Smart Home Automation

Your Name

March 15, 2025

Abstract

Home automation using Raspberry Pi has gained popularity due to its low cost and capacity to improve convenience and energy efficiency. This concept combines a smart home automation system with blockchain technology to enable safe and transparent energy invoicing. The system gathers energy usage data from a smart energy meter linked to an Arduino and sends it to a cloud-based IoT platform via MQTT and REST APIs for real-time monitoring.This data is retrieved by a Web3 application, which then interacts with an Ethereum smart contract to securely produce and handle energy invoices. Using blockchain technology, the system assures transparency, security, and decentralization in energy transactions. The integration of IoT and smart contracts eliminates intermediaries, lowering operating costs and increasing efficiency. This study exhibits a seamless transition from data collecting to invoicing and payment in a decentralized context, which helps to enhance smart energy management solutions.

Contents

Chapter 1

Introduction

Home automation using Raspberry Pi has gained popularity due to its numerous advantages and cost-effectiveness. These systems provide users with the ability to control household appliances through local networks or remote access, thereby enhancing convenience and energy efficiency[1].

Modern home automation technologies offer automatic meter reading, real-time monitoring, and remote control of electrical connections without the need for personal involvement[2]. The integration of Arduino controllers with GSM modules improves data transmission, allowing power companies to track energy use in kilowatt-hours (kWh) and create billing information[3]. In addition to energy monitoring, home automation systems include sensors, cameras, and web-based applications to increase security and device control[4]. Real-time data gathering and storage in databases, such as MySQL, ensures accurate monitoring and analysis. The use of the MQTT protocol improves data quality and dependability in IoT-based systems[5].

Blockchain and smart contracts have emerged as disruptive technologies with the potential to transform many sectors. Smart contracts are self-executing programs that autonomously enforce contractual conditions without the need for intermediaries, improving efficiency and lowering operational expenses[6]. These technologies offer advantages such as transparency, security, and decentralization, making them appropriate for a wide range of applications, from identity management to business process automation[2].

Home automation systems that combine IoT with blockchain-based smart contracts can improve trust and security in energy management. This project investigates the integration of a smart energy meter with a blockchain-based billing system, using Raspberry Pi, IoT platforms, and Ethereum smart contracts to create a decentralized, transparent, and automated energy billing solution.

Chapter 2

Project Workflow

2.1 Overview

This chapter depicts the workflow of a smart home automation system that is coupled with blockchain for energy billing. The smart meter captures energy usage data, which is then transmitted to a cloud-based IoT platform for real-time monitoring[1][2]. Data is processed using MQTT and REST APIs to ensure reliability[5].
A Web3 application obtains this information and communicates with an Ethereum smart contract to generate energy bills in a secure and transparent manner[6][7]. By merging IoT and blockchain, the solution automates billing and payments, increasing efficiency and security.

2.2 System Architecture

The architecture consists of multiple interconnected components, including sensors, microcontrollers, cloud platforms, and blockchain technology. The data flow and interactions are illustrated in Figure 2.1.

2.3 Components and Data Flow

2.3.1 Energy Measurement and Data Acquisition

The system starts with an Arduino connected to a smart energy meter, which measures power consumption. Since Arduino lacks a built-in analog-to-digital converter (ADC) with the required resolution, an external ADC is used for accurate readings. The measured data is then transmitted to a Raspberry Pi for further processing.

