of kernels stays the same but with linear time complexity.

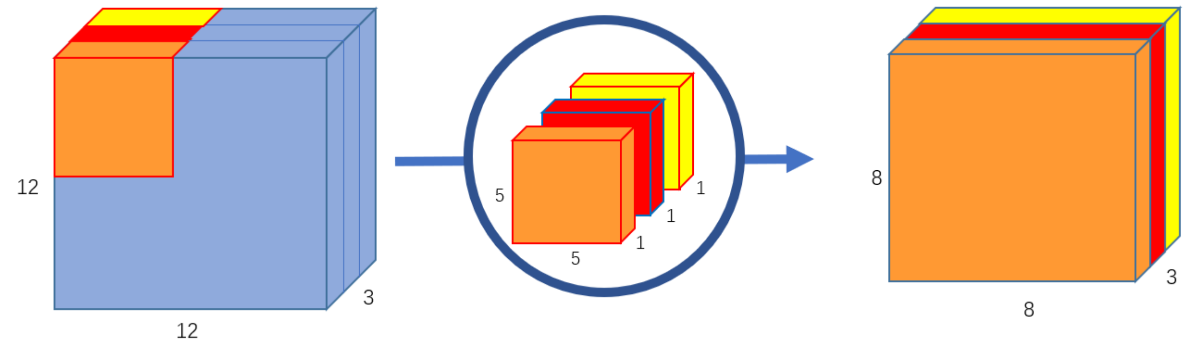


Figure 3.9: Depthwise convolution [28].

Pointwise convolution

As of now we have transformed the image into 8x8x3 but if we want to increase the number of channels to 256 as in normal convolution above we apply a pointwise convolution. As the name says pointwise convolution applies a kernel of size 1x1x3. At this step we come back again to normal convolution, so now it applies the kernel to the image 8x8x3 as in Figure 3.9 and combine it to form one channel image with output of size 8x8x1. To get 256 channels we apply the same kernel 256 times. Now we have the same feature map as in normal convolution above but with less number of computations.

Here is a simple calculation of both approaches, normal convolution and MobileNet approach. Assuming we have an input image of size 12x12x3 and the kernel is of size 5x5 and stride 1 with these parameters we get an image of size 8x8. With normal convolution since the input image has 3 channels we apply the kernel 5x5x3 and we get an output image of size 8x8x1. If we want an image with 126 channels, the total multiplications is 126x3x5x5x8x8=604800. With the MobileNets we have 126x1x1x3x8x8=24192. Hence with MobileNets the network performs faster.

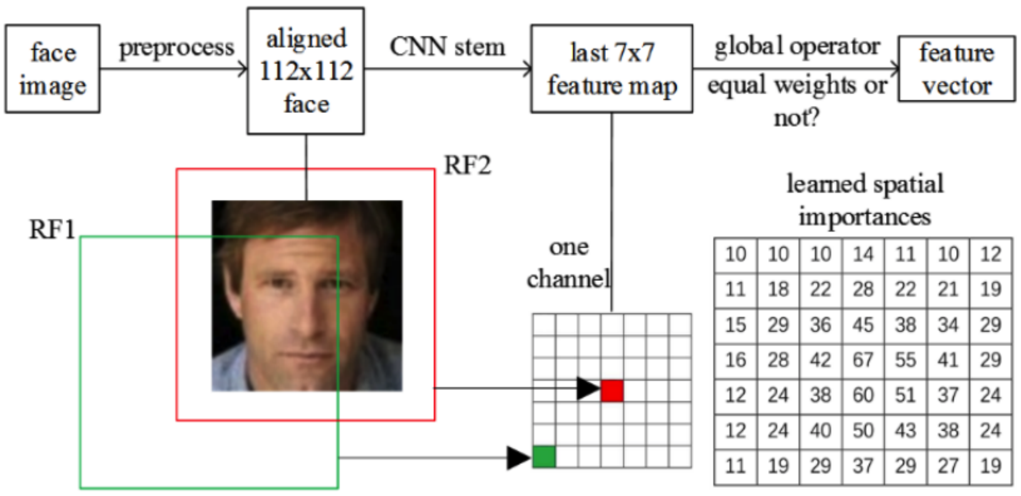
3.3 Face Recognition Model Based on CNN

Face recognition is done using MobileFaceNet [29] which consists of MobileNetv2 [30] and ArcFace algorithm. The output of face detection is the aligned face with human face coordinates: bounding box as an image and landmarks. This output will be the input for face recognition. Only if human is detected with the above mentioned model then face recognition algorithm fires which then generates a Face ID. In order to verify who that person is, it first has to check if that person exist in the database. The next step is to compare the newly generated Face ID with the existing Face IDs. Most likely there may not be an exact match for Face IDs even if it is from the same person. Hence the idea is to obtain the distance between the Face IDs which is done by Euclidean distance. In order to determine if the Face ID is from the same person it compares the distance between the Face IDs based on allowed threshold.

3.3.1 MobileFaceNet for face recognition using CNN

MobileFaceNet is one of state of the art approaches for face verification developed by the same authors who developed MobileNet [31]. Regarding this approach they also published a paper. MobileFaceNet [32] is claimed to be one of the very efficient model which addresses high-accuracy for real-time face verification on embedded devices. Offline face verification is important on mobile devices because many applications equipped with face recognition need to run offline. MobileFaceNets uses around one million parameters and can achieve a very high accuracy with 99.55 percent for face verification after being trained with ArcFace algorithm. The MobileFaceNet model takes the advantage of a global depthwise convolution layer rather than a global average pooling layer or a fully connected layer to output a discriminative feature 512-d vector.

Typically during face verification the models preprocess the images by extracting face features and eventually compare the new generated Face ID or vector with the already saved Face IDs by either comparing the similarities or the distance. Similarly MobileNetV2 is used here also for face feature embedding. The face verification process as seen in Figure 3.10 starts with MTCNN and then after a face is detected the aligned face is created. The aligned face is the bounding box with the face detected of size 112x112 which is then used as an input for face recognition. The aligned face is then mapped to a feature vector.



10	10	10	14	11	10	12
11	18	22	28	22	21	19
15	29	36	45	38	34	29
16	28	42	67	55	41	29
12	24	38	60	51	37	24
12	24	40	50	43	38	24
11	19	29	37	29	27	19

Figure 3.10: Face verification [32].

The MobileNetv2 is used similarly as in face detection until the last feature maps generated with size 7x7. At this step a Global Depthwise Convolution takes places, which uses a kernel of the same size as the input size. The output of this convolution is a 1x1xC while C being the number of channels. The reason behind using this approach in the last convolutional layer is that it reduces the last feature map to a vector. Which is then used as an identity for a person and the model uses it for facial recognition by computing the euclidean distance of this vector and other vectors.

Chapter 4

Hyperledger Fabric for Storing Images

The video surveillance monitoring system that we propose captures images once it detects a person and grants or rejects access. In order to keep evidence we store images of detected people in the Hyperledger Fabric. In this chapter we are going to show the Hyperledger Fabric architecture with its components.

4.1 Hyperledger Fabric Introduction

Public Blockchain as an immutable ledger allows for storing data in the form of transactions held by distributed network of untrusted peers and each peer can join or leave the network without any restrictions. In order to add information or create transactions normally peers ought to execute a consensus protocol which typically is a Proof of Work protocol in order to validate transactions. During this process the peers compete for a mathematical task to be solved which require a lot of computational power, this happens everytime a new block is created. Bitcoin and Ethereum fall in this category which are considered to be public permissionless Blockchains. This type of Blockchain is great when transparency is required and data privacy is not a concern. In addition due to the consensus algorithm in public Blockchains the number of transactions mined is relatively low, for example Bitcoin has an average of 7 transactions per second which makes it quite slow and energy intensive. Considering the above conditions public Blockchain does not offer the possibility for a wide range of use cases. Therefore many companies often chose private permissioned Blockchain over the public Blockchain because they:

- concern about data privacy and confidentiality;
- need a customizable blockchain to implement their specific use cases;
- provide more control over the resources;
- do not involve cryptocurrencies and provide better throughput;

Having considered our use case of video surveillance access control with face detection and recognition we found Hyperledger Fabric more appropriate than any other private permissioned Blockchain. Hyperledger Fabric is an open source permissioned Blockchain developed by Linux Foundation which has received contributions from IBM and some other companies. With its modular and configurable architecture it has become a standard for enterprise Blockchain.

4.2 Hyperledger Fabric Architecture

Hyperledger Fabric differs from other Blockchains as it provides solutions to the problems in the business world. So instead of having unknown participants, it has a membership service provider that only allows participants which are permitted. Hyperledger Fabric offers a pluggable consensus mechanism instead of using Proof of Work while giving a lot of freedom to chose. In other words the platform is very flexible which will be reflected once we look in more detail at each component. Figure 4.1 depicts Hyperledger Fabric architecture with its key features and components which we define:

- Roles in Hyperledger Fabric;
- Membership Service Provider;

- Certificate Authority;
- Consensus Mechanism;
- Chaincode / Smart Contracts ;
- Channels;
- Ledger and State Database;

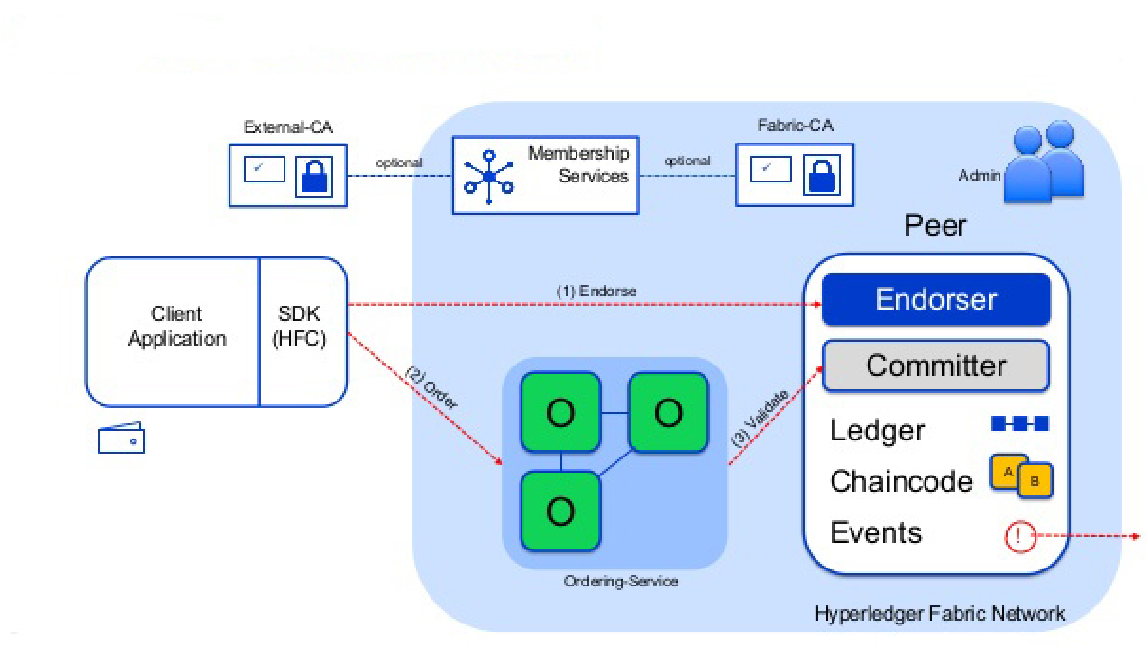


Figure 4.1: Fabric Architecture [33].

4.2.1 Roles within Hyperledger Fabric Network

There are a number of roles in Hyperledger Fabric but first of all the most important are the members of the organisations who operate the network and together form a consortium. Each of the nodes that participate in the HLF hold an identity issued by the Membership Service Provided (MSP) with each being responsible for one of these roles:

1. Clients or Participants: They are applications or devices that issue transactions in the network on behalf of the user. Every client is identified by the MSP and they

hold certificates issued from the Certificate Authority which allows them to sign transactions.

2. Peers. They are the fundamental element of the network because they operate the blockchain and execute transactions to the append only data structure.
3. Ordering Service: Ordering peers receive a transaction from the endorser it checks if the client is authorized by the MSP. If that is the case it orders the transactions in a block and delivers the blocks in atomic or total order broadcast.
4. Administrators: They stand one level above a member. They are responsible for adding or removing members from the Blockchain.

Types of peers:

1. Endorsing Peer: In Hyperledger Fabric it is not necessary that all peers execute the transactions but only a subset of them which are named as endorsing peers. Every chaincode holds an endorsement policy which defines which peers are allowed to execute the chaincode. The endorsement policy holds from the moment the chaincode is installed on the endorsing peers. Every chaincode can specify its own endorsement policy. Each and every peer that holds that particular chaincode has to do the endorsement and each organisation must have at least one endorser peer. The request for adding transactions comes from the client and the endorser ought to assure the client is valid, if the client has a true Certificate Authority and digital signature.
2. Ordering Peer: After an orderer peer receives a transaction from the client which is also signed by the endorser it checks if the client is authorized by the MSP. If that's the case it orders the transactions in a block and delivers the block in atomic or total order broadcast to all the peers. The task basically is to achieve consensus on the order of transactions and deliver the decision to the committer peer.
3. Committer Peer: The committer assumes the transaction has been validated by the order and its job is to transmit the valid transactions to other peers for inclusion in the ledger.

4.2.2 Membership Service Provider

Unlike public Blockchain in private Blockchain participants in the network are identified. Hence in Hyperledger Fabric MSP manages identities, each and every entity has to obtain an enrollment certificate from a Certificate Authority which is part of MSP. Since Fabric is permissioned, every interaction or messages that occurs between the nodes are authenticated with digital signature. MSP is responsible for issuing and managing credentials in order to authenticate and authorise every node in the network. Typically there is one MSP per organisation but it is still possible to have one MSP to serve multiple organisations. In addition MSP also defines roles, access rights and the rules for each type of peer. MSP makes use of Certificate Authority which makes it easier to verify and revoke certificates when needed. In other words MSP allows to find out which identity is to be trusted and which ones to be rejected.

4.2.3 Certificate Authority

Hyperledger Fabric uses Public Key Infrastructure (PKI) for the digital identity. Each digital identity is provided with X. 509 digital certificate in order to verify that a public key belongs to the entity who owns it. The role of public private key in Hyperledger Fabric is mainly to sign transactions. When an application user initiates a transaction they first sign the transactions with their private key issued by their organisation and it also attaches the public key and the signature.

By default Hyperledger Fabric provides a Fabric-ca docker container which is a certificate authority for handling digital identities reflected in Figure 4.1. Fabric-ca is in charge of registering identities, issuing of enrollment certificates and renewing or revoking certificates. However Hyperledger Fabric is not bound to Fabric-ca for managing digital identities but instead it allows for external CA also. This is another proof which shows the modular nature of Hyperledger Fabric.

4.2.4 Consensus Mechanism

Public Blockchains like Ethereum are permission-less which means the peers do not have to trust each other. In order to ensure trust there is a need for consensus algorithm such as Proof of Work. However with permissioned Blockchains the entities are all known and the consensus takes a different approach. Eventhough the participants may know each other but still they are not fully trusted.

In HLF every client that submits transactions belongs to an organisation which means that the client represents the organisation itself. Therefore in order to ensure the organisation is not doing any mischief there is a need for an agreement or consensus. In HLF there is a need to compromise on the ordering of transactions in the block and most importantly validating those transactions in the block that will eventually be inserted in the ledger. Consensus in HLF is broad and covers the entire transaction flow, starting from proposing a transaction till committing it to the ledger. But we need to keep in mind that transactions are executed before the ordering. For executing transactions there is no need for a consensus algorithm, it is the ordering peers that do it by holding to the rules in the endorsement policies applied. Consensus in Fabric ensures the correctness of all transactions in the proposed block and agreement on the ordering of transactions. This brings to the properties which are needed to guarantee agreement between nodes: safety and liveliness. “Safety” means a node receives the sequence of transactions to be inserted in the ledger, the same sequence and state occurs on all other nodes. “Liveliness” means even though some of the nodes may be faulty, the non faulty nodes should still be able to receive the committed transactions.

The ordering of transactions is assigned to a pluggable algorithm for consensus which is loosely coupled to the peers that execute and maintain the ledger. This allows the organisations running the Blockchain to chose the consensus algorithm that best fits their need and use case without limiting their own creativity. The component where the consensus is applied is the ordering service.

Consensus Algorithms Used in Hyperledger Fabric

With its pluggable architecture HLF allows developers to configure the development with the consensus algorithm which best suits the organisations needs. Currently the supported algorithms[34] are:

- **Solo**; As the name says solo, the ordering service is performed by only one peer. Clearly this is not the best option and it is neither decentralized nor fault tolerant. But since solo is used only for development purposes there is the advantage of letting the developers focus on other matters while developing the chaincode and application and it is not used for production.

- **Kafka**: The Apache Kafka provides a crash fault tolerant (CFT) but it is not Byzantine Fault Tolerant (BFT). Apache Kafka is an event streaming platform based on publisher-subscribe messaging which comes with high throughput and low latency. It follows a leader-follower model which means there is a leader peer and follower peers. The leader orders transactions and propagates them to all other peers and if the leader goes down then re-election takes place for choosing the leader.

- **Raft**: It is one of the latest mechanism recommend from Hyperledger Fabric which is there to improve the Kafka mechanism towards a (BFT) ordering service. Raft and Kafka provide the identical approach to the ordering service component. Both of them follow the leader and follower design and are crash fault tolerant. However Raft overtakes Kafka by a number of major advantages worth considering. In Raft is much easier to setup considering that deploying Kafka requires additional baggage such as Zookeeper. Secondly, Kafka is not scalable and it runs in a tight group of peers. There can be different organisations running Kafka but yet the nodes running Kafka all go to the same cluster which is managed by a single organisation. With Raft it is possible to spread the nodes in different data centers, so if one data center becomes unavailable the other one is there to recover and continue the operation.

- **Simplified Byzantine Fault Tolerance (SBFT)**: This is a new approach which is planned to be implemented by Hyperledger Fabric. SBFT is both crash fault tolerant and byzantine fault tolerant. With this approach the nodes can reach agreement eventhough there are faulty or malicious nodes.

4.2.5 Chaincode or Smart contracts

Since Hyperledger Fabric does not support cryprtocurrencies, it however comes with the notion of "asset" which typically is a tangible good. To simplify the trade between different participants and holders of the assets, there is a need for smart contract or chaincode which holds the governance rules. Literally chaincode and smart contract refer to the same thing but technically chaincode is more than a smart contract.

Fabric uses the term chaincode to refer to a group of related smart contracts. The difference is mainly in the way it is deployed in the network, smart contract manages transactions, however chaincode governs one or multiple contracts which is packaged as a single component. The deployment of chaincode takes a different approach, first it has to be packaged, then installed and only after that it can be used. Before it is installed every chaincode can specifiy the endorsement policies which defines the conditions necessary for a valid transactions. In addition the chaincode specifies the set of endorsing peers the chaincode will be installed. This allows multiple organisations to agree on the chaincode operation before it can be used.

During chaincode installation the endosring peers that are chosen to run the chaincode ought to approve the chaincode otherwise the chaincode will not be installed. After the chaincode is installed each of the endorsing peers must endorse a transaction before it is commited.

Since agreement takes place upfront and are explicit there is no need for a deterministic chaincode. This makes Fabric exceptional because then it allows to write chaincode in a deterministic programming language. This is why Fabric allows the chaincode to be developed in Go, Java and Javascript.

4.2.6 Channels

Channels allow the creation of sub networks within the same Fabric network. This is a huge advantage for organisations that are competitors and want to make their transactions invisible to their competitors. Channels allow the participating organisations to create and maintain their own private ledger. The ledger is not visible to any other organisations except to the members participating in that channel. Each organisation would normally want to have their own peers in the Blockchain. Suppose there are three organisations A, B and C. All of them are connected with one single Channel and maintain one single ledger. If A wants to buy a specific product from B, A does not want to share the details with C. Therefore A and B would need a new channel having their own peers. Since C is not part of channel it is not able to see their ledger. Each channels can have one ledger only. Technically a channel is a subnet between two or more members.

An endorser peer may participate in more than one channel therefore it allows them to hold two ledgers or more at the same time.

