PoT: Bridging IoT with Phone Technology

by

Haytham Khalil

A thesis submitted to the
School of Graduate and Postdoctoral Studies in partial
fulfillment of the requirements for the degree of

Doctor of Philosophy in **Electrical and Computer Engineering**

Faculty of Engineering and Applied Science
University of Ontario Institute of Technology (Ontario Tech University)
Oshawa, Ontario, Canada
September 2023

Thesis Examination Information

Submitted by: **Haytham Khalil**

Doctor of Philosophy in **Electrical and Computer Engineering**

Thesis title: PoT: Bridging IoT with Phone Technology

An oral defense of this thesis took place on September 14, 2023 in front of the following examining committee:

Examining Committee:

Chair of Examining Committee	Dr. Sanaa Alwidian
Research Supervisor	Dr. Khalid Elgazzar
Examining Committee Member	Dr. Mohamed El-Darieby
Examining Committee Member	Dr. Akramul Azim
Thesis Examiner	Dr. Masoud Makrehchi, Ontario Tech University
External Examiner	Dr. Tamer Nadeem, Virginia Commonwealth University

The above committee determined that the thesis is acceptable in form and content and that a satisfactory knowledge of the field covered by the thesis was demonstrated by the candidate during an oral examination. A signed copy of the Certificate of Approval is available from the School of Graduate and Postdoctoral Studies.

Abstract

The "Phone of Things" (PoT) introduces an innovative integration of IoT systems with the widely available telephone network infrastructure. It repurposes underused home landlines and existing communication servers, weaving them into the IoT fabric. By transforming IoT devices into SIP endpoints within the VoIP ecosystem, users can monitor and interact with these devices through regular phone calls, voice commands, or text messages. PoT presents a seamless user experience by capitalizing on ubiquitous phone network infrastructure while promoting context-aware telephony solutions. Using open-source technologies, PoT ensures affordability, interoperability, scalability, and security.

A tangible PoT prototype is developed using a Raspberry Pi equipped with Asterisk, a renowned open-source IP-PBX software. The Raspberry Pi acts as a gateway, facilitating communication between IoT devices and VoIP servers. Performance evaluation testing reveals that the Raspberry Pi 4 B can manage up to 182 concurrent calls, while the less performant Raspberry Pi Zero W can handle 12 simultaneous calls. These results highlight the potential of these compact, affordable boards as ideal PoT gateways for homes and small-to-medium businesses, making deployment of the framework more economical.

In addition, the thesis introduces "tSIP", a streamlined SIP version designed for PoT. It offers a concise message format, achieving up to 22% and 46% size reduction compared to traditional SIP and CoSIP messages. This compact format ensures quicker transmission, energy efficiency, and optimized network usage. The study also presents a decentralized registration and authentication mechanism for PoT, based on blockchain technology. A prototype is crafted on a private blockchain, emphasizing privacy, speed, and cost-effectiveness. This mechanism aligns with SIP's security standards and caters to embedded smart devices' constraints.

Lastly, to illustrate PoT's real-world application, the "Location Transparency Call" (LTC) system is introduced. LTC provides a context-aware telephony solution for businesses. It tracks employees via their RFID access tags, ensuring that incoming calls are redirected to the nearest phone to their current location, reducing missed business call occurrences.

Keywords— Phone of Things (PoT); Tiny Session Initiation Protocol (tSIP), Location Transparency Call (LTC); Internet of Things (IoT); Voice over Internet Protocol (VoIP); Asterisk; Blockchain

Author's Declaration

I hereby declare that this thesis consists of the original work I have authored. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize the University of Ontario Institute of Technology (Ontario Tech University) to lend this thesis to other institutions or individuals for the purpose of scholarly research. I further authorize the University of Ontario Institute of Technology (Ontario Tech University) to reproduce this thesis by photocopying or by other means, in total or part, at the request of other institutions or individuals for the purpose of scholarly research. I understand that my thesis will be made electronically available to the public.

_____ Haytham Khalil

Statement of Contribution

The work described in this thesis has been published as:

- Haytham Khalil; and Khalid Elgazzar, Phone of Things (PoT): Empowering IoT Systems Through VoIP Infrastructure and Voice Commands. In Proceedings of the 7th IEEE World Forum on the Internet of Things - WF-IoT 2021, New Orleans, Louisiana, USA, June 14 - July 31, 2021.

- Haytham Khalil; and Khalid Elgazzar, Performance Evaluation of Single-Board Embedded Linux Platforms as Asterisk Servers for Phone of Things (PoT) Applications. IWCMC 2022 IIIoT, Dubrovnik, Croatia, May 30 –June 03, 2022.

- Haytham Khalil; and Khalid Elgazzar, Location Transparency Call (LTC) System: An Intelligent Phone Dialing System Based on the Phone of Things (PoT) Architecture, Future Internet, MDPI, vol. 14(4), pages 1-22, March 2022.

- Haytham Khalil; and Khalid Elgazzar, Leveraging Blockchain for Device Registration and Authentication in tSIP-Based Phone-of-Things (PoT) Systems, IWCMC 2023 Smart & Sustainable Communications, Marrakesh, Morocco, June 19 - 23, 2023.

- Haytham Khalil; and Khalid Elgazzar, tSIP: A Lightweight SIP-Based Messaging Protocol for Resource-Constrained Embedded Devices, The 31st International Conference on Software, Telecommunications and Computer Networks (SoftCOM 2023), Split, Croatia, 21-23 September, 2023.

Acknowledgements

I want to express my heartfelt gratitude to all those who have supported and guided me throughout this incredible journey of completing my PhD thesis. Without their invaluable contributions, this accomplishment would not have been possible.

First and foremost, I extend my most profound appreciation to my supervisor, Professor Khalid Elgazzar. Your unwavering support, expert guidance, and immense knowledge have shaped my research and nurtured my academic growth. Your dedication to my development as a scholar has been truly inspiring, and I am grateful for the countless hours you have invested in mentoring and advising me.

I am also indebted to my thesis committee members, Professor Mohamed El-Darieby and Professor Akramul Azim. Your expertise, critical insights, and constructive feedback have immensely enriched my research. Your commitment to ensuring the quality and rigour of my work has been invaluable, and I am grateful for the time and effort you have dedicated to reviewing and evaluating my thesis.

I want to extend my appreciation to the faculty and staff of the University of Ontario Institute of Technology (Ontario Tech University), who have created a conducive academic environment that fosters intellectual growth and supports research endeavours. The resources, facilities, and opportunities provided by the university have been pivotal in successfully completing my doctoral studies.

I am grateful to my colleagues in the IoT Research Lab at Ontario Tech University, who have shared their knowledge, ideas, and experiences, fostering a stimulating intellectual community. Their collaboration, discussions, and support have enhanced the quality

of my research and have been a source of inspiration.

My heartfelt thanks also go to my family and friends for their unwavering love, encouragement, and belief in my abilities. Their constant support and understanding during the challenging moments of this journey have been a source of strength and motivation.

To everyone mentioned above and those who may not be specifically named but have played a part in my academic and personal growth, please accept my deepest appreciation. Your support and encouragement have been the cornerstones of my achievements. With sincere gratitude,

Haytham Khalil

List of Abbreviations

ACL	Access Control List
AGI	Application Gateway Interface
AMI	Asterisk Management Interface
API	Application Programming Interface
ASIC	Application Specific Integrated Circuits
BACnet	Building Automation and Control Network
BAS	Building Automation System
BLE	Bluetooth Low Energy
CA	Certificate Authority
CMO	Cellular Mobile Operator
CoAP	Constrained Application Protocol
CODEC	Coder-Decoder
COI	Community of Interest
CoSIP	Constrained Session Initiation Protocol
CPU	Central Processing Unit
DALI	Digital Addressable Lighting Interface
DIY	Do-It-Yourself
DoS	Denial of Service
DPoS	Delegated Proof of Stake
DSP	Digital Signal Processing
DTLS	Datagram Transport Layer Security
e2e	end-to-end
ECDSA	Elliptic Curve Digital Signature Algorithm
FXO	Foreign Exchange Office

GATT	Generic Attribute
GCP	Google Cloud Platform
GPIO	General Purpose Input/Output
GPL 2	General Public License Version 2.0
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HVAC	Heating, Ventilation, and Air Conditioning
IC	Integrated Circuit
IEEE	Institute of Electrical and Electronics Engineers
IM	Instant Messaging
IMS	IP Multimedia Subsystem
IOC	Indicator of Compromise
IoT	Internet of Things
IP-PBX	Internet Protocol Private Branch Exchange
IP	Internet Protocol
IPsec	IP Security
ISDN	Integrated Services Digital Network
ISP	Internet Service Provider
ITSP	Internet Telephony Service Provider
IVR	Interactive Voice Response
JSON	JavaScript Object Notation
LAN	Local Area Network
LON	Local Operating Network
LoRa	Long Range (a spread spectrum modulation technique)
LoRaWAN	Long Range Wide Area Network
LTC	Location Transparency Call
M2M	Machine-to-Machine

MOH	Music on Hold
MOS	Mean Opinion Score
MQTT	Message Queueing Telemetry Transport
MQTT-SN	Message Queueing Telemetry Transport for Sensor Network
NIC	Network Interface Card
NLU	Natural Language Understanding
NVS	Non-Volatile Storage
OS	Operating System
OTA	Over-the-Air
P2P	Peer-to-Peer
PBFT	Practical Byzantine Fault Tolerance
PBX	Private Branch Exchange
PCM	Pulse Code Modulation
PKI	Public Key Infrastructure
PLC	Powerline Communication
PoA	Proof of Authority
PoS	Proof of Stake
PoT	Phone of Things
POTS	Plain Old Telephone Service
PoW	Proof of Work
PSAP	Public Safety Answering Point
PSTN	Public-Switched Telephone Network
QoS	Quality of Service
RAM	Random Access Memory
REST	REpresentational State Transfer
RFID	Radio Frequency Identification
RTOS	Real-Time Operating System

RTT	Round-Trip Time
RTU	Remote Terminal Unit
S/MIME	Secure Multi-purpose Internet Mail Extensions
SBC	Single Board Computer
SCTP	Stream Control Transmission Protocol
SDN	Software-Defined Network
SHG	SIP-based Home Gateway
SIL	Single In-Line
SIP	Session Initiation Protocol
SNT	Social Network of Things
SPI	Serial Peripheral Interface
SUA	SIP User Agent
TCP	Transmission Control Protocol
TDM	Time Division Multiplexing
TI	Texas Instruments
TinyML	Tiny Machine Learning
TLS	Transport Layer Security
TPS	Transactions Per Second
tSIP	Tiny Session Initiation Protocol
UA	User Agent
UC	Unified Communication
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
USB	Universal Serial Bus
VIoT	Voice over Internet of Things
VLAN	Virtual Local Area Network
VoIP	Voice over Internet Protocol

VoLTE	Voice over Long Term Evolution
VPN	Virtual Private Network
VPS	Virtual Private Server
WDS	Wireless Distribution System
Wi-Fi	Wireless Fidelity
WLAN	Wireless Local Area Network
XML	Extensible Markup Language

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Internet of Things (IoT)

The Internet of Things (IoT) is a recent buzzword that has found its way into almost every domain, from home automation applications with simple sensors, actuators, and programmed on/off functionalities to medical, aviation, and industrial applications, featuring sensitive devices and timely critical missions [1]. Recent research statistics reveal that in 2021, there were over 10 billion connected IoT devices [2]. This number is projected to reach 41 billion by 2027, with more than 152,000 IoT devices expected to connect to the Internet every minute by 2025 [2]. IoT changes how we interact with our surroundings, including people themselves, by allowing IoT devices and wearable electronics to interactively infer from each other without human intervention. IoT systems leverage sensed information, helping people make better-informed decisions.

Since the emergence of the IoT, and as the name implies, the IoT mainly depends on the Internet to provide the ultimate connectivity between the compartments of the IoT ecosystem, namely the end users, the IoT devices, and the cloud-hosted back-end services. Internet connectivity requires things that constitute the IoT system to employ the TCP/IP protocol stack in their embedded firmware. Therefore, simple objects with

limited processing capacity or power consumption constraints that cannot implement the TCP/IP protocol stack in their embedded firmware can connect to the Internet through IoT gateways [3]. IoT gateways act as middleware that connects non-Internet-enabled devices to the Internet. However, in either case, this requires the constituent Internet-enabled IoT devices or the middleware gateways to employ medium- to large-scale embedded systems to achieve the designated task of IoT. This comes at a cost in terms of the price and the power sizing of the IoT system. Nevertheless, the consistent connectivity of devices to the Internet is the least energy-efficient piece of the whole IoT system framework. Moreover, wireless access technologies, which are dominant in IoT applications, are accounted for as the least energy-efficient access network technologies [4].

On the other hand, cybersecurity anxiety is still a significant concern that prevents the IoT from reaching its potential. With their fundamentally limited-resource embedded systems, IoT devices cannot be efficiently secured against cyberattacks. Cybersecurity represents a weak point in the overall IoT system. Hackers could exploit these intrinsically poorly secured IoT devices and pose a serious impersonation threat, allowing them to penetrate the underlying critical network infrastructure and harm its operation by reducing its availability, for example. Statistics reveal that 84% of companies adopting IoT solutions have reported security breaches related to IoT [5]. This, in turn, results in "walled gardens" of IoT systems [6]. Walled gardens imply that the utilization of sensed information from each IoT system is confined to its originators through their local or managed networks without being exposed to the Internet. This hinders the interoperability between different IoT systems and prevents the maximum utilization of sensed information.

1.2 Voice over Internet Protocol (VoIP)

Voice over Internet Protocol (VoIP) is a relatively mature technology that began in 1973 when a protocol for ARPANET was invented to enable network voice [7, 8]. However, it was not until 1995 that the first commercial VoIP solution hit the market [8]. Initially, the adaptation to VoIP technology appeared to be slow, from less than 1% of all business calls in 1998 to 25% in 2003 [9]. However, due to the advancement of digital transformation and the emergence of broadband technologies in the past decade, VoIP is currently prevailing and reaching new heights as a reliable and cost-effective means of telecommunication over the Internet. For example, there were more than one billion mobile VoIP users in 2017, with an estimated three billion mobile VoIP users in 2021 [9]. However, VoIP will soon replace the Integrated Services Digital Network (ISDN) and the Plain Old Telephone Service (POTS) as most telecom companies are planning to update their legacy telephone systems to VoIP at the end of 2025 [10].

Asterisk is open-source, Linux-based software for building feature-rich telephony applications [11]. It is a hybrid Time Division Multiplexing (TDM) and packet voice Private Branch Exchange (PBX). According to statistics published by Asterisk, there are over one million Asterisk server deployments in 170 countries, with over 2 million annual downloads [12]. A typical Asterisk system consists of different loadable modules, each responsible for specific phone functionality [13]. Asterisk empowers the creation of specialized modules to meet various telephony application needs thanks to the freedom that comes with open-source technology. In contrast to conventional PBX, the dialplan module in Asterisk is fully customizable and can be tailored to respond to external events. However, although Asterisk has its own scripting language to configure its operation, it can still be programmed using famous high-level programming languages, such as Python, Java, C++, PHP, and node.js, using the Asterisk Gateway Interface (AGI), which provides application-level control of selected features of the Asterisk system [12].

1.3 Session Initiation Protocol (SIP)

Session Initiation Protocol (SIP) is a mature signalling, presence, and instant messaging protocol used in VoIP [14]. Although an older signalling protocol exists, ITU-T H.323 [15], SIP utilization in VoIP prevails. This is due to its simpler architecture and fewer logical components [14]. SIP is used to establish and tear down media sessions between calling parties in the VoIP ecosystem. The main signalling functions of SIP include locating the called parties, querying their willingness to establish sessions with the calling parties, exchanging the information required to establish the sessions, and tearing down the established sessions when the communicating parties hang up [16]. SIP can run over many transport protocols, such as UDP, TCP, or SCTP. Moreover, it can run over secure transport protocols like TLS and DTLS [14]. However, UDP is commonly used in VoIP environments because it is a lightweight transport protocol with a cheap connection setup that fits VoIP applications' soft real-time constraints over statistical multiplexing mediums like the Internet. SIP is an HTTP-inspired protocol that inherits the same text-based message format as the HTTP protocol [17]. SIP's text-based messages are large in size, and it takes a lot of processing power (CPU and RAM) to construct and parse SIP messages [17], which comes at the cost of complex implementation in a constrained environment like IoT gadgets.

1.4 Blockchain

Blockchain is a revolutionary technology that has defined a secure and tamper-resistant paradigm for information exchange in a decentralized consensus [18]. Satoshi Nakamoto [19] firstly proposes blockchain as a purely peer-to-peer electronic cash payment system without the default crucial need for a financial institution in the middle for authentication and authorization purposes. However, blockchain's "distributed ledger" feature can be

adapted to any data exchange system, extending the horizon of blockchain applications. Therefore, while the first generation of blockchain focuses on financial transactions, the second generation has a much more comprehensive range of applications. Nevertheless, with the emergence of smart contracts [20], which are predefined scripts executed on each transaction, a new dimension of blockchain applications has emerged that builds on top of the distributed nature of blockchain to build decentralized incentive applications.

1.5 Problem Statement and Challenges

The thesis proposes the Phone of Things (PoT), an innovative framework that bridges the gap between IoT and VoIP technologies. PoT offers affordable, secure, and scalable solutions for designing IoT systems. By leveraging the widely available phone network infrastructure and assets, PoT seamlessly integrates them into the IoT architecture, extending gadgets' accessibility through pervasive phone networks. This approach enhances user engagement with surrounding devices. It capitalizes on the mature abstractions of phone technologies, catering to individuals who may need to be more technologically savvy or are more accustomed to pre-internet times. PoT's context-aware telephony solutions leverage the synergy between IoT and VoIP, empowering businesses to create innovative telephony applications that re-imagine mature phone features for modern times, leading to increased productivity and an improved user experience during phone calls. To delve into the specifics of the PoT framework, this section addresses Heilmeier's catechism [20], elucidating PoT's objectives, identifying shortcomings in current practices, and exploring the potential community of interest (COI) for PoT.

1.5.1 PoT Grand Vision

Despite the increasing emergence of IoT devices and many companies' widespread adoption of VoIP, these technologies are often handled separately. PoT aims to bridge the

gap between IoT and VoIP, forging a meaningful relationship between them. Currently, numerous companies either stick to their legacy telephony systems or migrate to VoIP, but in both cases, they primarily use these systems for conventional voice communication between parties. However, the integration of IoT presents exciting opportunities for VoIP users. VoIP can become an integral component of the IoT architecture, with existing Unified Communication (UC) solutions acting as central hubs for monitoring and controlling devices within the premises. Consequently, devices associated with PoT can be easily commanded and monitored using ubiquitous phone network infrastructure (such as VoIP and PSTN) and assets (such as POTS, IP phones, and softphones). Even without internet access, a simple phone call with intuitive voice commands can be used to interact with these devices effectively.

This thesis proposes tSIP, a streamlined version of the SIP protocol designed for resource-constrained smart objects. With its tiny footprint, tSIP can be deployed on limited-resource devices effectively. tSIP messages can be mapped to the original SIP messages using a proxy (i.e., PoT gateway). tSIP fosters PoT adoption and establishes a standardized, lightweight communication protocol between low-resource embedded devices and the existing Unified Communications (UC) solution within the premises. As a result, these devices can function seamlessly as typical SIP endpoints in the VoIP ecosystem, allowing for easier control and monitoring.

Additionally, the thesis proposes a lightweight decentralized registration and authentication mechanism based on the blockchain technology for smart gadgets in the PoT system. The proposed mechanism facilitates secure associations between the devices and the communication server without relying on high-end communication servers or placing trust in third-party entities. By leveraging the Ethereum blockchain and smart contracts, the proposed mechanism implements a programmatic, immutable access control mechanism. This ensures a secure, trustless, and scalable environment for PoT without disrupting the existing SIP-based VoIP architecture.

The thesis implements the Location Transparency Call (LTC) system as a use-case application of PoT, utilizing the PoT framework. LTC aims to enhance telephony solutions with a context-aware approach, leveraging IoT and VoIP technologies to create innovative telephony applications. By harnessing data inferred from the RFID-enabled door entry nodes within the premises, LTC effectively addresses the issue of missed business calls by automatically forwarding incoming calls based on the gathered information regarding the current location of the called party within the premises. This optimization reduces notoriously long call waiting times, increasing customer satisfaction.

1.5.2 Lacking in the Current Practices

Although IoT is increasingly emerging, it has yet to reach its potential promise. The following impede IoT from reaching its potential commitment:

1.5.2.1 Fragmented Ecosystems

The IoT landscape is characterized by numerous proprietary ecosystems, each with its own set of compatible devices, protocols, and frameworks. This is sometimes referred to as the "*Intranet of Things*" or "*vertical silos*" [21, 22]. This fragmentation of ecosystems poses challenges for interoperability, as devices from one ecosystem may not seamlessly integrate or communicate with devices from another [23]. It creates a dilemma for consumers, businesses, and developers who must navigate these different ecosystems, making it challenging to create cohesive IoT solutions or scale deployments across multiple platforms [23]. This lack of standardization can lead to vendor lock-in, limiting choice, and hindering innovation. Efforts are underway to address this dilemma through initiatives promoting interoperability and common standards, such as developing open-source frameworks and industry collaborations. Finding solutions to bridge the gap between different IoT ecosystems is crucial to unlocking the full potential of the IoT and enabling a more connected and unified digital future. The ultimate purpose of the proposed PoT

system is to eventually provide a modular, multi-tier, VoIP-enabled, and Do-It-Yourself (DIY) ecosystem for IoT system designers. The proposed framework utilizes open-source technologies and could arbitrarily encompass third-party cloud-hosted services according to specific application needs. This, in turn, allows incentive application concentration, architecture abstraction, and faster time to market to IoT system designers.

1.5.2.2 Walled-Off Data

Walled-off data in the context of the Internet of Things (IoT) refers to the practice of isolating and restricting access to data generated by IoT devices within specific boundaries or ecosystems, typically referred to as "*walled gardens*" [6]. Walled-off data involves implementing measures to control the flow and availability of data, typically within managed networks or proprietary platforms. Walled-off data addresses concerns regarding privacy, security, and ownership of IoT-generated information. By confining data within predefined boundaries, organizations or individuals can maintain tighter control over their data, allowing them to enforce stricter data governance policies and protect sensitive information from unauthorized access. However, the concept of walled-off data also raises questions about data interoperability, collaboration, and the potential for data silos, which may limit innovation and hinder the realization of the full potential of the IoT. Striking a balance between data protection and sharing remains crucial as the IoT continues evolving and transforming various industries.

In this context, PoT facilitates data exchange and sharing by promoting ubiquitous phone network infrastructure as a bridge between devices from various ecosystems. For example, VoIP technologies can extend access to sensed information in multi-site business domains through the established SIP trunks between the sites. Like cloud computing, which can provide shared data space across the Internet, PoT could provide shared data space between devices that are confined to the phone network. This, in turn, helps break down the barriers of walled-off data for businesses that are skeptical about their data

privacy and security and prefer to be air-gapped from the Internet.

1.5.2.3 Device Loads and Bandwidth

PoT leverages the VoIP network infrastructure within the premises to provide reliable and secure access to the surrounding smart gadgets within the premises. VoIP and data traffics are typically deployed on different subnets or VLANs (Virtual Local Area Networks) [24]. This ensures optimal performance, enhances security, and simplifies network management, making it a crucial practice in maintaining a reliable and efficient network infrastructure. By isolating VoIP traffic from data traffic, we prevent data-intensive operations from impeding the real-time requirements of voice calls. Additionally, segregating VoIP and data traffic helps prioritize voice packets, reducing the chances of jitter, packet loss, or latency issues that can degrade the call quality. Therefore, using VoIP networks in PoT improves the proposed framework's quality of service (QoS), ensuring a smoother and more reliable communication experience and facilitating better bandwidth allocation management as the PoT system scales and more devices are added.

