PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

FACULTY OF ENGINEERING

Dissertation on

"Securing IoT Based Surveillance System using Blockchain"

Submitted in partial fulfilment of the requirements for the award of degree of

Bachelor of Technology
in
Computer Science & Engineering

UE18CS390B – Capstone Project Phase - 2

Submitted by:

Shashank L	**PES1201800180**
Vallabhaneni Dhanush	**PES1201802041**
Vybhav Bhadri S	**PES1201801764**
Siri L Y	**PES1201801928**

Under the guidance of

Asst. Prof. Revathi G P
Assistant Professor
PES University

June - Dec 2021

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

FACULTY OF ENGINEERING

CERTIFICATE

This is to certify that the dissertation entitled

Securing IoT Based Surveillance System using Blockchain

is a bonafide work carried out by

Shashank L	**PES1201800180**
Vallabhaneni Dhanush	**PES1201802041**
Vybhav Bhadri S	**PES1201801764**
Siri L Y	**PES1201801928**

in partial fulfilment for the completion of seventh semester Capstone Project Phase - 2 (UE18CS390B) in the Program of Study - Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period June. 2021 – Dec. 2021. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The dissertation has been approved as it satisfies the 7th semester academic requirements in respect of project work.

Signature Asst. Prof. Revathi G P Designation	Signature Dr. Shylaja S S Chairperson	Signature Dr. B K Keshavan Dean of Faculty

External Viva

Name of the Examiners	**Signature with Date**
1. _____	_____
2. _____	_____

DECLARATION

We hereby declare that the Capstone Project Phase - 2 entitled **Securing IoT Based Surveillance System using Blockchain** has been carried out by us under the guidance of Asst. Prof. Revathi G P, Assistant Professor and submitted in partial fulfilment of the course requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering** of **PES University, Bengaluru** during the academic semester June – December 2021. The matter embodied in this report has not been submitted to any other university or institution for the award of any degree.

Shashank L
PES1201800180

Siri L Y
PES1201801928

Vallabhaneni Dhanush
PES1201802041

Vybhav Bhadri S
PES1201801764

ACKNOWLEDGEMENT

I would like to express my gratitude to Asst. Prof. Revathi G P, Department of Computer Science and Engineering, PES University, for her continuous guidance, assistance, and encouragement throughout the development of this UE18CS390B - Capstone Project Phase – 2.

I am grateful to the project coordinators, Prof. Sunitha R and Prof. Silviya Nancy J, for organizing, managing, and helping with the entire process.

I take this opportunity to thank Dr. Shylaja S S, Chairperson, Department of Computer Science and Engineering, PES University, for all the knowledge and support I have received from the department. I would like to thank Dr. B.K. Keshavan, Dean of Faculty, PES University for his help.

I am deeply grateful to Dr. M. R. Doreswamy, Chancellor, PES University, Prof. Jawahar Doreswamy, Pro Chancellor – PES University, Dr. Suryaprasad J, Vice-Chancellor, PES University for providing to me various opportunities and enlightenment every step of the way. Finally, this project could not have been completed without the continual support and encouragement I have received from my family and friends.

ABSTRACT

Surveillance means monitoring. Surveillance has been around for generations, evolving and changing forms but retaining the essence of information gathering. Security surveillance is a powerful means to gather evidence and discourage illicit activities. The digital era we live in is characterised by the heavy use of technology to perform tasks that were earlier done by humans. One such task is surveillance, an observation tactic using human eyes. Today, the Internet of Things allows embedded devices to perform surveillance.

Internet Protocol based cameras are IoT devices used to collect and transmit video footage remotely. The integrity of the stored video from private IP cameras cannot be ascertained due to the possibility of tampering by individuals who control access to the IP camera functionality. To ensure data integrity and implement access management, blockchain technology has been explored in this project. Blockchain technology allows surveillance events to be recorded as transactions on an immutable ledger.

In this project, we propose a surveillance system that makes use of machine learning techniques and blockchain technology to secure IP cameras. The IP camera is constructed using the Raspberry Pi family of devices. The Machine learning algorithms implement facial detection and facial recognition. The blockchain framework is Hyperledger Fabric, which is suitable for private IP cameras due to its permissioned nature. The entire surveillance system is divided into three major subsystems namely - IoT network, Blockchain network, and Client UI.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The study begins with an in-depth review of existing literature on existing developments in the IoT blockchain space followed by discussion on machine learning algorithms, which creates the foundation for the proposed surveillance system. The study proceeds to describe the proposed system from a conceptual point of view with emphasis on the practicality of the implementation. The requirements of the proposed system are laid out in detail from where the paper proceeds to explain the proposed methodology. Following this is the actual implementation and working demo of the proposed system. Results achieved via the proposed system are established and discussed. The paper concludes with a summary of work carried out in Phase-2 of the project and touches upon potential future considerations.

CHAPTER 2

PROBLEM STATEMENT

IP cameras record sensitive data during surveillance. The majority of IoT networks operate on centralized cloud solutions, both for storage and processing purposes. This architecture is subjected to challenges in data security, transparency, and data integrity.

This project is an attempt at utilising Hyperledger Fabric which provides a tamper-resistant database for secure and private access. In this paper, we show that the convergence of IoT, Machine Learning and Blockchain will solve the challenges of the traditional system. We designed and implemented an IoT based surveillance system incorporating the three domains. The proposed system is relevant for authenticating people who attempt to enter a building.

CHAPTER 3

LITERATURE SURVEY

[1]

3.1 A Video Surveillance System Based on Permissioned Blockchains and Edge Computing

This study provides an overview of the challenges associated with video surveillance systems and proposes a permissioned blockchain based architecture to ensure a reliable and robust surveillance system. In edge computing, processing of information is done at the source of data.

Notable characteristics of this paper are as follows:

- Addresses bandwidth limitations, real-time monitoring issues, vulnerability of devices to attacks, and requirement of sophisticated hardware for edge computation at the source.
- Compares centralized architectures with decentralized architectures and highlights the limitations and advantages of each system.
- Proposes a layered architecture consisting of Physical Layer, Data Service Layer and Application Layer.
- Physical Layer: Comprises IoT devices along with edge gateway which forms a local network. The edge gateway functions as a processing unit which collects data from edge devices, processes the collected data and pushes the data to Application Layer.
- Data Service Layer: Consists of permissioned blockchain and edge nodes. This layer provides computing power to standalone edge devices that cannot perform computation at the source end. The blockchain stores a hash of transactions, the transactions being the processing of video data that is carried out in the physical layer using Convolutional Neural Networks (CNN).

- Application Layer: Comprises InterPlanetary File System (IPFS) and monitoring center, which performs the task of storing and querying files. The file transferred from the physical layer is encrypted and a hash of the encrypted file is stored in a globally distributed hash table. This hash value is used for querying files.
- Use of CBFT consensus algorithm which ensures fault-tolerance.
- Privacy is ensured due to localisation of data processing at the source end and use of IPFS which stores encrypted data.

Future works that have been discussed in this paper are the use of smart contracts to implement access controls, use of upgraded video analysis algorithms, and large-scale testing of the system to measure latency and fault tolerance.

[2].

3.2 Public Security Surveillance System Using Blockchain Technology and Advanced Image Processing Techniques

This study provides an overview of a decentralized surveillance system proposed for use by government, public, and private operated entities. HyperLedger Fabric is the blockchain framework chosen for the particular use case because of the need for access control management and privacy among the entities involved. Performance of the system is tested against three main parameters namely - scalability, latency, and security of design.

Notable characteristics of this paper are as follows:

- Proposes an architecture with two main subsystems - surveillance subsystem and permissioned blockchain subsystem.
- The surveillance subsystem comprises Smart Camera Modules which is a combination of cameras and GPS modules to detect camera movement. Facial Detection is run by the Smart Camera Modules which sends the extracted features to a central processing unit called MPU. MPU is responsible for implementing facial recognition on obtained features and serves as a member of the blockchain.

- Each entity that is part of the Fabric can propose transactions based on predefined access controls.
- The smart contract contains logic for four transactions namely - Find, Issue, UpdateSuspect, and QuerySuspects.
- The experimental setup is a local network consisting of three nodes, each running on a PC with 8 to 16 GB RAM and 2-4 cores. The Smart Camera Modules are directly connected to the nodes.
- Transaction complexity and latency vary with the type of transaction. The average end-to-end latency of the system is about 12 seconds. This latency is calculated as the time from when a suspect appears in front of the Smart Camera Module to when an "Issue" transaction is executed by the MPU.

Future works that have been discussed in this paper are recognition of a suspect's apparel and behavior. Possible testing of security of the system against Dos attacks and malicious nodes has been mentioned.

[3].

3.3 Integration of IoT and Blockchain to in the Processes of a University Campus

This paper focuses on implementing IoT with blockchain in a university campus. This Paper explores two main concepts. The first is identifying weaknesses in the IoT system and the subsequent steps needed to integrate Blockchain for overcoming the weaknesses. The second is to integrate the entire campus architecture which interacts with blockchain to its full capacity.
Data validation process is one of the main processes in IoT-Blockchain architecture:
Flow of processes concentrates on order entry of IoT devices. Registration requests lead to creation of blocks in the blockchain network. Blocks are necessary to validate data and the

validation is performed by means of the unique hash value of a block. Main feature of blockchain is to send the processed record to different nodes that will validate the necessary information. If the state of stored data is changed, nodes are in charge of validating the information. After validation records are stored in a database and processed by Big Data, data analytics is used to improve performance of the system.

Role of IoT in university campus:

This paper discusses how IoT helps to utilise campus infrastructure resources both at computing level and network level. This paper specifically mentions how traditional university campuses use centralised models to manage IoT services. IoT services allow personalization for each campus member. Centralised system architecture has serious drawbacks which are mentioned below:

Vulnerability	Possible attack	Level of insecurity
Weak credentials	Brute force attack	Easily crackable, High Risk
Exposed ports and unencrypted network services	Attacks from outside on network services by using vulnerabilities	High Risk
Resources open in the network having weak privacy measures	Users connected to an open network are vulnerable to attacks. Intruders can take control of the user's private information.	High Risk
Many devices are provided with default configurations.	Default credentials are easily cracked and unsafe	High Risk

Table 3.1: Different vulnerabilities in IoT systems and the possible attack vectors

Blockchain as a solution to enhance security:

Blockchain architecture can be integrated with the IoT system to authenticate, standardize and protect data. Security is enhanced because of the inbuilt security features of blockchain which is capable of managing the information gathered by sensors without allowing for any duplication and maintains data integrity. Blockchain does not need any third party which can be used by IoT sensors to transfer data. Device autonomy, data integrity, virtual identity and point-to-point communication (P2P) are additional functionalities which help to get rid of technical vulnerabilities.

This Paper proposes an architecture which consists of five layers. Each layer has its own functionality. Integration of the blockchain layer includes several components that guarantee security in IoT processes. Each component of the Blockchain layer - Data Acquisition, Network, Consensus, Incentives, Services is responsible for managing specific tasks.

Integrating IoT with Blockchain invokes the ability to maintain the integrity of data. Each transaction is verified and validated by several nodes, which creates decentralisation and makes tampering virtually impossible.

[4].

3.4 Design and Implementation of an Integrated IoT Blockchain Platform for Sensing Data Integrity

Theme in this paper is the integration of Blockchain and control of IoT. Paper highlights the proposed IoT blockchain platform and its use-case implementation and deployment of the proposed system.