A few features that hold true:

- Every channel may have only one ledger;
- A channel may have one or multiply chincodes;
- Every chaincode has its own endorser peers and endorsement policies;
- A chaincode must be installed on peers that are members of a channel where the chaincode will be deployed;

4.2.7 Ledger and State Database

Like any other Blockchain platforms also in HLF the ledger keeps track of all important information by storing the history of all transactions. While the state of the assets may change, the history behind it is immutable. In HLF the ledger contains two distinct parts, the world state and the ledger itself. The ledger is kept by all peers except a subset of orderers. Figure 4.2 shows the structure of Blockchain with the blocks and the transactions in each block. Each Blockchain is structured as back-linked list of blocks which includes a hash of the prior block. More specifically the Block header holds the hash of the previous block as well as the hash of its own transactions. There are additional information stored in the block which can be seen in Figure 4.2. The transactions are stored in Block Data and the number of transactions is not necessarily the same. The job of the ordering peers is to assemble the transactions coming from the consensus algorithm. The conditions for creating and closing the block is based either on the maximum number of transactions allowed per block or when a timeout period has elapsed which is counted from the time the first transaction has been endorsed.

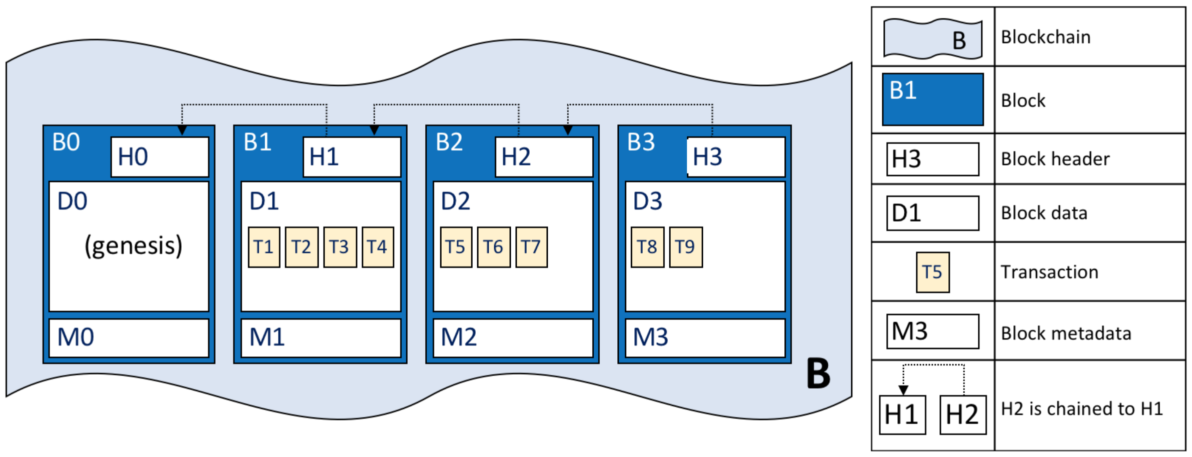


Figure 4.2: The structure of the ledger [35].

Blockchain data or asset in Hyperledger Fabric is represented as a collection of key value pairs. Assets can be in JSON form or binary. The state changes are recorded as a result of transaction execution in a set of key-value writes.

In addition to the Blockchain data there is a world state which represents the current state of the ledger. Word state similarly as the ledger holds a key value pair in a database which can be either CouchDB or LevelDB. With the world state the program can query the current most updated value of a state (key) without having to traverse the entire transaction log from the Blockchain ledger. The world state is updated upon transaction execution and follows the consensus algorithm of the network. The procedure is as follows, after the transaction is signed by endorser, executed and validated by orderer it then eventually results in the update of the world state. If the transaction is considered invalid and has not passed the lifecycle it does not update the world state.

Some peers may not join the network from the beginning so for those peers the world state is empty. However since every new transaction represents a valid change in the world state then the world state can be generated easily and within some time it will be able to fully hold the latest assets' state.

Chapter 5

Design and Implementation

This section elaborates and reasons the design decisions taken in order to materialize the idea of IoT based surveillance with Blockchain technology. In the section we discuss in detail the different approaches, technologies and communication protocols considered throughout the design decisions. We are going to also see how the IoT surveillance system with access control drives the design decisions.

5.1 Hardware Components

Before digging deeper in this section we would like to describe the hardware architecture and its components involved in the project. The video surveillance for face detection and recognition is handled by Esp Eye whereas the testing environment is based on an AMD Processor machine which in our case is macOS. In Figure 5.1 on the left hand side we have multiple Esp devices which communicate with the IoT gateway within the same network with Wi-Fi based on IEEE 802.11 standards for communication protocols.

The IoT Gateway serves as the middle man which waits for images coming from Esp Eye to be inserted into Blockchain. As the main experimental devices in our project we have Esp Eye and the raspberry Pi. But due to experimental purposes we had to run the Blockchain locally and that lead us to change the infrastructure and instead of the raspberry pi which is an ARM based processor machine we had to go for AMD based

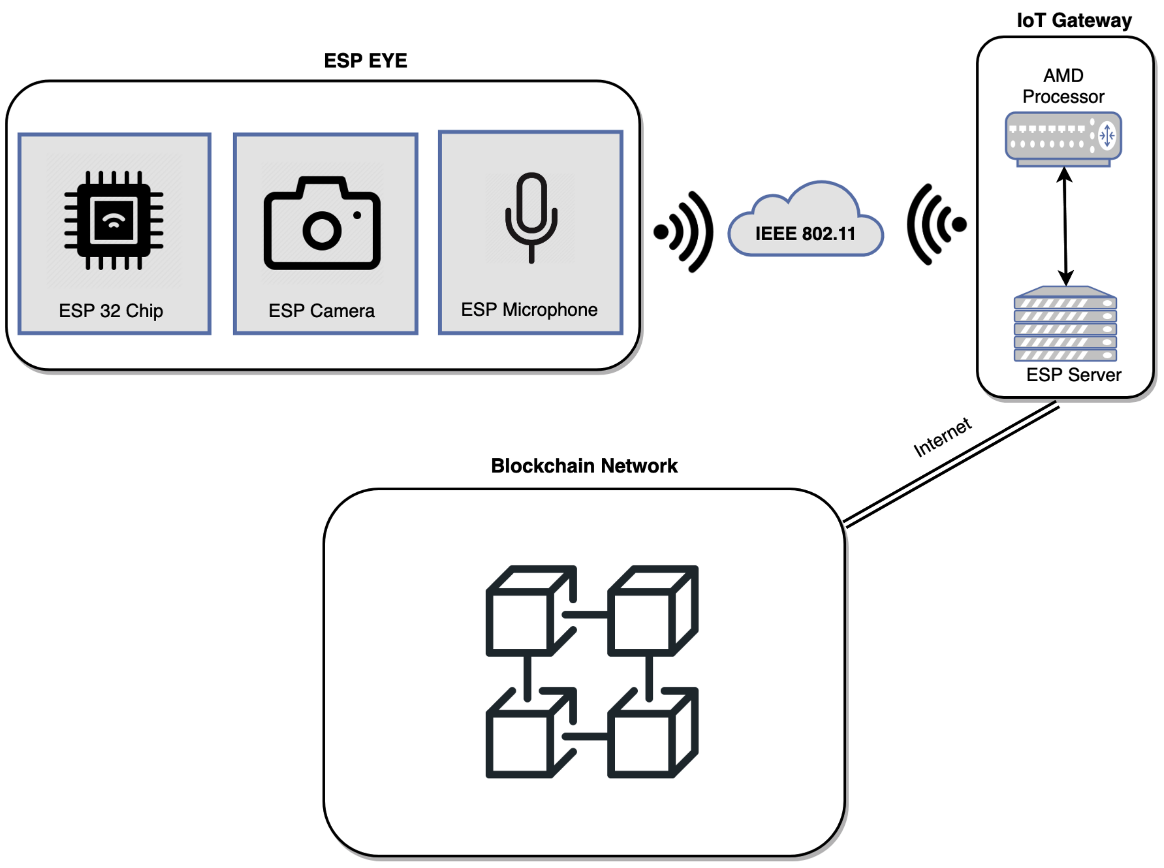


Figure 5.1: Hardware Architecture.

processor machine. This is due to the Hyperledger Fabric which runs in form of containers and its binaries are only available for an AMD processor.

5.1.1 Esp Eye for Face Detection

Esp Eye which can be seen in Figure 5.2 is an AI development board based on ESP32 chip. The device was first for sale in 2020 by the semiconductor company Espressif. There are many other devices which are based on the ESP32 family of chips. ESP32 is combined with extra components such as programming interfaces, different sensors which are used for the evaluation of the chip. At the core of the board is the dual core Tensilica LX6 processor with 8 MB PSRAM and 4MB flash.

The Esp Eye integrates an embedded microphone and a camera with 2-Megapixels. The camera is an OV2640 sensor with a maximum image size of 1600×1200 pixels. The board supports 2.4 GHz Wi-Fi technology to connect to internet through a wide area network. A Micro USB port provides the power supply and also debugging in order to use the AI APIs. It comes with an UART Chip which enables asynchronous serial communication with which the program is uploaded bits by bits.

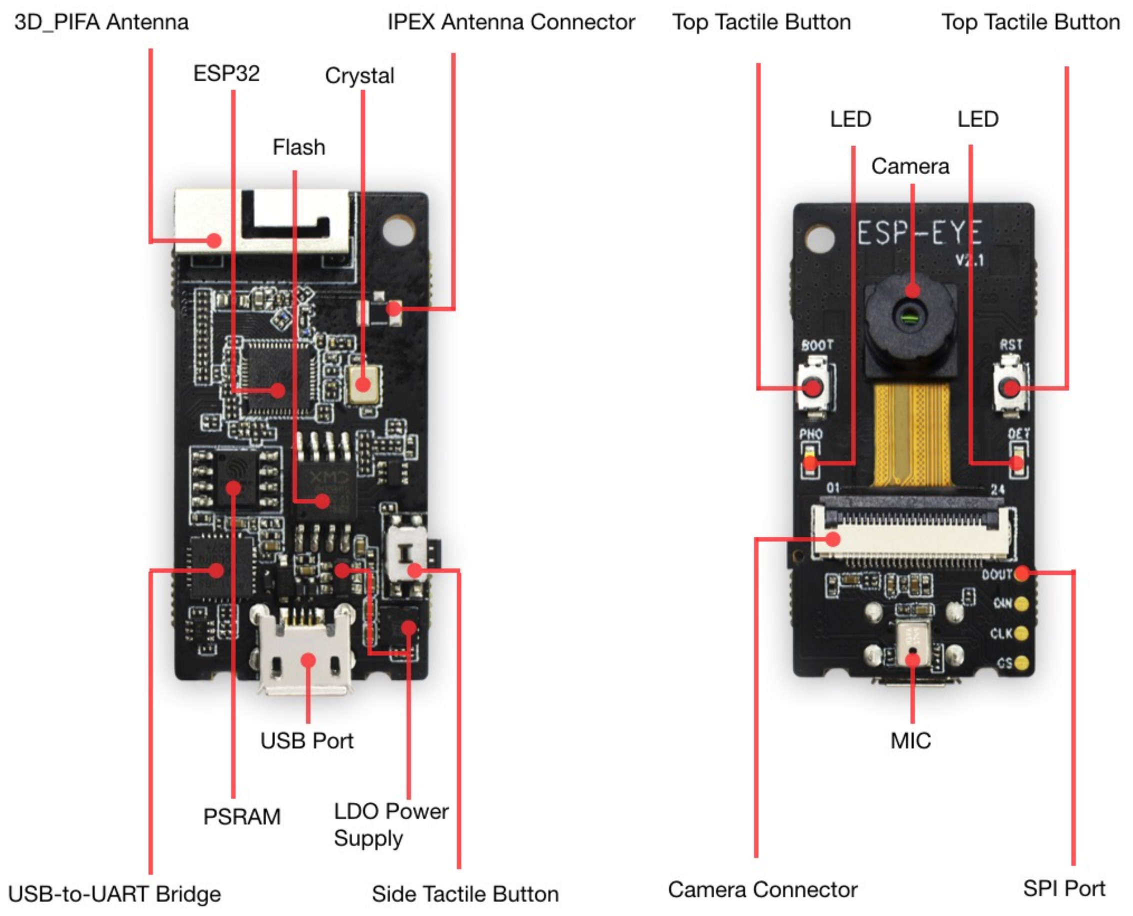


Figure 5.2: Esp Eye [36].

Esp Eye also supports additional security features: flash encryption feature and secure boot. The flash encryption is intended to secure the content of the flash memory. In Esp Eye flash encryption is done using AES-256 and the key is stored in the eFuse. eFuses keep the values intact and can not be changed by a software. Since the data in flash is encrypted upon reboot the physical readout will be impossible. On the other hand secure boot can protect the device from uploading unsigned code. The algorithm is a typical digital signature method with RSA, the public key is stored in the device itself whereas the private key is kept secret and used upon each code upload.

What makes the Esp Eye stand out of the crowd is its performance and the platform for face detection and recognition known as Esp-Who. As elaborated in Chapter 4 Esp-who comes with algorithms for face detection and recognition which makes it possible to run both in Esp Eye.

5.1.2 Raspberry PI

Raspberry Pi has been the development environment from the beginning of the project. It is very lightweight, supports many compilers and makes it ideal for experimental purposes. The esp32 family of devices are natively compiled and developed by the esp idf development platform. Similarly also the Esp-Who libraries for face detection and recognition were available in esp idf development platform. However, after many tries to setup Hyperledger Fabric in Rasperry Pi we realized that Hyperledger Fabric binaries were not possible to run in ARM processor. Therefore after that a computer with AMD processor was used.

5.2 Design

5.2.1 Surveillance System Architecture

Figure 5.3 shows a high level overview with the sotware components. It is important to mention that the design is compatible without many technological restrictions. At the beginning there were two approaches in mind: deployment of face detection and recognition on the cloud or in the Esp Eye itself. In the first approach AI is deployed on the cloud which means the camera consequently captures images and send them on the cloud for face recognition. This approach is costly, based on the previous studies in average the time required to detect a person is between 2 to 5 seconds. In addition it requires substantial amount of bandwidth which would be wasted if no one is present.

In the second approach we decided to implement the face detection and recognition run on the IoT device itself. This approach has a lot more benefits and images are captured by a tiny device powered with AA batteries. The previous approach they needed a machine on the cloud and a separate camera, whereas in this case both are converged into one single low powered device. When images are captured they are immediately passed to face detection and recognition algorithms, if no face is detected images are immediately deleted.

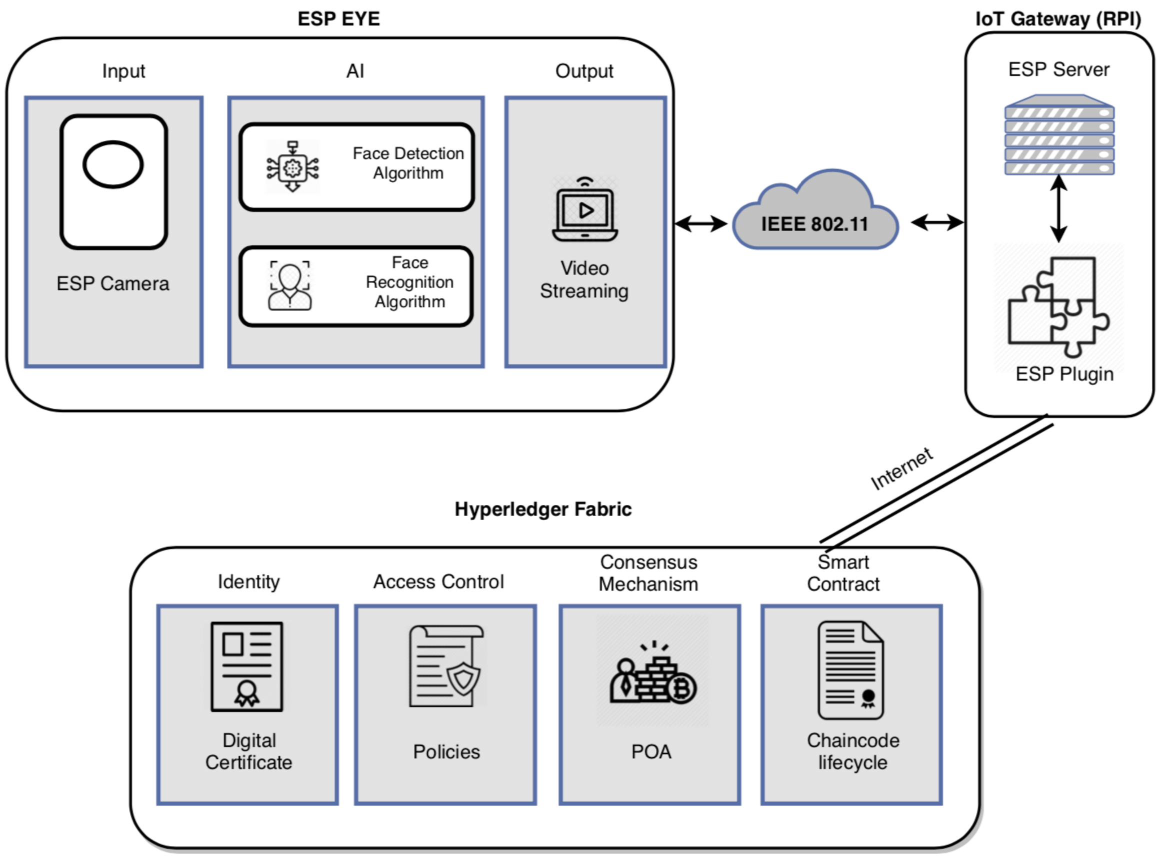


Figure 5.3: Software Architecture.

On the right hand side of the Figure 5.3 is the IoT Gateway which serves as a hub to do the rest of the job. The communication between the Esp Eye and the IoT gateway is achieved with a wireless communication compliant to IEEE 802.11 LAN protocols. It follows the client-server architecture, the Esp Eye is the client and the IoT Gateway as a server. Since we want to store the detected images in Blockchain having the Esp Eye as a client fulfills the requirements. The server will be waiting for requests coming from the Esp Eye. The other way around would not work, since the only way to send data from the server is when the client has sent a request. The first condition for the image to be stored in Blockchain is that the face has to be detected in the image. On the other hand ESP Plugin with the help of the SDK from Hyperledger Fabric will start the process of submitting the transaction to the ledger. Hyperledger Fabric enables the storage of the detected images coming from the Esp Eye.

5.2.2 Communication Protocols under consideration

When it comes to decide which communication protocol to use in Esp Eye, it is important to go for a lightweight protocol and among our choices is HTTP and MQTT both of them supported by Esp Eye with the help of IPEX antenna which enables the Wi-Fi connectivity. MQTT is an extreme lightweight protocol which follows the publish/subscribe model for communication. Whenever a client pushes a message on a subscribed topic then the message is forwarded to all the clients participating on that topic. Esp Eye clients do not have to pull information from a subscriber, but it pushes information to the server. Due to the use case where the Esp devices do not have to communicate in between them, MQTT does not qualify. Instead HTTP protocol is employed where Esp Eye and the IoT Gateway communicate in the client server architecture respectively. For the inter-machine communication the REST architectural style is used because anyways we do not want to bind the communication since we do not know when the face is detected. Besides REST is considered to be more lightweight compared to its counterparts. For the data-interchange format the JSON format is employed which is very lightweight and performs much better since its syntax follows the Javascript language so it makes it faster to read since the Esp server is also written in Javascript.

5.2.3 Face detection and recognition in Esp Eye

Although very small in size the Esp Eye still stands out of the crowd because of its features and communication standards. With 8 MB of PSRAM and 4 MB and Wi-Fi transmission of flash it gives room for deploying ESP-WHO a platform and a library for face detection and recognition. ESP-WHO is a framework and library which provides the face detection and recognition algorithms. At the beginning the implementation began with the ESP IDF platform, an SDK with software development tools which includes compiler, debugger and some plugins to integrate with additional IDEs. ESP IDF contains the essential software libraries for the face detection with MTCNN and MobileNets model and face recognition libraries implemented with MobileFaceNet model. In order to elaborate how face detection and recognition works, we have allocated Chapter 3 which explains explicitly.

Although we could have used ESP IDF development environment but rather we chose to develop it in Arduino IDE. Arduino IDE provides a lot of libraries including the libraries for ESP32 boards.

5.2.4 Hyperledger Fabric a tamper-resistant database

Hyperledger Fabric qualifies in our design because the main criteria were that Blockchain ought to be private and permissioned since the images should not be public and only available from within the organisation that applies our use case. Ethereum was another potential candidate to be deployed. Hence among the two we decided for Hyperledger Fabric because from the beginning it is meant to be private permissioned blockchain. Although Ethereum can be adapted to serve our needs but yet it comes with a consensus mechanism which is energy demanding and not flexible. Hyperledger Fabric provides a modular architecture and enables more flexibility and is tailored to the needs of our use case. For a better understanding of the architecture of HLF you can refer to Chapter 4. Hyperledger Fabric in our case stands out of the crows due to its features:

- High transaction throughput;
- Low latency for transaction confirmation;
- Confidentiality of transactions with the help of channels;
- Highly modular architecture;
- Identities supplied with digital certificate;

With its modular architecture HLF allows to be tailored to the needs and the use case. It has been designed to be highly configurable and offers the following modular components:

- A pluggable consensus or ordering service which is the core component for transaction execution.
- A pluggable Membership Service Provider responsible for managing entities and associating them with cryptographic information.
- A pluggable Certificate Authority responsible for issuing and managing the identities with digital signature cryptography.
- A smart contract implemented in a conventional programming languages known to developers.