On the other hand, IoT system designers are usually in a trade-off between implemented device capabilities on the one hand and the expected cost, power consumption, and bandwidth requirements on the other hand. In a later development stage of the proposed PoT framework, The PoT system could implement an autonomous gateway based on TinyML [25], enabling the embedding of tiny machine learning algorithms into the gateway itself. TinyML enables local inference on the PoT gateway, eliminating the need for constant data transmission to the cloud for processing. Performing ML computations on the PoT gateway reduces the dependency on high-bandwidth Internet connections and cloud computing resources, allowing the system to be fully functional through the phone network without needing Internet access or degrading the VoIP traffic. This reduces the deployment cost of PoT by eliminating the costs associated with maintaining and scaling the cloud infrastructure. In addition, it mitigates latency, network connectivity, and

privacy concerns, as data remains on the gateway, reducing the need for transmitting sensitive data to external servers.

1.5.2.4 Cloud Attacks

Cloud attacks pose a significant challenge to IoT security [26]. Given the cloud's capacity to handle vast amounts of IoT data, cloud providers have become prime targets for cyberattacks. This puts system designers in a challenging position, balancing the benefits and convenience of cloud computing for IoT systems with concerns about potential critical data leaks in the event of successful cyberattacks.

To address this issue, the proposed PoT framework leverages VoIP networks to counter cloud attacks. Using VoIP technologies for communication with smart gadgets, sensitive data remains within a local, segregated VoIP network, reducing dependence on cloud-based services and minimizing the attack surface. Also, in the proposed PoT framework, secured tunnels are implemented between distant gateways using VPN (Virtual Private Network) technologies [27], creating a more secure device environment and limiting data exposure to potential cloud-based attacks.

Moreover, VoIP technologies inherently incorporate encryption protocols to safeguard VoIP traffic during transmission, ensuring the confidentiality and integrity of sensitive information. By leveraging VoIP technologies, we strengthen data privacy and bolster the overall security of PoT deployments. Additionally, the proposed blockchain-based registration and authentication mechanism for smart gadgets in the proposed framework provides a robust and secure association between limited-resource gadgets and the gateway, mitigating the risk of impersonation threats on the associated devices.

1.5.2.5 Botnet Attacks

IoT gadgets present a broad surface for hackers to exploit their inherent vulnerabilities, jeopardizing the security of the underlying critical network infrastructure. PoT introduces

a novel approach to IoT system connectivity to mitigate this risk by leveraging the ubiquitous telephone network infrastructure and phone technologies.

In traditional IoT setups, connectivity between components heavily relies on the Internet and the dynamic IP addresses assigned by ISPs (Internet Service Providers). The problem with these dynamic IP addresses is that they lack uniqueness, making it challenging to identify and secure individual users when connecting to the Internet. This issue opens the door to cybersecurity attacks, as these IP addresses can be easily masqueraded or spoofed using freely available technologies.

In contrast, phone numbers offer an inherent global uniqueness that cannot be easily replicated. This characteristic provides an excellent opportunity for IoT system administrators to implement fine-grained access control based on the calling party's phone number, bolstering security. Unlike IP addresses, conceiving a calling phone number is practically impossible, further enhancing the system's protection.

Moreover, as the proposed PoT system matures, the gateway could implement dynamic VLAN assignments for associated devices, effectively isolating them from the critical network infrastructure. This isolation helps mitigate the impact of botnet attacks when devices are managed through the Internet.

Therefore, by utilizing the telephone network infrastructure and implementing robust access controls based on phone numbers, PoT offers a more secure and reliable connectivity solution for IoT systems, safeguarding against cyber threats and enhancing overall system resilience.

1.6 Research Questions

This section sheds light on the research questions regarding the intricate challenges and opportunities the proposed framework tries to address and achieve.

- How can IoT and VOIP technologies be effectively integrated?

- What are the technical challenges and considerations in leveraging the ubiquitous phone network infrastructure to extend the accessibility options of IoT systems?
- How can the assets of the phone network infrastructure be optimally utilized to enhance the functionality and efficiency of IoT devices and applications?
- What security and privacy concerns that need to be addressed when bridging IoT and VOIP technologies together?
- What communication protocols can be utilized in such a framework and what are the requirements for efficient communication?

1.7 Research Objectives

- Develop a Phone of Things (PoT) framework to integrate IoT and VoIP technologies.
- Investigate the adaptation of a tiny version of the SIP (Session Initiation Protocol) protocol to enable seamless integration between IoT and VOIP technologies within the Phone of Things (PoT) framework.
- Develop methods for unifying message exchange between embedded IoT sensors and established communication servers to enable meaningful interactions and communication between the two domains.
- Explore strategies to ensure interoperability between diverse IoT and VOIP components, considering the heterogenity of IoT ecosystems and devices.
- Evaluate encryption methods and techniques for securing data transmission within the PoT framework.
- Develop mechanisms for verifying the identity of IoT devices and users and controlling access to sensitive resources to ensure security and privacy.

- Design user-friendly interfaces to enable PoT users to monitor and control IoT devices seamlessly through phone technology and infrastructure.
- Explore practical use cases and applications where the PoT framework can provide significant value, with a focus on smart homes, industries, and businesses.
- Investigate the scalability of the PoT framework to accommodate a growing number of IoT devices and users.

1.8 Primary Beneficiaries

This subsection reviews the potential community of beneficiaries of the proposed PoT system:

1.8.0.1 Telecommunication Companies (PSTN & ITSP)

The first community of interest (COI) for the proposed PoT framework in the thesis would be the telecommunication companies, such as the public switched telephone network (PSTN) companies and the Internet Telephony Service Providers (ITSPs). Telecommunication companies can reap several benefits from integrating IoT and VoIP technologies. IoT integration allows telecom companies to expand their service offerings by providing connectivity solutions for a wide range of IoT devices. By leveraging their existing infrastructure, telecom companies can offer specialized IoT connectivity plans, device management services, and value-added solutions tailored to specific IoT use cases. For example, a call-barring service can help limit incoming and outgoing calls, which acts as an access control list to the proposed PoT system and hardens its security. This integration opens up new revenue streams and enables telecom companies to tap into the growing IoT market.

Furthermore, IoT and VoIP integration can improve telecom companies' operational efficiency. By utilizing IoT devices and sensors, telecom companies can monitor and

manage their network infrastructure more effectively. Real-time data collected from IoT devices can help identify network issues, optimize network performance, and proactively address potential outages.

Moreover, integrating IoT and VoIP could facilitate data-driven insights for telecom companies. The vast amount of data generated by IoT devices and VoIP systems can be leveraged for analytics and business intelligence purposes. Telecom companies can gain valuable insights into customer behaviour, network usage patterns, and service quality, enabling them to make data-driven decisions, improve customer experience, and optimize their service offerings.

1.8.0.2 Business and Industry

The following COI for PoT would be the business and industry domains. By utilizing PoT, companies can leverage their existing UC solutions within the premises beyond just making and receiving calls. The office telephone sets can be used as control hubs to command and monitor all appliances within the premises. Besides, PoT, empowered chatbot integration, can be used to build intelligent and dynamic Interactive Voice Response (IVR) systems for enterprises. Therefore, instead of building the conventional static IVR menus, which could not adapt to changes in the enterprise, we can exploit the PoT as an integral part of the existing UC to extract useful facts from what the caller says and forward the call accordingly to the suitable destination. This reduces the notorious long call queueing time and improves customer satisfaction.

The same concept can also be used to automatically query information from the suitable device(s) associated with the PoT gateway to answer users' questions, which will benefit public sensing and industrial applications. PoT can also be involved in incentive telephony applications for businesses that leverage the integration between IoT and VoIP to build context-aware, innovative telephony solutions that increase business productivity and availability. In Chapter 5, the thesis proposes the location transparency call (LTC)

system for enterprises as a use-case application of PoT. The LTC system intelligently mitigates the impact of missed business calls. It provides high availability and dynamic reachability to employees in the workspace based on the information inferred from the surrounding gadgets.

Third-party platforms, like Twilio, have been utilized to provide phone interface capabilities to businesses and applications [28]. These platforms are interfaced through APIs (Application Programming Interfaces) to build customer interactions on their preferred channels. However, third-party platforms need more flexibility to adapt to the dynamic requirements of applications. Nevertheless, it incurs an extra cost that hinders its widespread usage. PoT overcomes the drawbacks of third-party services and provides an interface to the existing phone network infrastructure within the premises of enterprises at no additional cost. PoT enables adaptive phone integration that fits the needs of different applications. Therefore, rather than relaying customer requests to a static inbound phone number of the business, which is the case with third-party platforms, PoT can be programmatically finely tuned to dynamically route calls to arbitrarily existing extensions within premises based on predefined criteria. This, combined with the potential of TinyML embedding in the PoT gateway, can be used in industry for automatic sensor readings in the field, inference of sensed information, auto-dialling predefined extensions in case of irregularities or emergencies, and more.

1.8.0.3 Home Residents

The following COI would be the home residents. There are enormous generations of people whose recollections originate before the Internet. For these people, the phone is, as yet, a valuable innovation. Therefore, if one wishes to do business with those people, one had better do an excellent job that can be handled using telephone calls.

The IoT systems are still expensive and are not widely spread in homes, not to mention the hassle of potential infrastructure modifications and device changes. This, in turn,

makes the home residents reluctant to install IoT systems due to their deployment cost. PoT provides a Do-It-Yourself (DIY) IoT framework based on open-source technologies, and it exploits the use of the existing and rarely-used telephone network infrastructure in homes. At its simplest form, home residents would plug the PoT gateway into the existing telephone landline at homes to control the associated IoT devices to the gateway through a simple phone call and intuitive spoken commands.

Besides, the PoT can be configured to automatically initiate phone calls to predefined destinations in case of emergencies or reading thresholds from the associated devices. Also, the PoT gateway could integrate a powerline communication (PLC) modem in a later stage of system development. This allows controlling dozens of simple and legacy devices at home, which are still in great use today, such as light bulbs and power strips. The PoT, with its embedded chatbot capability, provides an intuitive, natural conversational interface to home residents who do not need to be technically savvy by default. The PoT incurs no running cost if confined to the local or managed network and is more secure when managed through the phone network.

1.9 Thesis Contributions

The main contributions of the thesis are summarized as follows:

- We propose a boilerplate framework for the Phone of Things (PoT) based on open-source technologies. We delineate the framework's components and highlight the motivation behind choosing the enabling technologies they utilize. Also, we propose future directions for PoT development.

- We conduct a feasibility study of embedded Linux platforms acting as PoT gateways, promoting the utilization of tiny and cost-effective embedded Linux boards for PoT applications in homes and small-to-medium-sized business domains.

- We propose a tiny version of the Session Initiation Protocol (SIP), which we call tSIP. tSIP is a lightweight version of the SIP protocol deployable on limited-resource smart gadgets. tSIP messages can be mapped to their original SIP messages with the help of a proxy, i.e., a PoT gateway. tSIP allows smart gadgets to act as typical SIP endpoints in the VoIP ecosystem and facilitates their management through the communication servers.

- We propose a lightweight and blockchain-based registration and authentication mechanism between smart gadgets, the PoT gateway, and the communication server. The proposed mechanism provides a secure, trustless, and scalable environment for PoT without requiring high-end communication servers, affecting the existing SIP-based VoIP architecture, or mandating trust in third-party entities.

- As a use-case application of PoT, we propose the location transparency call (LTC) system. LTC is a context-aware telephony solution for businesses that leverages the information inferred from the surrounding gadgets to dynamically forward calls to the employees to the nearest extension at their current location, mitigating the effect of missed business calls.

1.10 Thesis Outline

- **Chapter 1** introduces the research topic, discussing the enabling technologies that form the basis of the study. It presents the problem statement and outlines the unique contribution of the thesis.

- **Chapter 2** comprehensively reviews the state-of-the-art, focusing on integrating IoT and VoIP, the performance of embedded Linux platforms as IP-PBX servers, the usage of SIP protocol in IoT applications and device deployments, and the application of blockchain in VoIP registration and authentication.

- **Chapter 3** introduces the proposed Phone of Things (PoT) framework in detail, comprehensively describing its implementation. The chapter provides deep insights into the proposed architecture.

- **Chapter 4** delves into tSIP, the proposed lightweight version of the SIP protocol. It also presents the blockchain-based registration and authentication mechanism explicitly designed for PoT.

- **Chapter 5** explores various use case scenarios of the proposed PoT framework, demonstrating its versatility and potential applications in different domains.

- **Chapter 6** presents and discusses the experimental results and performance evaluation of the proposed PoT system.

- **Chapter 7** offers concluding remarks, summarizing the key findings and contributions of the thesis. It also discusses potential future directions for further research and advancements of the proposed PoT framework in the thesis.

Chapter 2

Background and Literature Review

The thesis proposes the Phone of Things (PoT), a boilerplate framework for IoT. The poT represents a novel paradigm that leverages the ubiquitous presence of phone network infrastructure (i.e., PSTN and VoIP) and assets (i.e., communication servers, phone sets, softphones, etc.) as controllers for IoT devices. With VoIP's increasing popularity in all buildings, PoT represents a promising approach to extending the functionality and connectivity of IoT systems. Integrating VoIP with IoT devices enables enhanced user experiences, seamless control, and access to various services and applications. The literature review in this chapter explores the enabling technologies behind the PoT, focusing on key components of the proposed PoT framework, such as open-source technologies, messaging protocols, and the blockchain. By examining the current state of research and development in this area, this review aims to provide insights into the advancements, challenges, and potential applications of PoT. Understanding the underlying technologies is crucial for harnessing the full potential of PoT and driving the evolution of smart and interconnected IoT ecosystems. Lastly, Section 2.8 provides a summary of the chapter.

2.1 Open-Source IoT Frameworks

2.1.1 Background

The importance of IoT frameworks based on open-source technologies cannot be overstated. Open-source frameworks provide several advantages that contribute to the success and widespread adoption of IoT applications:

- Open-source frameworks foster collaboration and innovation among developers and the broader community. By providing access to the source code and allowing contributions from a diverse range of developers, these frameworks encourage sharing knowledge, best practices, and the development of new features and functionalities. This collaborative environment leads to faster iterations, improved reliability, and increased security through community-driven reviews and contributions.

- Open-source frameworks offer flexibility and customization options. Developers can modify and extend the frameworks to suit their specific needs, enabling the creation of tailored solutions. This adaptability allows for seamless integration with existing infrastructure and the ability to address the unique requirements of different IoT applications.

- Open-source frameworks often have active communities that provide support, documentation, and continuous improvements. This ecosystem of contributors ensures ongoing development, maintenance, and updates, enhancing the longevity and stability of IoT solutions.

- Open-source frameworks promote vendor neutrality and prevent vendor lock-in. Organizations can freely choose from various hardware and software components by avoiding proprietary solutions, fostering competition and reducing costs. This

freedom of choice and interoperability encourage a diverse IoT ecosystem where devices, applications, and platforms can seamlessly communicate and collaborate.

In conclusion, open-source IoT frameworks empower developers, enable customization, foster innovation, and promote interoperability, making them vital for the growth and success of IoT applications across various industries. The following represents some popular open-source IoT frameworks in the literature.

- **Eclipse IoT** [29]: Eclipse IoT is a set of open-source projects that provide a platform for building IoT applications. According to [30], the Eclipse IoT Working Group (WG) at the Eclipse Foundation is the biggest open-source community. Eclipse IoT includes projects like *Eclipse Paho* (for MQTT messaging), *Eclipse SmartHome* (for building smart home applications), *Eclipse Kura* (for building IoT gateways), and *Eclipse Californium* (for CoAP protocol support).

- **ThingsBoard** [31]: ThingsBoard is an open-source IoT platform that enables rapid development, management, and scaling of IoT applications. It offers device management, data collection, processing, and visualization capabilities.

- **OpenHAB** [32]: OpenHAB is a vendor and technology-agnostic open-source automation software for the home. It provides a framework and runtime for building smart home solutions. It supports various devices and technologies and offers features like rules engines, user interfaces, and integrations with other systems.

- **Home Assistant** [33]: Home Assistant is an open-source home automation platform that focuses on privacy and local control. It runs on various systems, including Raspberry Pi, and supports a wide range of devices and integrations.

- **Kaa** [34]: Kaa IoT is an open-source IoT platform for building, managing, and integrating connected products. It provides device management, data collection, analytics, and visualization features.

- **The Things Network (TTN)** [35]: The Things Network is an open-source and decentralized IoT network infrastructure that allows devices to connect and communicate with applications. It uses LoRaWAN (Long Range Wide Area Network) technology for long-range, low-power wireless communication.

- **Node-RED** [36]: Node-RED is an open-source visual programming tool for wiring together hardware devices, APIs, and online services. It provides a browser-based flow editor and a wide range of nodes that can be connected to create IoT applications.

2.1.2 Shortcomings

While open-source IoT frameworks offer numerous benefits, they come with their potential shortcomings. Here are some common shortcomings associated with these frameworks:

- **Complexity**: IoT frameworks can have a steep learning curve, especially for users who are new to IoT development. The complexity arises from the need to understand the framework's architecture, configuration, and integration with various hardware and software components. In the proposed PoT framework, we enable IoT devices to act as typical phone endpoints in the phone ecosystem. The ease of phone technology is undeniable, thanks to its maturity and widespread familiarity among people. The mature abstractions of phone technology would make PoT accessible and easy to use for individuals of all ages and technical backgrounds. Moreover, the widespread adoption of softphones has led to high comfort and familiarity among users, making them a natural choice for extending the reach of IoT applications through the Phone of Things (PoT) paradigm. Leveraging the mature phone technology in the context of PoT reduces users' learning curve and encourages seamless integration with existing IoT ecosystems. As a result, people can

effortlessly control and manage a diverse range of IoT devices from their standard phone sets at home and office or softphones on their mobile phones, contributing to a more connected, efficient, and user-centric IoT landscape.

- **Limited Hardware Support**: Some open-source IoT frameworks may have limited support for specific hardware devices or protocols. This can pose challenges when integrating devices that are not officially supported or require additional customization. In this context, PoT provides native support for IoT devices with the phone system. This is done by providing a tiny version of the SIP protocol called tSIP. tSIP is a lightweight messaging protocol based on the SIP protocol deployable on constrained IoT devices. With the help of a proxy, tSIP messages can be mapped to their original SIP counterparts. tSIP fosters greater interoperability, scalability, and energy efficiency, facilitating the seamless integration and operation of diverse devices through the proposed framework.

- **Scalability**: Scalability can be a concern when using specific open-source IoT frameworks. Depending on the design and architecture, some frameworks may struggle to handle a large number of devices or high data throughput, leading to performance issues. While integrating into the existing phone network infrastructure, the PoT framework offloads the communication servers from the potentially excessive resources needed to handle the registration and authentication of the surrounding devices. This is done by using blockchain technology to provide a decentralized registration and authentication mechanism for associated devices in the PoT framework. The proposed mechanism delegates the burden of device registration and authentication to PoT gateways, promoting PoT scalability while not affecting the existing VoIP deployments within the premises. Alongside, PoT allows enterprises to scale over their existing phone trunks, which can be easily provisioned, providing flexibility to adapt to changing call requirements as the PoT

system scales.

- **Security Considerations**: Security is critical to IoT applications. While open-source frameworks often have security features, the responsibility of implementing secure practices falls on the developer. Inadequate security measures can expose vulnerabilities and put the entire system at risk. In the proposed PoT framework, employing blockchain for device registration and authentication provides immutability and ensures the integrity of device registrations to the communication server. On the other hand, Phone numbers are generally considered more secure than IP addresses due to several factors. Phone numbers are associated with physical devices and are typically tied to specific individuals or organizations and regulated by telecommunications providers. This physical association adds a layer of accountability and traceability to communication. It also allows for building a finely-grained access control list (ACL) for PoT systems that can not be easily breached. However, IP addresses can be easily masked, hidden, or spoofed, making it challenging to reliably identify a communication's true origin or ownership.

2.2 The Evolution of Business Communication Systems

In the ever-evolving business communication landscape, PBX (Private Branch Exchange), Hybrid PBX, and IP-PBX systems have played pivotal roles in enabling efficient, reliable, future-proof communication solutions in the corporate world. The following subsections review their main characteristics.

2.2.1 Private Branch Exchange (PBX)

Traditionally, PBX systems are TDM (Time Division Multiplexing) and hardware-based communication systems. PBX systems rely on physical connections using copper wires [37]. They allow organizations to manage internal calls, transfer calls, and access essential telephony features within their premises. With the utilization of limited trunk lines, referred to as FXO (Foreign Exchange Office) in the telecommunication engineering parlance, PBX systems provide the outbound dialling capability to the Public-Switched Telephone Network (PSTN) or the Cellular Mobile Operators (CMO). Enterprises use PBX to reduce their calling expenditures since having a distinct phone line for each extension incurs high costs to their budget. However, PBX systems are an obsolete technology nowadays. They are vendor-specific hardware systems with various modules installed in cabinets to achieve the required functionalities, captivating their maintainability and scalability [38]. Nevertheless, they mandate dedicated phone network wiring within the premises, complicating their installation and increasing their deployment cost [38].

2.2.2 Hybrid PBX

A Hybrid PBX combines traditional TDM PBX and IP-based technologies, i.e., SIP [14] and H.323 [38]. It combines both analog and digital communication methods, allowing organizations to leverage existing infrastructure while gradually transitioning to more advanced IP telephony. Hybrid PBX systems typically support a mix of analog, digital, and IP-based endpoints. They offer flexibility by accommodating both traditional analog devices, such as analog phones and fax machines, and IP-based devices, like VoIP phones and softphones. Hybrid PBX systems allow organizations to upgrade their telephony infrastructure at their own pace and take advantage of IP-based features and cost savings while still utilizing their existing analog equipment. However, Hybrid PBXs are vendor-locked-in proprietary systems, increasing their maintenance and service costs [38].

Nonetheless, Hybrid PBX systems are not interoperable, impeding their scalability [38].

2.2.3 IP-PBX

An IP-PBX, or Internet Protocol Private Branch Exchange, is a telephony system that operates entirely on IP-based networks, such as local area networks (LANs) or the Internet [39]. IP PBX systems use VoIP technology to transmit voice calls over IP networks, converting voice into digital packets for transmission. IP-PBX offers numerous advantages over traditional PBX systems, including cost savings, scalability, flexibility, and integration with other IP-based applications. IP-PBX systems support a wide range of features, such as call routing, call recording, voicemail, video conferencing, and unified communications (UCs). They also allow remote users to connect to the system using secure VPN (Virtual Private Network) connections, enabling seamless communication across different locations. IP-PBX systems are highly adaptable and can integrate with various IP devices, making them suitable for modern communication needs.

2.3 Asterisk: The Open-Source PBX Framework

2.3.1 Overview

Asterisk is a powerful open-source PBX software that has gained significant popularity in the realm of telecommunications [15, 40]. Developed by Digium (now part of Sangoma Technologies), Asterisk offers a flexible and feature-rich platform for building customized communication systems. As an open-source PBX, Asterisk allows businesses to take advantage of cost savings and customization opportunities. It supports a wide range of telephony protocols, including VoIP, ISDN (Integrated Services Digital Network) [41], and traditional analog telephony. With Asterisk, users can create sophisticated call routing systems, interactive voice response (IVR) menus, call queuing, voicemail, and

more. Asterisk's modular architecture and extensive library of add-on modules enable users to tailor their PBX solution to their specific requirements. Asterisk's open-source nature fosters a vibrant community of developers contributing to its ongoing development, ensuring its continuous improvement and adaptability to evolving communication needs.

2.3.2 Leveraging Asterisk IP-PBX in IoT Applications

To the best of our knowledge, Phone of Things (PoT) is the first research work in the literature that proposes a general-purpose, VoIP-enabled, and multi-tier IoT framework that exploits the integration between phone and IoT technologies. Former researchers employ limited phone functionalities as the third party to assist their proposed IoT system, targeting specific applications. At this point, the literature contains research focusing on limited VoIP functionalities employed by some of the proposed IoT systems.