Design architecture of the proposed IoT blockchain platform comprises a peer-to-peer blockchain network consisting of IoT devices, users, data storage, and local bridges that interact with each other. IoT server acts as a service provider that interacts with local bridges and

blockchain networks, to provide services to end users. Data storage is an inbuilt feature in blockchain which provides data integrity. Data collected by sensors and other information are stored in the blockchain network. Users can read and write but cannot alter transactions in the network. Architecture proposes two approaches for communication with IoT devices - one via local bridges and other via direct wireless communication.

Proposed IoT Blockchain platform is represented in layer-based architecture comprising Application layer, IoT Blockchain Service Layer, Connectivity Layer, and IoT Physical Layer.

Highlights of layer-based architecture :

- Modular architecture is used, allowing developers to add modules and build different subsystems without altering the other.
- The IoT physical layer consists of devices capable of computation, communication and storage.
- IoT blockchain service has security features which can be used to identify users, maintain the state of the system using consensus mechanisms, and communicate via peer-to-peer protocols.
- API is used to access the functionalities of the blockchain network to integrate with the client application.
- Application layer is used for visualisation and to manage devices.

The Blockchain platform's interaction model is designed to have a user service framework which provides distributed ledger and smart contract as a service to the user application. Interaction model consists of an application client, IoT server, IoT Network, and Blockchain Network. Transactions submitted by clients are processed and written into the ledger. New device is registered by submitting a transaction request. Data stored in the ledger is processed by the smart contract. Smart contracts execute the necessary rules and conditions. If the value does not comply with the defined values, notification is generated and sent to the user as a feedback.

This paper summarises the development environment for the proposed platform. For Implementing IoT Blockchain network, Linux operating system is used consisting of Hyperledger fabric v1.2. For IDE composer-playground, CLI tools such as composer-cli, and composer-rest-server are used. Database is implemented using CouchDB and programming

language used is Node.js. IoT device server is developed using a Raspberry Pi3 Model B. User Interface for the blockchain web app is implemented using Californium CoAP, Notify.js, and programming languages - Java, HTML, CSS, and JavaScript.

This Paper highlights identity and interoperability of IoT devices and how blockchain technology is used to provide security through a decentralized approach which adds scalability, data security and integrity using a permissioned Hyperledger framework. Furthermore the work showcases performance and analysis of the proposed system by recording execution times of device registration, sensor reading, and data querying.

Main highlight of the paper is to use interoperability and scaling of IoT along with maintaining data integrity.

[5].

3.5 Blockchain-Based Secured Access Control in an IoT System

This paper primarily deals with fixing privacy and security flaws in IoT networks. It proposes a novel blockchain based architecture for securing the network. The blockchain helps protect IoT systems and also meets the security requirements of a network of computationally weak IoT devices.

The purpose of such an architecture is to make sure that the entire IoT system is completely secured including all communications between the devices of the network, cloud computing and the fog nodes. The proposed solution uses a mobile agent software. This software helps in decreasing traffic overhead and greatly increases mobility and excellence of the solution. It can be applied to various IoT applications.

By using a lightweight consensus mechanism and by using mobile agent software, the proposed architecture greatly reduces the traffic overheads that occur in the network. Mandatory Access Control (MAC) is introduced which is a concept based on hierarchical security levels. This works in line with the hierarchical private blockchain and helps set up a guaranteed Multilevel Security (MLS) policy.

The blockchain system contains four blockchain managers. Local Blockchain manager or LBCM includes all IoT devices. The FBCM or Fog Blockchain manager contains all the edge nodes. The CFBCM contains all the core edge nodes and the CBCM contains the cloud storage.

This framework meets the CIA security triads' requirement.

Any communication that occurs in the network is in the form of transactions. These transactions are created by BCMs to access data, add, remove or monitor the IoT devices within that network.

A MAC policy is established to give tighter security. This policy can only be edited by BCMs. A user's security clearance defines his MAC. Access is given to individuals on the basis of clearance level.

The signature verification agent makes sure that the integrity of a message is maintained and ensures that data is sent from a verified individual and was not changed during commute.

The authentication agent is responsible for making sure that a user is valid and is authentic based on a secret key being shared between the devices.

The Access control Policy is enforced by the Authorisation Agent. Granting permissions rights to a user is also done by this agent. It does so by verifying each user with their digital ID.

The confidentiality of data is maintained by using the encryption and decryption agent. This agent also helps guarantee that only users and agents can use the data being transferred in the system.

This framework relies on private, level based blockchain architecture and consists of blockchain managers (miners) to handle all the transactions and to provide protection for the IoT network. The architecture provides different levels of blockchain managers to manage the IoT's entire system. Smart homes are one of the best use cases for such an architecture.

[6].

3.6 Blockchain-based Management of Video Surveillance Systems

In the proposed architecture, the video received from the camera is encrypted and stored in an IPFS node connected to the blockchain network. The decryption key of the video is not recorded in the block, but it is stored in the private DB of the blockchain node which has the special authority and can be processed only by the chaincode of the blockchain. To export and decode video, anyone who wants to view the video must authenticate the export for the video in the blockchain network, then generate a license in the chaincode API. The proposed method is a method of safely storing and exporting video in a video surveillance system. The internal administrator does not know the decryption key of the video and therefore can not read or export the video without permission.

Fabric can enhance privacy and confidentiality by introducing the concept of a channel between participating organizations in the block chain. Even if they exist in the same blockchain network, the participants of the same channel share data. However, creating some of these separate channels for some data will incur additional overhead.

To solve the problem, Hyperledger fabric v1.2 provides a function called PDC(Private Data Collection) that can collect, commit and query private data without creating individual channels. PDC consists of a hash of private data and private data. Private data can be checked only by authorized peers who are authorized to view data in the same channel. The peer receiving the block checks the access rights to the private data. After that, if there is a privilege, private data is fetched through peer-to-peer protocol from another peer, hash of private data is validated, block is committed, and written to the private ledger.

The proposed system restricts unauthorized leakage and viewing from the internal administrator. In the proposed architecture, video is securely encrypted and stored, and licenses are applied and exported safely when exporting video.

[7].
3.7 Face Recognition in CCTV Systems

This paper proposes a face recognition technique for increasing the efficiency of a criminal identification process. The steps involved are face detection, pre-processing, feature extraction, and face recognition. The recognised person's details will be shown as the output. AdaBoost algorithm, hansdorf distance is an additional factor that improved ICA (Independent Component Analysis) for facial recognition. Efficient results can be achieved by using AdaBoost and improved ICA.

The proposed methodology works as follows:

- Face detection: Computer vision toolbox is used for detecting faces. Output of this phase is a cropped face image.
- Pre-processing: Noise is removed from the cropped image obtained from the previous step. Image enhancement is done using the Wiener filter. Output of this phase is an enhanced image.
- Feature extraction: Facial features such as position of eyes, nose, mouth, facial scars, and brightness of skin are extracted from the images. Output of this phase is the set of facial features obtained from the enhanced input images.
- Face recognition: If the face is recognised, the image will be shown in a box and if the recognition fails, no image is shown.

One issue with the face recognition process is the changing behavior of the images.
The output quality depends on the quality of the camera input. Quality here means the depth, resolution of the picture, etc. Hence, the quality of the captured image from live video must be enhanced to get better quality results.

[8].

3.8 Face Recognition based Home Security System Using IoT

Surveillance systems are installed in homes and banks for security purposes. This paper is based on providing adequate security to home occupants using IoT and face recognition. A person in the video feed captured by a camera will be matched against authorised individuals. Once the face matches, the door unlocks. In the event that the face does not match with any authorised face, the door remains closed.

The proposed methodology consists of a Raspberry Pi controller with face recognition algorithm, embedded sensors which are connected to the Raspberry Pi in order to transmit the collected data, a camera module, and a stepper motor for locking/unlocking the door. Face recognition is implemented using OpenCV and python.

The image that is captured from the live video is pre-processed and then cropped. Face detection on these cropped and enhanced images is done using Local Binary Pattern (LBP) technique. LBP is a simple and effective method for detecting faces through labelling the pixels of an image by thresholding the neighboring pixels and storing the result as a binary number. Faces can be represented as a data vector by combining LBP with histograms. After face detection, recognition is done by comparing the face with the database. Accuracy of this recognition is calculated on every feature of the detected image. In case an unauthorised person is recognised, the system sends an alert message to the home owner through a mobile application. There is also a buzzer installed which beeps when an unauthorised person tries to break the door lock.

IoT is the backbone for the system's functionalities. Development of a mobile application is very handy as it makes the system accessible from anywhere.

CHAPTER 4

PROJECT REQUIREMENT SPECIFICATION

4.1. Introduction

This is a requirement specification document for securing an IoT Based Surveillance System using Blockchain. The product will be accessible via the internet using a client UI which is backed by the entire IoT-Blockchain ecosystem.

4.2. Project Scope

This project serves the objective of securing surveillance footage obtained from IoT devices by preventing unauthorized access and enforcing access management on stored data. Such a system will prevent tampering of data and subsequent malicious activities.

4.3. Product Perspective

This product serves as an enhancement of existing surveillance systems. It is a standalone product and can be coupled with other security devices for improved security measures.

4.3.1. Product Features

- Picture of individual is captured when the individual attempts to enter a building
- Every time an individual requires access, a picture of the individual is captured and stored in the blockchain
- Access is revoked if individual is not registered in the system but the picture of individual is still stored in the blockchain
- Using an identification number associated with each individual, the image can be retrieved from the ledger and viewed in the client UI

4.3.2. Operating Environment

- Hardware requirements:
    - Raspberry Pi 3B
    - Raspberry Pi camera
    - AMD based processor laptop
    - Processor: Intel i5 or higher
    - Hard disk: 20 GB minimum

- Software requirements:
    - OS: Ubuntu 16.04 or higher
    - Programming languages: Python, Node.js

4.3.3. General Constraints, Assumptions and Dependencies

- Smooth integration of IoT network with blockchain architecture and easy tracking across the system as a whole.
- This system does not consider the possibility of manual intervention being employed as a means to subdue the system.
- Front end User Interface is seamlessly integrated with backend and latency, if any, during communication between the two interfaces is negligible to the extent that it does not have a significant impact on the performance of the system.
- It is assumed that participating users of the blockchain network do not harbor ill intentions which may cause damage to the logic on which the system is built.
- It is assumed that IoT sensor data can be efficiently stored and processed by facial detection algorithms and is independent of blockchain implementation.
- Round the clock internet access must be made available for optimal functioning of the system.
- Continuous power supply must be ensured for the camera to function.

4.3.4. Risks

- Failure of Raspberry Pi architecture which is central to the functioning of the surveillance system.

- The hardware devices and components used in the IoT system must be covered with plastic encasing for electrical insulation and protection from accidental physical damage.
- Blockchain networks may suffer from possible network disconnection.
- Facial detection algorithms is highly accurate, but do not guarantee 100% accuracy. There is a possible margin for error which needs to be accommodated.
- It is assumed that the Raspberry Pi module can easily handle significantly high data flow from the connected Pi camera.

4.4. Functional Requirements

- Our system should be able to detect images when individual appears in front of pi camera
- Pi camera must immediately forward detected image data to IoT server
- IoT server acts as middle man which waits for images from Raspberry pi and inserts images into blockchain
- IoT server uses Hyperledger Fabric SDK to submit transactions to the ledger.
- Hyperledger Fabric enables the storage of detected images coming from Pi cameras.