5.3 Implementation

In this section we exhibit what it takes to implement the above mentioned design decisions. First the data flow will be demonstrated on how data flows from Esp Eye until it reaches Hyperledger Fabric. In the second part the design decisions taken will be put into action by implementing the architecture. Therefore we will show with implementation details for each component starting from the Esp Eye until the implementation needed for HLF. A number of APIs and data structures are used to materialize the idea however still the Esp Eye comes with limitations such as the max array size due to which a dynamic memory allocation is needed.

5.3.1 Data Flow Overview

In this setup the Esp Eye devices and the IoT Gateway are all connected in LAN and operate within the same subnet. In Figure 5.4 the Data Flow Diagram is depicted which shows the flow from left to the right starting with Esp Eye. Esp devices act as stations and the reason is that there are multiple Esp devices and it would not be a good choice because Esp Eye may be able to handle all the traffic and propagate then again to the server. Once Esp Eye detects a face or eventually recognizes a face it will take that exact image and forward it to the IoT Gateway. This is achieved with a REST API call using the POST request where the image is included with additional metadata. On the other hand the nodejs server located in the IoT Gateway will receive the request in this case the image with a detected person and start activate the HLF client for inclusion of transaction in the ledger.

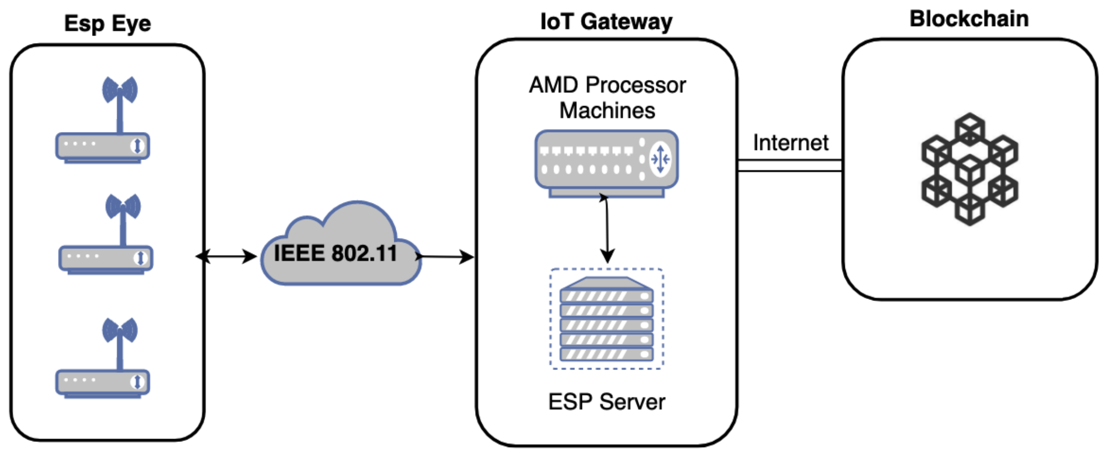


Figure 5.4: Data flow diagram

The nodejs server is responsible for receiving the data and forwarding it to the Hyperledger Fabric. In order to interact with HLF network the Fabric SDK for nodejs provides us with APIs to submit transactions to the ledger. The process of submission takes place right after the image has arrived. HLF transaction data is known as assets which are represented as a collection of key-value pairs. When the value of the key is modified by a chaincode simulated from transaction execution then the world state is updated as a new transaction on a ledger. Assets written by chaincode typically use the JSON form and or binary. Since for the image transfer the JSON format is used then this allows for sending the same JSON parameters to the blockchain. On the other hand the role of nodejs server is to receive and acknowledge the Esp Eye that data has arrived and then forward the exact same data to the blockchain. The process begins with transaction initiation where Esp Eye is assumed to be a client and such that it holds an identity issued from the MSP.

Images with recognized faces are stored in the Blockchain because of the following reasons: Firstly Hyperledger Fabric runs at no costs for transaction execution and the consensus mechanism is very lightweight and pluggable. Secondly, since Hyperledger Fabric allows for splitting the network into multiple smaller networks in order to preserve privacy between competitors then storing images in some other third party such as IPFS would vaporize the concept of channeling and IPFS does not offer the same characteristics.

5.3.2 Transmission Overview

The data transmission is depicted in Figure 5.5 which captures the interaction between all the components end to end. When an Esp Eye device established connection to the LAN it then immediately starts the process of video surveillance. With the current configurations it takes roughly 4 frames per second. Esp Eye analysis each frame for face detection and if a face is detected only then it puts into work the face recognition algorithm. Hence if a face is detected or eventually recognized the HTTP client is activated and eventually through an end point it will make a POST request.

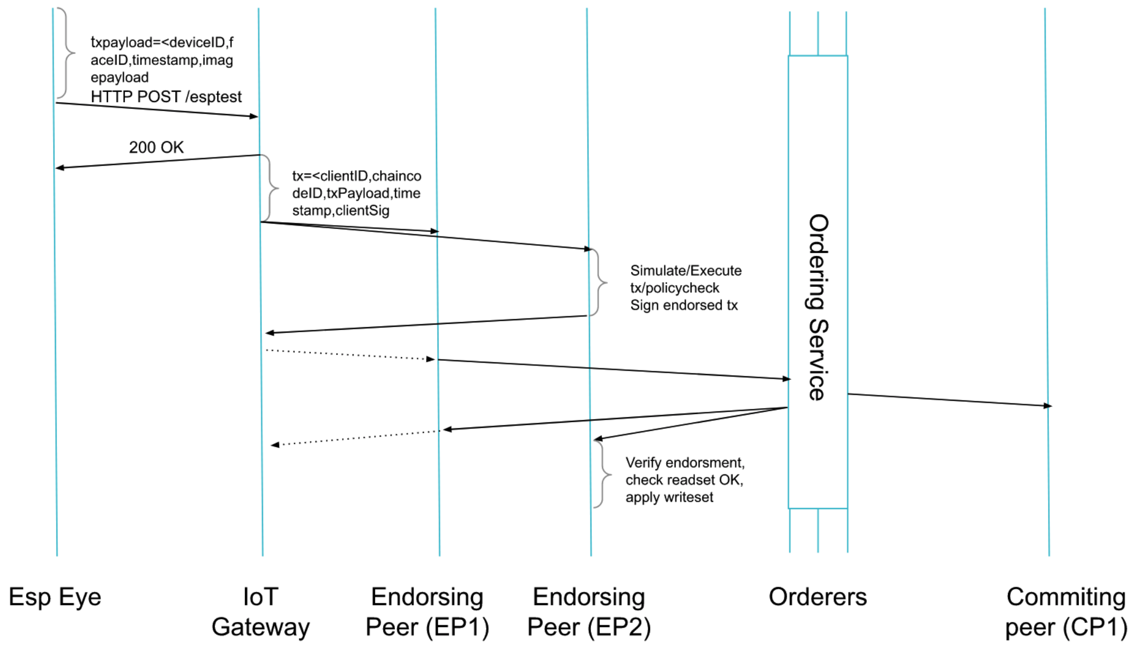


Figure 5.5: Sequence Diagram showing transaction execution

There are 4 important parameters needed to be stored in the Blockchain which are reflected in the Figure 5.6. Firstly, the DeviceID is important since we need to know from which device is this information coming as multiple devices may be employed in video surveillance. Secondly, with FaceID we will be able to identify the person without having to look on the captured frame. Additionally with timestamp we capture the time when the person is detected and finally the frame itself encoded in base64.

As in Figure 5.5 these 4 parameters will be sent first through the exposed API with an endpoint. The nodejs server located in IoT Gateway will then reply with the status code but no additional information.

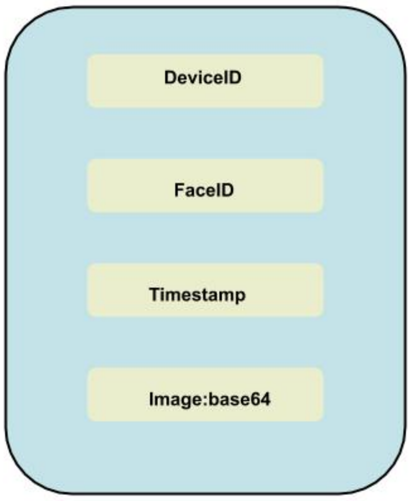


Figure 5.6: JSON parameters sent from Esp Eye

IoT Gateway initiates a transaction

Before transaction is initiated there are three assumptions necessary to keep in mind:

- The channel in Fabric is running.

- The Esp Eye is registered and enrolled with Certificate Authority and MSP contain the identity of each Esp.
- The chaincode with its endorsement policy is deployed on the peers.

Now the job of the IoT Gateway is to initiate the transaction with the 4 paremeters passing them as they are to the transaction payload. The IoT Gateways acts as a Fabric client for Hyperledger Fabric. Our endorsement policy states that both peers taking part in the consortium must endorse any transaction coming from this client. The transaction proposal now is easy constructed, since Fabric holds assets in key value pairs in our case the key will be the FaceID and all other information will be stored in a json document as values. This will make it easier to query the FaceIDs and for the world state. The transaction proposal utilizes an API for nodejs SDK to generate a transaction. The goal of the API is to invoke the functions of the chaincode with input paramters coming from Esp Eye. Each Esp Eye is issued a public private key pair which is used to sign the transaction before sending it to the endorsers for execution. In this case the Fabric-client SDK persists the public private key and signs the transaction on behalf of the Esp Eye. Although transaction signing is possible in offline mode, which means the user signs the transaction at his device and sends the signed transaction to the endorser node. In our case this was not possible to implement since the Fabric-client SDK is not available in Esp Eye devices.

Endorsing Peers EP1 and EP2 verify and execute the transaction

The endorser EP1 and EP2 from Figure 5.5 will now receive the transaction proposal which includes: clientID, chaincodeID, txPayload, timestamp and clientSignature as in Figure 5.7. Only the endorsing peers specified by the chaincode will receive this transaction proposal in our case both of them will receive a proposal. EP1 and EP2 first of all check the format of the transaction and if the signature is valid and if that is the client who claims to be using the MSP. Besides it also ensures that this client in this case Esp Eye is an authorized member of that channel. When all the conditions have passed then the endorsers invokes the chaincode function and pass the four proposed arguments. Eventually the transaction is executed, however it does not yet update the ledger. Now the endorsers sign the proposed transaction and sends back a proposal response to the fabric-client. The intent of the fabric-client now is to submit the transaction to the ordering service to update the ledger. If the fabric-client was to query the ledger then there is no need to submit anything to the ordering service. However in order for the fabric-client to submit the transaction to the ordering service both endorser EP1 and EP2 have to respond with same response proposal. However, yet at this stage the ledger is not updated.

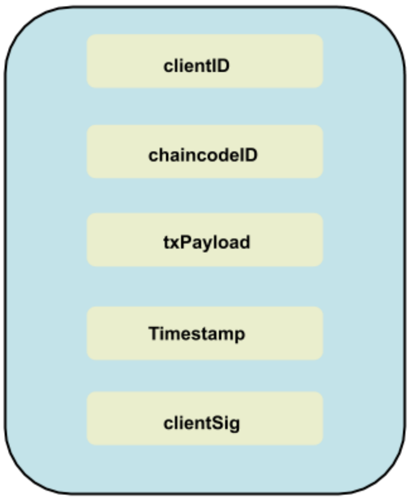


Figure 5.7: Parameters to be included in the proposed transaction.

Fabric-client submits transactions to the ordering service

Before the fabric-client submits the final version of the transaction, the application broadcasts the endorsed transaction with a message to the ordering service. The transaction proposal now contains the read/write sets besides the endorsing peers signatures. The ordering service nodes may receive transaction from other clients or other ESP devices.

The job of the ordering service is to order transactions on an agreed sequence in order to package them into blocks. After the maximum number of transactions allowed in a block is reached or the maximum time duration has passed then blocks are saved first to the ledger of the orderers.

Validate and commit the transactions to all peers

Upon receiving a broadcast message with the created block from the orderers the committing peers will again verify the signatures of the ordering nodes which are part of the ordering service. In HLF it is allowed to set which peers validate the transaction and they are known as committing peers, however all the peers in the channel ought to update the ledger. If the committing peers fail to verify the signature of the ordering peers that created the block then the block is rejected for inclusion in the ledger. Another very important validation step is to ensure that every transactions' read and write set does not lead to invalid world state. During the endorsement phase every transaction copies a set of read and write sets of the key pair that will get updated. Hyperledger Fabric in addition to key value pairs it implicitly keeps the third parameter which is the version number of that pair. When a new transaction is executed the versioning of the key pair gets updated, it is like a counter. The read set comes from the world state and the write set is the same key but with updated value and version. In our case the key will be the FaceID and the value is the new image.

The committing/validating peers will now verify the transaction by looking at the read write sets. Now the committing peer compares the read set it holds with the world state of the same key. If the versioning of the key does not match then a previous transaction could have updated the ledger. Then new version of key pair is created which means the transaction will be considered invalid and will be rejected from the block. This is how it prevents the double spending. Otherwise if the read set and its version has not been changed the transaction is then allowed to be inserted to the ledger. Eventually the write sets are committed to the world state and the detected image has reached all the peers.

Execution model

Typically in permissioned Blockchains such as Ethereum, the transaction is executed by all peers in the block level which means that the consensus protocol first orders the transactions and propagates to the peer nodes and only then the execution of transactions takes place. Therefore they operate in order-execute-validate model. The execution happens after all the transactions have been propagated to all peers by executing sequentially all the transactions in all peers.

Unlike the traditional approaches, HLF follows the execute-order-validate model which is reflected in the above described steps. This model has a lot of advantages. A transaction is not executed by all the peers but only by a subset of the peers. Besides, also the transaction endorsement and validation may be executed only by a subset of the peers which means not all the peers need to execute the chaincode. Since many peers are released from the transaction execution this can save energy and resources because some of the peers may be running far away from each other. This approach uses a non-deterministic chaincode because the peer agreement is explicit which allows to implement the chaincode in a non-deterministic programming language. In addition the Fabric allows to execute transactions in parallel in order to increase throughput. Besides the elapsed timeout for a block creation makes HLF much more efficient for transactions coming from IoT devices.

5.3.3 Esp Eye Setup

In the previous section with a sequence diagram the data flow from one object to another is elaborated. In this section we would like to stop on the most important parts of the implementation for face detection and recognition, which is entirely implemented using Arduino IDE. Since three domains are involved we would not be able to elaborate all the details here because the focus is merely on the interactions between them. Because of that the face detection and recognition algorithms implemented in Esp Eye are elaborated in more detail in Chapter 3. Therefore we will now take a look at the most important parts of implementation which materialize face detection and recognition.

Face Detection API

First of all we would like to start with introduction to the API for face detection: face_detect which is adapted from ESP WHO platform, its implementation is shown in the Listing 5.1. As it can be seen the function returns a pointer of type box_array_t which contains information about the face such as the bounding box and the face landmarks. When no face is detected the function returns NULL. In our code we just check if the the function returns NULL then do not proceed for face recognition. The face_detect function accepts two inputs of type dl_matrix3du_t and mtmn_config_t.

```
// Adapted from ESP-WHO Platform
 box_array_t *face_detect(dl_matrix3du_t *image_matrix, mtmn_config_t *config)
{
    net_config_t pnet_config = {0};
    pnet_config.w = 12;
    pnet_config.h = 12;
    pnet_config.threshold = config->p_threshold;
    box_array_t *pnet_boxes = NULL;
    if (FAST == config->type)
        pnet_boxes = pnet_forward_fast(image_matrix,
                                       config->min_face,
                                       config->pyramid_times,
                                       &pnet_config);
    else if (NORMAL == config->type)
        pnet_boxes = pnet_forward2(image_matrix,
                                   config->min_face,
                                   config->pyramid,
                                   config->pyramid_times,
                                   &pnet_config);
    if (NULL == pnet_boxes)
        return NULL;
    net_config_t rnet_config = {0};
    rnet_config.w = 24;
    rnet_config.h = 24;
    rnet_config.threshold = config->r_threshold;
    box_array_t *rnet_boxes = rnet_forward(image_matrix,
                                           pnet_boxes,
                                           &rnet_config);
    dl_lib_free(pnet_boxes->box);
    dl_lib_free(pnet_boxes);

    if (NULL == rnet_boxes)
        return NULL;
    net_config_t onet_config = {0};
    onet_config.w = 48;
    onet_config.h = 48;
    onet_config.threshold = config->o_threshold;

    box_array_t *onet_boxes = onet_forward(image_matrix,
                                            rnet_boxes,
                                            &onet_config);
    dl_lib_free(rnet_boxes->box);
    dl_lib_free(rnet_boxes);
    return onet_boxes;
} /*}}}*/
```

Listing 5.1: Face detection API [6].

The Listing 5.2 shows the definition of the image_matrix a 3 dimensional matrix with the image and additional metadata about the image unlike the box_array_t which only holds features about the face. These are information about the image width, height, number of channels, number of filters, stride and a pointer to image data. Since we are dealing with a memory constraint device typically the dynamic memory allocation is used. It allows for more efficient memory allocation and the memory is not wasted. So first the memory is dynamically allocated and then the captured image is assigned to it.

```
typedef struct
{
    int w;      /*!< Width */
    int h;      /*!< Height */
    int c;      /*!< Channel */
    int n;      /*!< Number of filter, input and output must be 1 */
    int stride; /*!< Step between lines */
    uc_t *item; /*!< Data */
} dl_matrix3du_t;
```

Listing 5.2: Image matrix structure.

The second parameter mtmn_config_t also shown in Listing 5.3 is a struct which contains configurations about the MTMN, the multi stage algorithm for face detection which is explained in detail in Chapter 3. There are a number of properties which are important because they can affect the performance of MTMN and also the processing speed. The first property is min_face which is the minimum detection size which by default is set to 80. It means that it should be able to detect all faces larger than 80x80 pixels. This property is important because we only want to detect individual faces and anyone else standing in the background is not important. Therefore the smaller the min_face is:

- the larger the number of images is generated in image pyramid;
- the higher the processing time;
- the smaller the size of detectable face is in the image;

In other words it means that MTMN will work harder to try to detect the face if the person is standing pretty far away from the camera. The pyramid holds the scale which monitors the generated pyramids with a range from 0 to 1.

Therefore the larger the pyramid is:

- the higher the number of images generated in the image pyramid;
- the higher the processing time;

The pyramid_times is another property which together with pyramid monitor the image pyramid. In addition the configuration also allows to set the threshold for each stage of the MTMN reflected in the struct threshold_config_t. The closer to 1 the lower the number of potential candidate bounding boxes and such the lower the probability of detecting a face. The struct mtmn_resize_type allows to chose between FAST and NORMAL. When it is set to FAST the pyramid equals 0.707106781. If we want to set our own value of pyramid then it has to be set to NORMAL. By FAST it means the processing time is faster compare to NORMAL mode.

```
typedef struct
  {
      float min_face;              /*!< The minimum size of a detectable face
         */
      float pyramid;               /*!< The scale of the gradient scaling for
         the input images */
      int pyramid_times;           /*!< The pyramid resizing times */
      threshold_config_t p_threshold; /*!< The thresholds for P-Net. */
      threshold_config_t r_threshold; /*!< The thresholds for R-Net. */
      threshold_config_t o_threshold; /*!< The thresholds for O-Net. */
      mtmn_resize_type type;       /*!< The image resize type. 'pyramid' will
         lose efficacy, when 'type'==FAST. */
  } mtmn_config_t;
```

Listing 5.3: MTMN configurations [6].

In the Listing 5.1 of the face_detect function the three stages of MTMN are implemented. As discussed in Chapter 3 in the P-net stage the input image size is 12x12 this is reflected with assigning the pnet_config.w and pnet_config.h to 12. Similarly this happens for R-net and O-net where the width and height is set to 24x24 and 48x48 respectively. Once the function has reached the end it will return either NULL if no face is detected or else will return the bounding box, the score and landmarks.