To that extent, the author of [42] proposes IoT and VoIP integration. He uses similar underlying hardware to that used in the proposed PoT framework. However, the system lacks real integration with VoIP. The author utilizes the built-in dialplan functions to scan for user input via the keypad and invokes a script that controls the I/O pins of the embedded platform accordingly. Besides, the author does not provide for system security and scalability. The authors of [43] propose a SIP-based IoT gateway that automatically calls healthcare providers in case of medical emergencies based on outlier readings of biomedical sensors attached to a human. Here, the IoT gateway does not employ a SIP server itself. However, it relies on an external SIP client responsible for initiating an emergency call to the appropriate healthcare destination, limiting its capability. It is an application-specific integration between IoT and VoIP. Nevertheless, since no VoIP server is incorporated within the IoT gateway, it is stripped of most VoIP functionalities. In [44], the authors propose an intelligent water heating system for buildings based on IoT technology that is activated through the use of IVR on a cloud-hosted Asterisk-based PBX. Again, the proposed system in the latter employs limited functionalities of VoIP

connectivity. Besides, it neither provides for system security nor scalability. In [45], the authors mediate an external Asterisk PBX server as a bridge between the user and an ESP32-based embedded device to control its pins using textual command inputs issued using a SIP client. Similar to the latter, the authors in [46] provide a similar approach to supervising home appliances by employing a dedicated Asterisk server and using WLAN, which confines system usage to the local home network. Another application of VoIP integration with an IoT system is introduced in [47]. The authors in the latter build an intelligent postbox system that auto-notifies the user by phone call when a shipment arrives. Besides, the system allows users to inspect and open the postbox remotely. However, there is no native support of VoIP since the system proxies VoIP functionalities to a third-party Asterisk server.

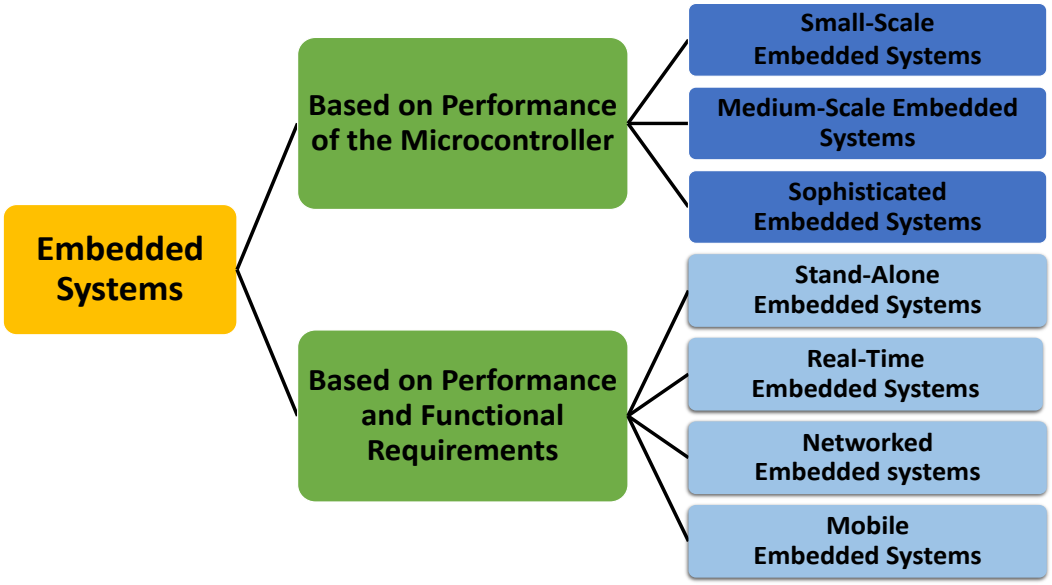

Figure 2.1: Embedded systems classification.

2.4 IoT Devices

IoT devices come in various forms, sizes, and complexities, from simple devices utilizing small-scale embedded systems to more complex devices empowered by medium-to-large

scale embedded systems and Single Board Computers (SBCs) [48]. Generally, embedded systems are classified based on either the embedded controller's performance or the system's functional requirement [49], as depicted in Figure 2.1. Based on the performance of the utilized controller, embedded systems are furtherly divided into three categories:

- **Small-scale embedded systems**: Simple embedded systems are characterized by their limited computational capabilities and dedicated functionality. They often consist of 8-bit microcontrollers or low-power processors, minimal memory, and specific sensors or actuators to perform a singular task efficiently. These devices are typically cost-effective, energy-efficient, and designed for resource-constrained environments. IoT systems typically utilize this category of embedded systems for sensor nodes with limited functionalities. The simple functions of multiple sensor nodes typically coalesce together through the use of an IoT gateway to achieve the designated complex task of the IoT system [50].

- **Medium-scale embedded systems**: This category typically employs 16-bit or 32-bit microcontrollers. As a result, they are more powerful than small-scale embedded systems. However, the integration between the underlying hardware of this category and the software to control it is more complex. High-level programming language tools, like compilers, debuggers, simulators, etc., are available to design applications using medium-scale embedded systems. Nevertheless, this category of embedded systems can employ operating systems (OS), mostly Linux-kernel-based OS, which furtherly ease the process of application development using this category of embedded platforms by masking the complexity of the underlying hardware interfacing through providing device drivers, file systems, network connectivity stacks, etc. which are available through easy-to-use libraries and/or graphical user interfaces (GUIs). This embedded system category includes many embedded Linux platforms like Raspberry Pi [51], Beaglebone [52], etc. They are usually used for

processing-intensive IoT applications or as gateways for simpler sensor nodes that constitute the IoT system.

- **sophisticated embedded systems**: This category of embedded systems is designed using one or multiple 32-bit or 64-bit microcontrollers. These systems possess very high hardware and software complexities. They are used for highly complex and processing-intensive functions that require hard real-time processing, like medical and military applications. They typically employ Real-Time Operating Systems (RTOS), which provide scheduling guarantees to ensure deterministic behaviour and timely response [53]. Essentially, this category of embedded systems outperforms the requirements of typical IoT applications with almost no-to-soft real-time constraints requirements.

The common characteristics across IoT devices include connectivity, enabling communication and data exchange with other devices and systems, and the ability to sense and collect data from the environment using sensors. IoT devices often have power efficiency considerations to ensure extended battery life or optimal power usage, mandating efficient, preferably event-triggered, messaging protocols. Security is another critical aspect, as IoT devices must protect sensitive data and resist potential threats. Scalability and interoperability are vital to accommodate expanding and integrating a diverse range of IoT devices and platforms.

Overall, IoT devices span a spectrum of complexity, from simple embedded devices performing dedicated functions to complex systems utilizing SBCs. Each category serves specific use cases and offers varying levels of computational power, connectivity, and customization options to fulfill the diverse requirements of IoT applications.

Table 2.1: Raspberry Pi Boards Comparison Matrix.

Model	RPi Zero	RPi Zero 2 W	RPi 3 B+	RPi 4 B
SOC Type	Broadcom BCM2835	Broadcom BCM2710	Broadcom BCM2837B0	Broadcom BCM2711
CPU Clock	1 × ARM 1176JZF-S, 1.0 GHz	4 × ARM Cortex-A53, 1.0 GHz	4 × Arm Cortex-A53, 1.4 GHz	4 × Arm Cortex-A72, 1.5 GHz
RAM	512 MB	512 MB	1 GB	1 GB/2 GB/4 GB
GPU	Broadcom VideoCore IV	Broadcom VideoCore IV	Broadcom VideoCore IV	Broadcom VideoCore VI
USB Ports	1	1	4	4 (2 × USB 3.0 + 2 × USB 2.0)
Ethernet	No	No	Gigabit Ethernet (max. 300 Mbps)	Gigabit Ethernet (no limit)
Power over Ethernet	No	No	Yes (requires separate PoE HAT)	Yes (requires separate PoE HAT)
WiFi	No	WiFi 802.11b/g/n dual-band	WiFi 802.11ac Dual Band	WiFi 802.11ac Dual Band
Bluetooth	No	5.0 BLE	4.2 BLE	5.0 BLE
Video Output	Mini HDMI	Mini HDMI	HDMI/3.5 mm Comp./DSI	micro-HDMI/3.5 mm Comp./DSI
Audio Output	I²S/Mini HDMI/3.5 mm Composite	I²S/Mini HDMI/3.5 mm Composite	I²S/HDMI/3.5 mm Composite	I²S/HDMI/3.5 mm Composite
Camera Input	15 Pin CS	15 Pin CS	15 Pin CS	15 Pin CS
GPIO Pins	40	40	40	40
Memory	MicroSD	MicroSD	MicroSD	MicroSD

2.4.1 Single Board Computers (SBCs)

Single board computers (SBCs) play a vital role in the IoT ecosystem, offering immense importance and value for IoT applications [54]. These compact and self-contained computing devices integrate all essential components of a computer onto a single circuit board. These credit card-sized computers feature robust processing capabilities, generous memory, storage options, and an array of input/output interfaces. Also, SBCs often come with built-in connectivity options such as Ethernet, Wi-Fi, or BlueTooth (BLE). Their compact size, low power consumption, and cost-effectiveness make them ideal for deploying IoT solutions in various domains. SBCs serve as the brain of capable IoT devices or a gateway for an array of simpler devices, enabling data processing, communication, and control functionalities. They can run operating systems, execute software applications, and connect to the Internet, allowing for real-time data acquisition, analysis, and decision-making. Furthermore, SBCs provide a flexible and customizable platform for IoT developers, empowering them to build and deploy tailored IoT solutions based on specific project requirements. Whether used in industrial automation, smart homes, or wearable devices, SBCs offer the processing power and connectivity needed to enable seamless integration and intelligent interaction between IoT devices, making them an essential component in the IoT landscape.

The Raspberry Pi line of products, compared in Table 2.1, is a prominent example of the SBC domain. Raspberry Pi models exemplify the power and versatility of SBCs.

The Raspberry Pi ecosystem offers a range of models tailored to different needs. The models vary in terms of computational power, memory capacity, connectivity options, and peripheral support. Each model caters to specific IoT requirements, from the entry-level Raspberry Pi Zero to the more advanced Raspberry Pi 4. The Raspberry Pi Foundation also provides a user-friendly Linux-based operating system called Raspberry Pi OS [55], along with a rich repository of software libraries and development tools. This ecosystem simplifies the software development process and fosters innovation, allowing developers to create custom IoT solutions easily. Moreover, the Raspberry Pi community is vibrant and enthusiastic, offering extensive online resources, forums, and projects for users to collaborate and share their experiences. This community-driven approach empowers individuals to explore and expand the capabilities of Raspberry Pi boards in diverse IoT applications, ranging from home automation to robotics, industrial monitoring, and beyond.

The Raspberry Pi is an excellent choice as a gateway in the proposed PoT system due to several compelling motivations. Its affordability makes it accessible to a wide range of user categories in home automation or business domains. The cost-effective nature of Raspberry Pi boards allows for PoT scalability, enabling the deployment of multiple PoT gateways across various premises or remote sites without substantial financial constraints. Besides, the Raspberry Pi's powerful processing capabilities and versatile connectivity options make it well-suited for PoT gateway functionality to aggregate simple devices, typically existing in homes and offices, into a coherent VoIP-enabled smarter system. Its onboard Ethernet and Wi-Fi connectivity and the availability of USB and GPIO interfaces facilitate seamless integration with a diverse range of IoT devices and networks. Moreover, Raspberry Pi's Linux-based operating system and its extensive software ecosystem allow us to build a modular software stack for the PoT gateway that can be tailored to the specific requirements of different application needs. This flexibility enables incorporating our proposed custom messaging protocol (i.e., tSIP) and security

mechanisms capabilities within the PoT gateway.

2.4.2 Embedded Linux

Embedded Linux is a powerful operating system that plays a crucial role in the IoT domain, enabling intelligent and connected devices to be deployed. Embedded Linux enables a robust and flexible platform for developing IoT solutions by offering various features and capabilities. Its open-source nature allows for customization, making it suitable for various IoT applications across industries such as manufacturing, healthcare, smart homes, agriculture, and transportation. With its small footprint and resource-efficient design, Embedded Linux can run on low-power devices, making it ideal for resource-constrained IoT environments. Its rich set of libraries, tools, and frameworks facilitates the development of applications, connectivity, and data management for IoT devices. Moreover, Embedded Linux benefits from extensive community support and frequent updates, ensuring the availability of security patches, bug fixes, and new features. Its versatility, reliability, and compatibility make Embedded Linux popular for building robust and scalable IoT solutions that seamlessly integrate with other devices, systems, and cloud platforms.

Measuring the performance of embedded Linux systems is of utmost importance in ensuring their optimal functionality and efficiency. Performance measurement provides valuable insights into the system's behaviour and resource utilization, allowing for identifying and resolving potential bottlenecks, inefficiencies, or areas for improvement. Developers can make better-informed design decisions by quantifying metrics such as response time, throughput, latency, memory usage, CPU utilization, and power consumption. It helps fine-tune the system configuration, improve resource allocation, and identify areas where system enhancements can be made. Additionally, performance measurement facilitates benchmarking and comparisons between different software and hardware configurations, enabling designers to select the most suitable options for their specific use

cases. This information is precious in the context of IoT, where embedded Linux systems often operate in resource-constrained environments with limited processing power, memory, and energy availability. By measuring and monitoring performance, developers can ensure that their embedded Linux systems in IoT deployments meet the desired requirements, provide reliable operation, and deliver optimal performance, leading to enhanced user experiences, increased efficiency, and improved overall system reliability. Listed below are some of the numerous tools available for the performance evaluation of Linux-based systems. However, this list is by no means exhaustive since, in Linux and open-source platforms in general, developers may implement their own approaches and set of tools to achieve the same targets, which is uncountable:

- `top/ps` [56]: The `top` and `ps` commands are typically installed on all Linux distributions by default. While the `ps` command provides an instantaneous snapshot of system activity per thread, the `top` command provides essentially similar information as `ps` at defined intervals, which can be as granular as one-hundredth of a second. Despite their powerfulness and readiness, these commands are overlooked by many Linux users.

- `free` [56]: It is a pre-installed and readily accessible application on most Linux distributions. `free` is a convenient command to view a snapshot of system memory usage and can be used to detect memory leakages, a situation where a computer program fails to release dynamically allocated memory after it is no longer needed, or disc instability caused by excessive swapping.

- `valgrind` [57]: `valgrind` is a popular, extensible, and GPL2-licensed open-source framework for advanced programmers that can be used to detect memory-related errors and threading issues. `valgrind` utility automatically and dynamically detects these errors as the code executes. However, it may produce false positives. Although it is a handy utility, it can be exceedingly intrusive because the code

executes 50 times slower than its actual execution speed [58].

In Chapter 4, we examine the feasibility of using embedded Linux platforms as PoT gateways in the proposed framework, focusing on evaluating the performance of Raspberry Pi's line of products in this role. Specifically, we assess the operating system's capabilities by simulating a growing number of simultaneous calls using the embedded Asterisk installation. The objective is to demonstrate the potential of cost-effective embedded Linux SBCs for PoT applications, particularly in residential and small-to-medium-sized settings. By analyzing the performance of these devices as PoT gateways, we aim to encourage their widespread adoption in practical PoT scenarios.

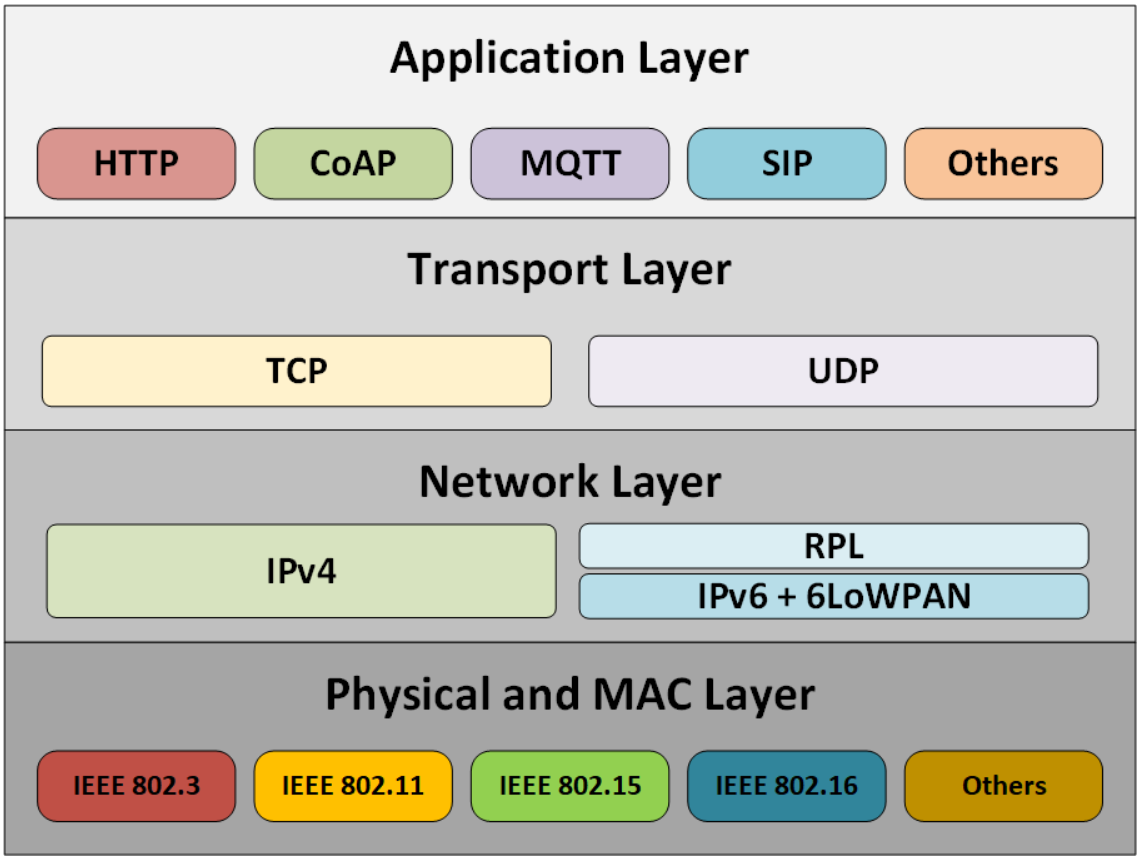

Figure 2.2: IoT protocol stack (Adapted from [59])

2.5 IoT Messaging Protocols

2.5.1 Overview

To foster adaptability and scalability in the IoT, the long-term interoperability of the underlying infrastructure of the IoT should be maintained. Concurrently, amidst the relentless rapid pace of technological progress, IoT networks must be flexible to easily incorporate and capitalize on novel, optimized hardware designs that are on the line. This requires good IoT standards to avoid the problems with vendor lock-in and backward compatibility issues that often come with hardware-driven solutions that are not open-source. Therefore, the application layer technologies, shown on top of the IoT protocol stack in Figure 2.2, that are based on robust standards and are certified by reputable standard development bodies (e.g., IEEE) provide a universal and transparent framework that encourages worldwide acceptance and cross-vendor support in the future.

Messaging protocols are application layer protocols that are crucial in the IoT ecosystem. They facilitate efficient and reliable communication between interconnected devices. These protocols enable seamless data exchange, command delivery, and real-time interactions among various IoT devices, systems, and services [60]. Messaging protocols are designed to address the unique challenges of IoT environments, including limited bandwidth, intermittent connectivity, and constrained resources. They offer lightweight, low-overhead communication mechanisms optimized for low-power devices, allowing efficient utilization of network resources. These protocols prioritize efficiency and scalability while ensuring secure and reliable data transmission. The importance of messaging protocols in IoT lies in their ability to enable interoperability and seamless integration. With a standardized messaging protocol, devices from different manufacturers running on different platforms can communicate and exchange data effortlessly. This interoperability fosters the development of robust and scalable IoT solutions by promoting device agnosticism

and avoiding vendor lock-in [61].

Moreover, messaging protocols facilitate decoupling IoT components, enabling asynchronous communication patterns. This loose coupling enhances system flexibility, scalability, and responsiveness, allowing real-time data analysis, event-driven automation, and intelligent decision-making. Messaging protocols also contribute to the reliability and resilience of IoT systems. Some messaging protocols provide mechanisms for quality of service (QoS) levels, ensuring message delivery and guaranteeing that critical data reaches its intended destination [62].

2.5.2 Challenges in IoT Messaging Protocols

Messaging protocols in IoT face several challenges, and one of the key aspects is the absence of a *one-size-fits-all* solution [63]. The diverse nature of IoT applications, devices, and network environments introduces complexity and variations that necessitate different messaging protocols to suit specific requirements. Some IoT deployments may prioritize low latency and real-time responsiveness, making protocols like MQTT (Message Queuing Telemetry Transport) an ideal fit [64]. However, other scenarios like constrained and resource-limited devices may call for lightweight protocols like CoAP (Constrained Application Protocol) to minimize overhead [65]. The challenge lies in selecting the most appropriate messaging protocol for a specific IoT application, considering factors such as bandwidth constraints, device capabilities, network topology, and security requirements.

Interoperability is another significant challenge [60]. As IoT ecosystems encompass many devices and platforms from different manufacturers, ensuring seamless communication and data exchange becomes complex. Although standardization efforts have led to the development of common messaging protocols, achieving interoperability between heterogeneous devices and platforms can still be challenging. Bridging the gap between protocols and establishing interoperable frameworks is an ongoing concern in the IoT community that still attracts many researchers.

Security is also a critical challenge [66]. Messaging protocols must ensure the confidentiality, integrity, and authenticity of transmitted data in IoT environments. Protecting sensitive information, preventing unauthorized access, and addressing potential vulnerabilities in messaging protocols are crucial to safeguard IoT ecosystems from data leakage and cyber threats.

Scalability is another challenge with the increasing number of devices and the exponential data growth in IoT deployments [60]. Messaging protocols must handle the growing volume of messages efficiently and scale to accommodate the expanding network without compromising performance or reliability.

Moreover, the varying network conditions and connectivity issues in IoT environments pose challenges for messaging protocols. Fluctuating network availability, low bandwidth, intermittent connectivity, and high packet loss mandate protocols that can gracefully handle such conditions, ensuring reliable communication and minimizing data loss [67].

Addressing these challenges requires carefully evaluating the specific IoT deployment, considering factors such as application requirements, device capabilities, network constraints, and security considerations. It calls for implementing appropriate protocols, protocol gateways, and mechanisms that enable efficient and secure communication, considering the unique characteristics and demands of the IoT landscape. We contrast the common messaging protocols typically used in IoT in the following subsections: *MQTT*, *CoAP*, and *HTTP* (Hypertext Transfer Protocol). We then overview the *SIP* (Session Initiation Protocol) protocol, the predominant protocol used in the VoIP ecosystem. We highlight the unique characteristics of the SIP protocol and the motivation behind adopting it in the IoT domain. Also, we review the state-of-the-art literature considering using the SIP protocol in IoT, highlighting the uniqueness of our proposed tSIP as native support for the SIP protocol in constrained PoT applications.

2.5.3 Common IoT messaging Protocols

2.5.3.1 MQTT

MQTT is a standard lightweight messaging protocol widely used in the IoT ecosystem. Its main characteristics make it suitable for IoT applications that require efficient, reliable, and scalable communication. MQTT is designed to be lightweight, minimizing network bandwidth and reducing the processing burden on resource-constrained devices typically found in the IoT domain. This characteristic makes it well-suited for IoT deployments where devices have limited power and bandwidth capabilities. MQTT follows a publish-subscribe messaging paradigm, enabling efficient and asynchronous communication [64]. Devices can publish messages on specific topics; others can subscribe to these topics to receive relevant interest data. This decoupling allows for scalable and flexible communication between devices and systems. MQTT supports three Quality of Service (QoS) levels to ensure message delivery reliability [68]. QoS levels range from "*at most once*" (fire and forget) to "*at least once*" (guaranteed delivery) and "*exactly once*" (assured single delivery). This flexibility enables system designers to choose the appropriate QoS level based on the specific requirements of their IoT applications. MQTT is designed to support intermittent and unreliable network connections. This includes session persistence, message retention, and the ability to store and forward messages, ensuring that data is reliably transmitted even during network disruptions. These characteristics make MQTT a popular choice for IoT applications that demand efficient, scalable, and reliable communication.

2.5.3.2 CoAP

CoAP is a lightweight messaging protocol designed specifically for resource-constrained IoT devices. Its main characteristics make it an efficient and suitable choice for IoT applications that require low overhead and energy-efficient communication [65]. CoAP is

designed to be lightweight and simple, minimizing IoT devices' processing and memory requirements. It utilizes a RESTful (REpresentational State Transfer) architecture, similar to HTTP, allowing easy integration with web-based services and interoperability with existing web technologies [69]. CoAP operates over UDP (User Datagram Protocol), which provides low-latency communication suitable for real-time applications. CoAP offers a compact header format and efficient message encoding, reducing the overhead and improving communication efficiency. Alongside, CoAP supports resource discovery and observation mechanisms, enabling devices to discover and interact with available resources in a network [65]. This capability allows for efficient monitoring and control of IoT devices and services. Additionally, CoAP supports various methods for communication, including *GET*, *POST*, *PUT*, and *DELETE*, enabling device-to-device and device-to-server interactions. Nonetheless, CoAP includes mechanisms for congestion control and reliable message delivery, such as retransmission and acknowledgement mechanisms. These characteristics make CoAP an ideal choice for IoT applications with constrained devices, limited bandwidth, and energy requirements, providing an efficient and scalable communication protocol for resource-constrained environments.