4.5. External Interface Requirements

4.5.1. User Interfaces

- The User Interface is a website application. User interacts with the website by hovering over website elements, clicking buttons and entering information using the keyboard.
- Small delay may exist between input and output for smooth display of web application elements.
- Functions will exist to view detected images in the ledger by inputting unique identifier associated with each image

- Relevant error messages are displayed incase of miscommunication, invalid transactions, and unauthorized actions.

4.5.2. Hardware Requirement

- Raspberry Pi 3b:

  Raspberry Pi 3B is a popular microprocessor which has 40 GPIO pins which support various communication protocols such as i2c, spi. The Pi 3B has 4 USB connection ports, an audio output port, HDMI connector and a CSI connector (Camera Serial Interface) which is used to access video feed from the Raspberry Pi official camera. The Raspberry Pi 3b has 1GB of RAM. It lacks on-board storage which has to be provided externally using a SD card. There is a built-in WiFi module and a Bluetooth module for communication purposes. Additionally, there exists an ethernet jack which can be used for LAN connectivity.

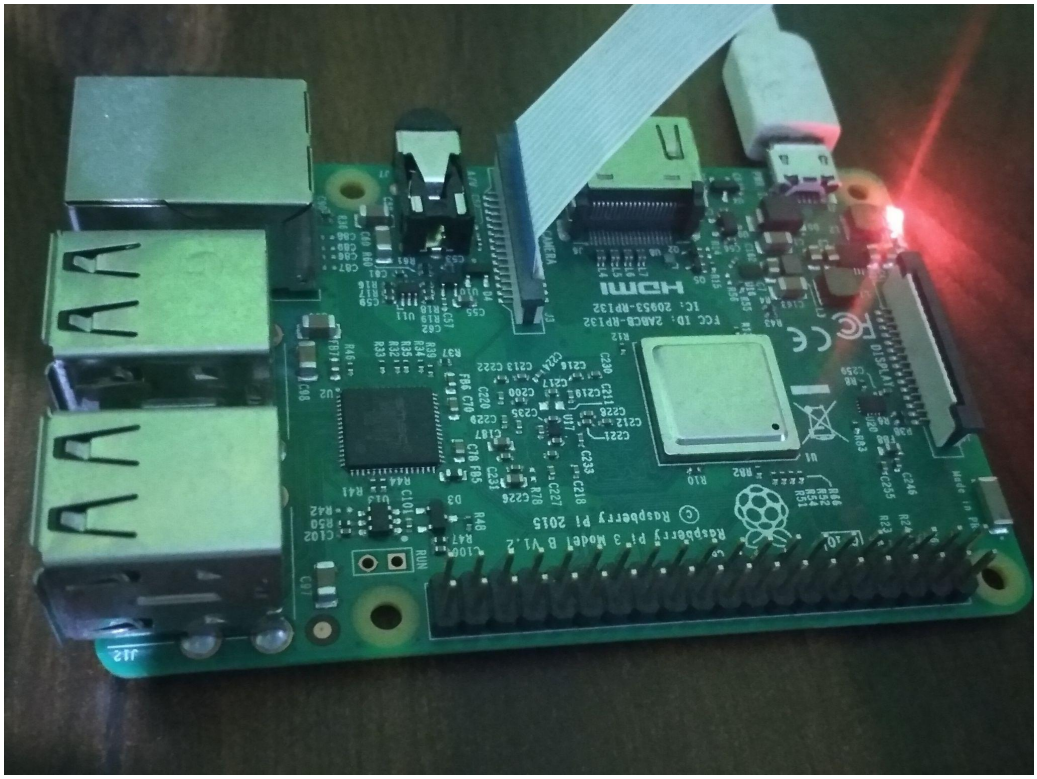

Fig. 4.1 Raspberry Pi 3B

- Pi Camera Module:

  The Pi camera module is a convenient mobile camera. It can be used in building IoT projects, and can be integrated with applications for surveillance purposes. The module houses a Sony IMX219 8-megapixel sensor. Pi camera is suitable for capturing quality video with high-resolution, as well as still photographs.

  The Camera is made up of a narrow circuit, dimensions of 25mm by 20mm by 9mm, and is connected to a Raspberry Pi's Camera Serial Interface. Camera supports still images and video with full resolution upto 2592x1944 as well as 1080p30, 720p60 and 640x480p60/90 options.

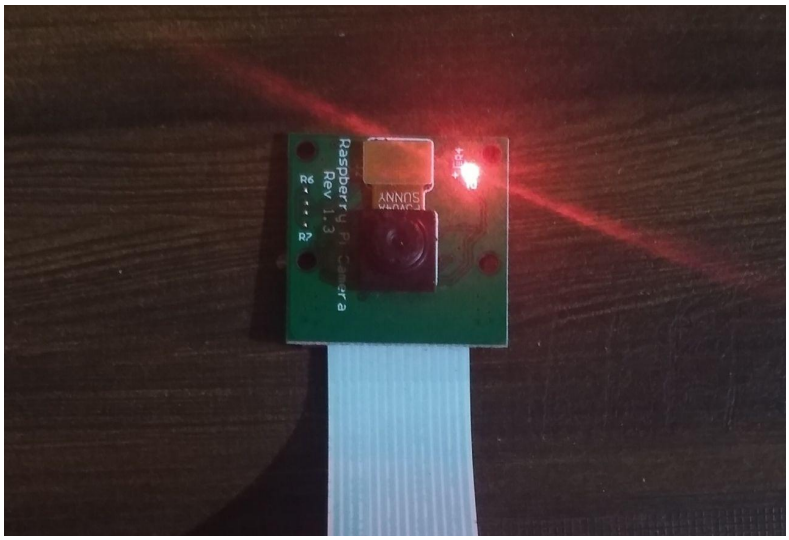

Fig. 4.2 Pi Camera

4.5.3. Software Requirements

- Python programming language
- Node.js programming language
- Node Package Manager (NPM)
- Node.js SDK
- Hyperledger Fabric

4.5.4. Communication Interfaces

- HTTP protocol
- Wifi based IEEE 802.11 LAN protocol
- REST architectural style
- JSON format for data-interchange

4.6. Non-Functional Requirements

4.6.1. Performance Requirement

- System should be reliable and robust
- System should perform efficiently with good response time
- System should be user friendly
- System should be able to handle continuous requests

4.6.2. Safety Requirements

- If unauthorized individual is detected multiple times, action must be taken against the individual
- Database containing the images of authorized people must be updated regularly.

4.6.3. Security Requirements

- Data protection laws of the user's country must be followed such as India's

upcoming PDPB (Personal Data Protection Bill) and the EU's GDPR (General Data Protection Regulation).

- Malpractice by any network user will create a transaction alert which will be reviewed by all users in the network.

4.6.4. Other requirements

- Should be scalable to handle large loads
- Application should be easy to maintain

CHAPTER 5

HIGH LEVEL SYSTEM DESIGN

5.1. Introduction

This document is designed to be a reference for an individual with an interest in the architecture of IoT based surveillance system using Blockchain, Blockchain based IoT network, IoT surveillance system client application, and IoT surveillance system server application. This document describes each system, the architecture of the application and the motivation behind the chosen design.

The objective of this High Level Design Document is to specify the necessary details required for implementation. This document highlights the high level interaction between different modules.

5.2. Current System

Current system is a centralized IoT surveillance system with cloud support. The data collected by sensors on the IoT device is transmitted to the cloud infrastructure which processes the incoming data and stores a backup for monitoring and data analysis.

5.3. Design Considerations

5.3.1. Design Goals

The major goal of the current design is to enhance the security and integrity of data in the IoT network. It is important to address vulnerabilities such as a single point of failure and unencrypted data. To achieve this goal, the application is geared towards utilising design principles of a permissioned blockchain - Hyperledger Fabric, which has properties such as decentralisation, hashing of blocks, consensus protocols, modularity etc.

The second major goal is to design a simple user service framework which provides functionalities of distributed ledger and smart contract in the form of a client application. Usually, a client application is provided with default credentials which lead to security risks. The proposed design has an identity management service to authorize credentials and permissions for submitting transactions.

The third major goal is design compatibility and technological restrictions. The IoT end is intended to have a design that is relatively simple to set up. Theoretically, a smart camera is capable of performing many tasks. For implementation purposes, a single-board computer and a camera module are combined to form an IP camera which performs facial detection.

This design attempts to keep the IoT-Blockchain network and client application as independent as possible. Modular approach is employed for easy development and deployment.

The reasons for opting for the above design principles:

- Scalability in terms of adding and removing IoT devices at will.
- Data stored in the ledger is virtually impossible to tamper with.

5.3.2. Architecture choices

Layer Based Centralised-IoT platform architecture:

Traditional centralized models use a central server to manage IoT services.

Pros:

- Low implementation cost.
- Efficiency in terms of lower latency.
- Simple deployment.

Cons:

- Single point of failure.
- Data Integrity is not ensured.

- Vulnerable user authentication mechanism.

General Architecture of a centralised IoT system:

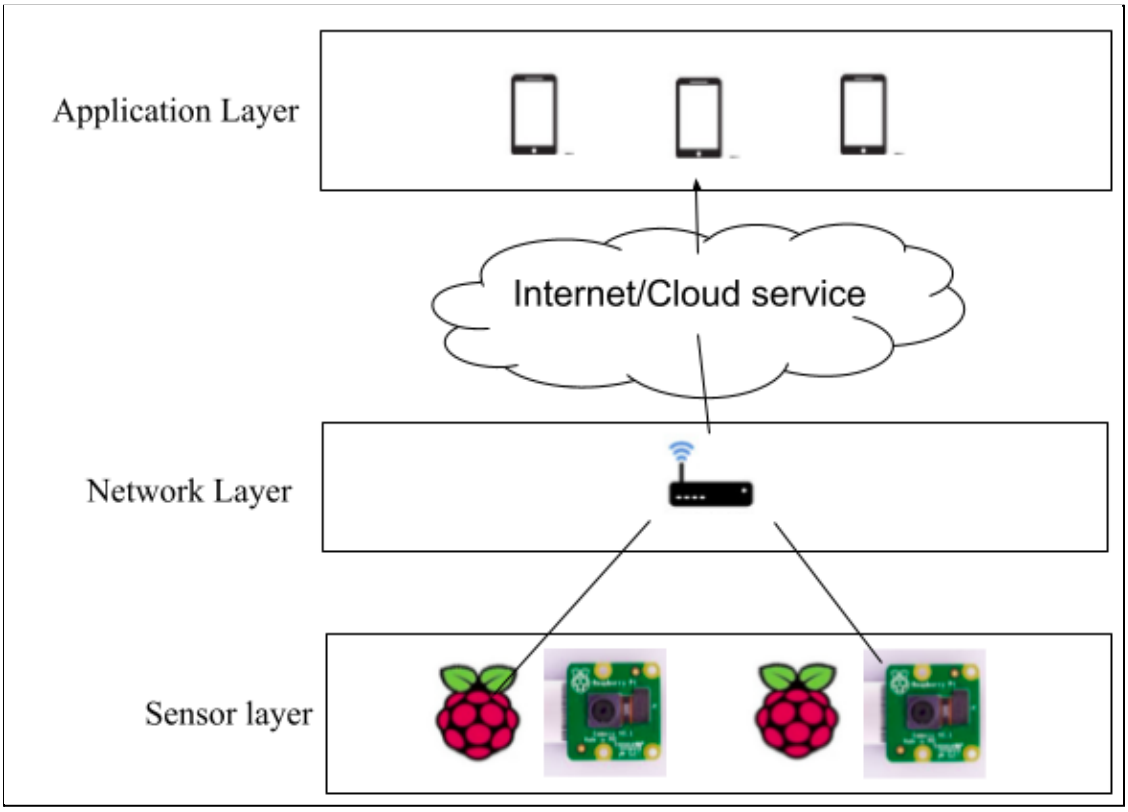

Fig. 5.1 Centralised IoT system architecture

Centralised system architecture has serious drawbacks in terms of privacy protection and insecure network services, which allow remote unauthorized access.