Face Recognition API

The input for face recognition is only the bounding box or the aligned face with landmarks for the position of eye, nose and mouth. The recognize_face function accepts two parameters, the face_id_list and the aligned_face.

```
int8_t recognize_face(face_id_list *l, dl_matrix3du_t *algined_face);
```

Listing 5.4: Face Recognition API

The face id list also shown in Listing 5.5 has a number of parameters that keeps track of the face IDs, where the most important information about the face is stored in a 3 dimensional vector id_list.

```
typedef struct
  {
      uint8_t head;          /*!< head index of the id list */
      uint8_t tail;          /*!< tail index of the id list */
      uint8_t count;         /*!< number of enrolled ids */
      uint8_t size;          /*!< max len of id list */
      uint8_t confirm_times; /*!< images needed for one enrolling */
      dl_matrix3d_t **id_list; /*!< stores face id vectors */
   } face_id_list;
```

Listing 5.5: Face Id List [6].

The job of recognize_face function is to generate a new face_id from the aligned_face parameter. The newly generated face_id is then compared with all the face ids stored in the face_id_list. With the help of Cosine similarity it measure how similar two vectors are. The Cosine similarity will then obtain the distance between the newly generated face_id with each face_id stored in the face_id_list. The closer the Cosine value to 1 the higher the match between two vectors and vice versa. As a matter of fact it is almost impossible to have a Cosine value 1 of two compared faceIDs. This is challenging due to image differences in terms of light, pose and facial expression. Due to that we can set a threshold `FACE_REC_THRESHOLD` to either increase or decrease the recognition rate. If it obtains a Cosine value greater than the threshold when it compares two face IDs then it is considered to be the same person.

Sending detected image to IoT Gateway

The image is sent to IoT Gateway either when a face is detected or recognized. The WiFiClient library in Arduino enables to establish the connection to the IoT Gateway by using the IP address of the IoT Gateway. The Esp Eye makes an HTTP request using the POST method through an exposed API from the IoT Gateway. Throughout this study for submitting the image to the IoT Gateway two ways were used:

1. multipart/form-data;
2. application/json;

With “multipart/form-data” the image is sent in series of parts. This approach allows sending a file or image without looking at its type. This approach was used at the beginning of the project implementation when a python server was used and we did not have information about the Hyperledger Fabric implementation which uses Javascript for interaction with Blockchain. This approach is beneficial when we do not want to capture the raw image in bytes.

The second approach “application/json” is a standard format of sending structured data. It is useful for sending plain text or any other type of data. Since HLF also uses JSON format to store assets and the only way to store the image in it is to convert the image in data type JSON supports. Hence the best option is to store it as a String and to do that the image is first converted to `base64` and when needed we can convert it back to an image type. So then we decided to do the conversion in Esp Eye itself and send the image as `base64` using JSON format. The Listing 5.6 shows the parameters that are sent to the IoT Gateway and the exact same parameters are inserted into Fabric ledger.

```
{
       DeviceId:
       Face_Id:
       Timestamp:
       ImageData:
}
```

Listing 5.6: JSON structure in Esp Eye.

5.3.4 IoT Gateway

The IoT Gateway holds two important components, the nodejs server and the Hyperledger Fabric SDK that serves as the client for Hyperledger Fabric Blockchain. The Express.js an application server framework enables to build the server in nodejs. With an exposed API in nodejs the Esp Eye is able to send the data as in the Listing5.7. The nodejs server will then pass the data to the Hyperledger Fabric SK for further propagation to the ledger.

```
app.post("/esptest", async (req, res) => {

   if (req.body.ImageData) {
      console.log(" Device Id is:-", req.body.DeviceId);
      console.log(" Face id is :-", req.body.Face_Id);
      console.log(" Base64 Image is:-", req.body.ImageData);
      console.log(" Date Time is:-", req.body.Timestamp);
   }

   res.sendStatus(200);
   console.log("Receided image and uploading to blockchain");
   main(req.body.Face_Id, req.body.DeviceId, req.body.Timestamp,
      req.body.ImageData);
});
```

Listing 5.7: The exposed API for handling requests coming from Esp Eye.

5.3.5 Hyperledger Fabric Setup

Hyperledger Fabric allows two or multiple organisations to be part of the ledger and the transaction lifecycle. In our use case of video surveillance we have two organisations and each organisation has one peer. The first organisation can be the institute itself which is willing to use and deploy the video surveillance and the other organisation would be the surveillance company which is monitoring the access control.

The Hyperledger Fabric implementation has two parts of the code: the chaincode implementation and the client implementation.

Chaincode implementation

The chaincode implementation which is deployed and executed inside the network is shown in the Listing 5.8. First of all the fabric-contract-api module for nodejs is used for the implementation of Hyperledger Fabric chaincode which provides the Contract interface. This API provides an entry to write the chaincode functions in order to allow the endorsing peers and Blockchain clients to communicate. HLF makes it possible to write chaincode in Go, Java and Javascript (nodejs). One of the biggest advantage is that no matter in which of the above languages the chaincode is implemented, it still allows the client to use a different language for communication but it has to be among one of the mentioned programming language. Hence we decided to go for Javascript implementation of the chaincode since the client is also implemented in Javascript.

There are a number of functions implemented each with its own purpose which can be split into two categories, the functions for inserting images into the ledger and querying images from the ledger. For inserting images coming from the Esp Eye the create_Image function is implemented. This function receives five arguments: ctx, faceid, device_id, timestamp and the imageitself. The first argument ctx is the transaction context which keeps the state of the transaction. First of all it allows to maintain variables when invoking transactions. Secondly, it offers a number of very important APIs to establish communication and perform transaction processing. As you can see in the Listing 5.8 ctx.stub.putState function assembles the arguments into key value pairs, where the key is the faceid and the rest of arguments will be inserted as values to that key. The ctx.stub is used to access the putState API which writes to the ledger. Besides ctx.stub provides very important APIs for inserting into the ledger and querying it.

```
'use strict';
```

```
const { Contract } = require('fabric-contract-api');

class FabImages extends Contract {

    async initLedger(ctx) {
        console.info('============= START : Initialize Ledger ===========');
        const images = [
            {
                device_id: 'Device 1',
                timestamp: '2020 - 12 - 13T01: 47: 38Z',
                docType: 'image',
                imageitself: 'AAAFCAYAAACNbybl',

            }
        ];

        for (let i = 0; i < images.length; i++) {
            images[i].docType = 'image';
            await ctx.stub.putState('Face id 0',
               Buffer.from(JSON.stringify(images[i])));
            console.info('Added <--> ', images[i]);
        }
        console.info('============= END : Initializing Ledger ===========');
    }

    async queryImage(ctx, faceid) {
        const imageAsBytes = await ctx.stub.getState(faceid); // get the image
            from chaincode state
        if (!imageAsBytes || imageAsBytes.length === 0) {
            throw new Error(`${faceid} does not exist`);
        }
        console.log(imageAsBytes.toString());
        return imageAsBytes.toString();
    }

    async createImage(ctx, faceid, device_id, timestamp, imageitself) {
        console.info('============= START : Create Image ===========');

        const image = {
            device_id,
            timestamp,
            docType: 'image',
            imageitself

        };

        await ctx.stub.putState(faceid, Buffer.from(JSON.stringify(image)));
        console.info('============= END : Create Image ===========');
    }
```

```
async queryAllImages(ctx) {
    const startKey = '';
    const endKey = '';
    const allResults = [];
    for await (const { key, value } of ctx.stub.getStateByRange(startKey,
        endKey)) {
        const strValue = Buffer.from(value).toString('utf8');
        let record;
        try {
            record = JSON.parse(strValue);
        } catch (err) {
            console.log(err);
            record = strValue;
        }
        allResults.push({ Key: key, Record: record });
    }
    console.info(allResults);
    return JSON.stringify(allResults);
}

async retrieveHistory(ctx, key) {
    console.info('getting history for key: ' + key);
    let iterator = await ctx.stub.getHistoryForKey(key);
    let result = [];
    let res = await iterator.next();
    while (!res.done) {
        if (res.value) {
            console.info(`found state update with value:
                ${res.value.value.toString('utf8')}`);
            const obj = JSON.parse(res.value.value.toString('utf8'));
            result.push(obj);
        }
        res = await iterator.next();
    }
    await iterator.close();
    return result;
}

async query(ctx, key) {
    console.info('querying for key: ' + key);
    let returnAsBytes = await ctx.stub.getState(key);
    let result = JSON.parse(returnAsBytes);
    return JSON.stringify(result);
}

async querySpecificKey(ctx, key) {
    console.info('querying for key: ' + key);
    let returnAsBytes = await ctx.stub.getState(key);
    let result = JSON.parse(returnAsBytes);
    return result;
    }
}

module.exports = FabImages;
```

Listing 5.8: Chaincode for interacting with the network.

Other functions implemented in the chaincode are useful for querying the ledger in different ways. For instance the function queryAllImages retrieves the world state of all key pairs in the ledger. This has been useful for testing purposes. In this case the function getStateByRange API is used to iterate over all sets of keys from the startKey until the endKey. Additionally the functions querySpecificKey allows to query the world state of a specific key. This is achieved with the help of ctx.stub.getState API which retrieves the current value of the passed key. When we want to see the history of all values for a key the function retrieveHistory can be used which takes in a key and returns the entire history of all values ever inserted for the given key.

Hyperledger Fabric Client implementation

The Hyperledger Fabric Client is how the outside world interacts with the Blockchain network. In order for the client to interact with the Fabric network and chaincodes the fabric-network module for nodejs is used. Typically in Bazo and Ethereum a developer needs to know the exposed gateway to interact with the network which normally the RESTfull services are used.

Hyperledger Fabric offers a number of classes through the module fabric-network for nodejs to make calls to the network. The most important class of this module is Gateway. It includes various methods to establish connection and interaction with the fabric network. The Listing 5.9 shows snippets of the client implementation. First of all the existing participant in our case the Esp Eye is a participant. This participant is connected to the fabric network in the right channel and will be allowed to access a particular chaincode which are all defined previously in the participant's access rights. Once the connection is established then the participant or client is allowed to submit transactions. Then the submitTransaction function will submit a transaction to the endorsing peers using a specified function of a chaincode by passing arguments needed for the function being used. The transaction function will then be evaluated by the endorsing peers which will then further be submitted to the ordering service to eventually be inserted into ledger.

```
// Create a new gateway
const gateway = new Gateway();
await gateway.connect(ccp, { wallet, identity: 'appUser', discovery: {
    enabled: true, asLocalhost: true } });

// Get the network (channel) where the contract is deployed.
const network = await gateway.getNetwork('mychannel');

// Get the contract.
const contract = network.getContract('imagecontractupdated');

// Submit a transaction.
await contract.submitTransaction('createImage', faceid, deviceid,
    timestamp, imagebase64);
console.log('Transaction has been submitted');

// Disconnect from the gateway.
await gateway.disconnect();
```

Listing 5.9: Client establishing the connection and submitting a transaction.

We have seen now how a transaction is submitted using the fabric-network module. In order to retrieve and check what has been submitted is in the ledger we can take the advantage of evaluateTransaction method. The purpose of this method is to query the ledger using the functions in the chaincode which resembles the GET method in HTTP requests. The chaincode function will only be evaluated by the endorsing peers and will not involve the ordering service and there will be no changes to the ledger state.

5.3.6 Frontend Implementation

Considering that the image from its origin Esp Eye until it reaches the Blockchain is of type base64 and we are not able to see visually the image for that reason a web application is developed. For the development of web application we are using react Javascript library. The web application will take the advantage of the same local nodejs server for the backend. In the nodejs server we have exposed another gateway which can be seen in the Listing 5.10. The client retrieves the image in base64 from the ledger by making a single REST call using the GET method. The getFile endpoint will expect a key which is the faceID of a person. After that another client for Hyperledger Fabric is employed which actually is the same client (participant) which inserts images to the ledger. Similarly as in the Listing 5.9 the client first establishes the connection and only then queries the ledger. As already mentioned the evaluateTransaction method of the contract instance allows to read the ledger state.

```
app.get("/getFile", async (req, res) => {
    if (req.query.filehash) {
        const chunks = [];
        console.log("Key value:", req.query.filehash);
        //let key = 'Face_id_0';
        const result = await queryfrontend.main(req.query.filehash);
        console.log(result);
        res.send(result);
        // res.send(Buffer.concat(chunks).toString());
    } else {
        res.status(501).send("Please provide face id");
    }
});
```

Listing 5.10: The API gateway serving the web application.

To retrieve the image the client has to query the ledger which means that we need to have a method in the chaincode to do so. Since we have implemented one chaincode then the method is reflected in the Listing 5.8 with the name querySpecificKey. The function expect the faceID of the person and only then the image will be returned to the client and be displayed on the web application.

Chapter 6

Evaluation

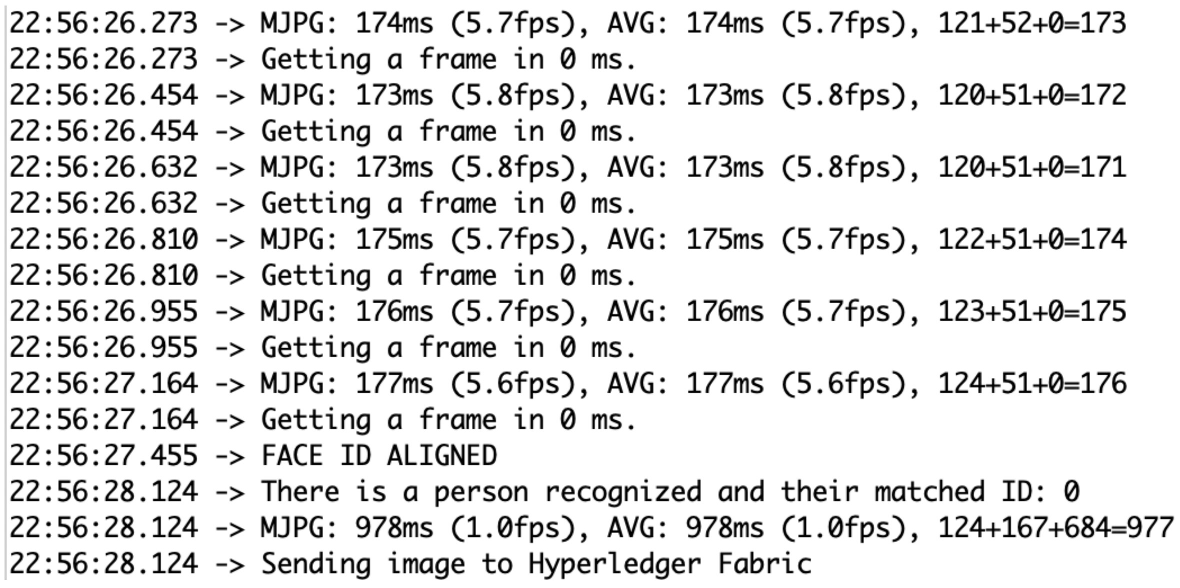
In this section the design and implementation will be evaluated from different perspectives. Some of the most important criteria are: Reliability of the overall design, efficiency and the time it takes from the moment the picture is captured and a person is recognized until the picture is inserted in the ledger, the energy efficiency of the Esp Eye and the quality of the image captured and stored in Hyperledger Fabric.

6.1 Evaluation and Discussion of the Proposed Approaches

6.1.1 Processing Delay of Face detection and Recognition

First of all the idea of performing face detection and recognition in Esp Eye while incorporating Hyperledger Fabric is an approach in itself because typically the face detection and recognition runs either on the cloud or on a local computer with the help of a camera which is only capable of taking images.

Based on our research shown in Chapter 2, we have seen that offloading face detection and recognition to the cloud takes at least 2 seconds and goes up to 5 seconds and much more processing power and energy is required. Therefore in this section we prove that face detection and recognition takes in average a second but with less energy and processing power than the offloading method and achieve the same results. In Figure 6.1 we can see a screen shot of the processing time by calculating the total time in milliseconds it takes for the face detection and recognition algorithm. For instance the first loop has these values 121+52+0+0 which equals 173ms. The first value is the time it takes for the loop until it reaches the face detection, which means the camera has already captured an image. The second value is 52 ms which is the time the face detection algorithm needs until it decides if there is a face or not. The third value represent the time it takes for the recognition algorithm but since no face has been detected therefore it is 0 ms. In the same screenshot towards the end we can see that a person has been detected and recognized. This time we can see that the third value is not empty but it holds 684 ms. Therefore the total time for face detection and recognition is 977 ms which includes the preparation time for the setup. But we can also see that it has taken longer this time for face detection algorithm. This is interesting because this time the Multi-task Cascaded Convolutional Networks comes into play. Since there have been more candidate windows for face detection therefore the algorithm will increase the granularity of detection using the three stages. This means when no face is detected in the first stage of the MTCNN algorithm then the other stages do not get activated.



```
22:56:26.273 -> MJPG: 174ms (5.7fps), AVG: 174ms (5.7fps), 121+52+0=173
22:56:26.273 -> Getting a frame in 0 ms.
22:56:26.454 -> MJPG: 173ms (5.8fps), AVG: 173ms (5.8fps), 120+51+0=172
22:56:26.454 -> Getting a frame in 0 ms.
22:56:26.632 -> MJPG: 173ms (5.8fps), AVG: 173ms (5.8fps), 120+51+0=171
22:56:26.632 -> Getting a frame in 0 ms.
22:56:26.810 -> MJPG: 175ms (5.7fps), AVG: 175ms (5.7fps), 122+51+0=174
22:56:26.810 -> Getting a frame in 0 ms.
22:56:26.955 -> MJPG: 176ms (5.7fps), AVG: 176ms (5.7fps), 123+51+0=175
22:56:26.955 -> Getting a frame in 0 ms.
22:56:27.164 -> MJPG: 177ms (5.6fps), AVG: 177ms (5.6fps), 124+51+0=176
22:56:27.164 -> Getting a frame in 0 ms.
22:56:27.455 -> FACE ID ALIGNED
22:56:28.124 -> There is a person recognized and their matched ID: 0
22:56:28.124 -> MJPG: 978ms (1.0fps), AVG: 978ms (1.0fps), 124+167+684=977
22:56:28.124 -> Sending image to Hyperledger Fabric
```

Figure 6.1: Screenshot of serial port showing the running time.

6.1.2 Data-interchange format for transporting the image

In order to store and transport the image from the Esp Eye to Hyperledger Fabric we employed the multipart/form-data and the application/JSON. Both approaches do the job however they differ in the way they send the data. Using the first approach the image is split into chunks as binary data and send as it is without providing a structure to represent fields. This approach does not allow to send additional fields such as faceID. Besides with multipart/form-data the IoT Gateway receives it into chunks therefore it has to write the data in a disk and cannot hold it on run time. Due to that then JSON format was used with which data is well structured and each field can be captured by the IoT Gateway as is. This allows to hold data on the run time which is beneficial since it will then anyways be inserted into the HLF ledger.

6.1.3 IPFS for storing images

IPFS a file sharing system is typically used by decentralized application to store files instead of storing files in the Blockchain ledger. The main reasons behind is saving costs and escaping from storing the files in multiple blocks due to the limit of amount of data allowed for a block to hold. This is typically the case when a cryptocurrency-based Blockchain is used for public Blockchain use cases. In the case of private and permissioned Blockchains it is not necessary to have cryptocurrencies when all entities are known and the Blockchain is used to store assets of collaborating organisations. Hyperledger Fabric overcomes such challenges from which the current Blockchain platforms are suffering from.

Although IPFS is implemented, it is not put into work in our design. We argue that IPFS is not necessary in our case and not beneficial to be used with Hyperledger Fabric. First of all in Hyperledger Fabric there are not many nodes like in public blockchains but a counting number of nodes. Besides HLF does not care about the types of values inserted in a key pair. Therefore almost any type of file can be converted in base64 which makes it possible to store the entire file in the ledger itself. The maximum number of bytes to be stored in a block can be configured and by default the block size can go up to 98 MB. In IPFS all traffic is public unless the content of files are encrypted. This is not useful for Hyperledger Fabric which allows subnetting the network into channels for privacy. Eventhough the files may be encrypted in IPFS it does not make sense to store all the files in a single filesharing system which may belong to different channels in Hyperledger Fabric.