2.5.3.3 HTTP

HTTP is a mature and widely used protocol in web applications. However, its characteristics limit its usability in IoT applications [70]. HTTP is a request-response protocol that operates over TCP (Transmission Control Protocol), offering reliable and connection-oriented communication. Its main characteristics include a well-defined structure, flexibility, and support for various data formats. HTTP utilizes the client-server model, where clients initiate requests and servers respond with the requested data. This makes HTTP suitable for web-based IoT applications, where data is fetched from servers or transmitted to cloud services.

The usability of HTTP in IoT applications stems from its widespread adoption and

Table 2.2: Comparing SIP Protocol with Popular Messaging Protocols for IoT.

Criteria	HTTP	CoAP	MQTT	SIP
Communication Architecture Model	Client/Server	Client/Server	Client/Server	Client/Server Peer-to-Peer
Message Semantic	Request/Response	Request/Response Publish/Subscribe	Publish/Subscribe	Request/Response Publish/Subscribe (long) Session-based
Header Size	Undefined	4 byte	2 byte	Undefined
Message Size	Large and undefined	Small and undefined	Small and undefined (256 MB max)	Large and undefined
Transport Protocol	TCP	UDP	TCP UDP (MQTT-SN)	TCP UDP
Encoding Format	Text	Binary	Binary	Text
Security	TLS/SSL	DTLS	TLS/SSL	TLS/SSL DTLS
QoS	Limited	Yes	Yes	Yes

compatibility with existing web technologies. It leverages the existing infrastructure and knowledge surrounding web development, making integrating IoT devices with web-based services and frameworks easier. Additionally, HTTP's support for various data formats, such as JSON (JavaScript Object Notation) and XML (Extensible Markup Language), enables seamless interoperability and easy integration with different systems [71]. However, HTTP also has limitations in the context of IoT. One major drawback is its relatively high overhead regarding message size and the need for persistent connections, delegating its deployments to capable devices in the IoT ecosystem, typically the gateways [70]. This can be problematic for IoT devices with limited resources and constrained networks, resulting in increased power consumption and decreased efficiency. Moreover, HTTP's request-response nature is unsuited for scenarios requiring real-time communication or asynchronous event-driven interactions.

2.6 Session Initiation Protocol (SIP)

2.6.1 Overview

The Session Initiation Protocol (SIP) is a signalling protocol widely used for establishing, modifying, and terminating multimedia sessions in the VoIP ecosystem [14]. SIP is an efficient candidate to provide a unified application protocol paradigm to IoT systems

typically populated with heterogeneous devices. This is because SIP inherently supports different communication architecture models (e.g., *client/server* and *peer-to-peer*) and different messaging semantics (e.g., *request/response*, *publish/subscribe*, and long *session-based*) typically encountered in simple and advanced IoT systems. The *request/response* message semantic can be used to issue a command to configure IoT devices or to query information from them. The *publish/subscribe* semantic is used to efficiently build event-based interaction with IoT devices that react to predefined thresholds of their readings or state changes. However, the *session-based* message semantics can be used to build advanced IoT systems incorporating rich media types that are exchanged by establishing communication channels over a period of time. Moreover, SIP operates at the application layer of the TCP/IP protocol stack. It can utilize the TCP and UDP transport layer protocols, giving system designers the flexibility to trade off between guaranteed delivery on the one hand and real-time constraints that require low latency on the other hand based on the application requirements. Also, one of the key characteristics of SIP is its flexibility and extensibility, allowing for the integration of various communication technologies and devices. SIP can facilitate seamless communication and interoperability between IoT devices, enabling them to establish real-time sessions, exchange data, and collaborate with other devices or services. This makes SIP suitable for IoT applications that require interactive and collaborative capabilities, such as smart homes, industrial automation, telemedicine, and smart cities. Table 2.2 contrasts the SIP protocol with the standard IoT messaging protocols, highlighting its unique characteristics.

2.6.2 SIP Methods and Transactions

In SIP, *methods* are commands or actions to initiate, modify, or terminate participant sessions. SIP defines various methods that enable communication and control within a session [14]. These methods provide the foundation for initiating and managing sessions within the SIP protocol. Each method serves a specific purpose, facilitating various

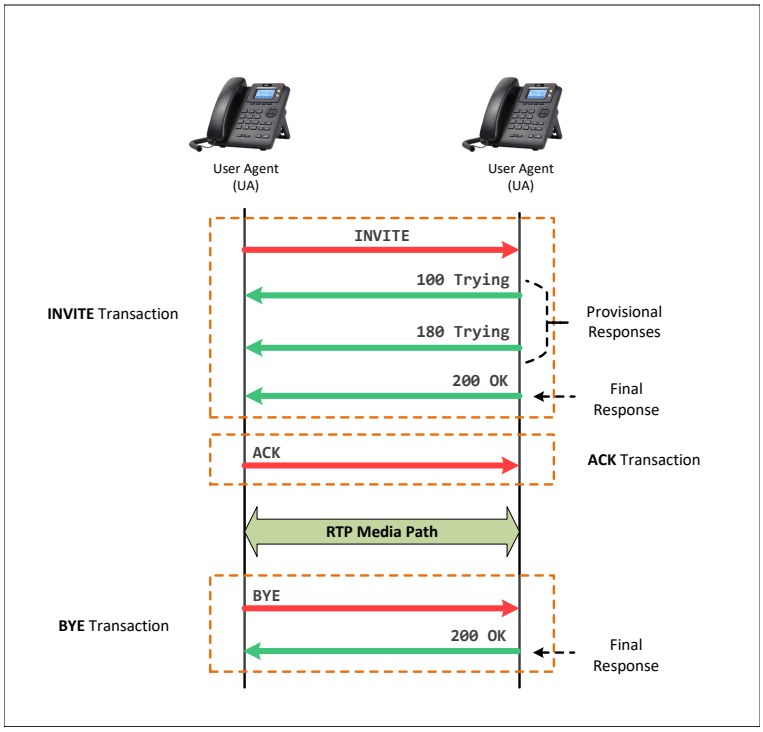

Figure 2.3: SIP Transactions (SIP Call Origination and Termination Example).

session establishment, modification, and termination stages. By utilizing these methods, SIP enables flexible and interactive communication among participants in real-time communication sessions. Some commonly used SIP methods include:

- `INVITE`: This method initiates a session and invites a user to participate in a communication session, such as a voice or video call. The `INVITE` method contains details about the call, including the participants' addresses, media capabilities, and session parameters.

- `ACK`: After receiving a successful response to an `INVITE` request, the `ACK` method confirms the acceptance of the response. It acknowledges the establishment of the session and ensures the reliable delivery of the response.

- `BYE`: The `BYE` method terminates an established session. It indicates the desire to end a call or session between participants. The `BYE` message includes information about the reason for ending the session, such as user-initiated termination or call

completion.

- `REGISTER`: The `REGISTER` method is used by a user agent to register its contact information with a SIP server. This allows the user agent to receive incoming calls and messages.

- `OPTIONS`: The `OPTIONS` method is used to query the capabilities and characteristics of a remote user agent or server. It allows a user agent to retrieve information about supported methods, media types, and other session-related features.

On the other hand, a *SIP transaction* represents a sequence of request and response messages exchanged between entities involved in a communication session [14]. A SIP transaction begins with transmitting a request message, such as an *INVITE* or *REGISTER*, and concludes with a final response message, such as a *200 OK* or a failure response. SIP transactions follow a specific state model that progresses through various states based on the messages exchanged. The initial state is the "*Trying*" state while the initial request is being processed. As the transaction progresses, it can transition to states such as "*Proceeding*" to indicate that the request is processed and provisional responses are being sent and "*Completed*" to indicate that the final response has been sent. SIP Transactions provide an essential mechanism for managing the reliability and consistency of SIP communications. They ensure that messages are delivered reliably and that the necessary acknowledgments and confirmations are received. SIP transactions help handle retransmissions, timeout management, and transaction-level consistency, ensuring the integrity and robustness of the communication process. Figure 2.3 depicts SIP transactions incurred through a typical SIP call session, from its origination to the termination by the participant user agent (UA).

2.6.3 IoT and SIP Integration: Challenges

Integrating SIP into the IoT domain presents several challenges:

- First, SIP is an HTTP-inspired text-based protocol. Therefore, despite the unique features of SIP, it often goes beyond most IoT devices' processing and storage limits. This means that the full protocol stack of SIP cannot be directly implemented on resource-constrained IoT devices. This mandates the provision of optimization and adaptation strategies to ensure efficient resource utilization.

- Second, the security of SIP-based communication in IoT environments is crucial. Protecting sensitive data, ensuring authentication and authorization, and safeguarding against potential attacks are paramount considerations. Implementing secure and robust mechanisms within SIP-based IoT systems is necessary to maintain the integrity and privacy of communication.

- Furthermore, scaling SIP-based IoT systems to accommodate a large number of devices and concurrent sessions can be challenging. Efficient session management, load balancing, and network optimization are essential to handle the increased traffic and ensure reliable and responsive communication. Additionally, interoperability among different SIP implementations and compatibility with diverse IoT platforms and devices can be complex, requiring adherence to standardized SIP profiles and support for industry-wide protocols.

2.6.4 IoT and SIP Integration: State of the Art

Numerous research in the literature has utilized the SIP protocol in their proposed IoT systems or frameworks to leverage its capabilities. For example, the authors of [72] proposed an IoT service platform called *iSIPtalk*. It leverages the inherent capabilities of the SIP protocol to provide developers with the full range of support, flexibility, and the rapid development cycle needed to deploy advanced IoT solutions demanding low latency, high data rates, and different quality of service (QoS). To justify the applicability of their proposed framework, they implemented a real testbed that targets a vehicular

service that tracks vehicle locations and monitors their status. The authors rely on capable computing devices that run the Linux OS to deploy the SIP user agents (SUA) and the SIP proxy. The authors of [73] proposed a SIP-based Home Gateway (*SHG*) to interface with Zigbee and Bluetooth smart objects in homes. *SHG* facilitates the integration between domotics devices and the existing SIP infrastructure, allowing users to control smart objects at home using SIP clients. *SHG* uses a subset of SIP methods to satisfy home automation applications. Again, *SHG* is based on a capable embedded Linux single-board computer (SBC) to implement the SIP protocol capabilities of the *SHG*. The authors of [74] proposed a SIP-based emergency framework called *SEEK* that automatically calls the appropriate Public Safety Answering Points (PSAPs) if events are triggered by body sensors that measure vital signs of the human body. Once an event is triggered, the framework establishes a call to the corresponding PSAP through a proprietary SIP client (softphone) that encapsulates critical data and location information into the body of the SIP messages using Sensor Model Language [75]. The sensor gateway of the *SEEK* platform does not possess SIP functionalities. However, it delegates SIP tasks to a proprietary SIP client that runs on a mobile phone. The authors of [76] proposed *VIoT* (Voice over Internet of Things), which integrates IoT devices with voice capabilities into the VoIP ecosystem to enable new customer applications and motivate further research on integrating IoT protocols and telephony applications.

The authors of [17] proposed *CoSIP*, a lightweight version of the SIP protocol for constrained devices. The purpose of *CoSIP* is to allow duty-cycled devices to instantiate communication sessions in a standardized fashion for M2M (machine-to-machine) application scenarios. *CoSIP* is a generic constrained binary encoding of the SIP protocol that follows the same message syntax as CoAP. However, there exist differences in the key concepts between SIP and CoAP. For example, there is no concept of *SIP dialogs* in CoAP. This requires some tweaks to the original message syntax of CoAP to accommodate SIP encoding by *CoSIP*. This complicates the implementation of *CoSIP*

and hinders its portability among intrinsically heterogeneous IoT devices with different hardware architectures and programming language support. Nonetheless, to the best of our knowledge, *CoSIP* is the only research in the literature to provide smart objects with SIP capabilities through a constrained version of the SIP protocol.

2.6.5 tSIP: A Lightweight Version of the SIP Protocol for Constrained Devices

To address IoT and SIP integration challenges, the thesis proposes *tSIP*. *tSIP* is a constrained version of the SIP protocol. It provides a portable, lightweight message format that can be deployed in limited-resource devices to enable them with SIP communication capabilities. *tSIP* allows for standardized peer-to-peer communication between smart objects in their communication network vicinity. It reduces processing power, network traffic, and energy consumption in IoT contexts. Alongside, with the help of a proxy, *tSIP* messages can be turned into their original SIP messages if a communication server exists on the premises. The proxy can be a PoT gateway or an add-on Asterisk module installed on top of the existing Asterisk-based communication server. This allows *tSIP*-enabled devices to act as typical SIP endpoints in the VoIP ecosystem, where each device would then be assigned a unique phone number by the communication server through which it can be administered. This provides seamless interaction and enhances user engagement with the surrounding devices. People could use the existing phone sets at home or within the premises, combined with simple and intuitive voice commands, as delineated in Chapter 3, as a unified interface to administer and query information from the surrounding devices. Furthermore, cost-effective embedded Linux platforms (such as the Raspberry Pi) can be used simultaneously as a PoT gateway and a PBX server, even in homes without a PBX server by default. As per the performance evaluation study in Chapter 4, embedded Linux platforms can provide adequate concurrent active channels when acting as Asterisk servers to provide VoIP functionalities to home residents. In

Chapter 6, we compare our proposed *tSIP* protocol to both the original SIP protocol and *CoSIP* to show how *tSIP* is better in terms of portability and compression ratio.

2.7 Blockchain

2.7.1 Overview

Blockchain technology has emerged as a revolutionary concept that can potentially transform various industries [77]. It originated in 2008 with the introduction of *Bitcoin*, the first decentralized cryptocurrency [19]. Blockchain is a distributed ledger that maintains a chronological chain of blocks containing transactions or data. Its main characteristics include decentralization, immutability, auditability, and fault tolerance.

Decentralization is a fundamental aspect of blockchain, as it eliminates the need for a central authority, such as a bank or government, to oversee transactions. Instead, a network of computer nodes collaboratively validate and record transactions, ensuring transparency and trust among participants through the utilization of an agreed consensus mechanism. The immutability of the blockchain ensures that once a transaction is recorded, it cannot be altered or tampered with, enhancing the integrity and reliability of data.

Blockchain technology has applications beyond cryptocurrency [78]. In finance, for example, it offers secure and transparent transactions, reducing the need for intermediaries. Supply chain management benefits from blockchain's ability to track and verify the authenticity and movement of goods, enhancing transparency and reducing fraud. Blockchain also enables secure and decentralized digital identity management, enabling individuals to control and protect their personal information. Additionally, it has potential applications in healthcare, intellectual property rights, and more which attract the intention of many researchers.

However, blockchain faces scalability, energy consumption, and regulatory concerns

Table 2.3: Comparison of Public, Private, and Consortium Blockchains [18].

	Public Blockchain	**Private Blockchain**	**Consortium Blockchain**
Participation in Consensus	All nodes	Single organization	Selected nodes in multiple organizations
Access	Public read/write	Can be restricted	Can be restricted
Identity	Pseudo-anonymous	Approved participants	Approved participants
Immutability	Yes	Partial	Partial
Transaction Processing Speed	Slow	Fast	Fast
Permissionless	Yes	No	No

[79]. As more transactions are added to the blockchain, scalability becomes a crucial factor that needs to be addressed. The energy-intensive consensus mechanisms used in blockchain networks have raised concerns regarding sustainability. Furthermore, regulations surrounding blockchain technology and cryptocurrencies vary across jurisdictions, requiring legal frameworks to catch up with technological advancements.

2.7.2 Types of Blockchains

Blockchain technology can be classified into three main types: public, private, and consortium blockchains [18].

- **Public blockchain**: Public blockchains, such as *Bitcoin* and *Ethereum* [80], are open and decentralized networks where anyone can participate [81]. They are characterized by a distributed network of nodes that collectively validate transactions and maintain the blockchain. Public blockchains offer transparency, immutability, and security through consensus mechanisms, such as *Proof of Work* (PoW) or *Proof of Stake* (PoS) [82]. They are typically used for cryptocurrencies and applications that require a trustless and permissionless environment.

- **Private blockchains**: Private blockchains are restricted and accessible only to a specific group or organization. These blockchains are centralized and governed by a single entity, giving them greater control over access and governance. Private blockchains provide privacy, scalability, and efficiency compared to public blockchains. They suit enterprise applications, where confidentiality and regulation

compliance are crucial. A private blockchain is utilized by the proposed registration and authentication mechanism in the thesis to securely associate the surrounding smart objects within the premises to the communication server of the enterprise due to its privacy characteristics, fast transaction processing, and cost-effectiveness.

- **Consortium blockchains**: This type of blockchain combines elements of both public and private blockchains [83]. They are governed by a group of organizations rather than a single entity, ensuring a more distributed and decentralized approach while maintaining a certain level of control. Consortium blockchains are designed for specific industry sectors or collaborations between organizations. They offer shared governance, enhanced privacy, and selective participation, allowing multiple entities to collaborate on a common blockchain infrastructure. As a future direction of the proposed PoT framework, consortium blockchains might be used to allow telecom companies to provide PoT as a service to their existing portfolio. In the context of PoT, consortium blockchains can facilitate secure and transparent interactions among telecom companies, IoT device manufacturers, service providers, and stakeholders to collaborate and share resources while maintaining a level of control and trust.

Table 2.3 contrasts the main characteristics of public, private, and consortium blockchains.

2.7.3 Consensus Algorithms

Consensus algorithms play an essential role in maintaining the integrity and agreement of data within a blockchain network. These algorithms are designed to enable distributed nodes to reach a consensus on the validity of transactions and ensure that the blockchain remains secure and consistent [82]. Several consensus algorithms exist, each with its own set of characteristics.

- **Proof of Work (PoW)**: PoW is a commonly used consensus algorithm notably

utilized by Bitcoin. PoW requires participants, known as *miners*, to solve complex mathematical puzzles to validate transactions and add new blocks to the blockchain. The main characteristic of PoW is its reliance on computational power and energy consumption. It ensures that consensus is achieved through the majority of computational work performed by honest nodes, making the network more resilient against attacks.

- **Proof of Stake (PoS)**: PoS is a consensus algorithm that addresses the energy consumption concerns associated with PoW. In PoS, participants, known as *validators*, are selected to validate transactions and create new blocks based on the amount of cryptocurrency they hold. The main characteristic of PoS is that the probability of a validator being chosen to create a new block is proportional to their stake in the network. This ensures that participants with a larger stake are more likely to be selected, theoretically aligning their interests with the network's security.

- **Proof of Authority (PoA)**: PoA is a consensus algorithm that relies on the identity and reputation of network participants rather than computational power or stake. PoA requires a predefined set of trusted nodes, known as *authorities*, to validate transactions and create new blocks. The main characteristic of PoA is that authority nodes are identified and selected based on their reputation, expertise, or a formalized governance process. The proposed registration and authentication mechanism in the thesis utilizes PoA. PoA ensures fast block creation times and high throughput but relies on the trustworthiness and accountability of the selected authorities. Alongside this, PoA allows the potential to roll back the blockchain at any time in the past, which is common during application development.

There are also consensus algorithms such as *Delegated Proof of Stake* (*DPoS*) [84], *Practical Byzantine Fault Tolerance PBFT*) [85], and *Raft* [86], each with its own unique

characteristics. *DPoS* introduces a voting system where participants elect a limited number of delegates to validate transactions. *PBFT* focuses on reaching consensus in networks where participants may be malicious or unreliable, ensuring agreement among a set of known validators. *Raft* is a consensus algorithm designed for distributed systems, providing strong leader-based consensus.

2.7.4 Smart Contracts

Smart contracts are self-executing agreements with predefined rules and conditions encoded on a blockchain [87]. They automate the execution of agreed obligations without intermediaries. The main characteristic of smart contracts is their ability to automatically enforce contract terms once predefined conditions are met, removing the need for manual intervention and reducing the risk of fraud or manipulation. Smart contracts are written in programming languages designed explicitly for blockchain platforms, such as *Solidity* for Ethereum [88]. They enable the exchange of digital assets, the transfer of ownership, and the implementation of complex business logic. Smart contracts have diverse applications across industries, including supply chain management, financial services, healthcare, and decentralized applications. They enable secure, tamper-resistant, and verifiable interactions between multiple parties, facilitating automated and trustworthy transactions. However, smart contracts are only as reliable as the code written, and potential vulnerabilities or bugs can have significant consequences. Therefore, careful auditing, testing, and security practices are essential to ensure the robustness and integrity of smart contracts deployed on a blockchain.

The proposed registration and authentication mechanism in the thesis uses the Ethereum blockchain and smart contracts to implement a programmatic, immutable access control mechanism to administer the association of devices with the communication server, avoiding the exhaustion of the communication server resources, overwhelming its design capacity, or altering the existing SIP-based VoIP architecture and configuration as

delineated in Chapter 4.

2.7.5 SIP and Blockchain Integration: Benefits and Possibilities

The integration of blockchain with SIP communication holds the potential to introduce several benefits and possibilities [89]. By leveraging blockchain technology, SIP communication can benefit from enhanced security, transparency, and trust. Blockchain's decentralized nature and immutability can provide a more secure and tamper-proof environment for SIP-based communication, reducing the risk of unauthorized access, fraud, and data manipulation. Additionally, blockchain's transparent and auditable nature can promote trust among participants by allowing them to verify the integrity and authenticity of communication transactions.

Integrating blockchain with SIP communication enables new business models and revenue streams. Smart contracts, for example, can automate and streamline the execution of SIP-based services, such as call routing, billing, and service provisioning. This can reduce administrative overhead, eliminate intermediaries, and enable new monetization models. Blockchain-based micropayments [90] and tokenization [91] can facilitate seamless and secure financial transactions between SIP participants, opening up possibilities for innovative value-added services for telecom companies.

Furthermore, blockchain can facilitate identity management and authentication in SIP communication. By utilizing blockchain for identity verification and storing verified user identities, trust can be established between SIP participants without relying on centralized identity providers. This can enhance privacy, reduce identity theft risk, and enable secure and trusted communication between parties, eliminating the notorious resource exhaustion of standard-based authentication algorithms.

Another possibility is the integration of blockchain and SIP for data management and auditing. Blockchain's immutable and transparent nature can facilitate the reliable tracking and auditing of communication data, such as call records and logs. This can

ensure regulatory compliance, simplify auditing processes, and provide a trustworthy record of communication activities.

2.7.6 SIP and Blockchain Integration: Challenges

Integrating blockchain technology into SIP communication poses several challenges that need to be addressed for successful implementation:

- **Latency**: Blockchain's inherent characteristics, such as its distributed nature and consensus mechanisms, introduce additional latency and complexity to real-time communication. The time required for transaction validation and block confirmation, especially in public blockchain networks, can impact the responsiveness and efficiency of SIP-based communication systems, potentially affecting the quality of real-time voice and video streams.

- **Scalability**: Scalability is a significant challenge when integrating blockchain with SIP. Blockchain networks, especially public ones, face limitations regarding transaction processing capacity and network throughput. The high transaction volumes generated by SIP-based communication can strain the blockchain network, potentially leading to delays, increased fees, or congestion.

- **Interoperability**: Ensuring seamless interoperability between blockchain-based SIP systems and existing traditional telecommunication infrastructures and protocols can be complex. Standardization efforts and the development of interoperability frameworks are necessary to bridge the gap between the blockchain and SIP domains.

- **Privacy**: Privacy and confidentiality pose challenges when integrating blockchain with SIP. Blockchain's transparent and immutable nature conflicts with the privacy requirements of certain communication scenarios. Ensuring the confidentiality of

call details, identities, and sensitive information while leveraging the benefits of blockchain technology demands careful design and the integration of appropriate privacy-enhancing mechanisms.

- **Compliance**: The regulatory landscape surrounding blockchain and communication technologies may present challenges. Compliance with data protection, privacy, and telecommunication regulations needs to be considered when integrating blockchain into SIP-based systems.