Our proposed architecture:

Figure 5.2 shows a high level overview with the software components. Facial detection is employed in the Raspberry Pi itself. This approach has a lot more benefits than the cloud based system. In this approach, there is no need for a cloud and a separate camera, both functionalities are combined into one single low powered device. On the left side, it can be seen that when images are captured by a raspberry pi camera, they are immediately passed to a face detection algorithm, and if no face is detected, images are deleted immediately.

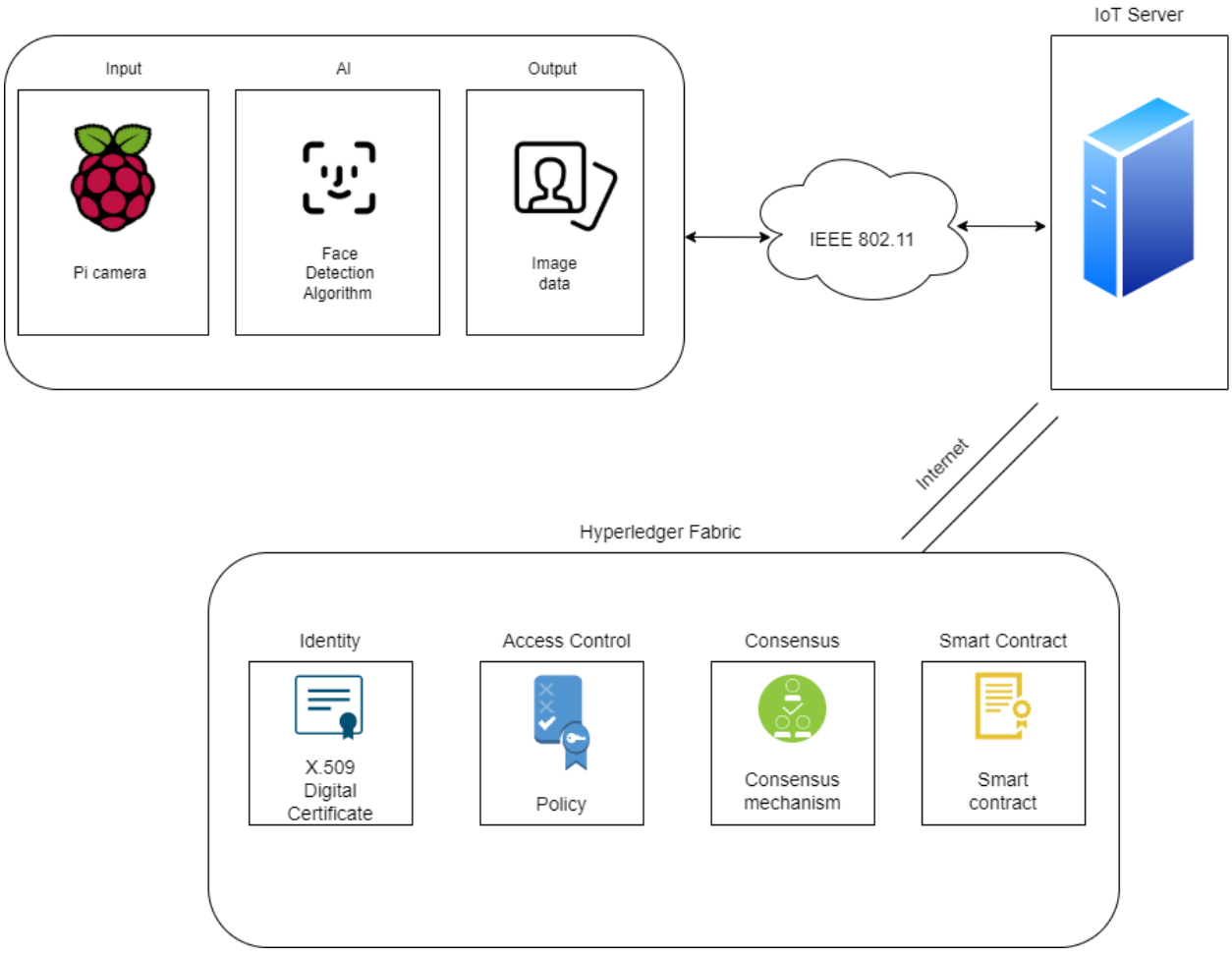

Fig. 5.2 Proposed system architecture

On the right hand side of Figure 5.2, the IoT server serves as a command center. The raspberry pi communicates with an IoT server via wireless communication compliant to IEEE 802.11 LAN protocols. Here, the Raspberry Pi behaves as a client to the IoT server. The IoT server waits for requests coming from the Raspberry Pi. The condition for the image to be stored in the blockchain is that a face has to be detected in the image. The IoT server with the help of Fabric SDK will issue transactions to the ledger, to store the detected images that are forwarded by the Raspberry pi.

Motivation behind proposed architecture:

- Access controls prevent misuse of application.
- Tampering is virtually impossible due to inherent properties of blockchain.

5.3.3. Communication Protocols under consideration

It is important to use a lightweight protocol in Raspberry Pi, which narrows the choices down to HTTP and MQTT, both of which are supported by Raspberry Pi. However, HTTP protocol was chosen because the Raspberry pi does not have to pull information, it simply has to push information to the IoT server. For inter-machine communication, REST architectural style is used due to its favourable characteristic of being lightweight compared to counterparts. For data-interchange, JSON format is used since its syntax follows Javascript language, which is also the language used to write the IoT server program.

5.3.4. Hyperledger Fabric for storing images

Hyperledger Fabric suits the design goals since it is a private blockchain which is open to use only to members of the network. Images should not be made public, and must only be available for viewing by the authorised network members. Hyperledger fabric has a highly modular and flexible architecture, which can be tailored to our use case of

identifying individuals that want access to a building. It's support for pluggable consensus protocols enables the platform to be more effectively customized to fit out particular use cases and trust models. Other notable characters that make it a compelling choice are:

- High transaction throughput performance
- Low latency of transaction confirmation
- Privacy and confidentiality of transactions and data pertaining to business transactions

5.3.5. Constraints, Assumptions and Dependencies

Assumptions:

- Smooth integration of IoT network with blockchain architecture and easy tracking across the system as a whole.
- No manual intervention to subdue the system.

Constraints:

- Selection of correct Hyperledger Fabric version based on release notes and matching viability with chosen use case.
- Ensuring data obtained from Pi camera is processed optimally keeping latency in mind.
- Using relevant API calls to gather required information.
- Using a practical approach when running the system for long hours to maintain optimal performance of the system.
- Ensuring Client UI is extremely responsive and does not cause cascading delay in the system.
- Ensuring the system can handle dynamic addition of devices and device owners, and heavier transaction flow rate.

5.4. High Level System Design

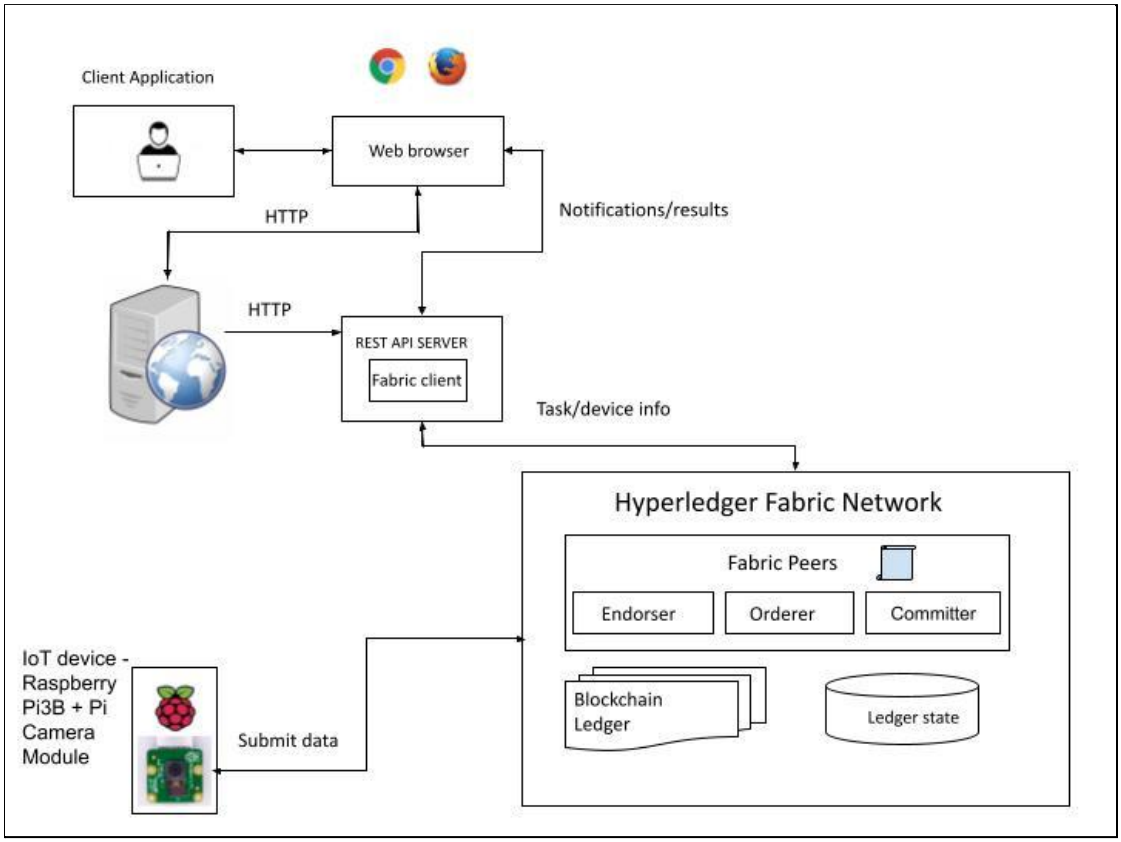

Fig. 5.3 High Level System Design

5.5. Data Flow Diagram

In this setup, the Raspberry pi device, pi camera, and IoT server are all connected in LAN and operate within the same subnet. In Figure 5.5, flow of data is depicted from left to right, starting with raspberry pi. Once Raspberry pi detects a face, it will take the captured image and forward the exact same image to the IoT server. This is achieved via a HTTP POST request that carries image and image metadata. The IoT server, located in the middle, which is a nodejs server, will receive a request with an image of the detected person. The IoT server will then activate the Hyperledger Fabric Client to issue a transaction for including the image in the ledger.