6.1.4 End to End Testing

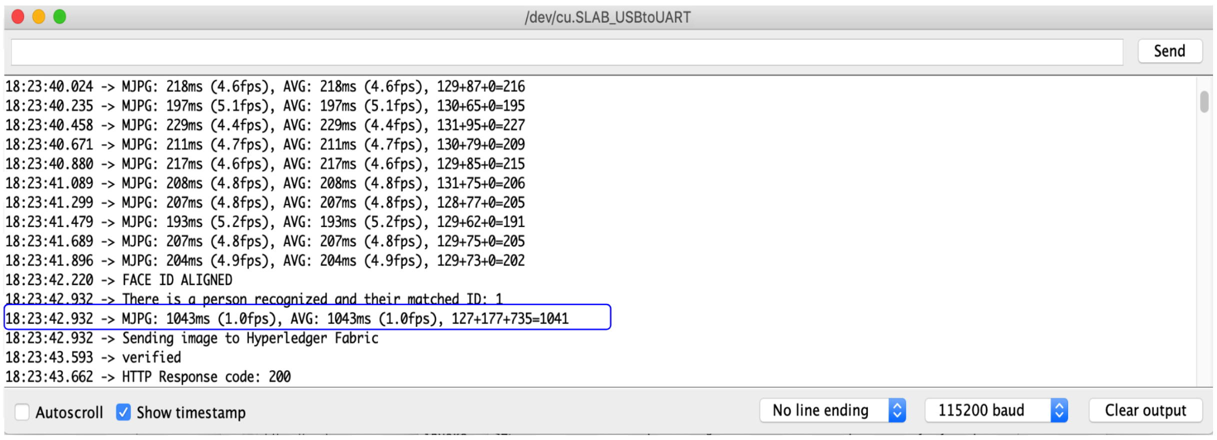
The entire end to end testing is performed in order to ensure that all the components work together and the software behaves as expected. In order to avoid looking at the logs on what has happened a small web application is implemented and besides we have integrated an easy to use open source browser Fabric explorer which fetches all the information about the ledger such as the number of transaction and blocks and so on. To begin with testing and evaluation these recommended steps need to be followed:

1. The Esp Eye is powered up using an external charger power bank.
2. Hyperledger Fabric is started with the help of docker. Once all the containers are running a new admin is registered and a new user is registered.
3. The IoT Gateway, more specifically the nodejs server is started to listen on port 8585.
4. The web application is started off using reactjs.
5. The Fabric Explorer is attached to the running Hyperledger Fabric network.

When all the above steps are running and working sound then we perform testing on a detected and recognized face.

Esp Eye Integration Testing

At this phase the assumption is that the faces are registered in the database and once the person position himself/herself in front of the device the Esp Eye should recognize the person and trigger an API call to the IoT Gateway. This can be seen in Figure 6.2 from the Serial monitor of Arduino and also reflected in the nodejs server where we print out the image in base64. For testing purposes 3 persons were registered in the database with each having faceIDs starting from 0 to 2. In Figure 6.2 it can be seen that the person with faceID 1 is recognized. Furthermore the total time for recognition is shown which is 1041 ms. It also can be seen that from the time the serial port prints "FACE ID ALIGNED" until the matched ID is displayed we have 732 ms which matches quite close with the third value calculating the total time which 735 ms. The http response shows that the image has reached the nodejs server successfully.

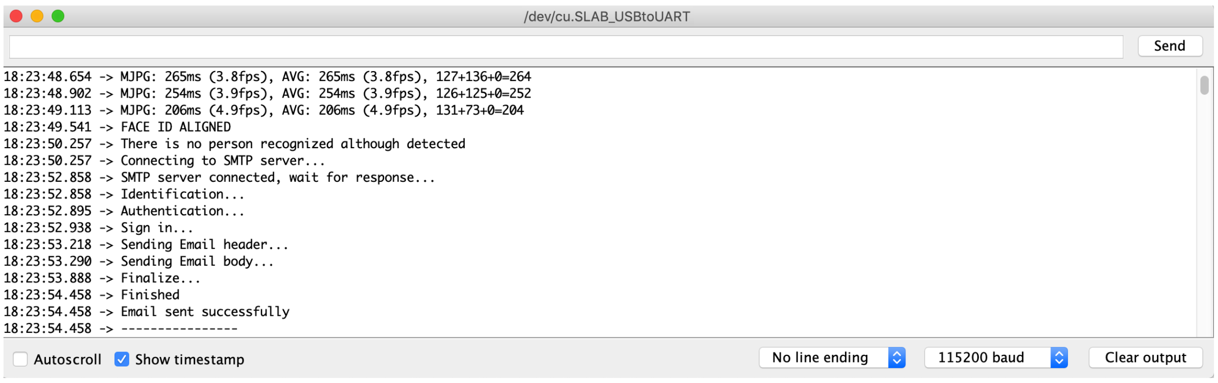


```
18:23:40.024 -> MJPG: 218ms (4.6fps), AVG: 218ms (4.6fps), 129+87+0=216
18:23:40.235 -> MJPG: 197ms (5.1fps), AVG: 197ms (5.1fps), 130+65+0=195
18:23:40.458 -> MJPG: 229ms (4.4fps), AVG: 229ms (4.4fps), 131+95+0=227
18:23:40.671 -> MJPG: 211ms (4.7fps), AVG: 211ms (4.7fps), 130+79+0=209
18:23:40.880 -> MJPG: 217ms (4.6fps), AVG: 217ms (4.6fps), 129+85+0=215
18:23:41.089 -> MJPG: 208ms (4.8fps), AVG: 208ms (4.8fps), 131+75+0=206
18:23:41.299 -> MJPG: 207ms (4.8fps), AVG: 207ms (4.8fps), 128+77+0=205
18:23:41.479 -> MJPG: 193ms (5.2fps), AVG: 193ms (5.2fps), 129+62+0=191
18:23:41.689 -> MJPG: 207ms (4.8fps), AVG: 207ms (4.8fps), 129+75+0=205
18:23:41.896 -> MJPG: 204ms (4.9fps), AVG: 204ms (4.9fps), 129+73+0=202
18:23:42.220 -> FACE ID ALIGNED
18:23:42.932 -> There is a person recognized and their matched ID: 1
18:23:42.932 -> MJPG: 1043ms (1.0fps), AVG: 1043ms (1.0fps), 127+177+735=1041
18:23:42.932 -> Sending image to Hyperledger Fabric
18:23:43.593 -> verified
18:23:43.662 -> HTTP Response code: 200
```

Autoscroll ✓ Show timestamp No line ending 115200 baud Clear output

Figure 6.2: Arduino serial monitor showing the person being recognized.

Figure 6.3 shows a case when a person is detected but not registered in the database. Although the person is detected but not recognized it takes roughly the same time as if the person was registered in database. In order to ensure a safety so that no intruder trespass a notification email is sent to the gatekeeper.

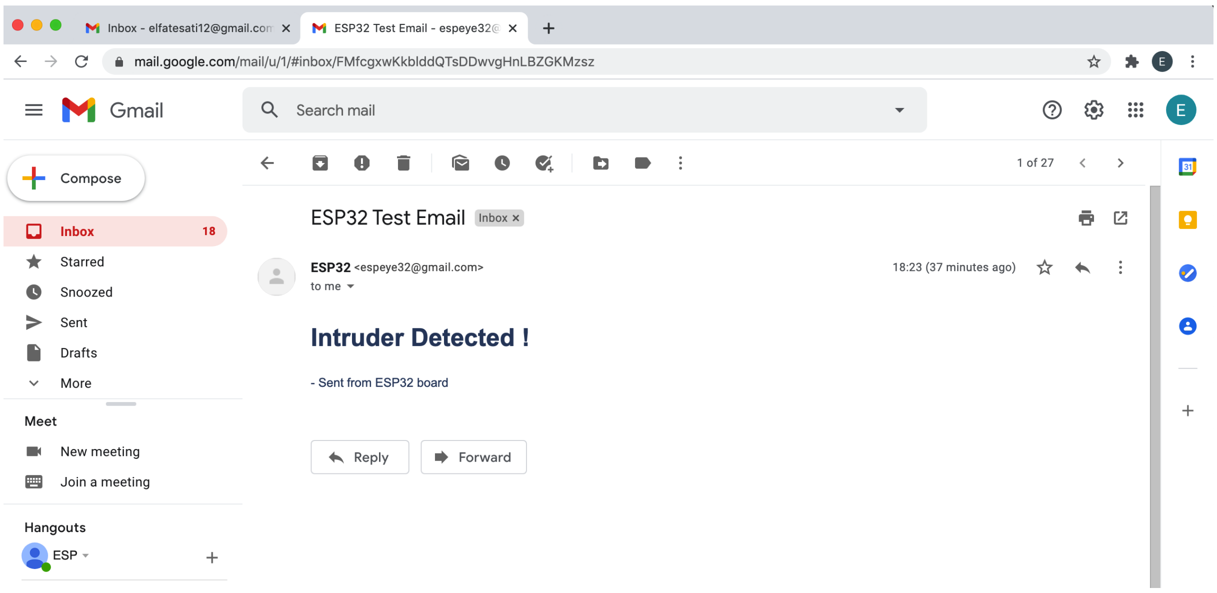


```
18:23:48.654 -> MJPG: 265ms (3.8fps), AVG: 265ms (3.8fps), 127+136+0=264
18:23:48.902 -> MJPG: 254ms (3.9fps), AVG: 254ms (3.9fps), 126+125+0=252
18:23:49.113 -> MJPG: 206ms (4.9fps), AVG: 206ms (4.9fps), 131+73+0=204
18:23:49.541 -> FACE ID ALIGNED
18:23:50.257 -> There is no person recognized although detected
18:23:50.257 -> Connecting to SMTP server...
18:23:52.858 -> SMTP server connected, wait for response...
18:23:52.858 -> Identification...
18:23:52.895 -> Authentication...
18:23:52.938 -> Sign in...
18:23:53.218 -> Sending Email header...
18:23:53.290 -> Sending Email body...
18:23:53.888 -> Finalize...
18:23:54.458 -> Finished
18:23:54.458 -> Email sent successfully
18:23:54.458 -> ----------------
```

Autoscroll ✓ Show timestamp No line ending 115200 baud Clear output

Figure 6.3: Arduino serial monitor showing an intruder being detected.

A dummy email account was created for testing purposes. Figure 6.3 confirms that an email is received for an intruder alert and the time matches with the time the email was sent which can is reflected in Figure 6.3.


Inbox - elfatesati12@gmail.com

ESP32 Test Email - espeye32@

mail.google.com/mail/u/1/#inbox/FMfcgxwKkbIddQTsDDwvgHnLBZGKMzsz

ESP32 Test Email Inbox

ESP32 <espeye32@gmail.com>
to me

18:23 (37 minutes ago)

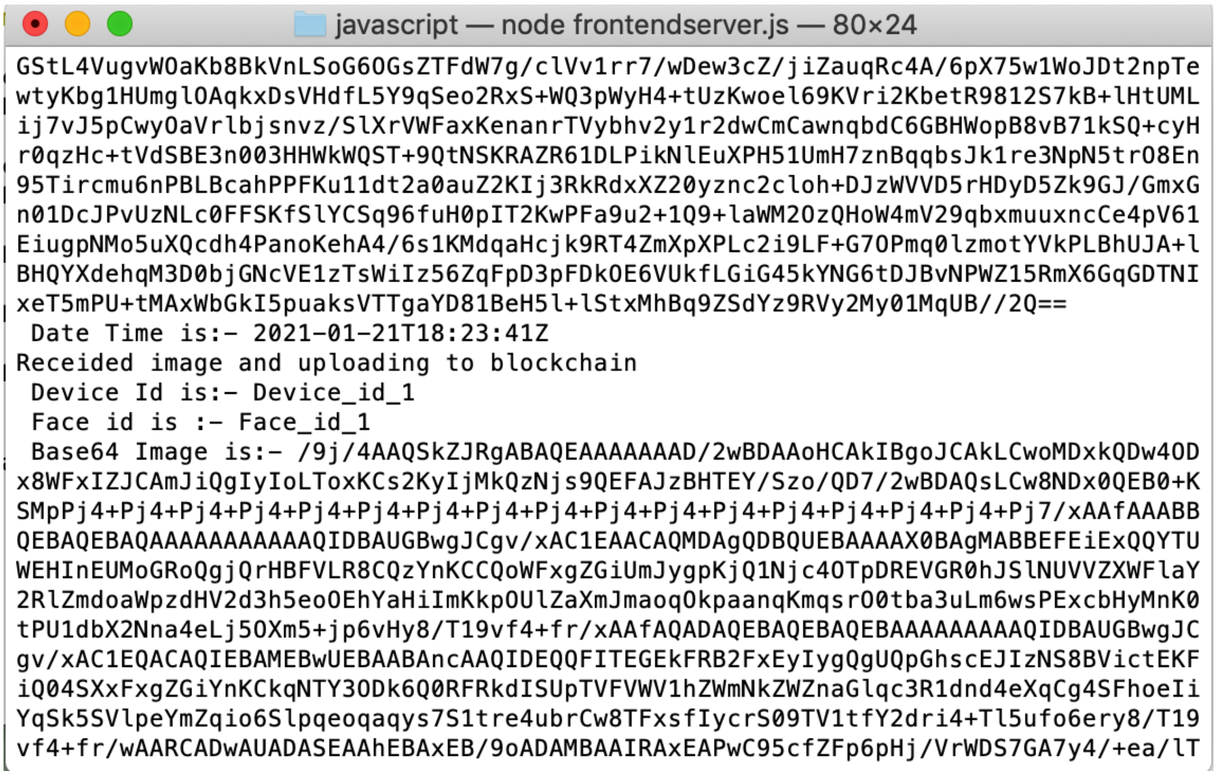
Intruder Detected !

- Sent from ESP32 board

Figure 6.4: Email account showing notification for intruder alert.

IoT Gateway Integration Testing

It is the responsibility of nodejs to handle the API calls and serve as the middle man for establishing communication between Esp Eye and Hyperledger Fabric. For experimental purposes we have printed out what is coming from Esp Eye, this can be seen in Figure 6.5. The four parameters sent are shown in the screenshot. Basically the last parameter is the timestamp. The screenshot in Figure 6.5 has captured two requests coming from Esp Eye, from the first request we can see part of the image in base64 and the Timestamp. The second request begins with deviceID followed with other parameters and the image. It is important to notice that the timestamp proves that this image is coming from the person with faceID 1, recognized within the timeframe shown in the Figure 6.2. There is no doubt that this is the same image we are referring in Figure 6.2.



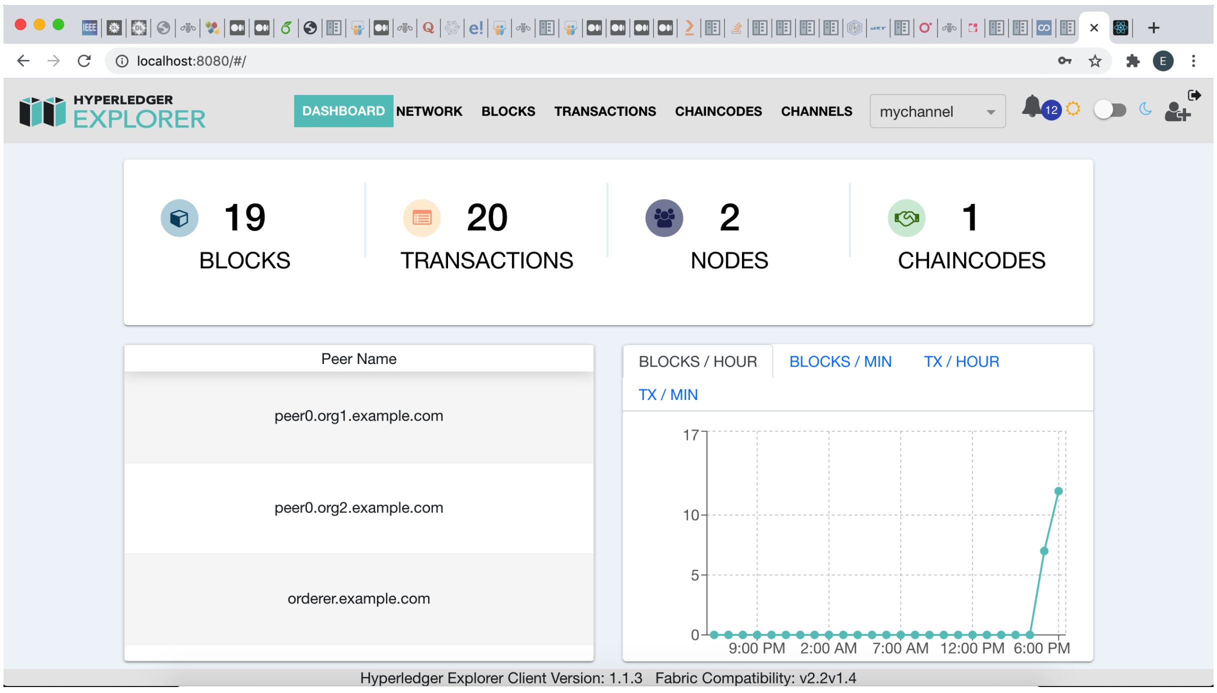
```
GStL4VugvWOaKb8BkVnLSoG6OGsZTFdW7g/clVv1rr7/wDew3cZ/jiZauqRc4A/6pX75w1WoJDt2npTe
wtyKbg1HUmglOAqkxDsVHdfL5Y9qSeo2RxS+WQ3pWyH4+tUzKwoel69KVri2KbetR9812S7kB+lHtUML
ij7vJ5pCwyOaVrlbjsnvz/SlXrVWFaxKenanrTVybhv2y1r2dwCmCawnqbdC6GBHWopB8vB71kSQ+cyH
r0qzHc+tVdSBE3n003HHWkWQST+9QtNSKRAZR61DLPikNlEuXPH51UmH7znBqqbsJk1re3NpN5tr08En
95Tircmu6nPBLBcahPPFKu11dt2a0auZ2KIj3RkRdxXZ20yznc2cloh+DJzWVVD5rHDyD5Zk9GJ/GmxG
n01DcJPvUzNLc0FFSKfSlYCSq96fuH0pIT2KwPFa9u2+1Q9+laWM2OzQHoW4mV29qbxmuuxncCe4pV61
EiugpNMo5uXQcdh4PanoKehA4/6s1KMdqaHcjk9RT4ZmXpXPLc2i9LF+G7OPmq0lzmotYVkPLBhUJA+l
BHQYXdehqM3D0bjGNcVE1zTsWiIz56ZqFpD3pFDkOE6VUkfLGiG45kYNG6tDJBvNPWZ15RmX6GqGDTNI
xeT5mPU+tMAxWbGkI5puaksVTTgaYD81BeH5l+lStxMhBq9ZSdYz9RVy2My01MqUB//2Q==
 Date Time is:- 2021-01-21T18:23:41Z
Receided image and uploading to blockchain
 Device Id is:- Device_id_1
 Face id is :- Face_id_1
 Base64 Image is:- /9j/4AAQSkZJRgABAQEAAAAAAAD/2wBDAAoHCAkIBgoJCAkLCwoMDxkQDw4OD
x8WFxIZJCAmJiQgIyIoLToxKCs2KyIjMkQzNjs9QEFAJzBHTEY/Szo/QD7/2wBDAQsLCw8NDx0QEB0+K
SMpPj4+Pj4+Pj4+Pj4+Pj4+Pj4+Pj4+Pj4+Pj4+Pj4+Pj4+Pj4+Pj4+Pj4+Pj4+Pj4+Pj7/xAAfAAABB
QEBAQEBAQAAAAAAAAAAAQIDBAUGBwgJCgv/xAC1EAACAQMDAgQDBQUEBAAAAX0BAgMABBEFEiExQQYTU
WEHInEUMoGRoQgjQrHBFVLR8CQzYnKCCQoWFxgZGiUmJygpKjQ1Njc4OTpDREVGR0hJSlNUVVZXWFlaY
2RlZmdoaWpzdHV2d3h5eoOEhYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0
tPU1dbX2Nna4eLj5OXm5+jp6vHy8/T19vf4+fr/xAAfAQADAQEBAQEBAQEBAAAAAAAAAQIDBAUGBwgJC
gv/xAC1EQACAQIEBAMEBwUEBAABAncAAQIDEQQFITEGEkFRB2FxEyIygQgUQpGhscEJIzNS8BVictEKF
iQ04SXxFxgZGiYnKCkqNTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqCg4SFhoeIi
YqSk5SVlpeYmZqio6Slpqeoqaqys7S1tre4ubrCw8TFxsfIycrS09TV1tfY2dri4+Tl5ufo6ery8/T19
vf4+fr/wAARCADwAUADASEAAhEBAxEB/9oADAMBAAIRAxEAPwC95cfZFp6pHj/VrWDS7GA7y4/+ea/lT
```

Figure 6.5: IoT Gateway handling the image and aggregating it to Hyperledger Fabric.

Hyperledger Fabric Integration Testing

One way to check the logs in Hyperledger Fabric is by just displaying the logs of a docker container or the logs of the orderer peer. However this is not ideal solution since we have to read a lot from the Terminal. Fabric Explorer a Web application tool is used to view additional details about the blockchain and the network deployed. It allows to view the number of bocks, transactions and the associated data, information about the network, chaincodes and any other relevant information about the network as in Figure 6.6. It provides interfaces for participants and for administrators. We are logged in as Administrator and can see all the details about the ledger and besides it also provides us with plots where it shows how many transaction each organisation created.