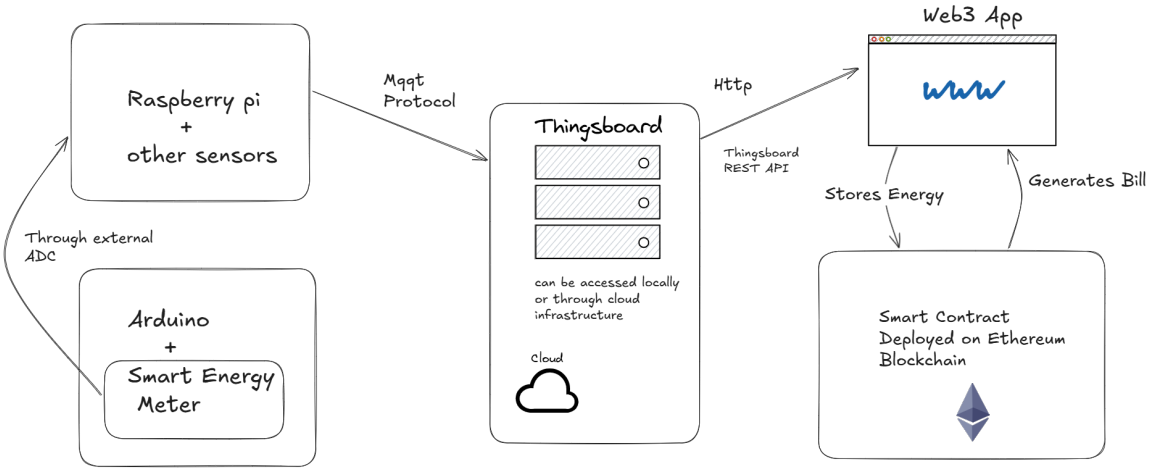

Figure 2.1: System Workflow

2.3.2 Data Transmission and Storage

This chapter illustrates the workflow of a smart home automation system that is integrated with blockchain for energy billing. The system's goal is to collect energy consumption data, store it in a cloud-based IoT platform, and bill and pay using a Web3 application that interacts with an Ethereum blockchain smart contract.

2.3.3 Web3 Integration and Smart Contracts

A Web3 application fetches energy consumption data from ThingsBoard via HTTP requests. The retrieved energy data is then stored in a smart contract deployed on the Ethereum blockchain. The smart contract automatically generates an energy bill based on the stored consumption values.

2.3.4 Billing and Payment System

Using HTTP queries, a Web3 application retrieves energy usage data from ThingsBoard. A smart contract that is implemented on the Ethereum blockchain then stores the energy data that has been recovered. Using the saved consumption values, the smart contract automatically creates an energy bill.

2.4 Conclusion

This process combines blockchain, cloud computing, and the Internet of Things to produce an automated and decentralized energy billing system. An effective and transparent billing

procedure is made possible by the modular architecture, which facilitates smooth data flow from energy monitoring to smart contract interactions.

Chapter 3

Schematic Design and Simulation in Proteus

3.1 Circuit Design

The circuit consists of:

- Voltage and current sensors connected to ESP8266
- UART communication between ESP8266 and Raspberry Pi

Figure 3.1: Schematic Diagram of the System

Chapter 4

ThingsBoard Integration

4.1 Introduction to ThingsBoard

Rapid breakthroughs in semiconductor technology and wireless communication have resulted in the development of low-cost sensor-based devices, which serve as the foundation for the Internet of Things (IoT) ecosystem[8]. These sensors produce massive volumes of data, demanding effective collection, processing, and management frameworks. IoT platforms play an important role in managing this data by offering connectivity, security, data visualization, and analytics capabilities[9][10]. Among these platforms, ThingsBoard has emerged as an effective open-source solution for IoT data collecting and management.

ThingsBoard is a Java 8-based IoT platform that acts as a gateway for devices that communicate using MQTT[11], CoAP[12], and HTTP[13]. These protocols allow for lightweight communication between resource-constrained IoT devices and cloud services. MQTT is a publish/subscribe protocol for small, low-power devices that enables efficient message exchange via a broker with varying Quality of Service (QoS) levels[11]. In contrast, CoAP is an UDP-based protocol designed for limited contexts, with lower overhead but lesser dependability than MQTT[12]. One of ThingsBoard's key features is the ability to build rules and plugins for message processing. Rules contain data filters, metadata enrichment processors, and action triggers that change messages into new formats before sending them to plugins. This rule-based system supports basic data processing, including threshold-based notifications. However, the technology does not automatically support complex data aggregation over time or across several devices[9].

ThingsBoard allows you to configure alerts for both devices and assets, which improves real-time monitoring and event-based automation. When aberrant situations are recognized, these alerts

alert users or initiate automated replies. Furthermore, the platform supports both lightweight communication protocols, such as MQTT and CoAP, and classic RESTful services[12].

4.2 Prerequisites for ThingsBoard MQTT Integration

Before integrating ThingsBoard with MQTT, ensure that the required software components are installed and configured correctly. The ThingsBoard platform must be running on either a local system or a cloud server. Although ThingsBoard has a built-in MQTT broker, an external broker such as Mosquitto can also be used if needed.

Additionally, MQTT communication requires appropriate network settings. Ensure that port **1883** is open for unencrypted communication. Each device must authenticate with ThingsBoard using an **Access Token**, which is assigned when the device is created in ThingsBoard.