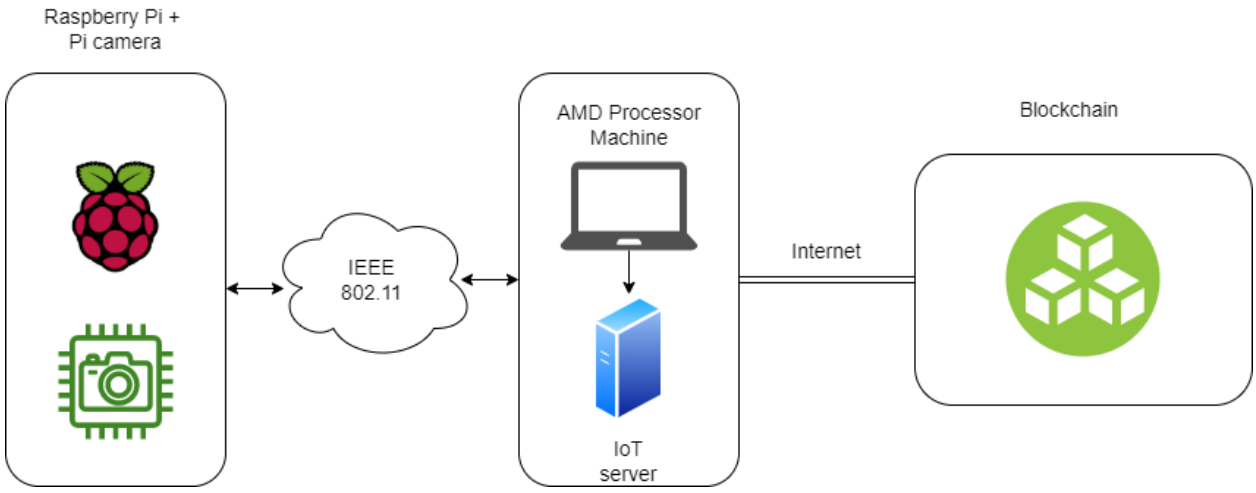

Fig. 5.5 Data Flow Diagram

The IoT server is responsible for receiving data from Raspberry pi and forwarding the data to the Hyperledger Fabric. To interact with the Hyperledger Fabric Network, the Fabric SDK for nodejs is used, which provides APIs to issue transactions to the ledger. The process of issuing a transaction occurs instantly after the image is received by the IoT server. Data is stored in a Fabric ledger in the form of assets. Assets are represented as a collection of

key-value pairs. When the value of the key is modified by a chaincode simulated from transaction execution, the world state is updated as a new transaction on the ledger. Chaincode controls asset state and typically represents assets in JSON form or binary form. Since the Raspberry pi sends image data in JSON format, it allows for easy transmission of the same JSON parameters to Hyperledger Fabric. The process begins with transaction initiation where the Raspberry Pi is assumed to be a client and holds a digital identity issued by Membership service provider.

Detected images are stored in Hyperledger Fabric which provides the following advantages:

- Hyperledger Fabric runs at no cost of transaction execution and uses a very lightweight and pluggable consensus protocol
- Hyperledger Fabric allows splitting the network into multiple smaller networks via channels, in order to preserve privacy between network members.

CHAPTER 6

PROPOSED METHODOLOGY

6.1. Raspberry Pi and Pi camera

- Raspberry pi is an ARM processor machine, low powered device
- Raspberry pi runs a flask server that serves the purpose of surveillance
- Raspberry pi runs face detection algorithm
- If face is detected, a unique identifier is generated for the image, called Face_Id
- Raspberry pi will convert image data into base64 format. The reason the image is converted into json format at the Raspberry pi end is because HyperLedger Fabric stores images in json format. The image is first converted into base64(binary to string) and sent using json format.
- Raspberry pi will send Device_Id, Face_Id, timestamp and image data in JSON format via python request.post function, to the IoT server

6.2. IoT Server

- Listens for POST requests from raspberry pi
- Serves as middle man which waits for image data from pi camera, to insert into blockchain
- The IoT server will then activate the HLF client and issue transaction for inclusion of detected image in the ledger
- The IoT server will issue transaction proposals with parameters of clientID, chaincodeID, txPayload, timestamp and clientSig. Then the process of endorsement and ordering will take place. Finally, ordered blocks are sent to committers which write to the ledger with asset key as Face_I and value as the detected image.

6.3 Hyperledger Fabric

- Has chaincode functions for querying and inserting images into the ledger
- Stores images in key-value pair format where the key is Face_Id and value is image data

CHAPTER 7

IMPLEMENTATION AND PSEUDO-CODE

7.1. Data Transmission Overview

Data transmission is shown in Figure 7.1 which highlights the interaction between all components in the system. When Raspberry Pi establishes connection to the LAN, it immediately starts the process of video surveillance. Raspberry pi analyses each frame for face detection and if a face is detected, HTTP client is activated and through an end point, a POST request is generated.

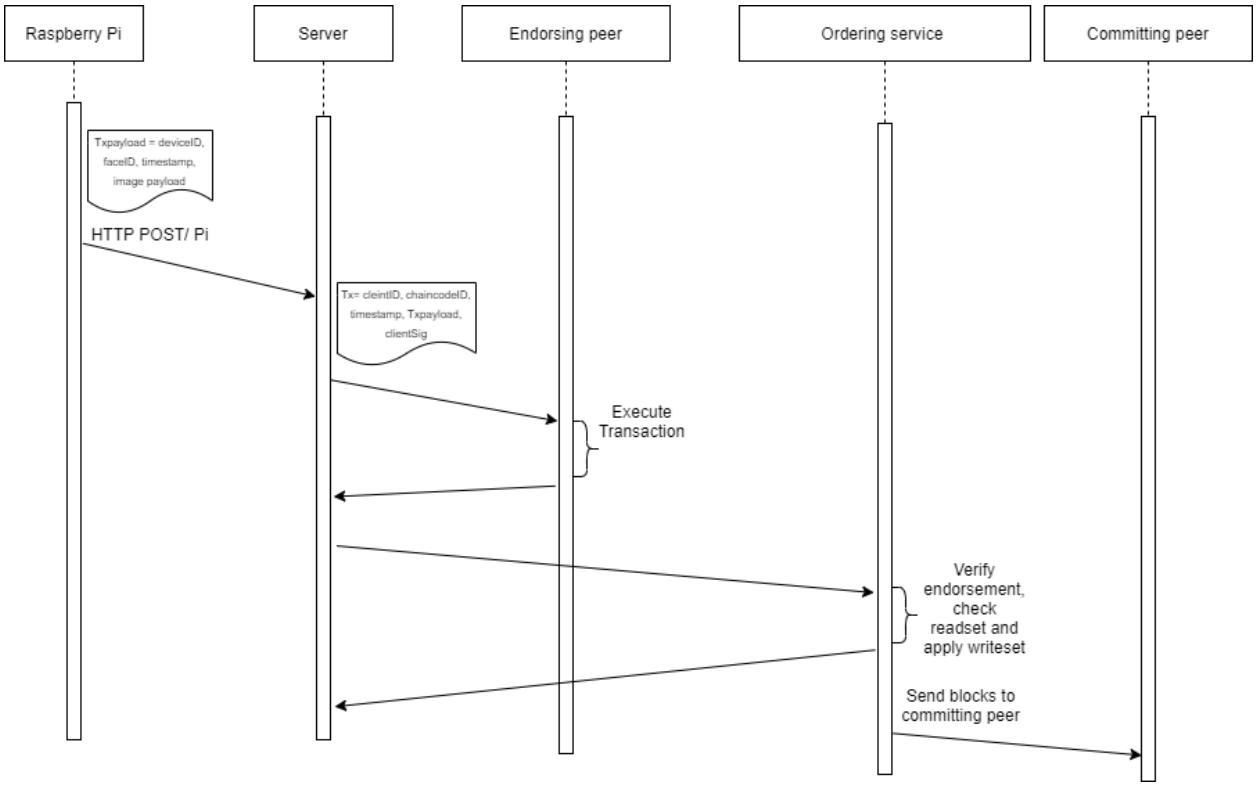

Figure 7.1 Sequence Diagram depicting transaction execution

Four important data parameters need to be stored in the blockchain,, which are shown in Figure 7.2. The first parameter is Device_Id, which signifies which device is transmitting the information, as in a typical scenario, it is highly likely that multiple devices will be employed in video surveillance, which makes distinction between devices a critical task. Second data parameter is Face_Id, which is a unique id assigned to every detected image. Third parameter is timestamp, which records the exact capture time of the image. Fourth parameter is the image frame itself, which is encoded in base 64 format.

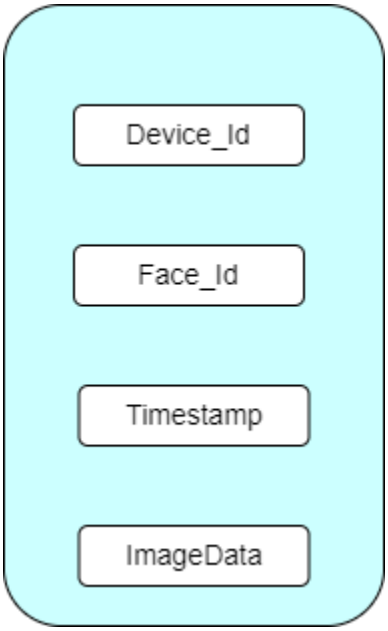

Figure 7.2: JSON parameters send by Raspberry Pi

As shown in Figure 7.1, these parameters will be sent through an exposed API with an endpoint from Raspberry Pi to IoT server.

7.2. IoT server working

Before IoT server issues transactions to the Hyperledger Fabric, the following conditions must be satisfied:

- Channel is running in Fabric
- Raspberry Pi is registered and enrolled with Certificate Authority and MSP contains digital identity of Raspberry pi
- Chaincode with endorsement policy is deployed on peers in the Fabric Network

Now, the IoT Server is responsible for initiating transactions with the four parameters obtained from Raspberry Pi and passing them as is, to the transaction payload. The IoT server behaves as a client for Hyperledger Fabric, since it interacts with the Hyperledger Fabric for issuing transactions. Our endorsement policy requires both participating organisations, one that houses the surveillance system, and the other that monitors access control, must endorse transactions coming for the client. The transaction proposal is now constructed and Fabric holds assets in key-value pair form. The key is Face_Id and the other three parameters constitute the image data, which will together be stored as the value part. This makes it easy to query the ledger using Face_Id as input parameter. The transaction proposal utilises API's provided by nodejs SDK to generate a transaction. The API invokes chaincode functions using input parameters that are sent from Raspberry Pi. The Fabric Client SDK holds the public-private key pair of Raspberry pi, which is used to sign transactions before they are sent to orders for execution.

7.3. Hyperledger Fabric working

Endorsing peers in Figure 7.1 will receive transaction proposals which includes clientID, chaincodeID, txPayload, timestamp and clientSignature. Only the endorsing peers specified by the chaincode will receive the transaction proposal, which in this case are two endorsing peers. The endorsing peers will check the format of the transaction, validity of signature and identity of client using MSP. It also ensures that the client is an authorized member of the channel. When all conditions have been satisfied, the endorses will invoke the chaincode function and pass the four proposed arguments. At this point, the transaction is executed but the ledger is not yet updated. The endorses sign the proposed transaction and send a proposal response back to the fabric client. The fabric client now issues a transaction to the ordering service to update the fabric ledger. The endorsed transaction is broadcasted to the ordering service. The transaction proposal now contains read/write sets besides endorsing peer signatures. The ordering service is responsible for ordering transactions in an agreed sequence, in order to package the transactions into blocks. The blocks are now broadcasted to the two peer nodes in the network.

The commiting/ validating peers will now verify the transactions in the blocks by viewing the read/ write sets. The commiting peers will compare the read set it holds with the world state of the same key. If the version of the key matches, write sets are committed to the world state and the detected image reaches all the peers.