As there were no error shown in the console of nodejs server then the image has arrived in Hyperledger Fabric. This can also be confirmed in Fabric Explorer where we can look on more details on the transactions. For this particular recognition the transaction created is highlighted in blue in Figure 6.7. The timestamp Hyperledger Fabric captures is the time when a transaction is being submitted to the Fabric for insertion in the ledger.

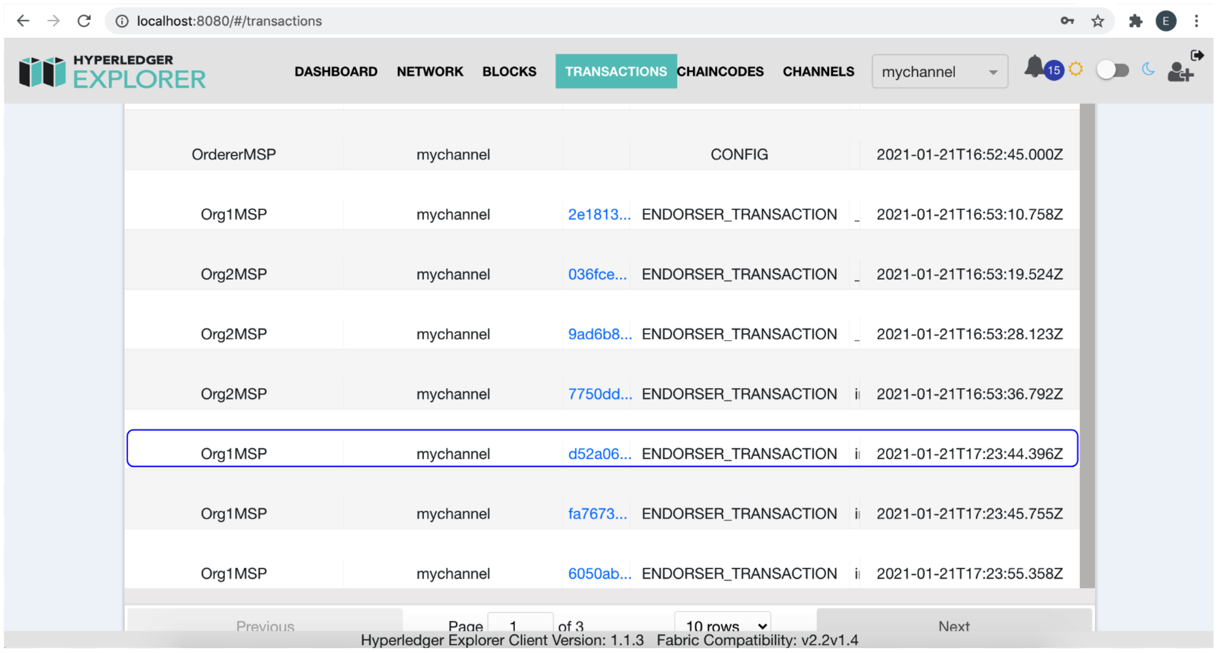


19 BLOCKS | 20 TRANSACTIONS | 2 NODES | 1 CHAINCODES

Peer Name
peer0.org1.example.com
peer0.org2.example.com
orderer.example.com

Hyperledger Explorer Client Version: 1.1.3 Fabric Compatibility: v2.2v1.4

Figure 6.6: Fabric Explorer showing information about the deployed network.



OrdererMSP	mychannel		CONFIG		2021-01-21T16:52:45.000Z
Org1MSP	mychannel	2e1813...	ENDORSER_TRANSACTION	_	2021-01-21T16:53:10.758Z
Org2MSP	mychannel	036fce...	ENDORSER_TRANSACTION	_	2021-01-21T16:53:19.524Z
Org2MSP	mychannel	9ad6b8...	ENDORSER_TRANSACTION	_	2021-01-21T16:53:28.123Z
Org2MSP	mychannel	7750dd...	ENDORSER_TRANSACTION	i	2021-01-21T16:53:36.792Z
Org1MSP	mychannel	d52a06...	ENDORSER_TRANSACTION	i	2021-01-21T17:23:44.396Z
Org1MSP	mychannel	fa7673...	ENDORSER_TRANSACTION	i	2021-01-21T17:23:45.755Z
Org1MSP	mychannel	6050ab...	ENDORSER_TRANSACTION	i	2021-01-21T17:23:55.358Z

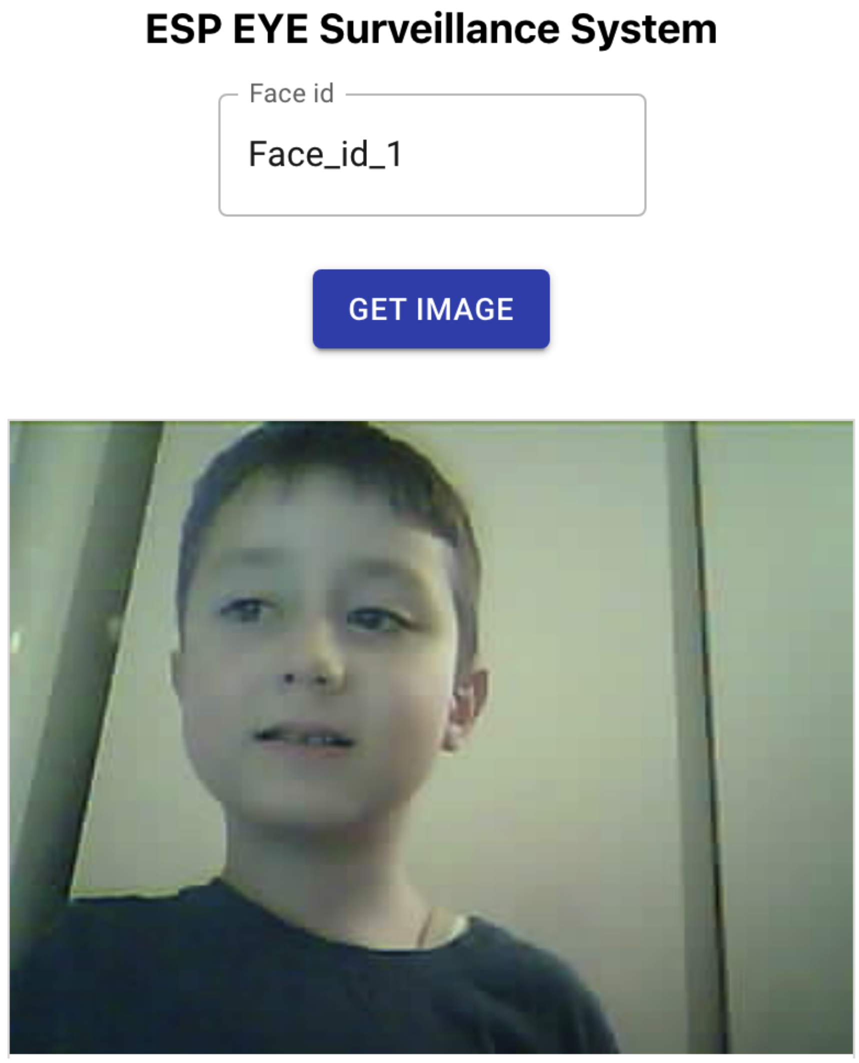
Previous Page 1 of 3 10 rows Next

Hyperledger Explorer Client Version: 1.1.3 Fabric Compatibility: v2.2v1.4

Figure 6.7: Fabric Explorer showing transactions and their timestamp.

Web Application Integration Testing

Figure 6.8 shows the recognized person which is fetched from the world state. The user has to type the faceID and only after that they can see the image of the person who is last detected and recognized. After performing multiple tests we can confirm that for each detected or recognized person the image is sent in Hyperledger Fabric although there is a time delay which we are going to look at in the next section.


ESP EYE Surveillance System

Face id: Face_id_1

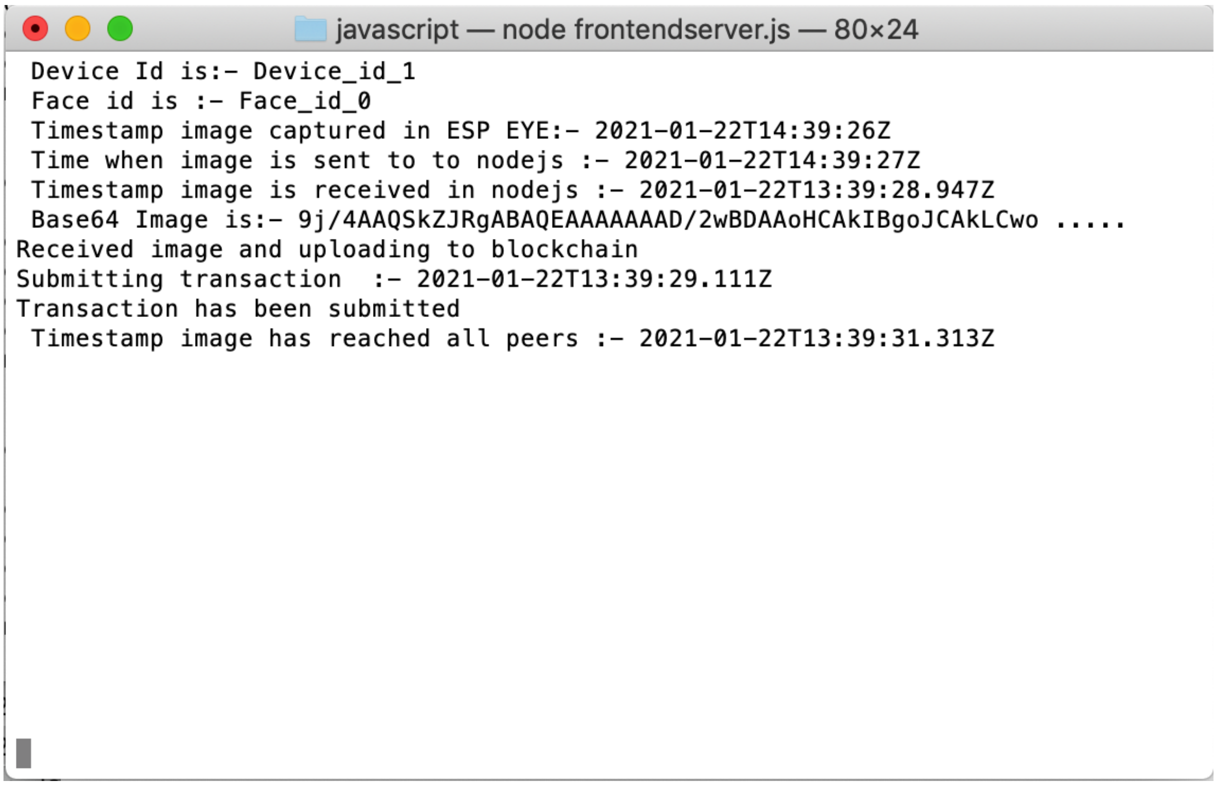
Figure 6.8: Web Application showing the person detected.

6.1.5 End to End Processing Delay

In order to measure efficiency and performance end to end we find that processing time is an important criteria. Many Blockchain platforms typically suffer from the low transaction rate. Fabric executes transactions before the transactions are being committed to the peers, so we measured the time by printing the timestamp at different time points to ensure a precise time measurement. The different time points we measure reflected in Figure 6.9 are:

- Image is captured no face detection or recognition has been performed yet;
- A person has been detected and or recognized and the image is sent to IoT Gateway;
- Image has reached the IoT Gateway;
- Image has been submitted to Fabric;
- Image is inserted in the ledger and ordering service is finished;

The nodejs timestamp is experiencing time zone issues therefore it is one hour ahead of Esp, therefore this is wrong and should be 1 hour less.



```
 Device Id is:- Device_id_1
 Face id is :- Face_id_0
 Timestamp image captured in ESP EYE:- 2021-01-22T14:39:26Z
 Time when image is sent to to nodejs :- 2021-01-22T14:39:27Z
 Timestamp image is received in nodejs :- 2021-01-22T13:39:28.947Z
 Base64 Image is:- 9j/4AAQSkZJRgABAQEAAAAAAAD/2wBDAAoHCAkIBgoJCAkLCwo .....
Received image and uploading to blockchain
Submitting transaction  :- 2021-01-22T13:39:29.111Z
Transaction has been submitted
 Timestamp image has reached all peers :- 2021-01-22T13:39:31.313Z
```

Figure 6.9: Measuring time consumption with different timestamps.

Table 6.1 shows action points of execution of each subtask starting from Esp Eye capturing the image until the image reaches its way in the ledger. Similarly as we have seen earlier face detection and recognition takes in average 1 second if there is a face detected in the image otherwise it is much less. Sending the image from Esp Eye to IoT Gateway takes almost 2 seconds. Furthermore the time it takes from IoT Gateway until the transaction is submitted to Fabric is very small. However Hyperledger Fabric has also consumed little more than 2 seconds which is considered to be high for Fabric. As a matter of fact Hyperledger Fabric has two configurable parameters, BatchTimeout and BatchSize. BatchTimeout refers to a mechanism in which a block has an upper bound time out on how long it takes for the block to be cut. The time out starts from the moment a transaction is submitted to the blockchain. BatchSize determines the maximum number of transactions allowed in a block. Hence in this case this is the only transaction then it is logical because the default BatchTimout is set to 2 seconds. Therefore the lower the number of transactions the higher the wasting time. This also leads to a high number of blocks but it minimizes the idle time of the ordering peers. Therefore depending on the use case these parameters may need to be tweaked such that there is no bottleneck in the transaction arrivals.

Action Point	**Timestamp**
Image is captured by Esp Eye	**2021-01-22T14:39:26Z**
Image is being sent to IoT Gateway	**2021-01-22T14:39:27Z**
Image is received by IoT Gateway	**2021-01-22T14:39:28.947Z**
Image is submitted to Fabric	**2021-01-22T14:39:29.111Z**
Image has reached all the peers	**2021-01-22T14:39:31.313Z**
Total Time	**5.313 seconds**

Table 6.1: Table with different action points and the total time needed end to end.

6.1.6 Image Quality and Energy Efficiency

From the end to end evaluation and the time consumption we can see that face recognition and HLF are time consuming. HLF can be configured to decrease the time out of the block cut. However, this comes with the trade off such that we restrict the transaction arrival rate which leads to having less number of Esp devices running at the same time. On the other hand the face detection and recognition can be traded off against two configurable parameters, image quality and MTMN or face detection algorithm which slightly allows some parameters to be changed. Both of the mentioned parameters may affect the recognition rate and the number of frames captured per second. Eventually this leads to increased energy consumption.

Image Quality

The OV2640 camera embedded in Esp Eye is also supported by Esp Who platform which can be configurable in terms of frame size, pixel format and quality features against the MTMN for face detection. The struct `camera_config_t` makes it possible to tweak these parameters.

The frame size may be set to one of the following options:

- FRAMESIZE_CIF (400 x 296);
- FRAMESIZE_QVGA (320 x 240);
- FRAMESIZE_VGA (640 x 480);
- FRAMESIZE_SVGA (800 x 600);
- FRAMESIZE_XGA (1024 x 768);
- FRAMESIZE_SXGA (1280 x 1024);
- FRAMESIZE_UXGA (1600 x 1200);

From the above mentioned framesizes available for OV2640 camera not all of them are possible for face recognition. One of the main reasons behind is that for each of the captured frame a 3 dimensional matrix has to be allocated for all 3 channels. The memory allocation is done using the function shown in Listing 6.1. The function takes in 4 parameters, n being the number of frames, the width and height of the frame and the number of channels. For each frame the memory to be allocated is performed by the function `dl_lib_calloc` which allocates memory space for each pixel in 3 channels, by multiplying all the parameters where each memory space takes 4 bytes of float data type. Assume we decide to capture images with FRAMESIZE_UXGA which has dimensions 1600 x 1200 and 3 channels. Therefore we have 1x1600x1200x3= 5760 000 and each space takes 4 bytes 5760000x4= 23040000 bytes which in total makes 23.05 mega bytes. Therefore this makes it impossible to allocate memory just for the image without considering the fact that face detection algorithm itself generates multiple candidate frames for detection. Hence FRAMESIZE_XGA, FRAMESIZE_SXGA and FRAMESIZE_UXGA have been tested and memory allocation failed and eventually cannot be considered for face detection and recognition. Therefore it is not necessary to further evaluate them based on other quality measures.

```c
static inline dl_matrix3d_t *dl_matrix3d_alloc(int n, int w, int h, int c)
{
    dl_matrix3d_t *r = (dl_matrix3d_t *)dl_lib_calloc(1,
        sizeof(dl_matrix3d_t), 0);
    if (NULL == r)
    {
        printf("internal r failed.\n");
        return NULL;
    }
    fptp_t *items = (fptp_t *)dl_lib_calloc(n * w * h * c, sizeof(fptp_t), 0);
    if (NULL == items)
    {
        printf("matrix3d item alloc failed.\n");
        dl_lib_free(r);
        return NULL;
    }

    r->w = w;
    r->h = h;
    r->c = c;
    r->n = n;
    r->stride = w * c;
    r->item = items;

    return r;
}
```

Listing 6.1: Allocating memory for three dimensional matrix.

For the FRAMESIZE_SVGA and FRAMESIZE_VGA memory could be allocated with size 5.76 MB and 3.68 MB respectively. However, this leaves little space for image pyramid and candidate frames for face detection algorithm. Although the video streaming is running the best we can get is 0.6 frames per second and it takes 750 ms to capture the image and 828 ms for face detection algorithm which is quite high compared to FRAMESIZE_QVGA as the best option for the face detection and recognition. For those frames that memory could be allocated other quality measures for face detection could be twisted to get the best results. These quality measures can be seen at the Listing 6.2. The minimum face size to be detected, the image pyramid, the type and number of candidates all affect the overall speed and recognition rate. After twisting these parameters for the FRAMESIZE_SVGA we could only get lower time of face detection but yet the face was not detected in all cases. After changing min_face to 200 the speed for face detection was dropped from 828 ms to 53 ms in average. Since the face was not detected after many tries these two option are not considered for deployment.

```
mtmn_config_t mtmn_config = {0};
 mtmn_config.type = NORMAL;
 mtmn_config.min_face = 80;
 mtmn_config.pyramid = 0.7;
 mtmn_config.pyramid_times = 4;
 mtmn_config.p_threshold.score = 0.6;
 mtmn_config.p_threshold.nms = 0.7;
 mtmn_config.p_threshold.candidate_number = 20;
 mtmn_config.r_threshold.score = 0.7;
 mtmn_config.r_threshold.nms = 0.7;
 mtmn_config.r_threshold.candidate_number = 10;
 mtmn_config.o_threshold.score = 0.7;
 mtmn_config.o_threshold.nms = 0.7;
 mtmn_config.o_threshold.candidate_number = 1;
```

Listing 6.2: Face detection configurable parameters.

With FRAMESIZE_QVGA and FRAMESIZE_CIF the Esp Eye with face detection and recognition run smoothly and both perform almost equally against quality measures. One difference between them is the number of frames per second. The larger the image in size the more processing time it takes. Therefore due to higher width the FRAMESIZE_CIF consumes more processing power and it can achieve in average 3.2 fps whereas with FRAMESIZE_QVGA we can have 5.2 fps. Both FRAMESIZE_QVGA and FRAMESIZE_CIF were evaluated against the parameters as in Listing 6.2.

The first parameter min_face was tested with values to 80 and 200. When min_face is set to 200 then face detection algorithm took less processing time. In average it took 700 ms for the face to be recognized whereas when min_face is 80 it took 1000 ms, the difference mainly lies in the processing time for face detection. The higher the min_face the less processing power is needed. However, the recognition rate may decrease since the person to be detected must stay close to the camera because it cannot reach to detect faces smaller than 200x200 in the frame.

Regarding the pyramid which accepts values between 0 and 1, the testing was performed with lowest to highest values, with value 2 the recognition rate fall and towards the highest value the recognition rate improved a lot. When pyramid was set lower then 0.3 Esp Eye was not able to detect any faces and the processing time remained unchanged. However, with pyramid_times we were not able to see any differences when the values between 1 and 100 were assigned.