Addressing these challenges requires research, innovation, and collaboration between the blockchain and SIP communities. Research and iterative development can help identify best practices and solutions. Ultimately, carefully considering the trade-offs and the specific requirements of the SIP communication context is necessary to determine the feasibility and benefits of integrating blockchain technology while mitigating the associated challenges.

2.7.7 SIP and Blockchain Integration: State of the Art

Efficient VoIP deployment mandates the provision of security mechanisms to fix the inherent security vulnerabilities of the SIP protocol and ensure authentication, confidentiality, integrity, and availability for participants. More solutions have yet to be proposed in the literature to secure SIP. The current solutions are based on well-known internet security standards, namely *HTTP digest*, *TLS* (transport layer security), *IPSec*, and *S/MIME* (secure multi-purpose internet mail extensions) [92]. However, these SIP security options could be better; they all have several drawbacks [93]. Also, VoIP systems follow a centralized architecture. The central SIP server relies on the public key infrastructure (PKI), utilizes several server certificates, and performs complex hash computations to authenticate communicating participants to the server. This burdens the server with many heavy processes, affecting its performance and hindering VoIP scalability. It also

renders the PBX servers more vulnerable to denial of service (DOS) attacks.

Recently, we are witnessing increasing research in the literature to enhance the security of VoIP communication, thanks to adopting remote work settings for businesses, especially after the pandemic. Standard-based research, like those proposed in [94, 95], focuses on enhancing the authentication algorithms used in SIP protocols while maintaining the default centralized architecture of the VoIP system. However, hardening the authentication algorithm often results in higher resource consumption for complex calculations. This burdens the VoIP server and impacts its efficiency. Nonetheless, it still includes the *single point of failure* predicament, affecting its liability.

Therefore, blockchain recently arose as a viable solution to tackle the drawbacks of the current security practices of VoIP systems. For example, The authors of [96] propose a secured end-to-end (e2e) *VoLTE* (Voice over Long Term Evolution) session based on the Ethereum blockchain. Their proposal depends on generating cryptographic key pairs (private and public) for user equipment (UE), where the public key is stored in the Ethereum blockchain. Their proposal mitigates the vulnerability concerns of the default end-to-access (e2a) security of *VoLTE*, where the sessions are just encrypted between the mobile terminals and the IP Multimedia Subsystem (IMS), which uses the SIP protocol as its primary protocol. Their proposal shows minimal overhead for the existing IMS network and negligible call setup time compared to the original *VoLTE* setting. Interestingly, they propose in [97] a similar mechanism to provide a secure end-to-end VoIP system based on the Ethereum Blockchain. The blockchain is used as a keystore for the cryptographic public keys of VoIP users, eliminating the notoriously crucial need for a centralized certificate authority (CA) to authenticate VoIP users. However, their proposed mechanism shows a higher call setup than the existing security practices of VoIP.

The authors of [98] propose the *SIPchain*, a SIP defense cluster based on blockchain technology. *SIPchain* utilizes the blockchain to maintain a decentralized, immutable

Indicator of Compromise (IOC) ledger. Each SIP node in the *SIPchain* implements its own firewall rules to protect the node. Once the node detects an attack, it issues a transaction to the blockchain that contains information about the recently recognized compromise. The peer nodes read this information and act proactively by implementing the appropriate firewall actions to protect themselves against the attack, even before it scratches their surface. *SIPchain* helps the scalability of secured VoIP systems and mitigates the drawbacks of limited expertise professionals within the organizations to maintain the security of the deployed VoIP systems. Nevertheless, the efficiency of the *SIPchain* is affected by the induced high latency of information dissemination in public blockchains.

The authors of [89] propose a lightweight registration and authentication mechanism for SIP-based VoIP systems. They implement a decentralized identity model to facilitate the registration and authentication process between the SIP clients and the SIP server. The results show negligible resource utilization compared to the current security practices in VoIP systems, namely TLS and IPsec. However, it still suffers from the default-induced high latency of block formation in the public blockchain.

The authors of [99] propose *VoIPChain*, a decentralized blockchain-based identity authentication mechanism for VoIP systems that enhances the average time delay of call setup compared to the existing blockchain-based authentication mechanisms.

On the other hand, the literature currently lacks sufficient research targeting resource-limited devices in the context of blockchain technology. While blockchain has gained significant attention and research focus, most studies have primarily concentrated on conventional computing environments with ample resources. However, resource-limited devices, such as IoT devices, present unique challenges that demand tailored solutions. Integrating blockchain into these devices requires addressing issues of limited processing power, memory constraints, energy efficiency, and communication protocols. Further research is still needed to explore lightweight consensus algorithms, optimized data stor-

age techniques, efficient transaction verification mechanisms, and secure communication protocols that can accommodate the resource limitations of these devices. Investigating the potential benefits and trade-offs of blockchain in resource-constrained environments is crucial to unlock the full potential of blockchain technology across diverse domains.

2.8 Summary

This chapter provides a comprehensive background and literature review on the enabling technologies of the proposed Phone of Things (PoT) framework in the thesis. The chapter begins with an overview of the open-source IoT (Internet of Things) frameworks in the literature, highlighting their shortcomings and paving the way for further research and development of frameworks to address the affordability, accessibility, interoperability, scalability, and privacy of IoT systems. Also, the chapter explores the evolution and ubiquitousness of business communication systems, leading to the motivation towards integrating them into IoT applications. In that context, the chapter focuses on Asterisk, a Swiss-knife open-source PBX (Private Branch eXchange) software for building customized telephony applications to fulfil the integration between IoT and communication systems (i.e., VoIP). It reviews the research in the literature that leverages Asterisk in IoT applications. The chapter also covers areas regarding IoT device classification and embedded Linux as a predominant open-source OS for SBCs, leading to the potential of tiny, capable, and cost-effective gateways for IoT applications.

The chapter provides a thorough literature review that explores the messaging protocols in the IoT domain to hit a key concept of the proposed research in the thesis. The review covers areas regarding their main characteristics and common challenges leading to the lack of a "one-size-fits-all" messaging protocol in IoT. Consequently, the chapter reviews the SIP (Session Initiation Protocol) protocol as a capable and suitable IoT messaging protocol candidate. The chapter reviews the main characteristics of the SIP

protocol, contrasting them with their standard IoT messaging protocols counterparts to highlight its superiority. The chapter overviews the challenges of the SIP protocol that still impede its wide adoption in IoT applications and reviews state of the art in the literature regarding the integration between IoT and SIP, leading to the motivation of the proposed tiny version of the SIP protocol in the thesis (i.e., tSIP).

Finally, the chapter reviews the literature considering integrating blockchain technology with the SIP protocol. It provides an in-depth analysis of existing research and literature concerning the integration of blockchain and SIP, exploring the potential benefits, challenges, and future directions in this emerging field. The chapter describes the blockchain concept, highlighting its respective features and functionalities. It then delves into the literature review, covering key aspects such as the role of blockchain in enhancing security and trust in SIP communication, the impact on identity management and authentication, the potential for smart contracts in SIP-based transactions, and the challenges associated with integrating the blockchain with SIP.

Chapter 3

Proposed PoT Framework

This chapter introduces the proposed Phone of Things (PoT) framework. PoT is an extensible open-source IoT framework that addresses the limitations observed in the literature regarding state-of-the-art IoT frameworks. IoT is unique in its native accessibility through existing phone technologies, assets, and infrastructure. This is achieved through the utilization of open-source technologies and hardware platforms, the provision of a tiny native wrapper for the SIP (Session Initiation Protocol) protocol deployable on heterogeneous and constrained devices, and the proposal of a lightweight registration and authentication mechanism to facilitate the association of devices and PoT gateways to the existing communication servers without overwhelming their resources, affecting their design capacities, or changing their existing configurations. This reduces the deployment cost of the proposed PoT framework and provides widespread coverage and connectivity without the need for additional infrastructure investments. Also, PoT allows IoT systems to benefit from pervasive phone network coverage, reaching both urban and rural areas where the phone network is already present. Furthermore, PoT can leverage the existing capabilities of phone networks, such as voice communication, messaging services, and location-based services, to enhance the system's functionality and provide a more comprehensive user experience. The following sections provide an in-depth breakdown of the

system's component hierarchy, outlining each element's specific roles and responsibilities.

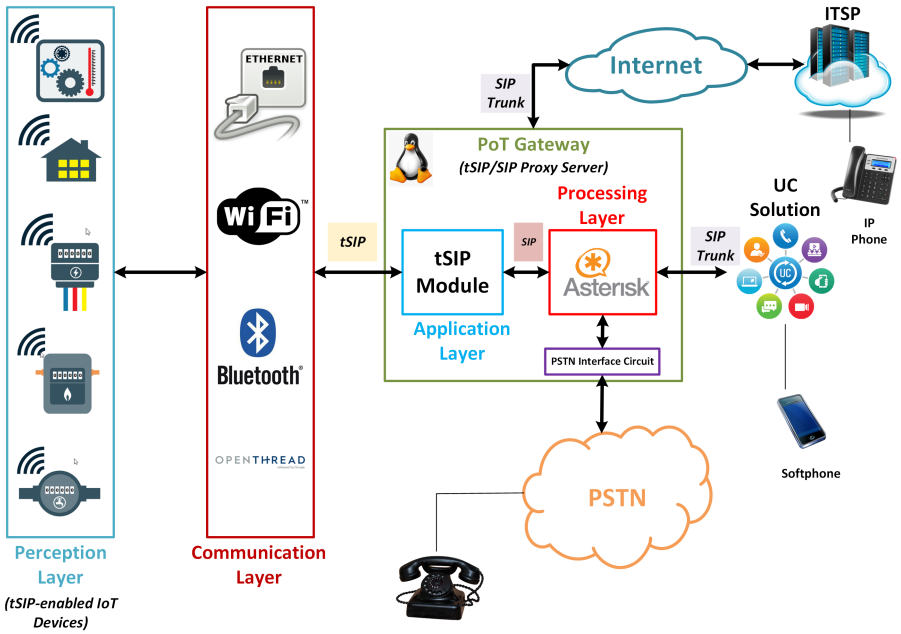

Figure 3.1: Overview of the Phone of Things (PoT) framework.

3.1 Overview

The proposed PoT framework is depicted in Figure 3.1. The PoT framework comprises a hierarchy of interconnected devices with different capabilities operating on different layers. Various heterogeneous and typically constrained devices at the perception layer sense and actuate the surroundings and communicate with the PoT gateway using many communication protocols stack [100]. These communication protocol stacks depend on device capability, distance, data rate, power consumption, and environmental considerations. At the lower end of the spectrum, serial protocol stacks such as RS232 and RS485 are commonly utilized by devices for short-range communication over wires to provide reliable point-to-point or multi-drop connections, making them suitable for scenarios where devices are in close proximity to the PoT gateway [101]. As the need for broader coverage and higher data rates arises, capable communication protocol stacks like Ethernet and

Wi-Fi come into play, enabling fast and reliable communication over local area networks (LANs) and Wireless Local Area Networks (WLANs) for capable devices and making them suitable for IoT deployments within the enterprise or industrial settings [100].

The PoT gateway plays a crucial role in enabling seamless VoIP connectivity to heterogeneous IoT devices, each utilizing different communication protocol stacks. The PoT gateway has the flexibility to support multiple communication protocols, accommodating the diverse connectivity requirements of various devices within the typical IoT ecosystem. This flexibility allows for integrating devices operating on protocols like Ethernet, Wi-Fi, Zigbee, Z-Wave, Bluetooth, and more. To support multiple protocol stacks, the PoT gateway adopts different approaches. One approach involves natively supporting some protocol stacks directly on the gateway hardware. This means that the necessary hardware components, such as radio modules and transceivers, are built into the gateway itself, enabling direct communication with devices using different protocols. This native support offers seamless connectivity and simplifies the deployment process. Another approach involves using dongles or add-on modules that support other protocol stacks that are not natively supported by the PoT gateway's hardware. These dongles are plugged into the IoT gateway, expanding its compatibility with additional communication protocols. The latter approach allows for greater flexibility, as new protocols can be added or upgraded through the use of compatible dongles without requiring significant hardware changes or replacements, maintaining the system's modularity and allowing for seamless expansion and adaptation as new devices and protocols emerge in the ever-evolving IoT landscape.

On top of the communication layer between the constituent surrounding devices and the gateway, the proposed system utilizes a proposed lightweight messaging protocol, which we call *tSIP*, tailored for small code footprints, efficient data exchange, and seamless interaction with the device through the existing communication server within the premises. tSIP provides lightweight abstractions of the SIP protocol methods that can

be translated back to their original SIP messages with the help of the PoT gateway. Consequently, the PoT gateway acts as a central hub that receives tSIP messages from the perception layer devices and maps them to their corresponding SIP messages. The gateway ensures seamless integration between the IoT devices and the communication server by leveraging this mapping process, rendering devices as typical SIP endpoints in the VoIP ecosystem. Furthermore, the gateway is responsible for registering the devices to the communication server and authenticating their identity before granting their messages exchange with the communication server. This registration and authentication mechanism is implemented through the provision of a decentralized algorithm based on blockchain technology and smart contracts deployment. The proposed registration and authentication mechanism is meant to secure devices association with the communication servers while preventing the exhaustion of their resources or affecting their designed capacity, promoting the adoption of the proposed framework. Therefore, the proposed PoT framework provides a cohesive, modular, and standardized approach to facilitate communication and collaboration between IoT devices and the existing SIP-based communication infrastructure, enabling enhanced connectivity and interoperability within the IoT ecosystem.

3.2 PoT Gateway

3.2.1 PoT Gateway as an IP-PBX Server

The PoT gateway is built upon an embedded Linux platform, namely the Raspberry Pi 3 Model B+ board [102]. The PoT gateway is equipped with an open-source IP-PBX server through the installation of bare-metal Asterisk [40] software on top of the Raspberry Pi OS [103] (previously called Raspbian). Integrating Asterisk into the PoT gateway brings numerous benefits to the realm of telecommunications and smart objects:

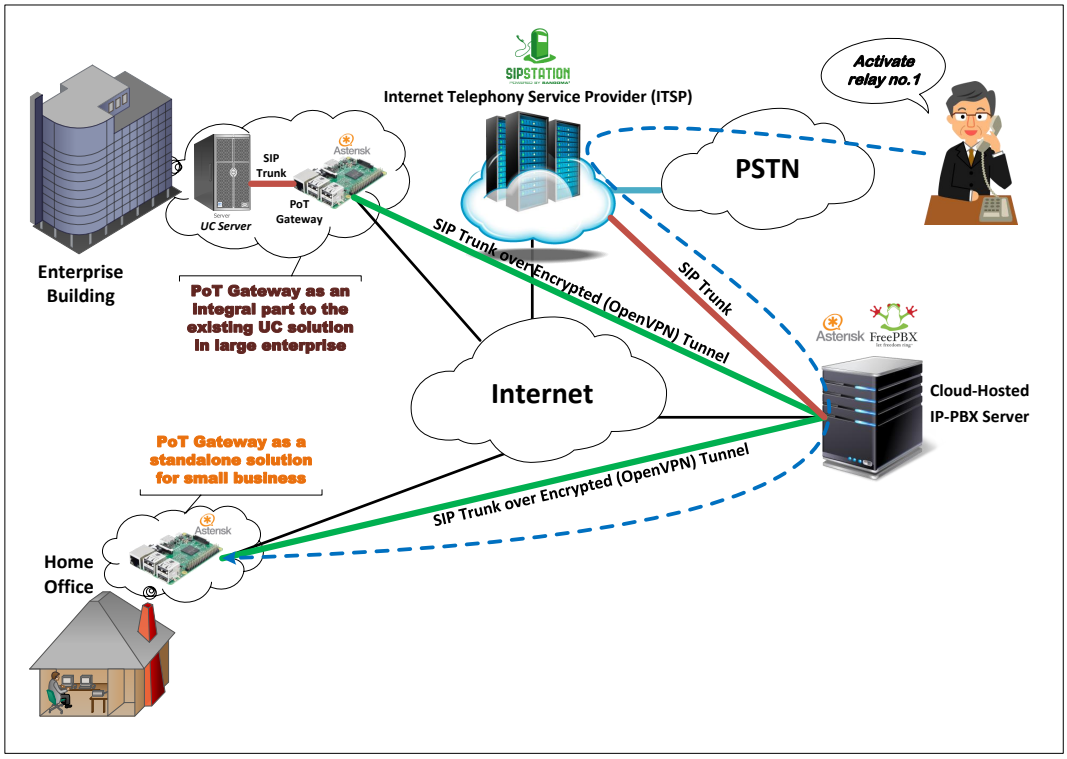

Figure 3.2: Scalability of PoT through SIP Trunks: Facilitating Multi-Site Interconnections.

- Asterisk, being an open-source telephony platform, offers flexibility and scalability to the proposed system, allowing seamless integration with IoT devices and gateways through the existing phone network infrastructure. This enables the creation of a robust and adaptable framework that can seamlessly handle diverse device types through the existing phone assets (i.e., phone sets, softphones, etc.).

- Asterisk's advanced call routing capabilities enable efficient call management and routing for IoT devices, facilitating smooth communication between devices, users, and applications.

- Asterisk's support for various telephony protocols and codecs ensures compatibility with various telecommunication systems (i.e., IP-PBX, legacy TDM PBX, PSTN, ITSP, etc.), ensuring interoperability and eliminating communication barriers.

- Asterisk's extensive feature set, including voice recognition, text-to-speech, and call

recording, enhances the functionality of PoT systems, enabling voice-controlled IoT devices and comprehensive monitoring capabilities through the mature technologies and abstractions of phone systems that people of all ages and backgrounds are already familiar with.

- Integrating Asterisk into the proposed framework to provide VoIP communication brings significant advantages, especially for homes and small-to-medium-sized businesses that may not have an IP-PBX system by default. This, combined with the utilization of powerful yet cheap SBCs, offers an affordable, scalable, and feature-rich solution for them to enjoy the advantages of modern telephony systems without the need for complex and expensive infrastructure, especially the significant investments in dedicated IP-PBX equipment.

- It allows interfacing the proposed framework with remote communication systems through standardized SIP trunks [104]. SIP trunking is a mature virtual connection that leverages the existing TCP/IP network infrastructure, ultimately the Internet, to interconnect distant communication servers. SIP trunks eliminate the use of a dedicated physical connection between communication servers, reducing the cost and eradicating wasted resources. Also, since SIP trunks are virtual connections, adding or removing trunks is seamless and dynamic, enhancing the PoT system's scalability, as illustrated in Figure 3.2.

Ultimately, integrating Asterisk into the proposed framework empowers businesses and individuals to build reliable and intelligent communication infrastructures that drive the seamless integration of surrounding IoT devices into our everyday lives.

3.2.2 PoT Gateway as an OpenVPN Client

The PoT gateway can be connected to the Internet using a wired connection utilizing the built-in Fast Ethernet port on the Raspberry Pi board. The Raspberry Pi board is

Table 3.1: Basic configuration of the VPS instance utilized by the proposed system.

Property	Configuration
Server Location	Toronto, Canada
Server Type	Linux, Ubuntu 18.04 × 64
No. of CPUs	1
RAM	1 GB
Storage	25 GB SSD
Bandwidth	500 GB
Cost	$5 USD/month

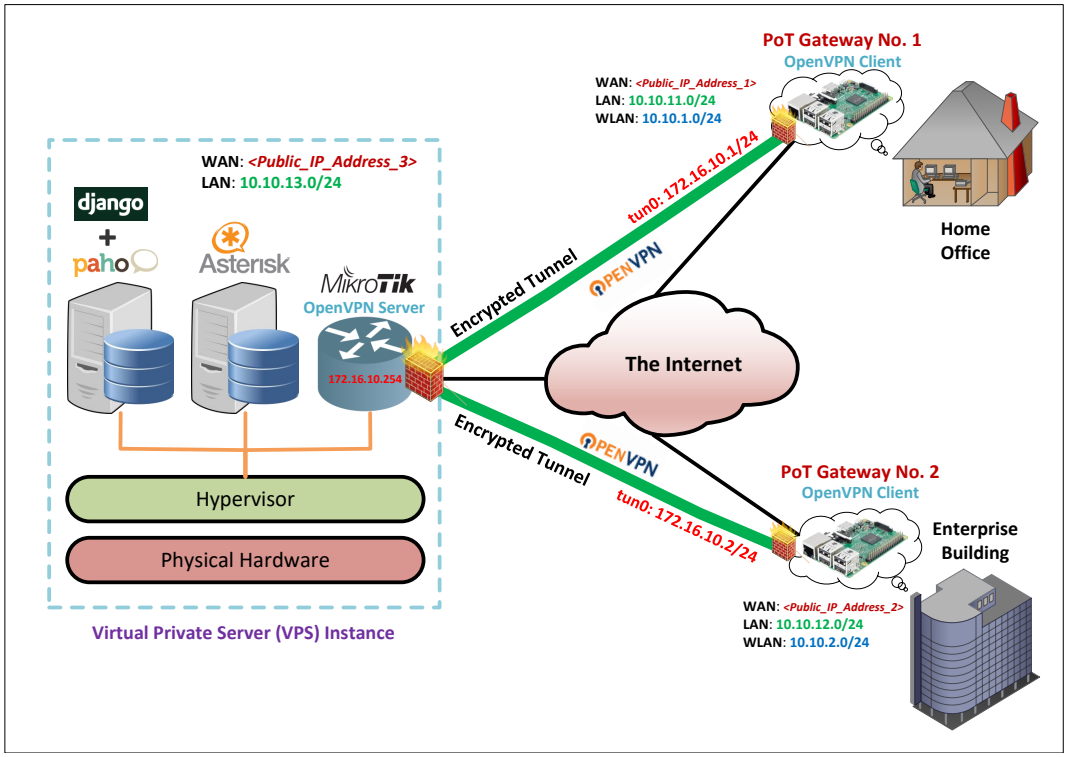

Figure 3.3: Overview of secure tunnel establishment between distant PoT gateways through VPN connection.

configured as an OpenVPN client [105]. At the startup of the board, it automatically connects through the Internet to an OpenVPN server running on a Linux-based virtual private server (VPS) instance hosted by Vultr [106], as illustrated in Figure 3.3. Table 3.1 lists the basic configuration of the VPS instance. Enabling the PoT gateway with a VPN (Virtual Private Network) connection brings several advantages, particularly in terms of secure communication and enhanced management capabilities. Integrating a VPN into the PoT gateway ensures all communications between distant gateways are encrypted, ensuring privacy and protection against unauthorized access. This is especially crucial when dealing with sensitive data or exchanging messages between distant gateways over the network. Additionally, the VPN connection enables secure remote access to the PoT gateway, allowing system administrators to manage and monitor the gateways from anywhere with an internet connection. This facilitates efficient fleet management, as administrators can remotely configure and troubleshoot devices without the need for physical access. Moreover, the VPN connection enables seamless firmware upgrades for the gateway. By securely transmitting firmware updates over the VPN, administrators can ensure that the gateways run the latest software versions with improved features, bug fixes, and security patches. Overall, enabling a PoT gateway with a VPN connection offers a robust and secure communication infrastructure, convenient fleet management, and firmware upgrade capabilities, empowering businesses and individuals to maintain a reliable and up-to-date framework.

3.2.3 PoT Gateway as a Wi-Fi Access Point

The PoT gateway is configured as an access point using the built-in dual-band WLAN interface on the Raspberry Pi board. Enabling the gateway as a Wi-Fi access point offers several benefits and holds great potential for future upgrades to the proposed framework. The gateway becomes a versatile communication hub by incorporating Wi-Fi capabilities into the gateway, providing wireless connectivity to nearby Wi-Fi-enabled devices.

This enables seamless integration and communication between Wi-Fi-enabled devices, creating a standalone, unified, and interconnected ecosystem. Moreover, leveraging the SDN (Software-Defined Network) [107] paradigm as a future upgrade direction for the proposed framework, the gateway can be dynamically managed and configured to adapt to changing network conditions. SDN allows for centralized control and programmability, enabling efficient allocation of network resources and optimizing performance. The gateway, acting as a Wi-Fi access point, can easily integrate with SDN controllers, enabling administrators to monitor and manage network traffic, apply security policies, and prioritize traffic based on specific requirements. This level of control and flexibility enhances the network's overall efficiency, scalability, and security. Additionally, combining the PoT gateway as a Wi-Fi access point with SDN paves the way for innovative applications such as location-based services, context-aware networking, and intelligent resource allocation. Ultimately, enabling the PoT gateway as a Wi-Fi access point in the context of SDN brings significant advantages in terms of seamless connectivity, efficient network management, and the potential for transformative IoT applications.