Typically, in permissionless systems such as Ethereum, an order-execute architecture is followed where transactions are validated, ordered and then broadcasted to the peers. The peers then execute the transactions sequentially. Because of sequential execution, performance and scale is limited. Since smart contracts are executed by each peer in the network, complex measures must be taken in an order-execute architecture to protect the overall system from potentially malicious smart contract code.

Unlike the traditional approach, Hyperledger Fabric follows execute-order-validate model, whose working is as follows

- First, transactions are executed by peer nodes, its correctness is checked and the transactions are endorsed.
- Second, the endorsed transactions are sent back to the client application which checks if enough endorsements have been received. If enough endorsements have been received, the client application sends the endorsed transactions to the ordering service. The ordering service orders the transactions and puts the ordered transactions in blocks. These blocks are then broadcasted to all the peer nodes in the network.
- Third, transactions in a block are validated against application specific endorsement policy and then committed to local copy of ledger of each peer.

In this architecture, the transactions need to be executed(endorsed) by only a subset of peer nodes. This allows for parallel execution, thus increasing performance and scale of the system. This eliminates non-determinism, as inconsistent results are filtered out before ordering.

7.5. Raspberry Pi surveillance and detection

This chapter explains face detection using raspberry pi.

In Figure 7.3, code structure used to implement surveillance and face detection in Raspberry pi is displayed.

```
whybow@vybhavpes1201801764:~/capstone/face_detection(testjson)$ tree
.
├── README.md
├── requirements.txt
├── run.py
└── SmartCamera
    ├── auth.py
    ├── camera.py
    ├── database.db
    ├── db_models.py
    ├── __init__.py
    ├── main.py
    ├── models
    │   ├── facial_recognition_model.xml
    │   ├── fullbody_recognition_model.xml
    │   └── upperbody_recognition_model.xml
    ├── __pycache__
    │   ├── auth.cpython-36.pyc
    │   ├── camera.cpython-36.pyc
    │   ├── db_models.cpython-36.pyc
    │   ├── __init__.cpython-36.pyc
    │   └── main.cpython-36.pyc
    ├── static
    │   └── main.css
    └── templates
        ├── base.html
        ├── index.html
        ├── login.html
        └── signup.html

5 directories, 22 files
```

Figure 7.3

In figure 7.4, Cascade Classifiers and Haar Features are the methods used for Object Detection. There are different types of cascade classifiers according to different target objects. In our project, we have used a classifier that considers the human face to recognize it as the target object.

Haar Feature selection technique has a target to extract human face features. Haar features are like convolution kernels. These features are different permutations of black and white rectangles. In each feature calculation, we find the sum of pixels under white and black rectangles.

OpenCV provides pre-trained models on Haar features and Cascade classifiers. These models are located in OpenCV installation. necessary XML files are at /face_detection/SmartCamera/models.

```python
class VideoCamera(object):
    def __init__(self, flip = False):
        self.vs = VideoStream().start()
        # self.flip = flip
        time.sleep(2.0)

    def __del__(self):
        self.vs.stop()

    def get_frame(self):
        frame = self.vs.read()
        ret, jpeg = cv2.imencode('.jpg', frame)
        return jpeg.tobytes()

    def get_object(self, classifier):
        found_objects = False
        frame = self.vs.read().copy()
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        objects = classifier.detectMultiScale(
            gray,
            scaleFactor=1.1,
            minNeighbors=5,
            minSize=(30, 30),
            flags=cv2.CASCADE_SCALE_IMAGE
        )

        if len(objects) > 0:
            found_objects = True

        # Draw a rectangle around the objects
        for (x, y, w, h) in objects:
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

        ret, jpeg = cv2.imencode('.jpg', frame)
        return (jpeg.tobytes(), found_objects)
```

Figure 7.4

Camera.py contains face detection logic.

In figure 7.5, implementation logic for returning captured frames is shown. If object is detected, image will be converted to base64 format and sent using POST request to IoT server via python requests module.

```
def check_for_objects():
    global last_epoch
    global face_id
    while True:
        try:
            frame, found_obj = video_camera.get_object(object_classifier)
            if found_obj and (time.time() - last_epoch) > image_update_interval:
                last_epoch = time.time()
                timestamp = datetime.datetime.now()
                # timestamp = timestamp.strftime("%c")
                print("sending image to api..")
                im_bytes = frame
                # print('hey')
                im_b64 = base64.b64encode(im_bytes).decode("utf8")
                #json object(face_id,)
                # print('hey-1')
                payload = json.dumps({"Face_Id":face_id,"ImageData": im_b64,"Timestamp":timestamp,"Device_Id":device_id},default=str)
                # print('hey-2')
                response = requests.post(api, data=payload, headers=headers)
                # print('hey-3')
                face_id = face_id + 1
                print('done!')
        except:
            print ("Error sending image: ", sys.exc_info())
```

Figure 7.5

7.6. Transmission of Image Data to IoT Server

Image is sent from Raspberry Pi to the IoT server when a face is detected. Raspberry pi communicates with Raspberry Pi using HTTP protocol. POST request is sent to the IoT server containing Face_Id and image data.

Data format used throughout the study for submitting images to the IoT server is application/JSON. It is a standard format of sending plain text or any other type of data. Since assets in Hyperledger Fabric are stored in JSON format, the only way to store images in the ledger is by converting the image into a data type that JSON supports. The image frame is converted into base64 format and sent in JSON format. The figure 7.3 shows the parameters that are sent to the IoT server and exact same parameters are inserted into the Fabric ledger.

```
{
"Face_Id":face_id,
"ImageData": im_b64,
"Timestamp":timestamp,
"Device_Id":device_id
}
```

Figure 7.3: JSON structure data

7.7. IoT Server

The IoT Server has two important components, the nodejs server and the Hyperledger Fabric SDK which serves as client for Hyperledger Fabric. Express.js application server framework is used to build the server in nodejs. With an exposed API in nodejs, the Raspberry pi is able to send the data in the format mentioned in Figure 7.4. The nodejs server will pass the data to the Hyperledger Fabric SDK for committing transactions in the ledger.

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

    if (req.body.ImageData) {
        console.log(" Device Id is:-", req.body.Device_Id);
        console.log(" Face id is :-", req.body.Face_Id);
        console.log(" Timestamp image captured in ESP EYE:-", req.body.Timestamp);
        console.log(" Time when image is sent to to nodejs :-", req.body.TimeStamp);
        console.log(" Timestamp image is received in nodejs :-", new Date().toISOString());
        console.log(" Base64 Image is:-", req.body.ImageData);
    }

    res.sendStatus(200);
    console.log("Received image and uploading to blockchain");
    // sending parameters to main function to initiate hyper ledger fabric
    // main(req.body.Face_Id, req.body.DeviceId, req.body.Timestamp, req.body.ImageData);
});
```

Figure 7.4

7.8. Hyperledger Fabric Setup

Our use case involves two organizations, each employing one peer. One organisation owns the building that individuals will enter and leave on a daily basis, and this organisation will house the video surveillance system. The other organisation monitors access control.

The Fabric implementation is divided into two distinct components - chaincode implementation and client implementation.

7.8.1. Chaincode Implementation

The chaincode implementation which is deployed and executed inside the network is shown in Figure 7.5. fabric-contract-api module for nodejs is used for implementation of Hyperledger Fabric chaincode which provides the Contract interface. This API provides an entry point to write chaincode functions in Go, Java, and Javascript (nodejs). The client need not be in the same programming language as the chaincode implementation.

The implemented functions can be broadly categorised into those for inserting images into the ledger and those for querying images from the ledger.

For inserting images coming from the Raspberry pi, create_Image function is implemented. This function takes five arguments - ctx, Face_Id, Device_Id, Timestamp and ImageData. The first argument, ctx, refers to the state of the transaction. ctx allows us to maintain variables when invoking transactions. ctx is also important because it offers a large number of APIs to establish communication and perform transaction processing.

As it can be seen in Figure 7.5, ctx.stub is used to access APIs such as putState API which creates key-value pairs where the key is ‘Face_Id’ and value is the image data.

```
async createImage(ctx, faceid, device_id, timestamp, imagedata) {
    console.info('START : Create Image ');
    const image = {
    device_id,
    timestamp,
    docType: 'image',
    imagedata
};
```

Figure 7.5

For querying the ledger, the queryImage function allows the client to query the world state of a particular Face_Id. It takes two parameters - ctx and Face_Id and returns the image data corresponding to that particular Face_Id.

```
async queryImage(ctx, faceid) {
    const imageAsBytes = await ctx.stub.getState(faceid);
    if (!imageAsBytes || imageAsBytes.length === 0) {
        throw new Error(`${faceid} does not exist`);
}