Additionally the MTMN for face detection allows to maintain a threshold around the number of candidate windows generated which contain the face and other landmarks. In order to detect a person it is not possible to be done once but instead a number of probable faces or candidate windows is generated. In each stage of MTMN a number of candidate windows is generated, from P-net to R-net the more potential candidate windows drop and fine grained window remains. Each candidate window has a score which is the probability that it contains a human face. For that reason there is a score threshold, the larger the score is the larger will be the number of filtered out candidate windows and due to that the detection rate may lower. Candidates with highest score are selected and forwarded to the next stage. After several experiments by tweaking the score and the number of candidate windows to be generated we can conclude that only when the score is 0.8 and above we could notice that the detection rate lowered down and hardly the algorithm could detect a face and vice versa. However after setting the candidate number low and high there was no affect either on the detection rate or processing time.

Energy Efficiency of Esp Eye

A similar setup as in the above section the energy consumption of Esp Eye is measured against image quality and quality features for face detection. The measurements in mWh collected are depicted in the Figure 6.10. After several testing with different image quality and parameters there was no difference in terms of energy consumption. Throughout the experiment the energy consumption increased gradually. The Figure 6.10 illustrates the linear growth by the same amount in each unit of time. There is an increase of 10 mWh per minute. There is no difference in the energy consumption in the case when a face is detected and when no face is detected.

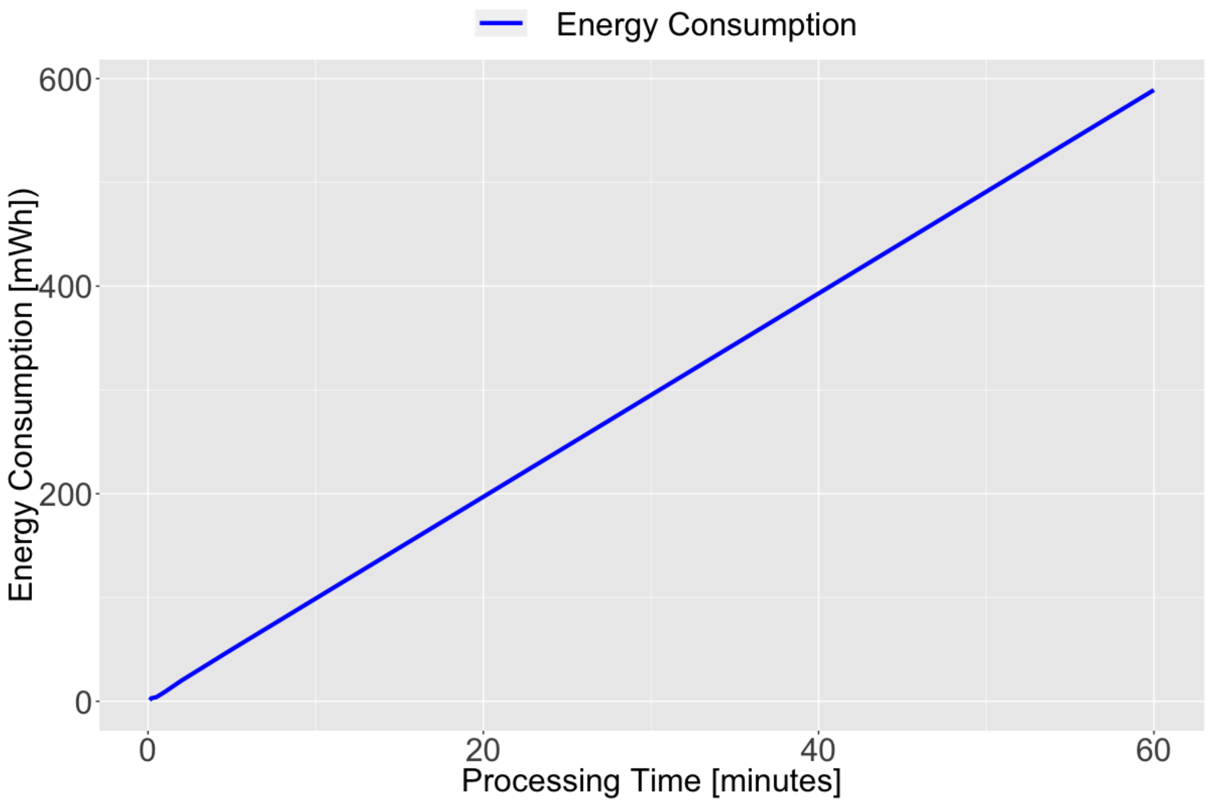


Figure 6.10: Energy Performance.

Chapter 7

Future Work

The proposed design for video surveillance with access control using Esp Eye and HLF has been materialized. The implemented design is steered by the use case and technologies used. In Figure 7.1 is a diagram which depicts how it could be realized in practice. Esp Eye has the ability to control the relay through the help of GPIOs. The Relay will then activate the electric door strike to unlock the door. So when a face is recognized GPIO pins will activate the Relay and it closes and then the current flows from the high power supply.

7.0.1 Limitations and Improvements

The first limitation is that the Esp Who platform for face detection and recognition does not provide any documentation. Nevertheless after getting familiar with it, it was possible to understand the APIs. Secondly, after a lot of research and tries we did not find any working libraries for asymmetric or symmetric encryption in Esp Eye. This is the major drawback of Esp Eye since digital integrity is important in our case.

On the other hand Hyperledger Fabric offers signing of transactions either online or offline. In offline mode the transaction must be fabric-signed before reaching endorsing peers. Unfortunately, Hyperledger Fabric APIs for offline signing of transactions are not available for IoT devices. First of all, signing of transaction require to have Elliptic Curve Digital

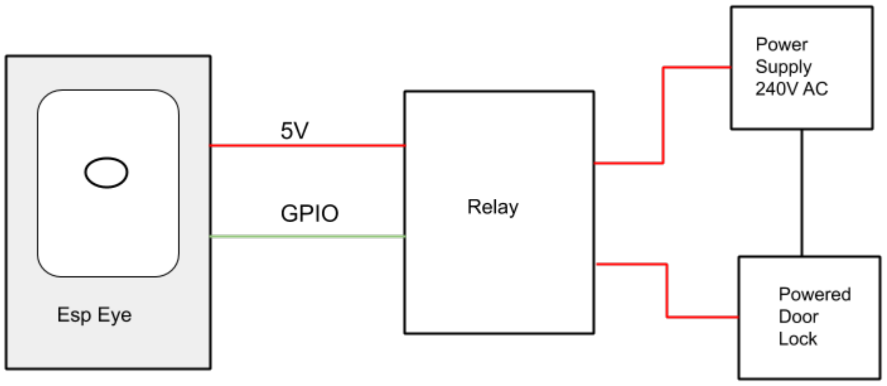


Figure 7.1: Wiring Diagram.

Signature Algorithm (ECDSA). Once this is possible in Esp Eye then the further step would be to see if HLF provides an endpoint to submit the signed transaction.

Furthermore, Esp Eye stores the registered face IDs in the flash itself. However the idea is to fetch face IDs upon device startup from a different source which could be on the cloud. However the limitation is that the face id vector is dynamically allocated in the memory. This is due to a technical limitation in esp32 boards because the maximum statically allocated memory is 160 KB. There is already some contribution in this direction and the workaround is to register the face IDs from the images being fetched from an API gateway. This is implemented however the face was not recognized in the first try.

Chapter 8

Summary and Conclusions

The main contribution of the thesis is to give a solution for the integrity and trust issues by incorporating the three domains: AI, Blockchain and IoT. With extensive research performed we could barely see the integration of the trio, however the advantages of them show promising future on how the three technologies complement each other to form a reliable architecture. In one hand we have materialized the idea of running face recognition entirely on Esp Eye. In such a memory constrained device the idea is accomplished with the Esp Who library which offers important APIs for face detection and recognition. At the beginning the implementation started with the ESP-IDF (Espressif IoT Development Framework) in order to upload firmware onto Esp Eye however Arduino IDE offered a better level of abstraction. On the other hand, after carefully considering available BCs we resulted in selecting Hyperledger Fabric for storing images of recognized faces. Although there are a number of existing Blockchain platforms but yet none of them is oriented towards the compatibility with IoT devices, this is reasoned with the fact that API endpoints are locked with Fabric SDK in order to interact with Blockchain. The work followed with the implementation of a chaincode with multiple APIs for establishing communication with Farbric network. Additionally, many different technologies and communication protocols were considered. A bottom-up approach was employed for the implementation of technologies since the state of the art technologies were considered. Due to that also different programming languages starting from C, Javascript as well Python which is was used at the beginning. In order to show feasibility of the design Hyperledger Fabric is run in LAN but eventually it can run on the cloud. Although RPI was used as a development environment Hyperledger Fabric was installed in an AMD processor supported machine due to Hyperledger Fabric not supporting the ARM architecture. The idea of performing face recognition in Esp Eye itself comes also with limitations. After many attempts trying different libraries for encryption and digital signature we could not manage to run it.

To conclude the current solution stands out of the crowd due to its novel architecture which is proved successful in achieving the goal while keeping in mind the restrictions that come from the use case and the available technologies on market.

Bibliography

[1] H. F. Atlam, M. A. Azad, A. G. Alzahrani, and G. Wills, “A review of blockchain in internet of things and ai,” *Big Data and Cognitive Computing*, vol. 4, no. 4, 2020.

[2] B. Parker and C. Bach, “The synthesis of blockchain, artificial intelligence and internet of things,” *European Journal of Engineering and Technology Research*, vol. 5, pp. 588–593, May 2020.

[3] S. K. Singh, S. Rathore, and J. Park, “Blockiotintelligence: A blockchain-enabled intelligent iot architecture with artificial intelligence,” *Future Generation Computer Systems*, vol. 110, 09 2019.

[4] E. Schiller, E. Esati, S. R. Niya, and B. Stiller, “Blockchain on msp430 with ieee 802.15.4,” in *2020 IEEE 45th Conference on Local Computer Networks (LCN)*, pp. 345–348, 2020.

[5] N. Waheed, X. He, M. Ikram, M. Usman, S. S. Hashmi, and M. Usman, “Security and privacy in iot using machine learning and blockchain: Threats and countermeasures,” *ACM Comput. Surv.*, vol. 53, Dec. 2020.

[6] Espressif, “ ESP-WHO Platform.” `https://github.com/espressif/esp-who`. [Online; accessed 19-September-2020].

[7] “Overview of the Internet of things http://handle.itu.int/11.1002/1000/11559.”

[8] H. F. Atlam, A. Alenezi, M. O. Alassafi, and G. Wills, “Blockchain with internet of things: benefits, challenges, and future directions,” *International Journal of Intelligent Systems and Applications*, vol. 10, pp. 40–48, June 2018.

[9] S. F. Abedin, M. G. R. Alam, R. Haw, and C. S. Hong, "A system model for energy efficient green-iot network," in *2015 International Conference on Information Networking (ICOIN)*, pp. 177–182, 2015.

[10] J. Tang, D. Sun, S. Liu, and J. Gaudiot, "Enabling deep learning on iot devices," *Computer*, vol. 50, no. 10, pp. 92–96, 2017.

[11] H. F. Atlam, R. J. Walters, and G. B. Wills, "Intelligence of things: Opportunities challenges," in *2018 3rd Cloudification of the Internet of Things (CIoT)*, pp. 1–6, 2018.

[12] AISOLAB, " ARTIFICIAL INTELLIGENCE, MACHINE LEARNING, NEURAL NETWORKS AND DEEP LEARNING: WHAT ARE THE DIFFERENCES?." `https://aiso-lab.com/intro-to-ai-1-artificial-intelligence-machine-learning-neural-networks-and-deep-lea` [Online; accessed 29-November-2020].

[13] D. Ackerman, " System brings deep learning to "internet of things" devices." `https://news.mit.edu/2020/iot-deep-learning-1113`. [Online; accessed 10-December-2020].

[14] T. N. Dinh and M. T. Thai, "Ai and blockchain: A disruptive integration," *Computer*, vol. 51, no. 9, pp. 48–53, 2018.

[15] Z. Shae and J. Tsai, "Ai blockchain platform for trusting news," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1610–1619, 2019.

[16] G. B. Rajendran, K. .P, and V. Rajan, *Secure IoT Systems Using Raspberry Pi Machine Learning Artificial Intelligence*, pp. 797–805. 01 2020.

[17] K. Salah, M. H. U. Rehman, N. Nizamuddin, and A. Al-Fuqaha, "Blockchain for ai: Review and open research challenges," *IEEE Access*, vol. 7, pp. 10127–10149, 2019.

[18] T. Kurian, " Transformational Technologies: Today How IoT, AI, and blockchain will revolutionize business. ." `http://www.oracle.com/us/solutions/cloud/`

tt-technologies-white-paper-4498079.pdf. [Online; accessed 10-December-2020].

[19] S. Singh, P. K. Sharma, B. Yoon, M. Shojafar, G. H. Cho, and I.-H. Ra, “Convergence of blockchain and artificial intelligence in iot network for the sustainable smart city,” *Sustainable Cities and Society*, vol. 63, p. 102364, 2020.

[20] S. A. I. Quadri and P. Sathish, “Iot based home automation and surveillance system,” in *2017 International Conference on Intelligent Computing and Control Systems (ICICCS)*, pp. 861–866, 2017.

[21] T. Amberg, R. Häfeli, and u. . h. C. Özyurt, journal=Kamera-basierter Sensor für Raumbelegung

[22] O. Source, “ ESP-WHO Overview.” https://github.com/espressif/esp-who. [Online; accessed 23-September-2020].

[23] Espressif, “ ESP-EYE Development Board.” https://www.espressif.com/en/products/devkits/esp-eye/overview. [Online; accessed 10-December-2020].

[24] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, “Joint face detection and alignment using multitask cascaded convolutional networks,” *IEEE Signal Processing Letters*, vol. 23, no. 10, pp. 1499–1503, 2016.

[25] C. Guru, “ Image Pyramid Example.” http://coding-guru.com/image-pyramid-expanding-image-2/. [Online; accessed 17-December-2020].

[26] S. K, “ Non-maximum Suppression (NMS).” https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c. [Online; accessed 18-December-2020].

[27] M. Stewart, “ Simple Introduction to Convolutional Neural Networks.” https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac. [Online; accessed 18-December-2020].

[28] C.-F. Wang, “ A Basic Introduction to Separable Convolutions.” https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728. [Online; accessed 19-December-2020].

[29] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *CoRR*, vol. abs/1704.04861, 2017.

[30] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, “Mobilenetv2 :inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation,” *CoRR*, vol. abs/1801.04381, 2018.

[31] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” 2017.

[32] S. Chen, Y. Liu, X. Gao, and Z. Han, “Mobilefacenets: Efficient cnns for accurate real-time face verification on mobile devices,” *ArXiv*, vol. abs/1804.07573, 2018.

[33] A. L. Hors, “ Hyperledger Fabric Technical Deep Dive.” https://www.slideshare.net/alehors/hyperledger-fabric-technical-deep-dive-20190618. [Online; accessed 29-December-2020].

[34] H. Documentation, “ The Ordering Service.” https://hyperledger-fabric.readthedocs.io/en/release-2.2/orderer/ordering_service.html. [Online; accessed 29-December-2020].

[35] H. Fabric, “ Hyperledger Fabric Ledger.” https://hyperledger-fabric.readthedocs.io/en/release-2.2/ledger/ledger.html. [Online; accessed 29-December-2020].

[36] I. Technologies, “ ESP EYE.” http://icorptechnologies.co.za/product/esp32-eye/. [Online; accessed 20-December-2020].

[37] Espressif, “ MTMN a lightweight Human Face Detection Model.” `https://github.com/espressif/esp-face/blob/master/face_detection/README.md`. [Online; accessed 19-August-2020].

[38] R. Robinson, “ Convolutional Neural Networks - Basics An Introduction to CNNs and Deep Learning.” `https://mlnotebook.github.io/post/CNN1/`. [Online; accessed 19-September-2020].

[39] D. Sandberg, “ Deep learning five, MTCNN face detection and alignment and FaceNet face recognition.” `https://www.programmersought.com/article/2837770080/`. [Online; accessed 20-September-2020].

[40] J. Du, “High-Precision Portrait Classification Based on MTCNN and Its Application on Similarity Judgement,” in *Journal of Physics Conference Series*, vol. 1518 of *Journal of Physics Conference Series*, p. 012066, Apr. 2020.

[41] H. Ku and W. Dong, “Face recognition based on mtcnn and convolutional neural network,” *Frontiers in Signal Processing*, vol. 4, 01 2020.

[42] M. C. T. Manullang, A. L. Ramdani, and N. Migotuwio, “Design and architecture of a public satisfaction detection camera based on facial emotional analysis,” 2020.

[43] Y. Xie, H. Wang, and S. Guo, “Research on mtcnn face recognition system in low computing power scenarios,” *Journal of Internet Technology*, vol. 21, pp. 1463–1475, 2020.

[44] Y. Sun, L. Xie, Z. Wang, and Y. An, “An embedded system of face recognition based on arm and hmm,” in *Entertainment Computing – ICEC 2007* (L. Ma, M. Rauterberg, and R. Nakatsu, eds.), (Berlin, Heidelberg), pp. 389–394, Springer Berlin Heidelberg, 2007.

[45] N. Ismail, M. Idayu, and M. I. Md Sabri, “Review of existing algorithms for face detection and recognition,” 12 2009.

[46] A. H. M. Amin, N. M. Ahmad, and A. M. M. Ali, “Decentralized face recognition scheme for distributed video surveillance in iot-cloud infrastructure,” in *2016 IEEE Region 10 Symposium (TENSYMP)*, pp. 119–124, 2016.

[47] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram, "Blockchain for iot security and privacy: The case study of a smart home," in *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pp. 618–623, 2017.

[48] M. Hollemans, " New mobile neural network architectures ." `https://machinethink.net/blog/mobile-architectures/`. [Online; accessed 23-November-2020].

[49] M. Pradhan, " Image Classification using MobileNet in the browser ." `https://medium.com/analytics-vidhya/image-classification-using-mobilenet-in-the-browser-b69f2f57abf`. [Online; accessed 23-November-2020].

[50] M. Valenta and P. Sandner, "Comparison of ethereum, hyperledger fabric and corda," 2017.

[51] C. Gorenflo, S. Lee, L. Golab, and S. Keshav, "Fastfabric: Scaling hyperledger fabric to 20,000 transactions per second," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pp. 455–463, 2019.

[52] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, and J. Yellick, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, (New York, NY, USA), Association for Computing Machinery, 2018.

[53] C. Cachin, "Architecture of the hyperledger blockchain fabric," 2016.

[54] S. V, " Hyperledger Fabric — Part 1 — Components and Architecture." `https://blog.clairvoyantsoft.com/hyperledger-fabric-components-and-architecture-b874b36c4af5`. [Online; accessed 23-December-2020].

[55] Y. Jeong, D. Hwang, and K. Kim, “Blockchain-based management of video surveillance systems,” in *2019 International Conference on Information Networking (ICOIN)*, pp. 465–468, 2019.

[56] N. Kshetri, “Can blockchain strengthen the internet of things?,” *IT Professional*, vol. 19, no. 4, pp. 68–72, 2017.

Abbreviations

IOT	Internet of Things
AI	Artificial Intelligence
BC	Blockchain
LAN	Local Area Network
API	Application Programming Interface
CNN	Convolutional Neural Network
MTCNN	Multitask Cascaded Convolutional Neural Networks
CA	Certificate Authority
PSRAM	Pseudostatic Random-access memory
FRMN	Human Face Recognition Model
ARM	Advanced RISC Machines
ARM	Advanced Micro Devices
MSP	Membership Service Provider
NMS	Non-Maximum Suppression
HLF	Hyper Ledger Fabric
CFT	Crash Fault Tolerant
BFT	Byzantine Fault Tolerant
SBFT	Simplified Byzantine Fault Tolerance

Glossary

MTMN is a lightweight Human Face Detection Model, which is built around a new mobile architecture called MobileNetV2 and Multi-task Cascaded Convolutional Networks, and is specially designed for embedded devices.

Chaincode is a program which can be invoked to update or query the ledger.

Convolutional Neural Network is a type of neural networks which applies convolution or filters to an image to find the presence of detected features.

List of Figures

List of Tables

Listings

Appendix A

Installation Guidelines

A.0.1 Mandatory Steps for preparing the environment

In order to successfully replicate the implementation in your environment some prior information is needed. First of all, we strongly encourage to install HLF in an AMD processor based computer since HLF does not support ARM architectures. However uploading code to Esp Eye it does not matter from which OS you are using. Throughout this guideline it is assumed that Git is installed.