4.3 Connecting ThingsBoard with MQTT

To send telemetry data to ThingsBoard using MQTT, follow these steps:

4.3.1 Step 1: Define Connection Parameters

The first step is to configure the MQTT connection by specifying the ThingsBoard host, access token, and MQTT port. Replace the `ACCESS_TOKEN` with the token obtained from the ThingsBoard device.

```
1 THINGSBOARD_HOST = "localhost"
2 ACCESS_TOKEN = "dU6S0YIAPX5WwfmB3wUi"  # Replace with your actual token
3 MQTT_PORT = 1883
4
```

4.3.2 Step 2: Install and Import MQTT Library

After setting up the connection parameters, ensure that the `paho-mqtt` library is installed. This library is required to establish an MQTT connection. Once installed, import the necessary modules in the Python script.

```
1 import paho.mqtt.client as mqtt
2 import json
3
```

4.3.3 Step 3: Create MQTT Client and Connect

The next step is to create an MQTT client instance and authenticate using the access token. The client must then connect to the ThingsBoard host on the specified port.

```
client = mqtt.Client()
client.username_pw_set(ACCESS_TOKEN)  # Use Access Token for authentication
client.connect(THINGSBOARD_HOST, MQTT_PORT, 60)

```

4.3.4 Step 4: Publish Sensor Data

Once connected, sensor data must be prepared in JSON format and published to the ThingsBoard MQTT topic. The following code snippet demonstrates how to send temperature and humidity data.

```
telemetry_data = {"temperature": 25.5, "humidity": 60}
client.publish("v1/devices/me/telemetry", json.dumps(telemetry_data))
print("Data sent successfully!")
client.disconnect()

```

4.3.5 Step 5: Verify Data in ThingsBoard

After publishing the data, it is essential to verify whether ThingsBoard has received it. This can be done by navigating to the ThingsBoard UI and checking the **Latest Telemetry** section of the configured device.

4.4 Prerequisites for ThingsBoard API Integration

Before integrating ThingsBoard with the API, ensure that the required software and authentication mechanisms are set up properly. The ThingsBoard platform must be installed on a local system or a cloud server to facilitate communication. A REST client such as Postman or the `fetch` API in JavaScript is necessary for interacting with ThingsBoard's API. Additionally, proper network access should be configured to allow API requests.

To ensure security, ThingsBoard requires authentication via a JSON Web Token (JWT). The authentication process involves sending a username and password to ThingsBoard's login API.

Upon successful authentication, a token is generated, which must be included in all subsequent API requests. Without this token, ThingsBoard will deny access to its resources.

4.5 Connecting to ThingsBoard API

To fetch telemetry data from ThingsBoard, follow these steps:

4.5.1 Step 1: Generate Access Token

The first step is to authenticate with ThingsBoard and obtain an access token. This is done by sending a POST request to the authentication API with valid credentials. The request should be formatted in JSON, containing a username and password.

```
POST http://localhost:9090/api/auth/login
Content-Type: application/json

{
    "username": "tenant@thingsboard.org",
    "password": "tenant"
}

```

4.5.2 Step 2: Extract JWT Token

Once the authentication request is processed, the API will return a response containing a JWT token. This token is essential for making authorized requests to ThingsBoard. The response will be in JSON format, and the token can be extracted from the `token` field.

```
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI...",
    "refreshToken": "eyJhbGciOiJIUzI1NiIsIn..."
}

```

4.5.3 Step 3: Fetch Telemetry Data

After obtaining the authentication token, the next step is to retrieve telemetry data from a specific device. The device ID must be specified in the request URL, and the JWT token must

be included in the request headers under the `X-Authorization` field. The following JavaScript code demonstrates how to fetch power telemetry data from a ThingsBoard device.

```
const response = await fetch(
    `http://localhost:9090/api/plugins/telemetry/DEVICE/0eb8ae80-ffe6-11ef-b36a-71656502eb9c/values/timeseries?keys=power`,
    {
        headers: { "X-Authorization": `Bearer ${thingsboardToken}` },
    }
);
const data = await response.json();
console.log(data);

```

4.5.4 Step 4: Process Response Data

Once the telemetry data is retrieved, it will be returned in JSON format. The response will contain key-value pairs representing the requested telemetry values. The received data can then be processed, displayed, or stored as needed, depending on the application requirements.

4.6 Conclusion

Integrating ThingsBoard with MQTT allows for efficient and secure real-time data transmission for IoT devices. By following the methods mentioned, devices can authenticate, publish telemetry data, and take advantage of ThingsBoard's monitoring and automation capabilities. This lightweight and scalable strategy improves IoT installations by providing consistent connectivity while also allowing for future enhancements such as encryption and improved data handling.