    console.log(imageAsBytes.toString());
    return imageAsBytes.toString();
}
```

7.8.2. Hyperledger Fabric Client implementation

The Hyperledger Fabric Client is a means for the outside world to interact with the blockchain network. In order for the client to interact with the Fabric Network and invoke chaincode, the fabric-network module for nodejs is used.

The fabric-network module provides a number of important classes for making calls to the network. The Gateway class has various methods to establish connection and interact with the fabric network. The submitTransaction function is used to submit a transaction to the endorsing peers by passing the name of the chaincode function, and parameters associated with that particular chaincode function. This transaction will then be evaluated by endorsing peers. The endorsed transactions will be returned to the client. The client will send the endorsed transactions to the ordering service. The ordering service will package the transactions into blocks and broadcast them to peers in the network and eventually blocks will be committed to the ledger.

CHAPTER 8

RESULTS AND DISCUSSION

The IoT blockchain network has been set up in the local environment and surveillance was carried out successfully.

Image Detection

Figure 8.1 represents data capture from raspberry pi. Flask server is started on port 5000 and image is captured and sent to the IoT server.

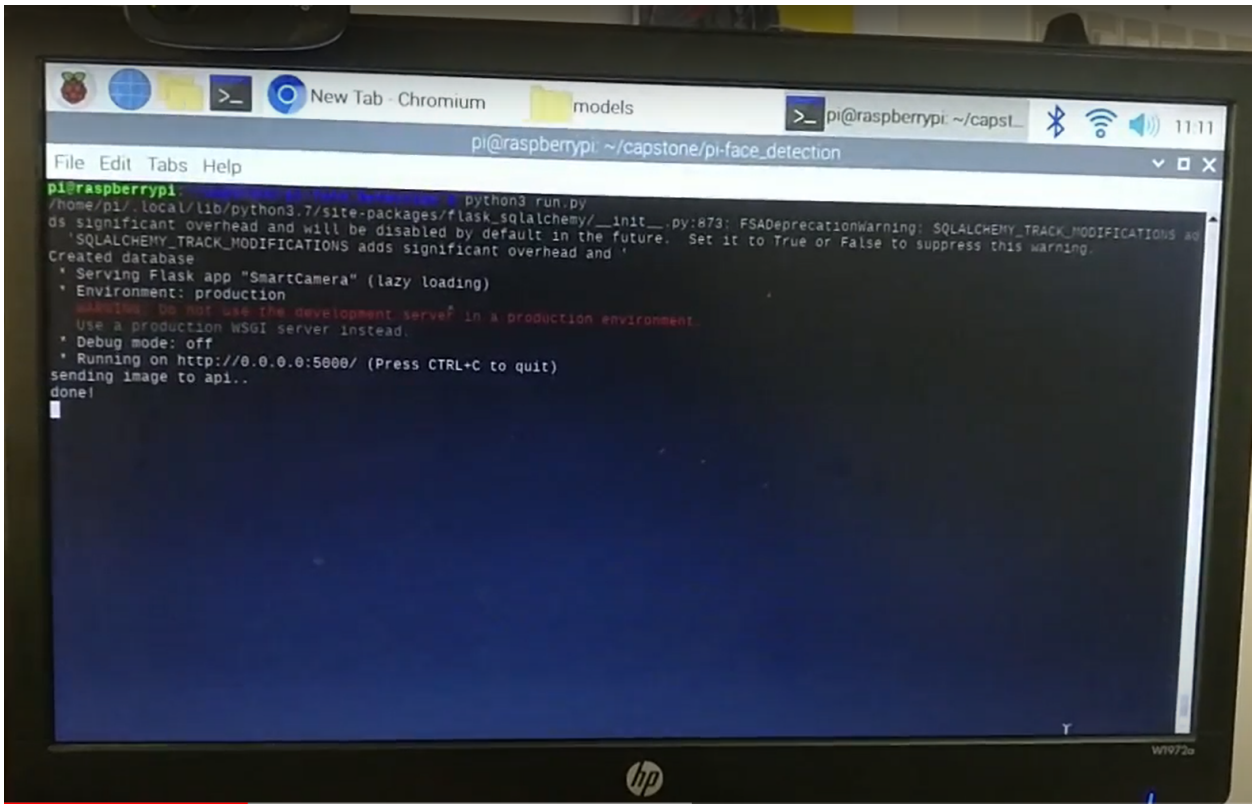

Figure 8.1 Raspberry pi running flask server

In figure 8.2, the detected image can be viewed on the right hand side of the screen. The bounding box represents the part of the subject that is detected.

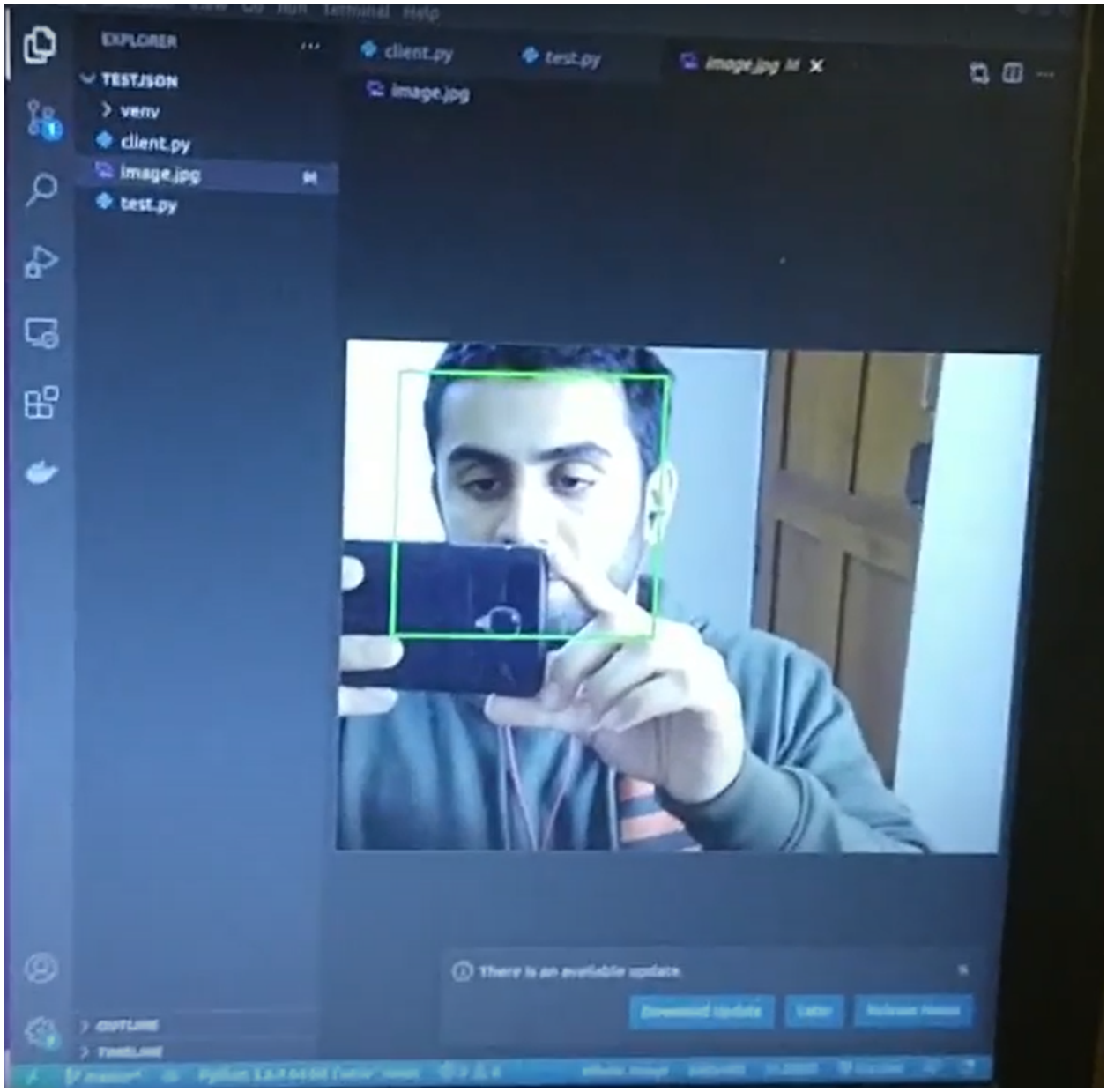

Figure 8.2 Detected Image

Raspberry pi and IoT server interaction

The figure 8.3 represents the IoT server that is developed using express.js framework. When raspberry pi captures images, it sends the images in JSON format to the IoT Server. The images have been successfully received at the IoT server end as it can be viewed below.

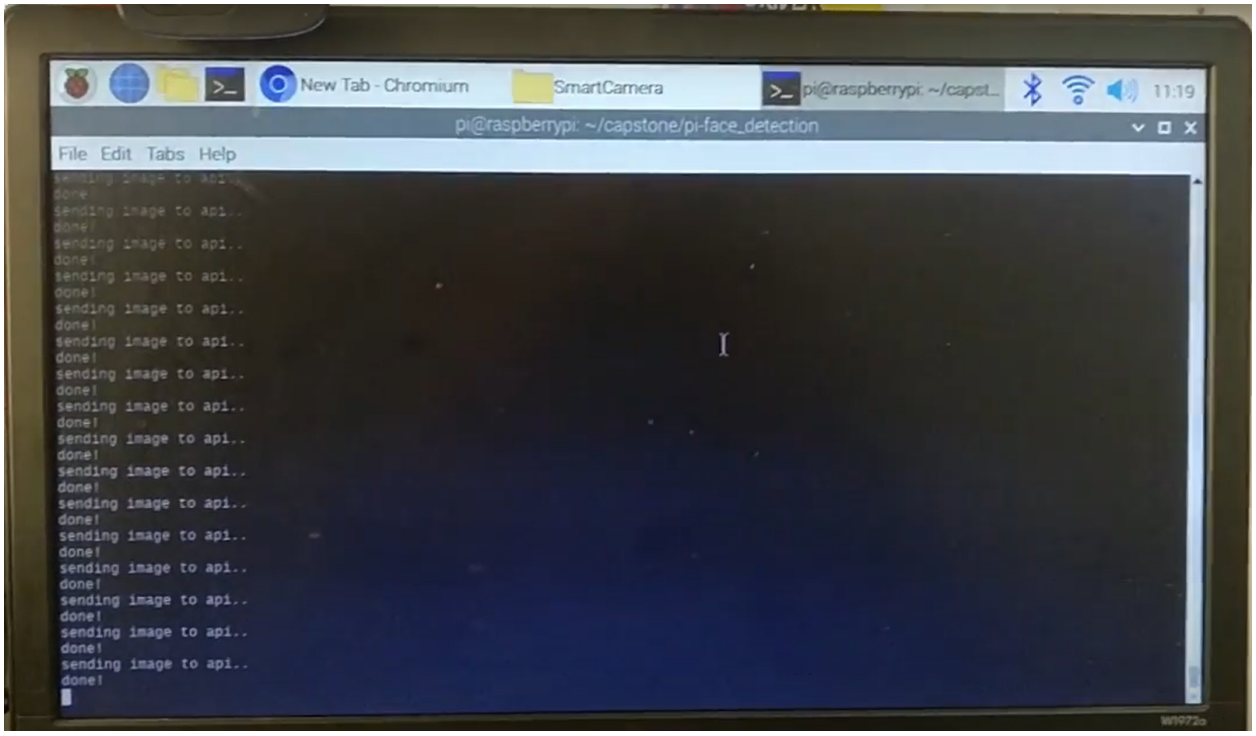

Figure 8.3 Image sent to IoT server

Working Hyperledger Fabric

Figure 8.4 represents the start of fabric network using startNetwork bash script. This script will start the network, create docker containers and deploy smart contract chaincode in the channel.

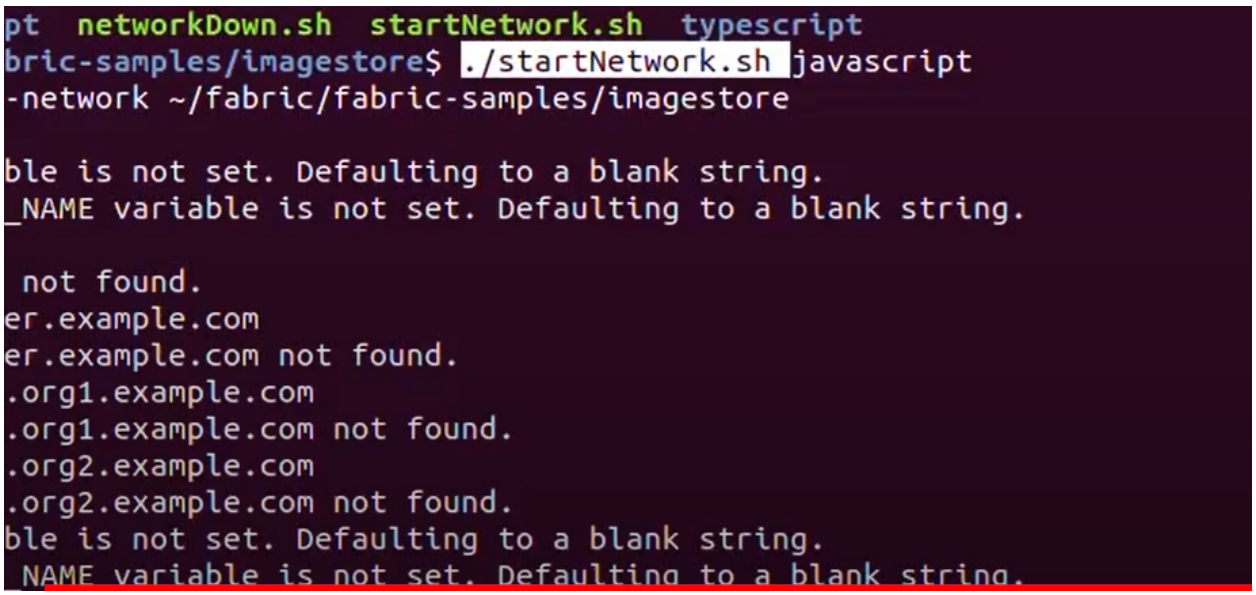

Figure 8.4

Once the network has been started, two peers, an order and a database are created and deployed on a channel. One peer belongs to the organisation that houses the surveillance system and the other peer belongs to the organisation that monitors access control. Figure 8.5 shows the two peers, orderers and database.

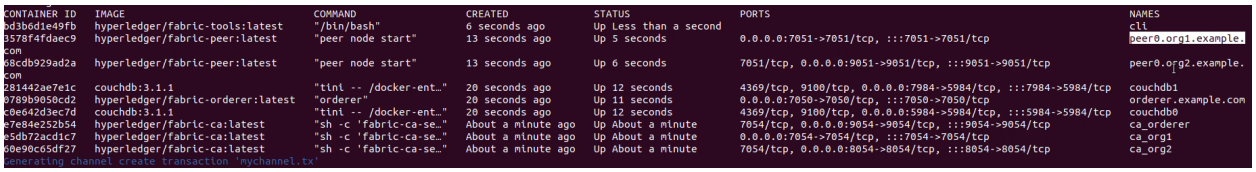

Figure 8.5

Committing image in Fabric ledger

Figure 8.6 shows the base64 format of the detected image, which is committed into the Fabric Ledger. The key is face id and value is base64 format of the image. The two together form a key value pair asset that is stored in the ledger.

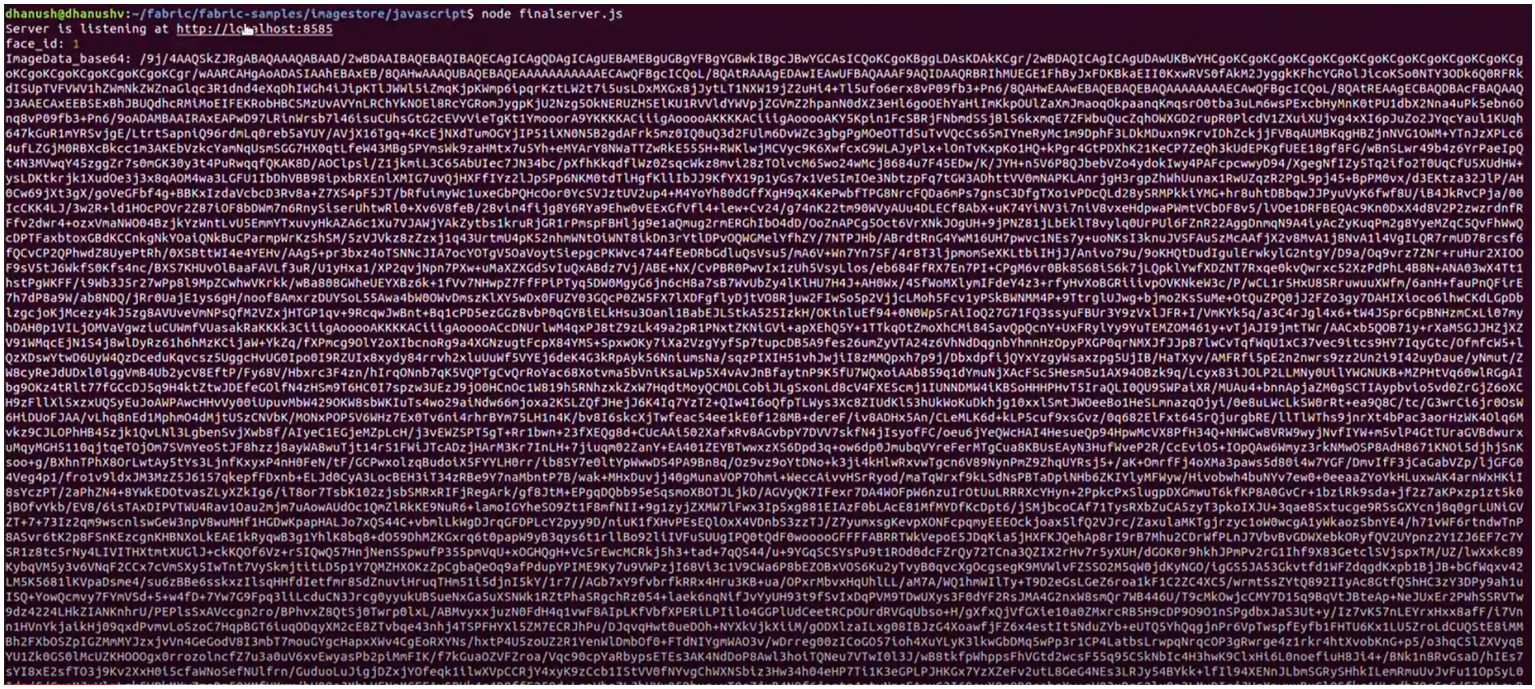

Figure 8.6

Finally, the transaction details are represented in Figure 8.7. The timestamp of when the image data is recorded in the ledger, along with confirmation that the ledger has been updated is shown.

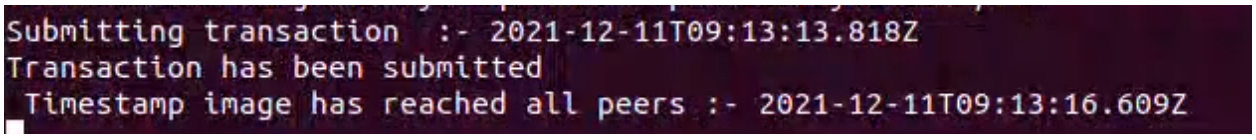

Figure 8.7

We successfully implemented an IoT surveillance system that is integrated with blockchain for storing images in a secure and immutable manner.

CHAPTER 9

CONCLUSION AND FUTURE WORK

Surveillance is a sensitive task that has been continuously improving with technological advances. This paper attempts to solve privacy and data integrity challenges by incorporating Machine Learning, Blockchain and IoT. We have better understood our problem statement after reviewing dozens of research papers and understanding the intersection of the three domains.

In a memory constrained device such as the Raspberry Pi, it is important to realise what computation can be performed at the device end and what must be shifted to the IoT server. The selection of the blockchain framework is also vital for feasible implementation of the proposed methodology. After careful consideration, Hyperledger Fabric was chosen for storing images of individuals. In order to show feasibility of design, Hyperledger Fabric is run in LAN but can also be deployed in the cloud.