A.0.2 Arduino Firmware for Esp Eye

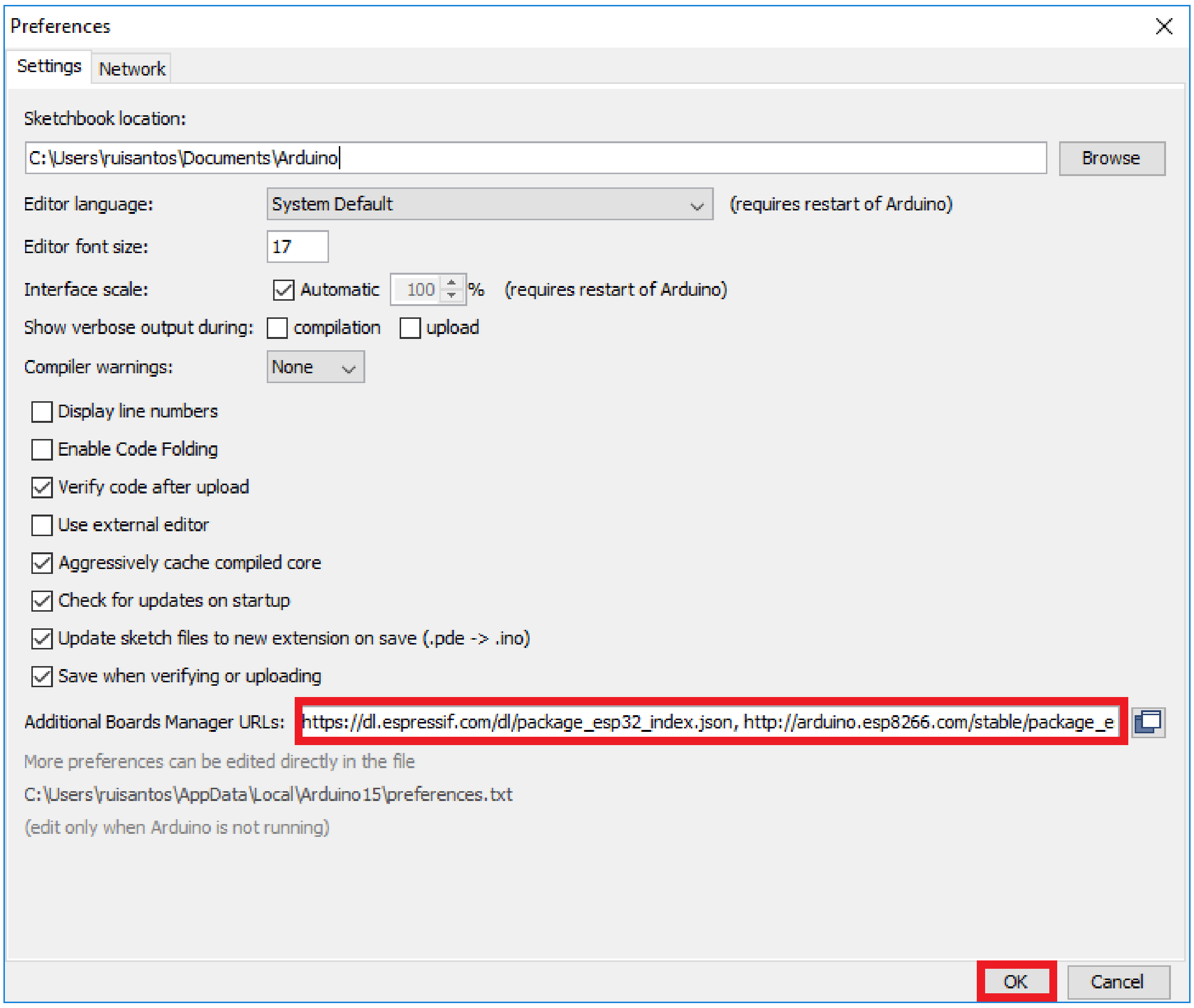
All the files for the Esp Eye, IoT Gateway and HLF reside in the shared git repository. To start with use the command below in you desired directory where you want to start replicating this project.

```
git clone https://github.com/eesati/Master-Thesis.git
```

After having done the above there are a number of directories, the ones which we are interested now is the `Arduino` folder and `Arduino15` folder. In order to upload firmware Arduino Ide must be installed. In order to do so follow this link for installation depending on your environment `https://www.arduino.cc/en/Guide`

In order to program Esp Eye with Arduino IDE the ESP32 board needs to be installed by following these steps:

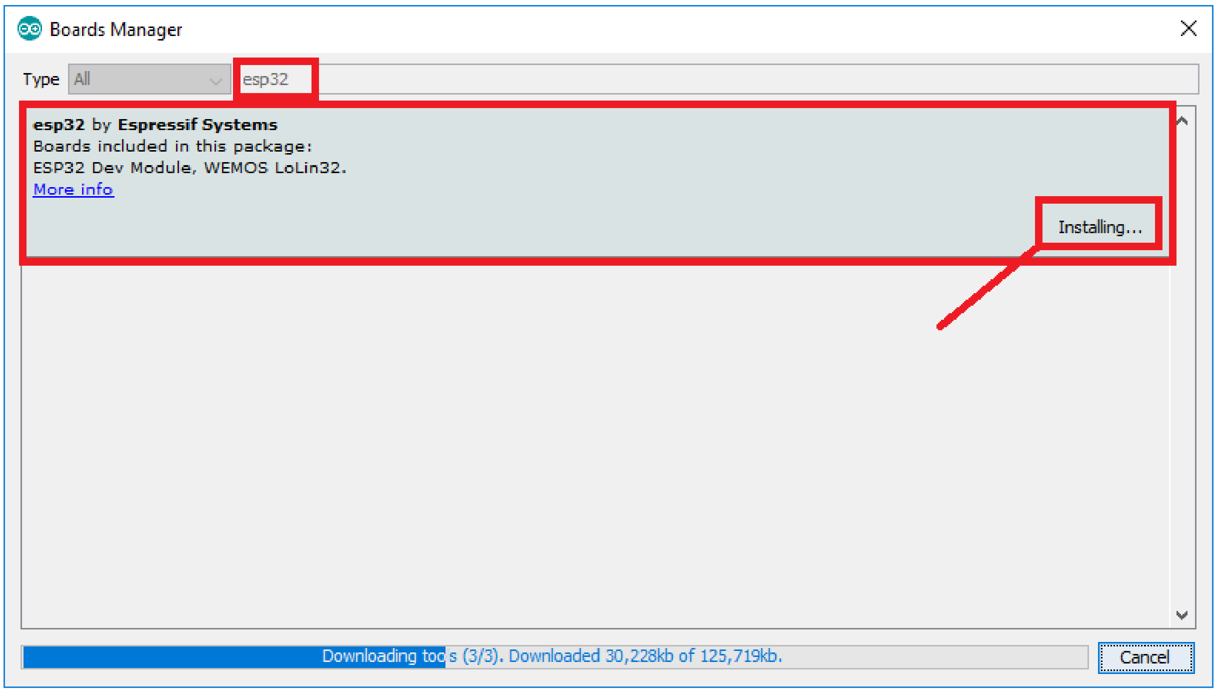
1. In Arduino, go to File> Preferences
2. Paste the following link https://dl.espressif.com/dl/package_esp32_index.json under Additional Boards Manager and click OK.



Sketchbook location:
C:\Users\ruisantos\Documents\Arduino
Editor language: System Default (requires restart of Arduino)
Editor font size: 17
Interface scale: ☑ Automatic 100 % (requires restart of Arduino)
Show verbose output during: ☐ compilation ☐ upload
Compiler warnings: None
☐ Display line numbers
☐ Enable Code Folding
☑ Verify code after upload
☐ Use external editor
☑ Aggressively cache compiled core
☑ Check for updates on startup
☑ Update sketch files to new extension on save (.pde -> .ino)
☑ Save when verifying or uploading
Additional Boards Manager URLs: https://dl.espressif.com/dl/package_esp32_index.json, http://arduino.esp8266.com/stable/package_e
More preferences can be edited directly in the file
C:\Users\ruisantos\AppData\Local\Arduino15\preferences.txt
(edit only when Arduino is not running)

Figure A.1: Installing ESP32 board.

3. After that Go to Tools > Board > Boards Manager... and search for esp32 and install the latest version of it. See Figure A.2.
4. It should be done. Now you can go to Tools > Board and select AI THINKER.



esp32 by Espressif Systems
Boards included in this package:
ESP32 Dev Module, WEMOS LoLin32.

Downloading tools (3/3). Downloaded 30,228kb of 125,719kb.

Figure A.2: Installing ESP32 library.

After that a number of libraries have to be installed in Arduino IDE: Arduino_JSON, Wifi, Wifi101, ESP32_Mail_Client and NTPClient. By going to Tools > Manage Libraries you can install each one by one.

The Arduino folder contains the firmware for Esp Eye and additional libraries. There are multiple Arduino files which show the step by step implementation. The last file to be uploaded in Arduino is:

```
Arduino/Final_for_submission/Final_for_submission.ino
```

Registering Face IDs

However in order to register face ids there is another sketch that needs to be uploaded to do that. Since the face ids are stored in flash then some changes need to be done in installation folder where Arduino IDE is installed in your computer. Before uploading the code for registration of faces there is a need to partition the flash to save little space for face Ids. Now you need to go to the partition folder which is typically located under this directory:

For windows downloaded Arduino Ide from Arduino Website:

```
C > Users > *your-user-name* > AppData > Local > Arduino15 > packages > esp32
> hardware > esp32 > 1.0.4 > tools > partitions
```

For Mac or Linux:

```
Users > *your-user-name* > Library > Arduino15 > packages > esp32 > hardware >
esp32 > 1.0.4 > tools > partitions
```

Now copy the file faceid_partitions.csv found in the root directory of the repository and paste in the directory above.

Now to let the board know about the partitioning there is a need to update the file board.txt located under the folder 1.0.4 which is in the two directories back from the partition folder and add the following lines:

```
esp32wrover.menu.PartitionScheme.faceid_partition=Face Recognition (2621440 bytes
with OTA)
esp32wrover.menu.PartitionScheme.faceid_partition.build.partitions=faceid_partitions
esp32wrover.menu.PartitionScheme.faceid_partition.upload.maximum_size = 2621440
```

Now use the Enroll_faces.ino sketch to enroll face ids. After enrolling faces then upload the Final_for_submission.ino sketch. Finally, since the Esp Eye firmware acts as a station there is a need to update the SSID and password to connect to.

A.0.3 Hyperledger Fabric installation

The Fabric network with implemented chaincode is located under HLF directory. At the time when we installed it HLF was the latest version but now it is no longer anymore and it is known as v2.1.

We strongly encourage to have a look at the official HLB site for installation guidelines located at :

```
https://hyperledger-fabric.readthedocs.io/en/release-2.1/getting_started.html
```

Prerequisites

Before beginning the installation, there a number of prerequisites that need to be followed. Which can be seen also here in the link below:

```
https://hyperledger-fabric.readthedocs.io/en/release-2.1/prereqs.html
```

1. The latest version of git for your OS
2. The latest version of cURL
3. The latest version of Docker and Docker Compose

Installing Hyperledger Fabric binaries and docker images

First method

Now it is time to install Hyperledger Fabric binaries and docker images. For now there are two ways to do that, one way is to install a clean copy of HLP. Which can be seen in more detail in this link :

```
https://hyperledger-fabric.readthedocs.io/en/release-2.1/install.html
```

The github repository for HLF need to be cloned in a local directory, for macOS it has to use a location under /Users:

```
git clone https://github.com/hyperledger/fabric-samples.git
```

Now it is the time to install binaries and docker images. To do so there should be a fabric-sample directory. In terminal under that directory execute the following command:

```
curl -sSL https://bit.ly/2ysbOFE | bash -s – <fabric_version> <fabric-ca_version>
<thirdparty_version>
```

Which in our case would translate to v2.1 as below:

```
curl -sSL https://bit.ly/2ysbOFE | bash -s – 2.1.1 1.4.7 0.4.20
```

To pick up the docker containers without going to the path for each binary, it is recommended to do this:

```
export PATH=<path to the cloned HLF location>/bin:PATH
```

After that copy the imagestore folder under HLF and paste at the cloned copy of the Hyperledger Fabric. Under chaincode folder of HLF we have the chaincode with a folder named imagestore copy that and paste under chaincode of the newly cloned Hyperledger Fabric under chaincode. It should by now be ready to run the project.

Second method

The second way is to use the HLF directory of ours but a number of changes need to be done. First the bin folder has to be deleted because it contains the previous installed docker images.

After that the binaries and docker images need to be installed locally. Similarly as above under the HLF directory the binaries need to be installed using the following command in Terminal:

```
curl -sSL https://bit.ly/2ysbOFE | bash -s – 2.1.1 1.4.7 0.4.20
```

To pick up the docker containers without going to the path for each binary, it is recommended to do this:

```
export PATH=<path to the cloned HLF location>/bin:PATH
```

It is now expected that Hyperledger Fabric is installed and ready to be used. The two methods apply to all OS however the prerequisites may have to be installed using different package manager. For any issues it is recommended to follow the official HLF documentation links as above.

Hyperledger Fabric application SDKs

Hyperledger Fabric provides a number of SDKs to support developing applications. Since we are using nodejs for the server, similarly nodejs SDK for fabric client is used. The prerequisites for nodejs SDK is that the following must be installed first:

- Node.js, version 10 is supported from 10.15.3 and higher
- Node.js, version 12 is supported from 12.13.1 and higher
- npm tool version 6 or higher

To install the nodejs SDK:

```
npm install fabric-network
```

Hyperledger Fabric also offers a contract API for developing smart contacts. For nodejs use following command to install it:

```
npm install –save fabric-contract-api
```

The main folders where our implementation lies in the source code of our project is imagestore folder. We strongly recommend to do :

```
npm install
```

under the folder /imagestore/javascript where the nodejs server and HLF client are and under /imagestore/frontend which is the react web application.

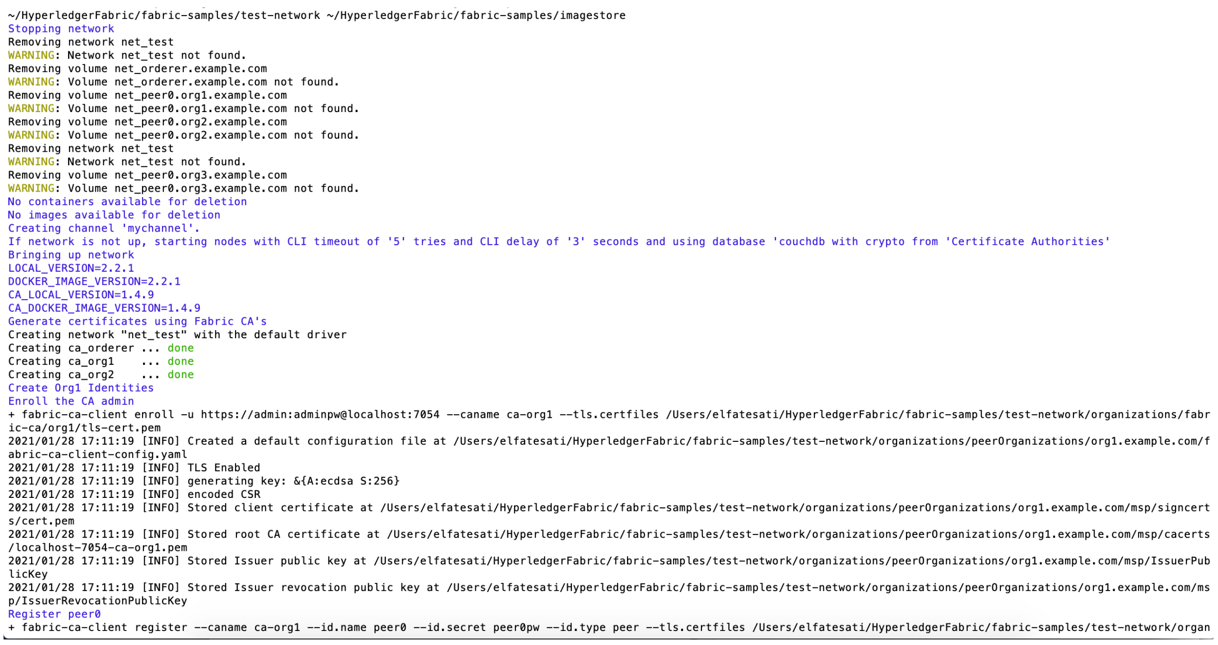
Running Hyperledger Fabric

To start the Fabric network, docker must be running. To start with in Terminal go to imagestore and do:

```
./startNetwork.sh javascript
```

And it should look like the Figure Figure A.3.

This script will start the network and docker containers, besides it will simultaneously deploy the smart contract specified in the chaincode. If you want to know more see the startNetwork.sh file. Basically it creates the network with 2 organisations and 2 ordering peers. The chaincode also gets deployed which is located under /chaincode/imagestore/-javascript/.



```
~/HyperledgerFabric/fabric-samples/test-network ~/HyperledgerFabric/fabric-samples/imagestore
Stopping network
Removing network net_test
WARNING: Network net_test not found.
Removing volume net_orderer.example.com
WARNING: Volume net_orderer.example.com not found.
Removing volume net_peer0.org1.example.com
WARNING: Volume net_peer0.org1.example.com not found.
Removing volume net_peer0.org2.example.com
WARNING: Volume net_peer0.org2.example.com not found.
Removing network net_test
WARNING: Network net_test not found.
Removing volume net_peer0.org3.example.com
WARNING: Volume net_peer0.org3.example.com not found.
No containers available for deletion
No images available for deletion
Creating channel 'mychannel'.
If network is not up, starting nodes with CLI timeout of '5' tries and CLI delay of '3' seconds and using database 'couchdb with crypto from 'Certificate Authorities'
Bringing up network
LOCAL_VERSION=2.2.1
DOCKER_IMAGE_VERSION=2.2.1
CA_LOCAL_VERSION=1.4.9
CA_DOCKER_IMAGE_VERSION=1.4.9
Generate certificates using Fabric CA's
Creating network "net_test" with the default driver
Creating ca_orderer ... done
Creating ca_org1    ... done
Creating ca_org2    ... done
Create Org1 Identities
Enroll the CA admin
+ fabric-ca-client enroll -u https://admin:adminpw@localhost:7054 --caname ca-org1 --tls.certfiles /Users/elfatesati/HyperledgerFabric/fabric-samples/test-network/organizations/fabr
ic-ca/org1/tls-cert.pem
2021/01/28 17:11:19 [INFO] Created a default configuration file at /Users/elfatesati/HyperledgerFabric/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/f
abric-ca-client-config.yaml
2021/01/28 17:11:19 [INFO] TLS Enabled
2021/01/28 17:11:19 [INFO] generating key: &{A:ecdsa S:256}
2021/01/28 17:11:19 [INFO] encoded CSR
2021/01/28 17:11:19 [INFO] Stored client certificate at /Users/elfatesati/HyperledgerFabric/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/msp/signcert
s/cert.pem
2021/01/28 17:11:19 [INFO] Stored root CA certificate at /Users/elfatesati/HyperledgerFabric/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/msp/cacerts
/localhost-7054-ca-org1.pem
2021/01/28 17:11:19 [INFO] Stored Issuer public key at /Users/elfatesati/HyperledgerFabric/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/msp/IssuerPub
licKey
2021/01/28 17:11:19 [INFO] Stored Issuer revocation public key at /Users/elfatesati/HyperledgerFabric/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/ms
p/IssuerRevocationPublicKey
Register peer0
+ fabric-ca-client register --caname ca-org1 --id.name peer0 --id.secret peer0pw --id.type peer --tls.certfiles /Users/elfatesati/HyperledgerFabric/fabric-samples/test-network/organ
```

Figure A.3

The last step wow to create an Administrator and a user move to /imagestore/javascript/ and run:

```
node enrollAdmin.js
```

```
node registerUser.js
```

A.0.4 Running the Web Application

The web application is located under /imagestore/frontend, cd into this folder and do :

```
npm install
```

```
npm start
```

A.0.5 Fabric Explorer

Fabric Explorer is optional. To install it you can use the following link and it offers a number of options:

```
https://github.com/hyperledger/blockchain-explorer
```

We recommend installing it using docker. The Fabric Explorer container comes automatically when binaries are installed . First of all it assumes the Fabric network is already running. Under the root directory of the source of this project there is a folder named **Fabricexplorer** which holds the necessary files to make Explorer running. The file that should be updated is the **connection-profile/test-network.json**. In this file the **adminPrivateKey** has to be updated with the newly generated private key with location in Hyperledger Fabric:

```
/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp/keystore
```

Appendix B

Contents of the CD

The CD contains the following:

- a folder with the Arduino source code
- a folder with the Arduino Libraries
- a folder for Hyperledger Fabric
- a folder with Web Application
- a folder with R Code
- a folder with Fabric Explorer
- a folder with the thesis source code and the final thesis pdf

All the above mentioned folders are available in a Zipped file.