3.2.4 PoT Gateway as an MQTT Broker/Publisher

The PoT gateway acts as a message broker, namely an MQTT broker. This brings numerous benefits to the proposed framework. Firstly, it enables seamless communication and data exchange between ubiquitous and MQTT-supported IoT devices, applications, and cloud services on the one hand and the PoT gateway on the other hand, facilitating their monitoring and control through the phone network infrastructure. By serving as a centralized MQTT hub, the PoT gateway allows devices with different protocols and standards to interact seamlessly and effortlessly with the proposed framework through MQTT message exchange, facilitating its interoperability. This promotes the integration of diverse devices and systems into the proposed framework, ensuring smooth operation and efficient utilization of surrounding devices. Secondly, leveraging the PoT gateway as

a message broker enhances the scalability and flexibility of the proposed framework, as it can handle multiple connections and efficiently distribute messages across the network. This optimizes the overall performance and responsiveness of the proposed framework, enabling real-time data processing and analysis.

The PoT gateway could also publish messages to cloud-hosted real-time databases, like the Firebase Realtime database [108] from Google Cloud Platform (GCP). This database could provide a shared data space between the PoT gateways. It also acts as an external interface to potential third-party services and applications. Besides, the shared data space between the PoT gateways would coalesce into a social network of things (SNT) and support assistive services in the public sensing domain. A sample web application is implemented using a Python-based web development framework, namely Django, to monitor and control the proposed framework using an instance of the Firebase Realtime database interfaced to the PoT gateway through MQTT over WebSocket [109].

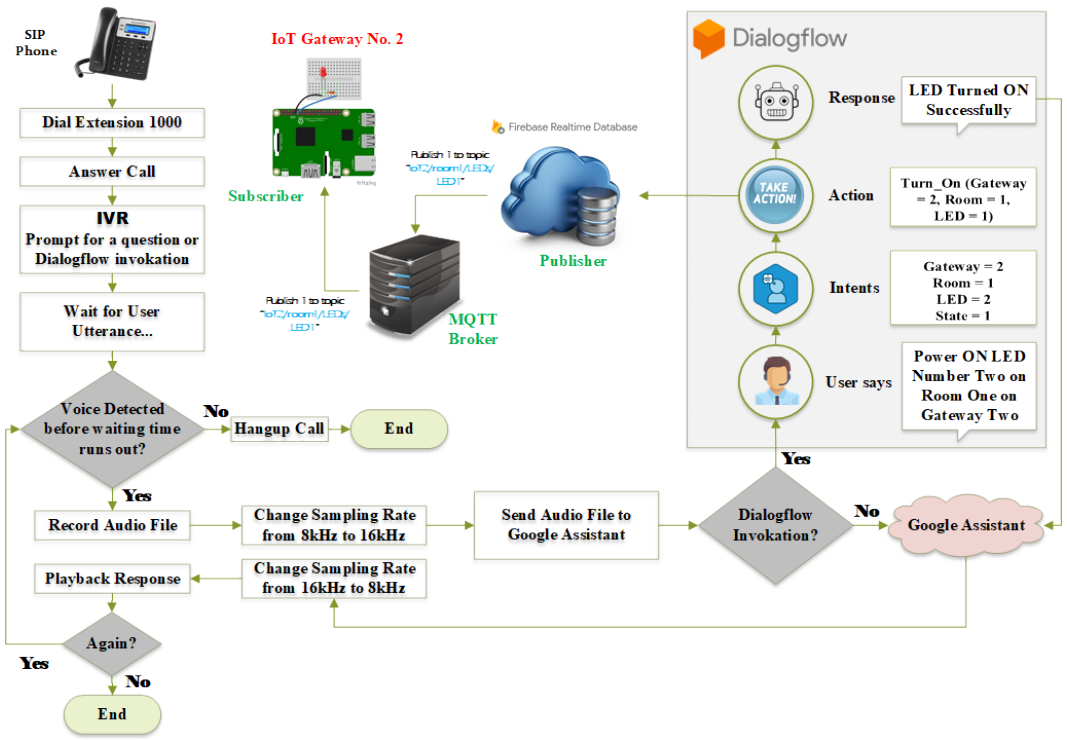

Figure 3.4: Flowchart of the integration between the embedded Asterisk server to the PoT gateway and Google's Dialogflow chatbot agent.

3.2.5 PoT Gateway and Chatbots Integration

Integrating the PoT gateway with a chatbot agent, such as Dialogflow from Google [110], offers numerous benefits and holds great potential in the realms of telephony and IoT applications. By interfacing the PoT gateway with a chatbot agent, users can leverage natural language processing and machine learning capabilities to enable interactive and intelligent communication with the surrounding gadgets. This integration allows users to interact with their IoT devices using intuitive voice commands and text messages through regular phone calls using their existing phone sets, making the interaction more user-friendly and intuitive. For telephony applications, users can make voice calls to the PoT gateway, which, in turn, interacts with the chatbot agent to perform various tasks, such as making phone calls, sending messages, creating dynamic interactive voice response (IVR) for businesses, or retrieving information from specific surrounding gadgets to answer the caller's queries. This enhances the overall telephony experience by adding intelligent, automated, and personalized features to the communication process. In terms of IoT applications, integrating the PoT gateway with a chatbot agent enables voice-controlled device management, status monitoring, and real-time notifications. Users can use voice commands to control and inquire about the status of their surrounding gadgets, facilitating seamless and efficient user engagement with the surrounding devices. This integration opens up opportunities for innovative IoT applications, such as smart home automation, voice-controlled industrial processes, personalized IoT services, and context-aware telephony solutions.

In the proposed PoT framework, the PoT gateway is interfaced with Google's Dialogflow [110] through the embedded Asterisk server on the gateway, as an example of chatbot integration with the gateway. We configured the Asterisk server running on the PoT gateway such that we built an interactive voice response (IVR) menu on a specific extension number. Upon dialling this number, the IVR instructs the caller to ask Google

Assistant a question or invoke the developed Dialogflow chatbot application using predefined word(s). Asterisk then waits for a predetermined amount of time for the user's utterance. Upon detecting the user's voice, it records the user's utterances as an audio file and saves it to the gateway. The system then changes the sampling rate of the recorded audio file from 8kHz (the sampling rate of phone calls) to 16kHz (suitable for Google Assistant), sends the file using the low-level APIs offered by Google to Google Assistant via the Internet, and waits for a reply. Upon receiving the audio file from Google Assistant, the system first changes the sampling rate of the received audio file from 16kHz to 8kHz and plays back the response to the user. The received responses may include responses to the user's query or the status of actions carried out on the associated devices with the PoT gateway. Asterisk interfacing and audio file manipulation are carried out using scripts written in the Python programming language. These scripts are invoked by Asterisk using the AGI [111]. Figure 3.4 depicts a flowchart of the abovementioned process.

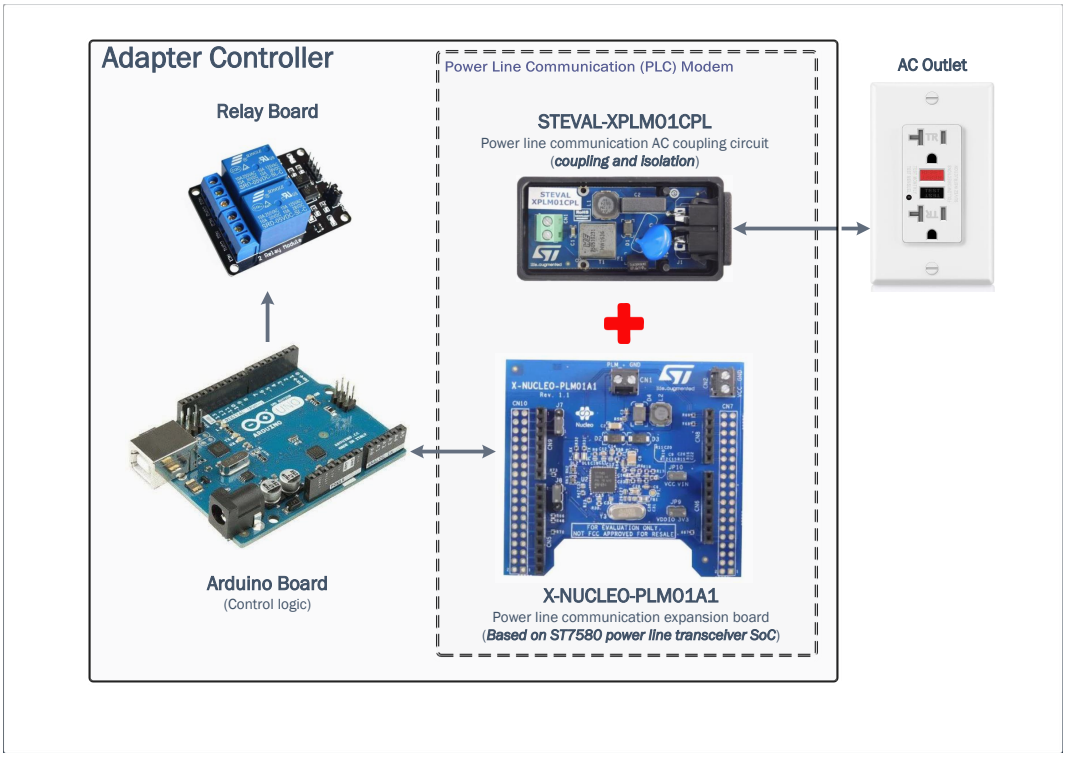

Figure 3.5: Architecture of the PLC adapter controller for the proposed PoT framework.

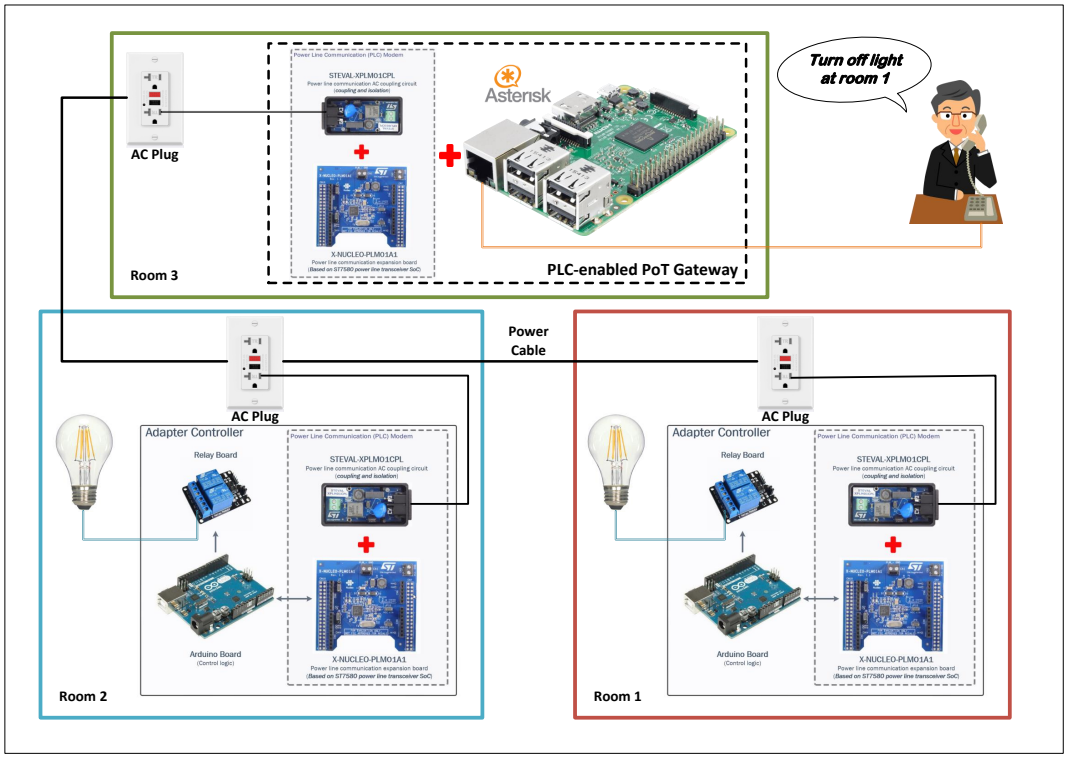

Figure 3.6: Overview of the proposed PLC-enabled PoT framework.

3.2.6 Enabling the PoT gateway with Powerline Communication (PLC) Capability

Empowering the PoT gateway with Powerline Communication (PLC) modem offers notable benefits in terms of expanding connectivity options, enhancing the reach of surrounding devices, and hardening the proposed system security. PLC utilizes existing powerlines within the premises to transmit data, effectively turning electrical wiring into a communication medium [112]. PLC highly extends the range of connected devices to the PoT gateway, including all mains-powered devices at home and in the enterprise. By integrating a PLC modem into the PoT gateway, PLC-enabled devices can utilize powerlines to establish communication links with the gateway, eliminating the need for additional wired or wireless infrastructure. This allows the conversion of billions of legacy and non-IoT-enabled devices, which are still in production and are widely used today,

to controllable objects. The range of legacy devices includes conventional light bulbs, power strips, and home appliances. In the proposed framework, this is done through the development of an adapter controller which integrates a powerline communication modem, as depicted in Figure 3.5. The adapter controller fits between the device and the mains. It connects to the PoT gateway, which utilizes a PLC modem, through the existing powerlines, as illustrated in Figure 3.6.

PLC opens up new possibilities for connecting devices in areas where traditional wireless signals may have limitations or face interference, like industrial settings. PLC-enabled devices can seamlessly communicate and exchange data over the powerlines, enabling a wider range of IoT applications, including smart home automation, energy management, and industrial control systems. Furthermore, PLC communication provides robustness and reliability, as powerlines are typically stable and offer consistent connectivity. This integration improves connectivity in environments with unreliable wireless signals or congestion. Moreover, PLC offers several security benefits when integrated into the proposed framework. One of the key advantages is the inherent encryption. PLC modems often utilize encryption algorithms to protect data transmitted over powerlines, ensuring that sensitive information remains secure. This encryption provides an additional layer of privacy, preventing unauthorized access and eavesdropping on data transmissions. Additionally, PLC systems typically operate within a closed network, limiting access to authorized devices connected to the same powerlines. This isolation reduces the risk of external attacks and unauthorized access attempts. Moreover, PLC communication is less susceptible to wireless signal interception or interference as it utilizes the existing power infrastructure. This makes it more challenging for potential attackers to tamper with or disrupt the communication flow. Furthermore, combined with other security measures of the proposed framework, such as extension number-based ACL, PLC helps to enhance the overall system security. Therefore, with proper configuration and monitoring, potential vulnerabilities of the proposed system are largely mitigated and

can be identified and addressed promptly.

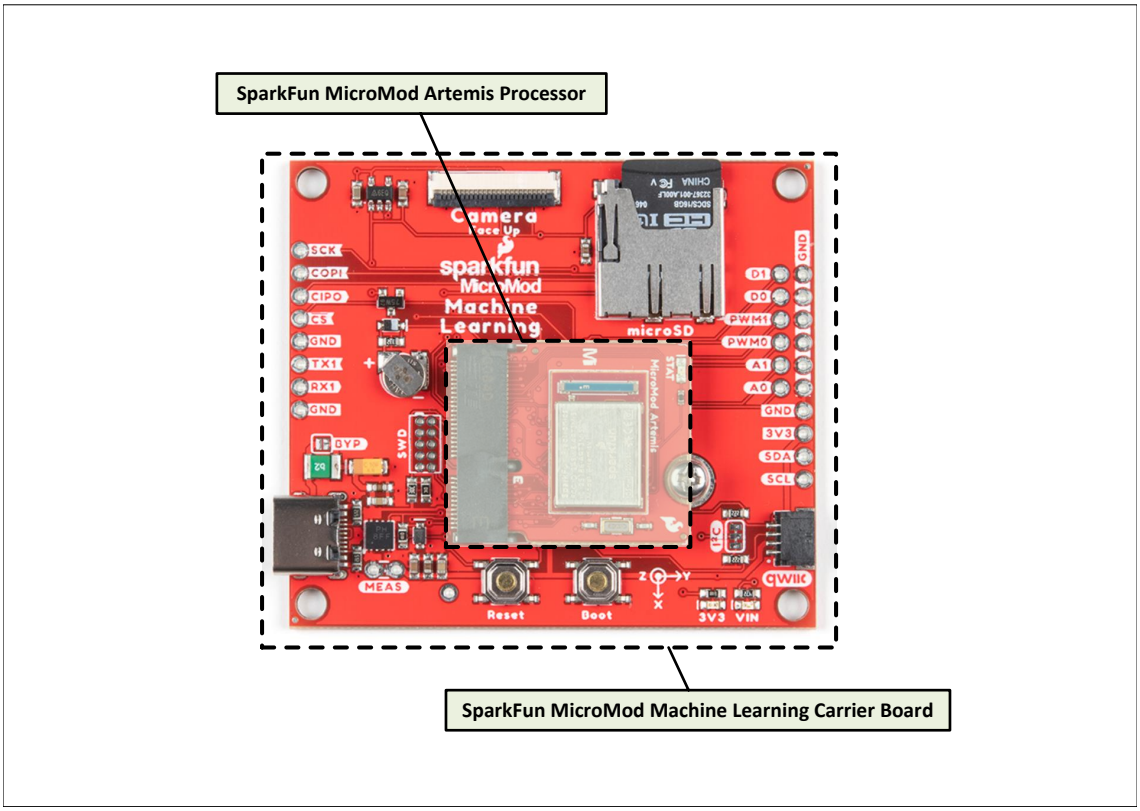

Figure 3.7: SparkFun's TinyML development board (MicroMod Artemis Processor board).

3.2.7 Enabling the PoT Gateway with Tiny Chatbot Agent Capability

Enabling the proposed framework with a tiny chatbot agent using TinyML technologies offers several compelling benefits to the framework. TinyML refers to deploying machine learning models on resource-constrained devices, such as microcontrollers and SBCs [25]. By integrating tiny chatbot agents into the PoT gateway, users can directly experience localized, fast, and intelligent communication with the gateway. The tiny chatbot agent embedded in the gateway leverages the power of TinyML to process natural language and deliver personalized responses in real time, decoupling the proposed framework's dependence on the Internet to be fully functional. This local processing

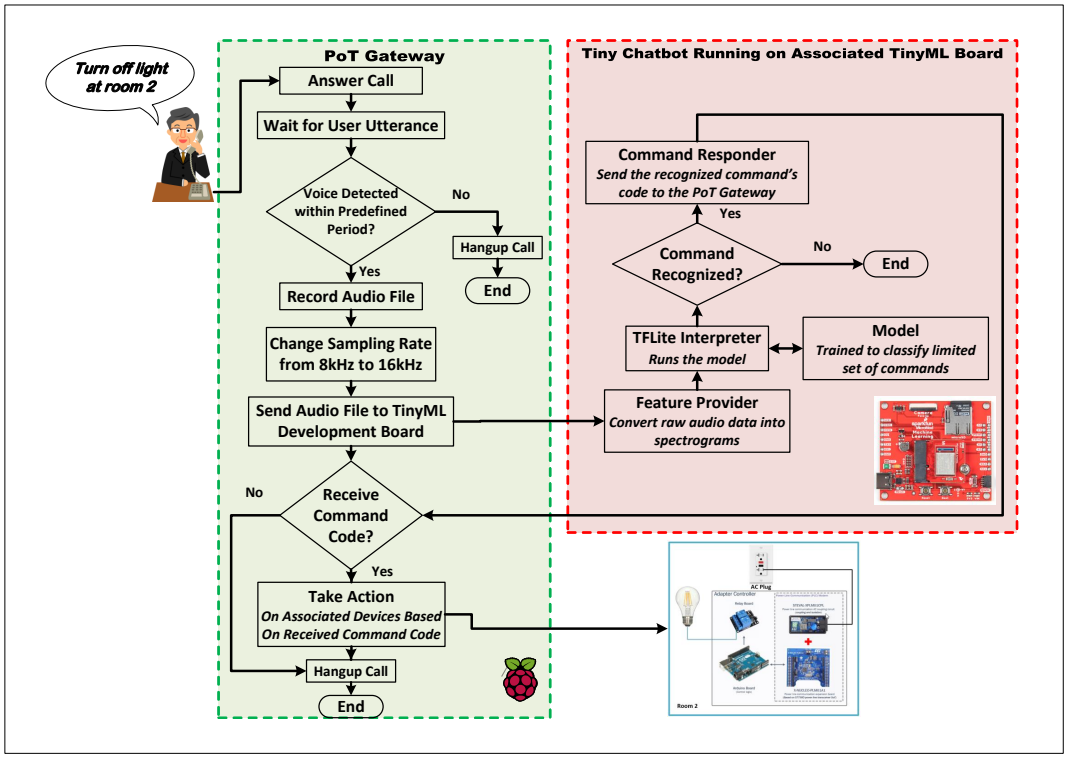

Figure 3.8: Flowchart of tiny chatbot embedding in the PoT gateway using TinyML.

reduces reliance on cloud services, enabling faster response times, enhancing privacy, and improving the offline capabilities of the proposed framework. Furthermore, deploying a tiny chatbot agent, suitable for IoT applications with limited command sets, on the gateway optimizes its resource utilization, minimizing its power consumption and memory requirements. This allows for efficient operation of the gateway, extending its battery life and reducing the operational costs of the framework. Figure 3.7 gives an example of a TinyML module, namely Sparkfun's MicroMod Artemis Processor board, plugged into Sparkfun's MicroMod Carrier Board. Figure 3.8 shows the flow chart that illustrates embedding a tiny chatbot into the PoT gateway.

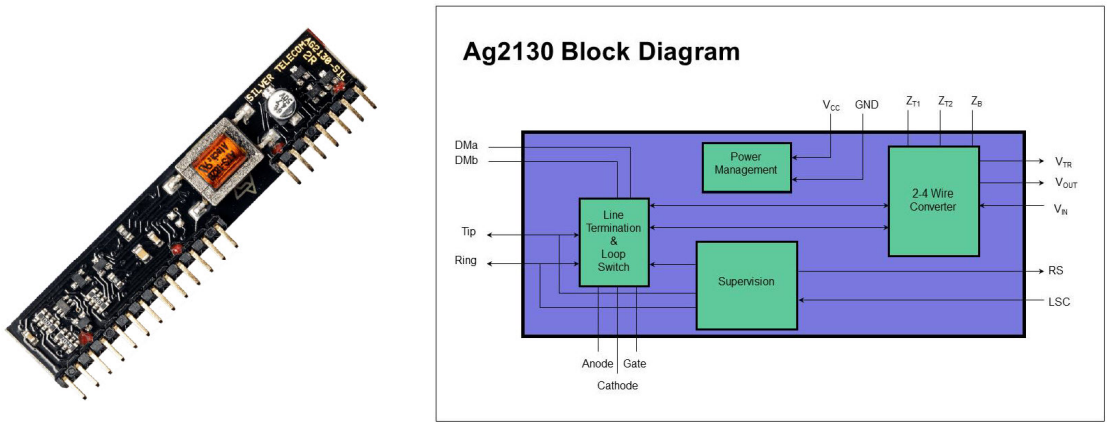

Figure 3.9: Silvertel's Ag2130 PSTN interface module.

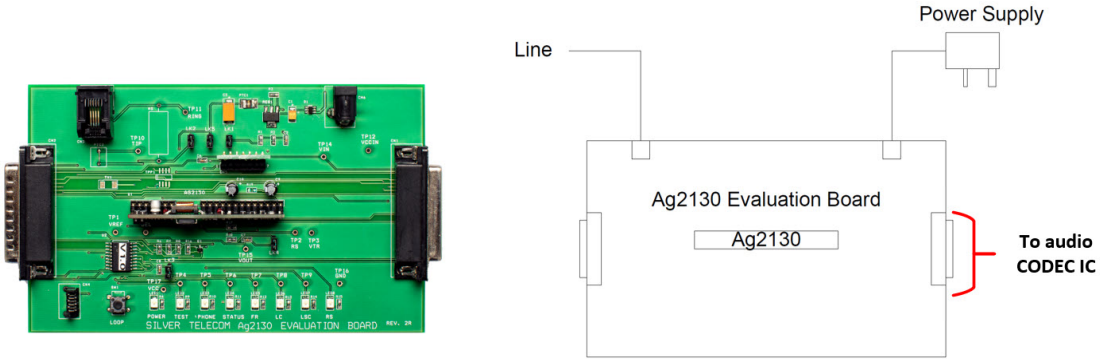

Figure 3.10: Silvertel's Ag2130 evaluation board.