We were able to successfully create a surveillance system that utilises blockchain technology and achieved this goal while accounting for the restrictions that come from the use case and available technologies.

Future scope of our project is to be able to scale the system to setup a real life surveillance system which comprises multiple cameras and thousands of individuals that require access to enter a building.

CHAPTER 10

REFERENCES / BIBLIOGRAPHY

[1] R. Wang, W. -T. Tsai, J. He, C. Liu, Q. Li and E. Deng, "A Video Surveillance System Based on Permissioned Blockchains and Edge Computing", 2019 IEEE International Conference on Big Data and Smart Computing (BigComp), pp. 1-6, 2019.

[2] L. Al-Sahan, F. Al-Jabiri, N. Abdelsalam, A. Mohamed, T. Elfouly and M. Abdallah, "Public Security Surveillance System Using Blockchain Technology and Advanced Image Processing Techniques", IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT), Doha, Qatar, pp. 104-111, 2020.

[3] W. Villegas-Ch, X. Palacios-Pacheco, and M. Román-Cañizares, "Integration of IoT and Blockchain to in the Processes of a University Campus" *Sustainability*, vol. 12, no. 12, pp. 4970, Jun. 2020.

[4] L. Hang and D.-H. Kim, "Design and Implementation of an Integrated IoT Blockchain Platform for Sensing Data Integrity", *Sensors*, vol. 19, no. 10, pp. 2228, May. 2019.

[5] S. Algarni, F. Eassa, K. Almarhabi, A. Almalaise, E. Albassam, K. Alsubhi, and M. Yamin, "Blockchain-Based Secured Access Control in an IoT System", *Applied Sciences*, vol. 11, no. 4, p. 1772, Feb. 2021.

[6] Y. Jeong, D. Hwang, and K. Kim, "Blockchain-Based Management of Video Surveillance Systems", 2019 International Conference on Information Networking (ICOIN), pp. 465-468, 2019.

[7] J. Celine and S. A. A, "Face Recognition in CCTV Systems", 2019 International Conference on Smart Systems and Inventive Technology (ICSSIT), pp. 111-116, 2019.

[8] Pallavi Gupta and Manisha Rajoriya, "Face Recognition based Home Security System Using IoT" *Journal of Critical Reviews* 7, 1001-1006, 2020.

[9] S. Roy, M. Ashaduzzaman, M. Hassan and A. R. Chowdhury, "BlockChain for IoT Security and Management: Current Prospects, Challenges and Future Directions", 2018 5th International Conference on Networking, Systems and Security (NSysS), pp. 1-9, 2018.

[10] P. Khan, Y.-C. Byun, and N. Park, "A Data Verification System for CCTV Surveillance Cameras Using Blockchain Technology in Smart Cities", *Electronics*, vol. 9, no. 3, pp. 484, Mar. 2020.

[11] S. Kumar, S. Swetha, V. T. Kiran and P. Johri, "IoT based Smart Home Surveillance and Automation", 2018 International Conference on Computing, Power and Communication Technologies (GUCON), pp. 786-790, 2018.

[12] D. Danko, S. Mercan, M. Cebe and K. Akkaya, "Assuring the Integrity of Videos from Wireless-Based IoT Devices using Blockchain", 2019 IEEE 16th International Conference on Mobile Ad Hoc and Sensor Systems Workshops (MASSW), pp. 48-52, 2019.

[13] A. Fitwi, Y. Chen, and S. Zhu, "A Lightweight Blockchain-Based Privacy Protection for Smart Surveillance at the Edge", 2019 IEEE International Conference on Blockchain (Blockchain), pp. 552-555, 2019.

[14] Mohammed I.M., Al-Dabagh M.Z.N., Ahmad M.I., and Isa M.N.M, "Face Recognition Using PCA Implemented on Raspberry Pi". In: Md Zain Z. et al. (eds) Proceedings of the 11th National Technical Seminar on Unmanned System Technology 2019. Lecture Notes in Electrical Engineering, vol 666. Springer, Singapore, 2021.

[15] P. Gallo, S. Pongnumkul, and U. Quoc Nguyen, "BlockSee: Blockchain for IoT Video Surveillance in Smart Cities", 2018 IEEE International Conference on Environment and Electrical Engineering and 2018 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I&CPS Europe), pp. 1-6, 2018.

CHAPTER 11

APPENDIX A: DEFINITIONS, ACRONYMS AND ABBREVIATIONS

1. IoT : Internet of Things
2. IP : Internet Protocol
3. Blockchain : Chain of blocks containing information
4. UI : User Interface
5. Fabric CA : Certificate Authority for Hyperledger Fabric
6. GPIO : General Purpose Input/Output
7. I2c : Serial communication protocol
8. SPI : Serial Peripheral Interface
9. ARM : Acorn RISC Machine
10. RISC : Reduced Instruction Set Computer
11. HDMI : High Definition Multimedia Interface
12. CSI : Camera Serial Interface
13. LAN : Local Area Network
14. HTTP : Hypertext Transfer Protocol
15. HTTPS : Hypertext Transfer Protocol Secure
16. API : Application Programming Interface
17. REQ: Request
18. Fig: Figure
19. SDK: Software Development Kit