Chapter 5

Blockchain and Smart Contracts

5.1 Introduction to Solidity

Solidity is the Ethereum smart contract language. Below is a basic example:

```
pragma solidity ^0.8.0;

contract EnergyBilling {
    uint public energyConsumed;

    function recordEnergy(uint _energy) public {
        energyConsumed = _energy;
    }
}
```

5.2 Advantages of Blockchain for Billing

- Ensures transparency in transactions
- Provides security with immutable records
- Removes the need for third-party billing systems

5.3 Ethereum Gas Estimation

Smart contract transactions require gas. Example of estimating gas:

```
web3.eth.estimateGas({ data: "0x600160..." })
```

Chapter 6

Local Ganache and Dockerized Ganache

6.1 Setting Up Local Ganache

Ganache is used for local Ethereum blockchain simulation:

```
ganache-cli -p 7545
```

6.2 Running Ganache in a Docker Container

To run Ganache in a container:

```
docker run -d -p 8545:8545 trufflesuite/ganache-cli
```

Chapter 7

Building the Web3 Application

7.1 Fetching Energy Data from ThingsBoard

Using Web3.js, the app retrieves energy data from ThingsBoard and sends it to the smart contract:

```
const web3 = new Web3("http://localhost:8545");
const contract = new web3.eth.Contract(abi, contractAddress);

async function sendEnergyData(energy) {
    await contract.methods.recordEnergy(energy).send({ from: accounts[0] });
}
```

7.2 Deploying the Smart Contract

Using Truffle, deploy the contract:

```
truffle migrate --network development
```

7.3 Transaction History Recording

Each energy billing transaction is recorded on the blockchain for transparency.

Chapter 8

Dockerizing the Complete System

8.1 Creating a Docker Compose File

To ensure all services run together in containers:

```
version: '3'
services:
  thingsboard:
    image: thingsboard/tb-postgres
    ports:
      - "1883:1883"
      - "8080:8080"

  ganache:
    image: trufflesuite/ganache-cli
    ports:
      - "8545:8545"

  web-app:
    build: ./web
    ports:
      - "3000:3000"
    depends_on:
      - ganache
      - thingsboard
```

Chapter 9

Conclusion

This project successfully integrates IoT, cloud, and blockchain technologies to create an automated and transparent energy billing system. Future enhancements include adding AI-based energy predictions and integrating real-world payments.

References

[1] S. Jain, A. Vaibhav, and L. Goyal, "Raspberry pi based interactive home automation system through e-mail," in *2014 International Conference on Reliability Optimization and Information Technology (ICROIT)*. IEEE, 2014, pp. 277–280.

[2] S. Chaudhari, P. Rathod, A. Shaikh, D. Vora, and J. Ahir, "Smart energy meter using arduino and gsm," in *2017 International Conference on Trends in Electronics and Informatics (ICEI)*. IEEE, 2017, pp. 598–601.

[3] M. M. Rahman, M. O. Islam, M. S. Salakin *et al.*, "Arduino and gsm based smart energy meter for advanced metering and billing system," in *2015 International Conference on Electrical Engineering and Information Communication Technology (ICEEICT)*. IEEE, 2015, pp. 1–6.

[4] V. Patchava, H. B. Kandala, and P. R. Babu, "A smart home automation technique with raspberry pi using iot," in *2015 International conference on smart sensors and systems (IC-SSS)*. IEEE, 2015, pp. 1–4.

[5] R. A. Atmoko, R. Riantini, and M. K. Hasin, "Iot real time data acquisition using mqtt protocol," *Journal of Physics: Conference Series*, vol. 853, no. 1, p. 012003, may 2017. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/853/1/012003

[6] M. Abdelhamid and G. Hassan, "Blockchain and smart contracts," in *Proceedings of the 8th International Conference on Software and Information Engineering*, ser. ICSIE '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 91–95. [Online]. Available: https://doi.org/10.1145/3328833.3328857

[7] Y. Hu, M. Liyanage, A. Mansoor, K. Thilakarathna, G. Jourjon, and A. P. Seneviratne, "Blockchain-based smart contracts - applications and challenges," *arXiv: Computers and Society*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:182952419

[8] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "Next century challenges: Scalable coordination in sensor networks," in *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, 1999, pp. 263–270.

[9] B. Hammi, R. Khatoun, S. Zeadally, A. Fayad, and L. Khoukhi, "Iot technologies¡? show [aq id= q1]?¿ for smart cities," *IET networks*, vol. 7, no. 1, pp. 1–13, 2018.

[10] V. Gazis, M. Görtz, M. Huber, A. Leonardi, K. Mathioudakis, A. Wiesmaier, F. Zeiger, and E. Vasilomanolakis, "A survey of technologies for the internet of things," in *2015 international wireless communications and mobile computing conference (IWCMC)*. IEEE, 2015, pp. 1090–1095.

[11] Z. Kegenbekov and A. Saparova, "Using the mqtt protocol to transmit vehicle telemetry data," *Transportation Research Procedia*, vol. 61, pp. 410–417, 2022.

[12] Z. Shelby, K. Hartke, and C. Bormann, "Rfc 7252: The constrained application protocol (coap)," 2014.

[13] M. B. Yassein, M. Q. Shatnawi *et al.*, "Application layer protocols for the internet of things: A survey," in *2016 International Conference on Engineering & MIS (ICEMIS)*. IEEE, 2016, pp. 1–4.