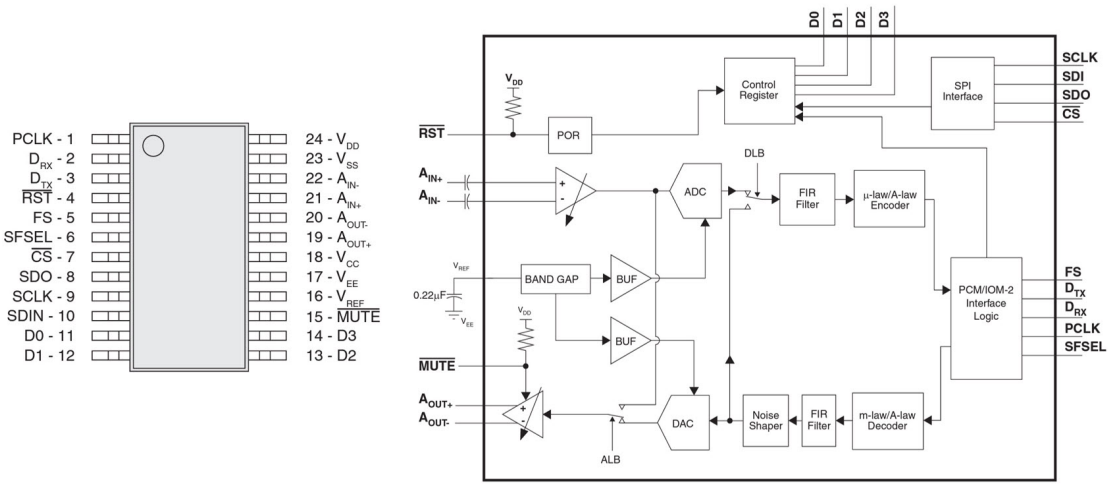

Figure 3.11: CPC5750 single-channel voice band codec IC.

3.2.8 Enabling the PoT Gateway with PSTN Interface Circuitry

Integrating a PSTN (Public Switched Telephone Network) interface circuit into the PoT gateway offers significant benefits in terms of interoperability and enhanced communication capabilities. The PSTN interface circuit enables seamless connectivity and communication between legacy telephony systems (i.e., TDM PBX and PSTN) and the proposed framework. By integrating PSTN support, the PoT gateway can bridge the gap between legacy and modern telephone networks, allowing framework reachability from both traditional landline phones and modern communication servers. This integration enhances the versatility and accessibility of the PoT system, enabling users to leverage existing legacy telephone infrastructure while incorporating the benefits of IP-PBX functionalities integrated into the gateway. Moreover, the PSTN interface circuit facilitates reliable and high-quality voice communication by connecting to the robust and well-established PSTN network. This ensures compatibility with various telephony services, including emergency calls. Additionally, integrating the PSTN interface circuit with the PoT gateway offers the potential for advanced features such as call routing, voicemail, and call forwarding to the pervasive PSTN using the existing and rarely used landline phones at homes without the need for third-party SIP trunking services like ITSP, reducing the deployment cost of the proposed framework.

The PSTN interface circuit embedded in the PoT gateway is built around a PSTN interface module from Silvertel, namely the Ag2130 [113], shown in Figure 3.9. Ag2130 is a self-contained, highly integrated PSTN interface circuit in a single in-line (SIL) module. Ag2130 connects to the analogue phone line utilizing an evaluation board for the module, shown in Figure 3.10, with all the proper impedance and isolation. The input and output analogue audio pins from the Ag2130 are connected to the CPC5750, a single-channel voice band audio codec integrated chip (IC) shown in Figure 3.11, for digitization. The

gateway controls different settings of the audio codec IC through the built-in SPI (Serial Peripheral Interface) interface of the gateway's SBC. In contrast, the audio data is sent and received from the audio codec IC using the PCM pins of the gateway's SBC.

3.3 PoT Devices

The PoT devices in the proposed system include all connected objects to the PoT gateway for developing and testing the proposed PoT framework as a proof-of-concept. Unlike IoT devices, PoT devices do not necessarily implement the TCP/IP protocol stack to connect to the PoT gateway. Hence, PoT devices are a superset of IoT devices. The PoT gateway provides different, less processing-intensive communication protocol options for simple devices with processing or power consumption constraints. The employed PoT devices in the proposed framework include:

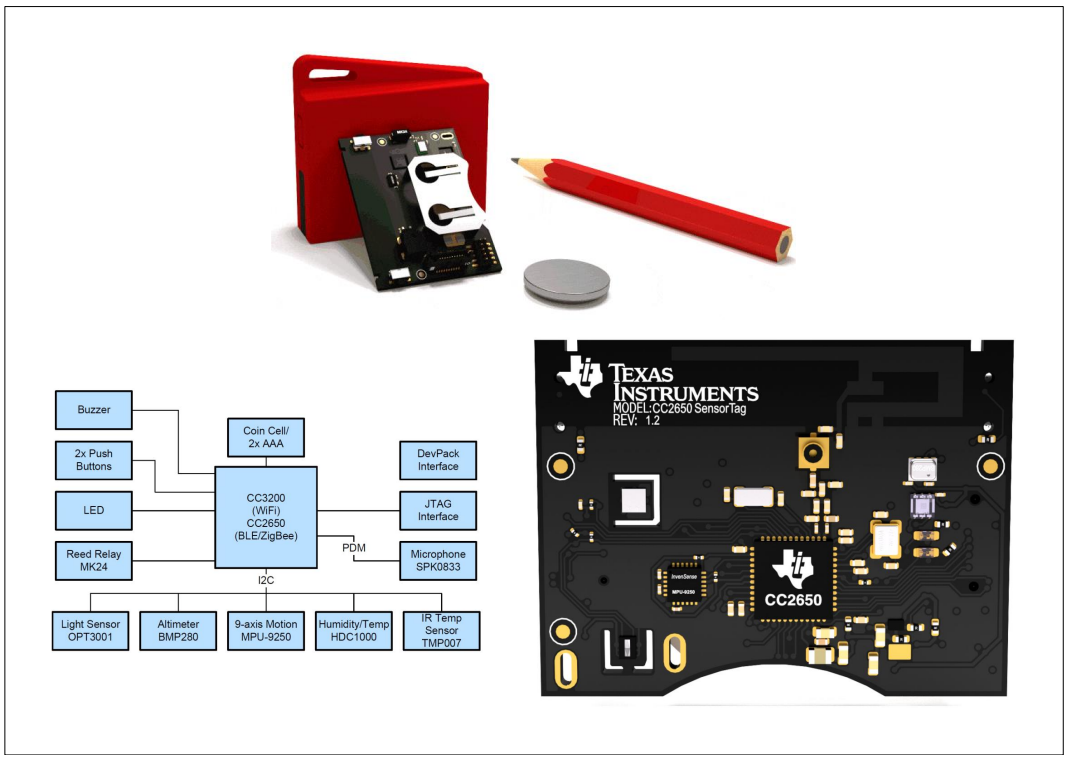

Figure 3.12: Multi-standard CC2650 SensorTag from Texas Instruments (TI).

- **CC2650 SensorTag**: It is a multi-standard IoT device from Texas Instruments (TI) that offers a multitude of functionalities and features, shown in Figure 3.12. Equipped with a wide range of sensors such as temperature, humidity, accelerometer, gyroscope, magnetometer, and ambient light sensor, the SensorTag provides comprehensive environmental data about its surroundings for monitoring and analysis [114]. To interface the SensorTag with the PoT gateway, we utilize the Bluetooth capabilities of both devices, namely BLE. The Raspberry Pi supports BLE connectivity, allowing it to establish a connection with the SensorTag. We establish a communication link between the two devices by installing the necessary Bluetooth software and libraries to the PoT gateway. Once connected, the PoT gateway can access the SensorTag's sensor data by reading and parsing the Bluetooth GATT (Generic Attribute) profiles exposed by the SensorTag. These profiles provide access to the sensors on the SensorTag, such as temperature, humidity, accelerometer, etc. Using Python programming language, we developed scripts on the PoT gateway to retrieve data from the SensorTag's sensors. The gateway invokes These scripts autonomously to respond to specific user queries through phone calls.

- **M5Stack**: It is an innovative and versatile development platform offering a compact, all-in-one solution for building IoT prototypes and projects. Its compact form and rechargeable battery make it highly portable and ideal for on-the-go prototyping. The M5Stack platform features a modular design, combining an ESP32-based microcontroller with various stackable modules, including displays, sensors, communication modules, and input devices. This modular approach allows for easy customization throughout the development of tSIP, and the proposed registration and authentication mechanism of the devices to the PoT gateway, as delineated in Chapter 4. Additionally, M5Stack supports various programming languages, such as MicroPython [115] and Arduino, facilitating its development cycle. The M5Stack can seamlessly connect to the PoT gateway with its onboard Wi-Fi and Bluetooth

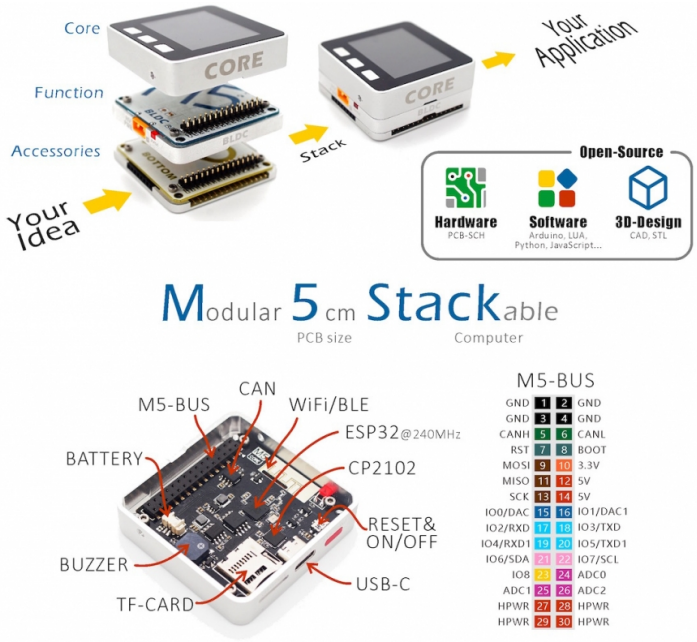

Figure 3.13: M5Stack: Core2 ESP32 IoT development kit.

capabilities.

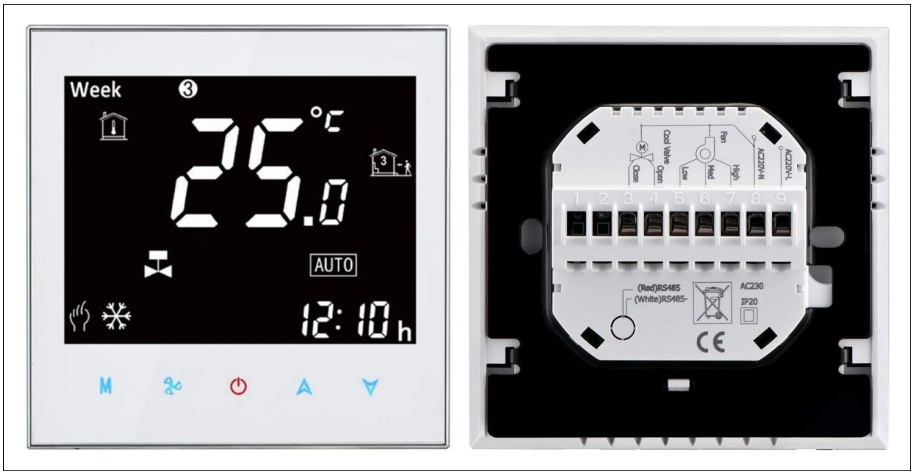

Figure 3.14: ModBus thermostat.

- **ModBus device**: The Modbus protocol, specifically Modbus RTU or Modbus TCP, provides a standardized and efficient method for monitoring and controlling devices. Modbus is crucial in building automation systems (BAS), providing a standardized and reliable communication protocol for seamlessly integrating diverse devices and systems. In a BAS, various components such as sensors, actuators,

controllers, and supervisory software must exchange data to monitor and control building systems effectively. Modbus, with its simplicity and widespread adoption, facilitates this data exchange by enabling interoperability between different devices and vendors. For example, Modbus can connect temperature sensors, HVAC controllers, lighting systems, energy meters, and more, allowing the BAS to monitor and adjust environmental conditions, optimize energy usage, and ensure occupant comfort. By implementing Modbus capability to the PoT gateway, the proposed framework can efficiently gather data from distributed devices on the premises and provide centralized control and monitoring to these devices through the existing phone network infrastructure. We designed a ModBus driver for the PoT gateway to facilitate integration with ModBus devices. As proof of concept, we interface the gateway to an RTU ModBus temperature sensor. To integrate the temperature sensor with the PoT gateway, we install a USB-to-RS485 dongle to the PoT gateway to connect it to the temperature sensor, install the `pyserial` library to enable Modbus communication in the gateway and write Python code to establish communication with the Modbus device. We can read and write data from/to the ModBus resisters in the thermostat to respond to user commands and queries invoked through a phone call regarding the sensor.

- **Relay board**: Integrating a relay board into the PoT gateway offers the proposed framework advanced control and automation capabilities. A relay board consists of multiple relays that can be controlled individually and electronically, enabling the PoT system to switch and control external devices or circuits. Leveraging relay board integration with the PoT gateway, users can remotely control electrical appliances, lights, motors, or other electronic devices. This integration allows for implementing advanced automation scenarios and intelligent control of various systems as future work for the proposed PoT framework. For example, in a smart home context, integrating a relay board enables the PoT system to control lights,

HVAC systems, or security devices, providing convenience, energy savings, and enhanced security. Moreover, relay boards offer flexibility and expandability, as they can handle multiple relays and support different voltages and currents. This allows for the customization and scalability of the PoT system based on specific needs and requirements. By integrating a relay board into the PoT infrastructure, users can remotely control and automate a wide range of devices, enhancing convenience, efficiency, and the overall functionality of the connected ecosystem to the framework.

3.4 Summary

This chapter presents the proposed Phone of Things (PoT) framework. The PoT framework is proposed as a paradigm shift for IoT system design. It extends the accessibility options for the surrounding devices within the premises through the ubiquitous phone network infrastructure and assets. This provides seamless interaction with the surrounding objects through mature phone technologies whose abstractions are recognized by all people of different ages, enhancing user engagement with the surrounding objects.

The chapter begins by discussing the role of the PoT gateway within the framework and its different configuration possibilities. It explores the concept of the PoT gateway as an IP-PBX server, OpenVPN client, Wi-Fi access point, and MQTT broker/publisher. This is intended to highlight the wide range of configuration possibilities regarding the gateway to accommodate various PoT application needs, thanks to the modular design philosophy of the gateway.

The chapter also explores integrating the gateway with chatbot agents for innovative telephony applications and the possibility of a tiny chatbot embedding within the PoT gateway using TinyML technology, obviating the need for Internet access to be fully functional. The chapter also overviews integrating a powerline communication (PLC)

modem with the gateway to manage mains-powered devices by simple phone calls using the PoT gateway through the existing powerline cables. The chapter also reviews interfacing the gateway to PSTN and legacy TDM-PBX systems by developing an FXO circuit interfaced with the gateway, emphasizing the importance of interoperability and compatibility.

The chapter then delves into the types of devices that can be associated with the PoT gateway, highlighting the wide range of possibilities, including sensors, actuators, wearables electronics, and BAS devices. Each device type is explained in detail, discussing its functionalities and potential applications within the framework.

Overall, the proposed PoT framework chapter provides a comprehensive understanding of the gateway's role, the diverse range of devices that can be integrated, and the various configuration options available within the framework. This knowledge lays the foundation for the subsequent chapters, where the framework's implementation, testing, and evaluation will be discussed in greater detail.

Chapter 4

PoT Framework Implementation

This chapter focuses on the implementation details of the proposed PoT (Phone of Things) framework introduced in the previous chapter. It delves into the technical aspects and provides a comprehensive development process overview. The implementation phase transforms the theoretical framework into a tangible and functional system. This chapter will explore the tools, technologies, and methodologies to realize the proposed framework. We start with a feasibility study of embedded Linux SBCs as a suitable candidate to act as PoT gateways in the proposed framework. This allows for efficient development and deployment of PoT applications based on these tiny and cost-effective SBCs, as developers can leverage the rich ecosystem of Linux-based software and frameworks while optimizing the deployment cost of the framework. Additionally, embedded Linux platforms offer excellent scalability, enabling them to handle diverse PoT requirements and accommodate varying computational power and memory levels.

The chapter then focuses on the two main contributions of the thesis to realize PoT. Firstly, it starts with developing a lightweight messaging protocol based on the SIP (Session Initiation Protocol) protocol, called tSIP, for PoT applications. tSIP allows constrained devices to act as typical SIP endpoints in the VoIP ecosystem, facilitating their command and monitoring through communication servers. Secondly, the chapter

goes through the development of a lightweight registration and authentication mechanism based on blockchain technology. The proposed mechanism provides a robust and secure association of surrounding devices with the existing communication server within the premises without exhausting its resources.

This chapter aims to provide a detailed account of the development process, offering insights into the practical considerations, challenges faced, and rationale behind the chosen implementation approaches. The information presented here will serve as a valuable resource for understanding the inner workings of the framework and pave the way for the subsequent chapter, where the evaluation and performance analysis of the implemented system will be discussed.

4.1 Feasibility Study

4.1.1 Introduction

A feasibility study is crucial in determining the viability and success of implementing embedded Linux SBCs as PoT gateways in the proposed framework. This study helps to assess the practicality and potential challenges associated with this implementation. By conducting a thorough feasibility study, various aspects are carefully evaluated, including technical requirements, the maximum number of simultaneous VoIP calls the utilized SBC can gracefully handle, and the recommended configurations of the embedded Asterisk server within the PoT gateway for different application scenarios, ensuring better system responsiveness and standardized VoIP call quality metrics. This helps identify potential risks, limitations, and possible solutions to ensure a seamless integration and efficient operation of the PoT gateway as the system scales. The feasibility study also sets realistic expectations for a successful implementation.

4.1.2 VoIP Codecs

VoIP Codecs are essential components in the VoIP ecosystem. Codecs are responsible for encoding and compressing voice signals into digital packets for efficient transmission over IP networks [116]. They also handle the decoding process at the receiving end to restore the voice signals to their original form. VoIP codecs are critical in determining the audio quality, bandwidth utilization, and overall performance of VoIP calls. Various codecs are available, each with its own characteristics, trade-offs, and compression algorithms [116].

G.711 audio codec is a widely used codec that provides excellent audio quality but consumes higher bandwidth due to its uncompressed nature. *G.729*, on the other hand, is a low-bit-rate codec, typically utilized by ITSPs, that offers good voice quality while significantly reducing bandwidth requirements. When conducting a feasibility study in VoIP, it is beneficial to consider both the *G.711* and *G.729* audio codecs due to the following reasons:

- **Audio Quality vs. Bandwidth Efficiency**: *G.711* offers superior audio quality as it transmits voice signals in an uncompressed format. This makes it ideal for scenarios where audio fidelity is a priority, such as in professional environments or when maintaining call clarity is crucial. On the other hand, *G.729* is a low-bit-rate codec that provides reasonable audio quality while significantly reducing bandwidth requirements. Considering both codecs allows for evaluating the trade-off between audio quality and bandwidth efficiency based on the specific needs and limitations of the VoIP deployment.

- **Bandwidth Considerations**: Bandwidth availability and limitations are key factors in VoIP implementations. *G.711* consumes higher bandwidth due to its uncompressed nature, making it suitable for scenarios with abundant network resources. *G.729*, being a compressed codec, optimizes bandwidth usage and is ideal for environments with limited bandwidth or when transmitting calls over low-bandwidth

connections, such as communication with ITSPs. By evaluating both codecs, the feasibility study can determine the codec that best aligns with the available network infrastructure and the desired quality of service.

- **Compatibility and Interoperability**: *G.711* and *G.729* are widely supported and have broad compatibility across VoIP systems, devices, and networks. Both codecs ensure compatibility with different endpoints, softphones, and SIP-based systems. It allows for seamless communication and interconnectivity between diverse devices and networks, enhancing the feasibility and versatility of VoIP deployment.

- **Scalability and Cost**: Scalability is important when planning VoIP deployments. *G.729* requires fewer network resources due to its compression algorithm, making it more scalable for large-scale implementations where the number of simultaneous calls is high. This scalability can have cost implications, such as reduced infrastructure requirements and potential cost savings on bandwidth usage. By evaluating *G.729* alongside *G.711*, the feasibility study can assess the scalability and cost-effectiveness of the VoIP solution.

Therefore, considering *G.711* and *G.729* codecs in the feasibility study allows for a comprehensive evaluation of the trade-offs between audio quality, bandwidth efficiency, compatibility, scalability, and cost. It ensures that the chosen codec aligns with the specific requirements and constraints of the VoIP deployment, ultimately leading to an informed decision on the feasibility and viability of the project.

4.1.3 Passthrough vs. Transcoding VoIP Calls

Passthrough and transcoding VoIP calls represent two distinct approaches to handling voice communication over IP networks, each with advantages and considerations [117].

In passthrough VoIP calls, the voice data is transmitted without modification or compression. This method is advantageous as it preserves the original audio quality and minimizes latency since the voice packets are transmitted as-is. However, passthrough calls require high network bandwidth, making them more suitable for scenarios where bandwidth is abundant, such as local networks or dedicated VoIP connections.

On the other hand, transcoding VoIP involves compressing the voice data to reduce bandwidth requirements. This compression can be lossy, resulting in a slight degradation of audio quality. However, transcoding reduces bandwidth consumption, making it a more practical choice for environments with constrained network resources or when transmitting calls over the Internet. Transcoding also allows for compatibility between different VoIP protocols or codecs, enabling seamless communication between various devices or VoIP systems.

Transcoding VoIP typically requires more processing capability than passthrough calls due to the additional computational tasks involved. When transcoding, the voice data must be decoded from the source codec, processed or modified as needed, and then encoded into the target codec before transmission. This involves significant computational overhead and requires specialized hardware or software resources to handle the transcoding operations efficiently. The more intensive the transcoding operation (e.g., converting between high-complexity codecs or handling multiple simultaneous transcoding calls), the higher the processing requirements. Transcoding also introduces additional latency to the VoIP call due to the decoding and encoding processes. This latency can impact real-time communication and may require appropriate buffering mechanisms to mitigate any adverse effects on call quality. Dedicated hardware components or specialized software solutions can be employed to address the increased processing demands of transcoding. Hardware solutions like digital signal processors (DSPs) or application-specific integrated circuits (ASICs) can offload the transcoding tasks from the main processor, enabling efficient and scalable processing. Software-based transcoding can leverage multi-core pro-

Table 4.1: Acceptable VoIP call quality metrics set by Cisco.

Metric	Acceptable Value
Jitter	<30 msec
Packet Loss	<1%
RTT	<300 msec

cessors or utilize specialized libraries and frameworks optimized for audio processing. The system's processing capability is important when implementing transcoding in a VoIP infrastructure. Sufficient processing power and appropriate hardware or software resources should be allocated to handle the transcoding workload effectively, ensuring smooth, high-quality voice communication.

Choosing between passthrough and transcoding VoIP calls depends on the specific requirements of the communication system. Passthrough is ideal when maintaining high audio quality is critical, and bandwidth is not a constraint. Conversely, transcoding balances audio quality and bandwidth efficiency, making it suitable for conserving network resources. Ultimately, the decision should be based on available bandwidth, network conditions, and the desired audio fidelity for the VoIP communication system.

4.1.4 VoIP Call Quality Metrics

VoIP call quality metrics are essential for assessing and monitoring voice communication performance over IP networks [118]. These metrics provide objective measurements that help evaluate the overall call quality and user experience. Common VoIP call quality metrics include:

- **MOS (Mean Opinion Score)**: MOS is a subjective measurement representing a call's overall perceived voice quality. It is typically rated on a scale from 1 to 5, with 5 being excellent quality. MOS is obtained by conducting subjective listening tests where human listeners rate the call quality based on factors such as clarity, background noise, and distortion. MOS provides a reliable indication of the user's

satisfaction with the call quality.

- *Jitter*: Jitter refers to the variation in the arrival time of voice packets at the destination. It is caused by network congestion, packet loss, or inconsistent network conditions. High jitter can result in choppy or distorted voices during calls. Monitoring and measuring jitter help identify network issues that impact call quality and enable appropriate measures to mitigate its effects.

- *Packet Loss*: Packet loss occurs when voice packets are not delivered to the destination successfully. It can lead to gaps or interruptions in the conversation, affecting call quality. Measuring packet loss helps identify network issues or congestion that may require troubleshooting or network optimization to minimize its impact on voice calls.

- *Latency*: Latency, often called delay, is the time it takes for voice packets to travel from the source to the destination. Excessive latency can cause noticeable conversation delays, leading to natural and interactive communication difficulties. Monitoring and managing latency help ensure real-time voice transmission and a seamless user experience.

By monitoring and analyzing these VoIP call quality metrics, designers can identify and address issues that impact call performance, enhance the user experience, and optimize the VoIP infrastructure for consistent and high-quality voice communication. Cisco, a leading networking and communication solutions provider, has established a set of acceptable VoIP quality metrics that serve as industry benchmarks [119]. These metrics define the desired thresholds for various parameters contributing to a satisfactory VoIP call experience. For example, Cisco's acceptable VoIP quality metrics often include jitter, packet loss, and round-trip time (RTT) criteria. Table 4.1 lists the acceptable VoIP call quality metrics set by Cisco. It shows that *jitter* is typically expected to be *less than 30 milliseconds* to ensure smooth and consistent voice transmission. To maintain audio

continuity, *packet loss* is typically *less than 1%*. *RTT*, which measures the round-trip time between the source and destination, must be *less than 300 milliseconds* to minimize delays and maintain real-time communication. These metrics established by Cisco provide a framework for evaluating and maintaining the quality of VoIP calls, ensuring optimal performance and user satisfaction in communication networks.

4.1.5 Motivation

When conducting a feasibility study of using Raspberry Pi boards as Asterisk servers in the proposed PoT framework, testing both passthrough and transcoding scenarios to comprehensively evaluate the gateway's capabilities is crucial. Testing passthrough scenarios allows for assessing the Raspberry Pi's ability to handle VoIP calls without any codec conversion or modification. This provides insight into the board's call routing, signalling, and basic voice transmission performance. On the other hand, testing transcoding scenarios allows for a more in-depth analysis of Raspberry Pi's processing capacity and efficiency. It helps determine how well the gateway can handle the additional computational load of decoding and encoding voice data between different codecs, providing a clearer picture of its performance under more demanding conditions.

The feasibility study can identify any limitations or bottlenecks in Raspberry Pi's processing capabilities by testing both passthrough and transcoding scenarios. It helps determine if the device can handle the expected call volumes and concurrent transcoding tasks and provide satisfactory voice quality. Additionally, this approach enables the evaluation of resource allocation, such as CPU utilization, memory usage, and network bandwidth requirements, to ensure that the Raspberry Pi as Asterisk server operates within acceptable limits.

Testing both scenarios also helps assess the scalability of the system. It provides insights into the number of concurrent calls the Raspberry Pi can gracefully handle in passthrough and transcoding modes. This information is valuable in determining if the

embedded Asterisk server to the PoT gateway can meet the projected demands of a VoIP infrastructure and whether additional hardware resources or optimization techniques are necessary for future scalability.

Testing both passthrough and transcoding scenarios during the feasibility study of Raspberry Pi boards as Asterisk servers in PoT applications allows for a comprehensive evaluation of its performance, capacity, and suitability for the intended use case scenario. It also helps identify potential limitations or performance issues, allowing PoT system designers to make informed decisions and determine the feasibility of effectively using the SBCs as gateways in the proposed framework.

4.1.6 Methodology

In the following subsections, we evaluate the performance of Raspberry Pi boards that run Raspberry Pi OS when acting as IP-PBX servers using the open-source Asterisk 16 IP-PBX installed on top of the Raspberry Pi OS. This comparative study promotes tiny and cost-effective embedded Linux SBCs as suitable candidates for PoT applications in home and small- to medium-sized business domains or as an integral part of the UC solutions in large enterprises to empower them with PoT capability.

We gauge the resource utilization and the operating system responsiveness of different Raspberry Pi boards with differentiated capabilities while programmatically initiating an increasing number of simultaneous VoIP calls. Both passthrough and transcoded VoIP calls are implemented, tested, and contrasted to give a comprehensive overview of the boards' performance at different VoIP setups. We also monitor the VoIP call quality metrics, namely, jitter and packet loss, throughout each scenario to ensure it falls within the acceptable limits set by Cisco.

We utilize the following Python libraries in the developed Python testing scripts:

- `pycall`: It is used for creating and using Asterisk files, and it enables programmatic origination of VoIP calls through the testing script

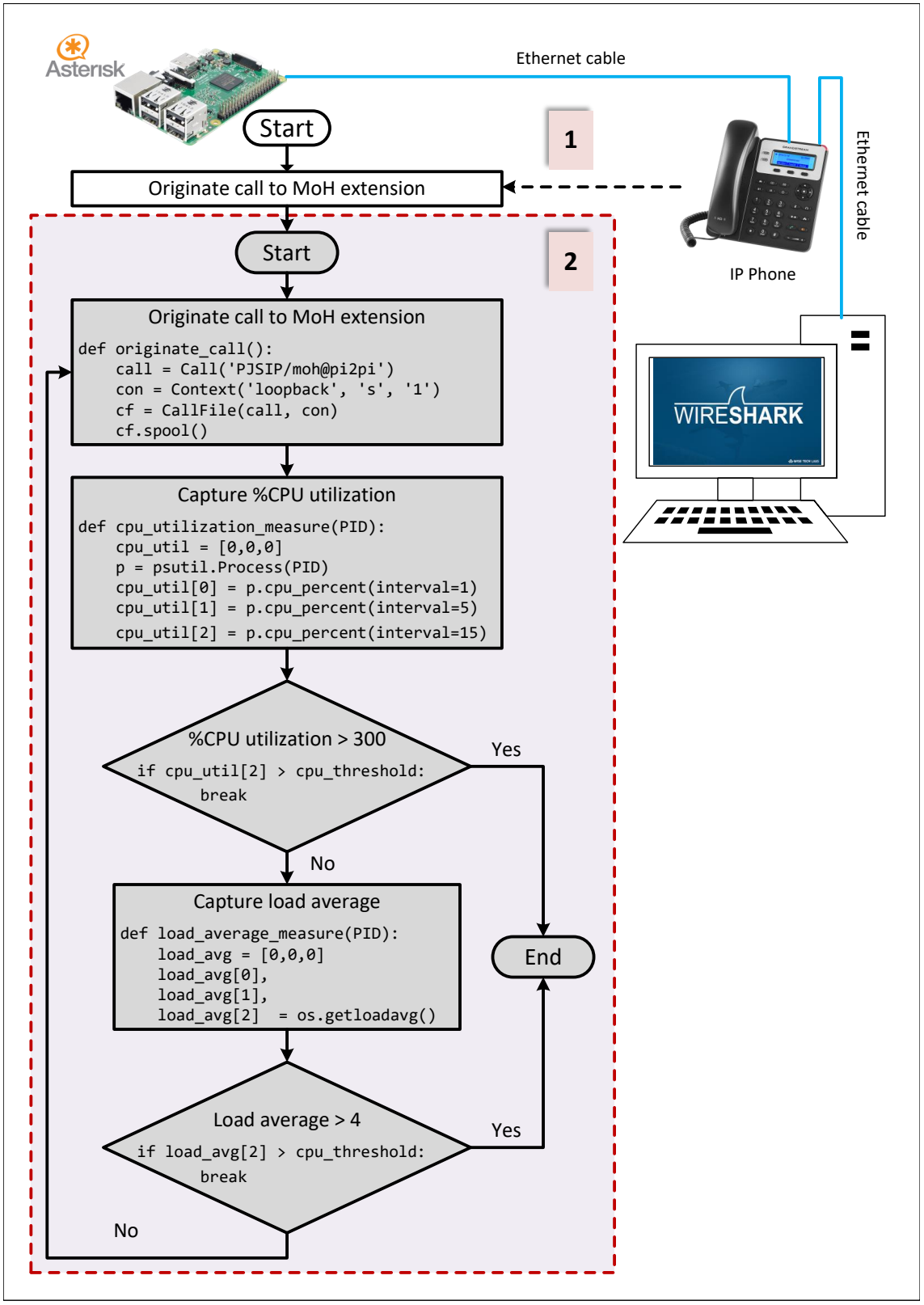

Figure 4.1: Flowchart of the evaluation methodology for SBCs as PoT gateways with embedded Asterisk server in the proposed framework.

- `psutil`: It is used to retrieve information on running processes and the system performance metric (i.e., %CPU utilization and load average).

- `os`: It runs Linux bash commands from within the testing scripts and sends remote commands to the peer PoT gateway used during the transcoding VoIP testing.

- `subprocess`: it is used to spawn new processes, connect to their input/output/error pipes, and get their return codes.

The test script records the metrics of the processes running on the gateway after each call generation. The records are saved by the test script within the gateway and used for further analysis and visualization using data visualization software, namely, Tableau Desktop. Figure 4.1 shows the flow diagram of the proposed performance testing algorithm, which is summarized as follows:

1. A *music-on-hold* (*MOH*) extension is configured on the embedded Asterisk server into the PoT gateway. MOH maintains each originated call to that extension alive until the calling party hangs up the call. MOH allows programmatically originating an increasing number of simultaneous calls at the PoT gateway.

2. After each call origination, the operating system metrics are captured by the testing script and compared against predefined threshold values defined in the script. Thresholds are set such that %CPU utilization $<75\%$ for single-core ARM boards and $<300\%$ for quad-core ARM boards (equivalent to 75% per core) and the load average (5-minute interval) < 4. These thresholds aim at:

   (a) Protecting the embedded platform from potential hardware damage due to the excessive CPU utilization by the Asterisk process.

   (b) Maintaining the responsiveness of the operating system to be able to carry out the evaluation measurements.

(c) Maintaining the quality metrics of the VoIP calls at acceptable values.

(d) Determining the maximum number of simultaneous VoIP calls the embedded platform can gracefully serve.

3. Another VoIP call originated at the beginning of the test using an IP phone set. It calls a MOH extension and is kept alive throughout the testing procedure. The traffic of this VoIP call is captured using Wireshark. During the test procedure, the traffic capture monitors the VoIP call quality metrics while originating an increasing number of VoIP calls to ensure that both jitter and packet loss of the established VoIP calls still fall within the standard acceptable limits.

4.1.7 Passthrough VoIP Testing

Figure 4.2 overviews the passthrough VoIP testing procedure. In passthrough, the calling parties use the same codec to transform analog audio signals into a digital format suitable for transmission over the network. For passthrough performance evaluation, the testing script forces the embedded Asterisk server into the PoT gateway to originate a call to the MOH extension through the loopback interface of the board. This resembles a SIP trunk operation. The call origination by the script creates a communication channel with the Asterisk server (i.e., Channel #1). On the other hand, passing the call to the called party (i.e., MOH extension) is carried out by a SIP trunk interface defined in `pjsip_wizard.conf` configuration file that uses the loopback interface of the board (i.e., Channel #2). Since both Channel #1 and Channel #2 are configured to use the same audio codec (i.e., ulaw), a passthrough VoIP call is established between the script and the MOH extension. This configuration has two advantages:

1. It allows passthrough testing using a single Raspberry Pi board.

2. It shortens the testing time; since each originated call represents two active channels. As a result, it reduces the testing time by 50%.

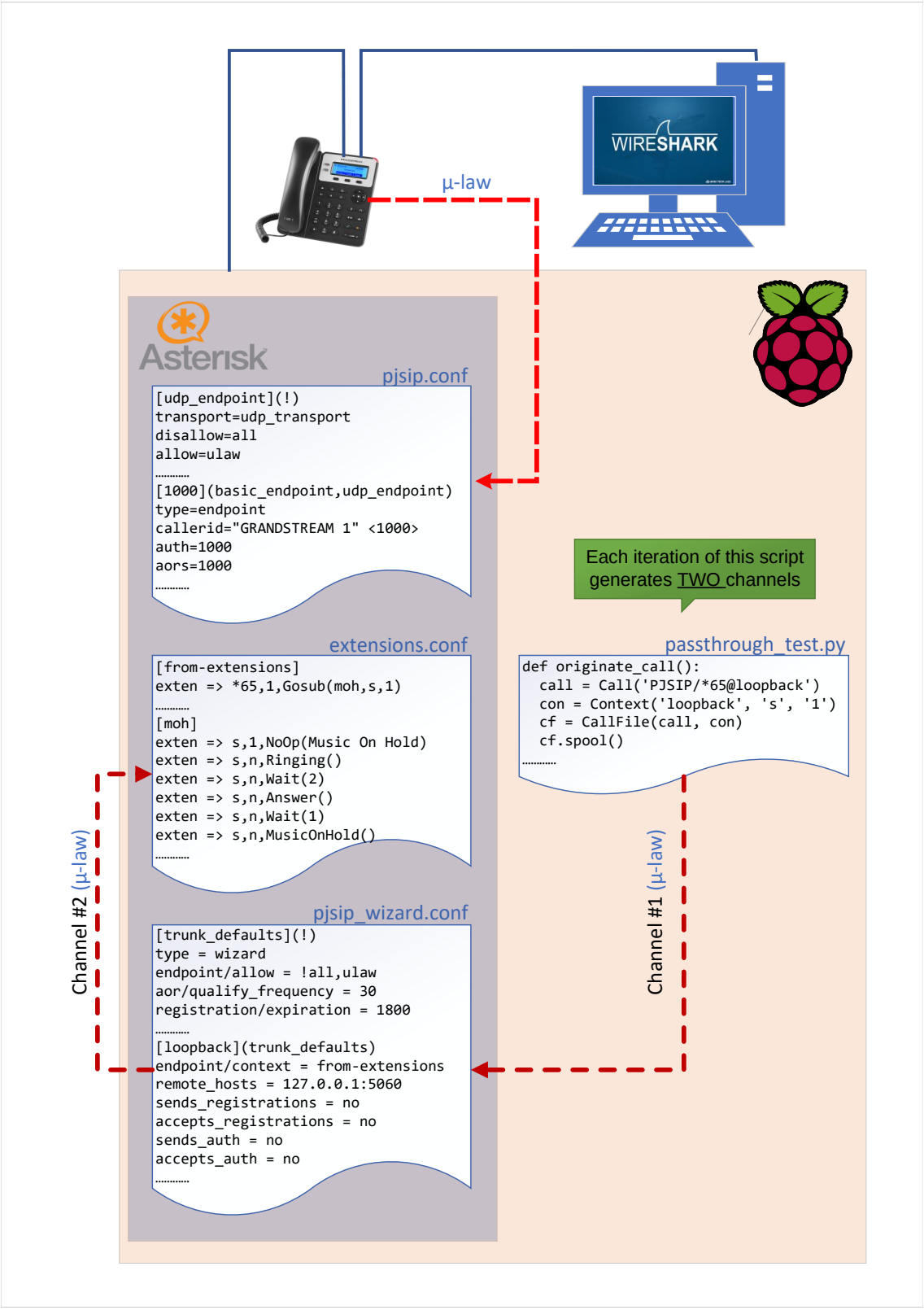

Figure 4.2: Overview of passthrough VoIP testing workflow.

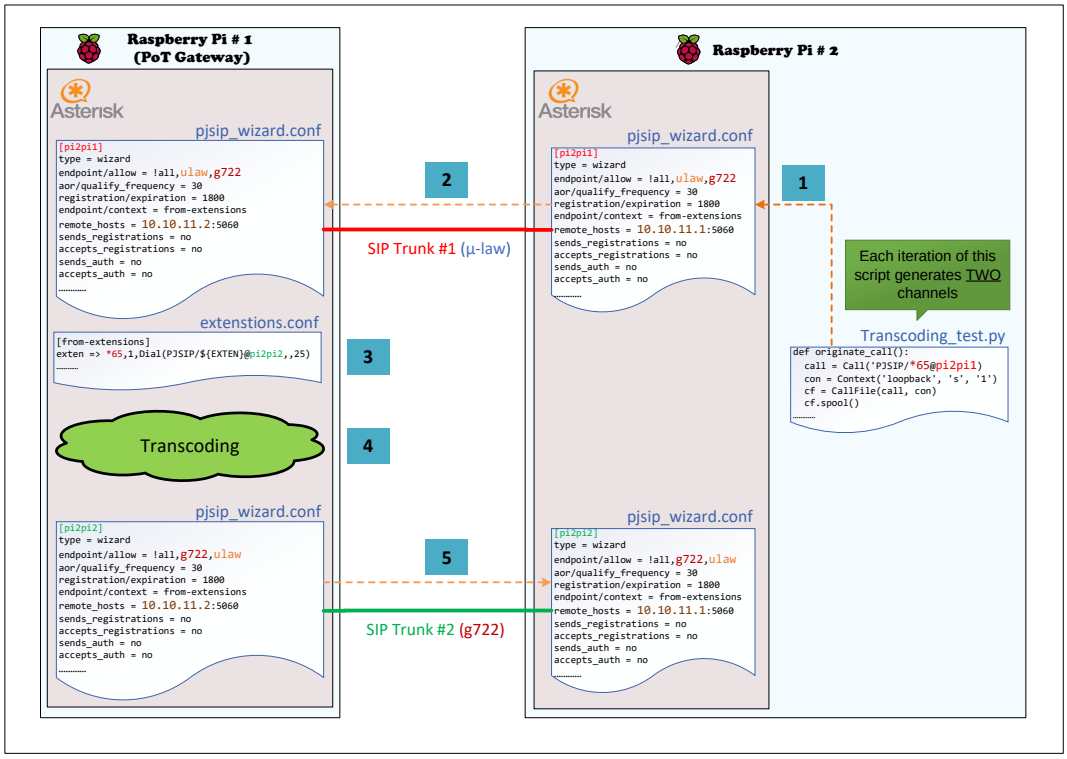

Figure 4.3: Overview of transcoding VoIP testing workflow.

4.1.8 Transcoded VoIP Testing

Figure 4.3 shows an overview of transcoding VoIP testing procedures. In the transcoding VoIP testing, we utilize another Raspberry Pi board. The second board is equipped with a bare-metal Asterisk server without the need for other applications or services running on the PoT gateway. The embedded Asterisk servers on the two boards are interconnected using two distinct SIP trunks. The SIP trunks are configured such that they use different audio codecs. SIP trunk #1 utilizes ulaw audio codec, and SIP trunk #2 utilizes g729 audio codec. We use these codecs because they are popular, open-source, and can be freely utilized by any VoIP system. The second Raspberry Pi board is used to run the test script. The test script calls a MOH extension on the helper board (i.e., Raspberry Pi 2). However, the script forces the call to go through SIP trunk #1, which utilizes ulaw audio codec. Once the call is received at the PoT gateway (i.e., Raspberry Pi 1), it goes through

the extensions defined in the `extensions.conf` configuration file on the PoT gateway to find a match. The extension definition of MOH on the PoT gateway is defined such that it goes through SIP trunk #2. Since the audio codecs of the incoming channel (i.e., SIP trunk #1) and the outgoing channel (i.e., SIP trunk #2) are different, transcoding is performed on the PoT gateway before passing the call back to the helper board (i.e., Raspberry Pi 2) through SIP trunk #2. Therefore, we can monitor the performance of the PoT gateway while carrying out transcoding operations. Since the test script runs on the helper board (i.e., Raspberry Pi 2), we utilize `os` and `subprocess` Python libraries to send remote Linux bash commands from the helper board to the PoT gateway. These commands are used to retrieve information about the performance measurements of the PoT gateway (i.e., %CPU utilization and load average).

4.2 tSIP: A lightweight SIP-Based Messaging Protocol for PoT

4.2.1 tSIP: Overview

The following subsections delve into the details of designing a SIP-based messaging protocol for the PoT applications called tSIP. tSIP is a lightweight Session Initiation Protocol (SIP) implementation designed for resource-constrained devices and low-power communication scenarios in PoT applications. The primary goal of the proposed tSIP is to provide a simplified and efficient SIP implementation that consumes minimal memory, processing power, and energy. tSIP is tailored to meet the unique requirements of PoT devices, which often operate with limited resources, such as microcontrollers or low-power embedded systems. By optimizing the SIP functionality and reducing the protocol's footprint, tSIP enables seamless integration of VoIP capabilities into a wide range of PoT devices without adversely affecting their performance. Using binary encoding, tSIP messages

can efficiently be transmitted and processed by PoT devices. The binary format reduces the message size and eliminates the need for extensive parsing and text-based processing, which are resource-intensive operations. This streamlined encoding enables the seamless integration of tSIP into devices without overburdening their limited resources or mandating extensive firmware modifications, which can be impractical. The reduced complexity and seamless integration of tSIP make it easier for developers to implement SIP functionality in their devices, accelerating the development cycle and time to market. The reduced energy consumption of tSIP contributes to prolonged battery life, which is crucial for PoT devices that may be battery-powered or have limited access to power sources. Therefore, PoT devices can extend their operational lifespan and enhance their overall efficiency by utilizing the minimal energy required for SIP operations. The tSIP integration process typically involves the addition of a software layer that handles tSIP messages exchange with a proxy (i.e., PoT gateway), allowing for seamless communication between the device and the SIP network.

To this extent, the PoT gateway as a tSIP proxy is an intermediary component crucial in mapping constrained tSIP messages to their original SIP counterparts. The PoT gateway bridges resource-constrained PoT devices utilizing tSIP messages and the larger SIP infrastructure (i.e., the communication server). Its primary function is facilitating communication between tSIP-enabled devices and standard SIP devices or services by translating and mapping the constrained tSIP messages to their original SIP format. The PoT gateway understands the limitations and optimizations of tSIP and the specific requirements of the constrained devices it serves. It receives the tSIP messages from the devices and then converts them to their corresponding standard SIP messages to ensure compatibility with the broader SIP ecosystem. This mapping process involves transforming the message structure, headers, and other elements to align with the standard SIP protocol. This mapping allows PoT devices to virtually act as typical SIP endpoints to the communication server, facilitating their management and control through the exist-

ing SIP infrastructure within the premises. It allows these devices to communicate with standard SIP endpoints, participate in VoIP calls, exchange messages, and take advantage of the rich features and functionalities the standard SIP ecosystem offers without requiring the constrained devices to handle the complexity of full-scale SIP protocols. It abstracts the intricacies of the mapping process, allowing the constrained devices to focus on their core functions while maintaining compatibility with standard SIP.

4.2.2 tSIP: Binary Encoding of SIP Messages

4.2.2.1 Google Protocol Buffers (Protobuf)

Binary encoding is a data representation method that encodes information using only two symbols: 0 and 1, effectively converting complex data into a sequence of bits [120]. Protocol Buffers is a popular and efficient solution among the various binary encoding formats [121]. Developed by Google, Protocol Buffers, or *Protobuf*, offers a compact and language-agnostic serialization format of structured data that enables seamless data interchange between systems written in different programming languages. Its characteristics include high performance, as Protobuf efficiently packs data, resulting in smaller message sizes, reduced bandwidth requirements, and optimized serialization and deserialization times. Additionally, Protocol Buffers support backward and forward compatibility, allowing users to evolve their data structures without breaking existing applications. Moreover, Protobuf provides schema definition through a simple language, ensuring better maintainability and ease of versioning. Due to these qualities, Protobuf has become a preferred choice in distributed systems, enabling faster and more reliable data exchange while minimizing resource consumption.

Protobuf plays a vital role in IoT applications, addressing the challenges of data communication and device interoperability in a resource-constrained environment [121]. In the IoT, where millions of devices generate and exchange vast amounts of data, efficiency

and compactness of data representation become crucial factors. Protobuf's characteristics make it well-suited for IoT applications. Firstly, its compact binary format reduces the size of data payloads, minimizing the bandwidth required for communication, a crucial consideration in low-power and limited connectivity scenarios. This efficiency is especially advantageous when dealing with data transmission over wireless networks, where every byte saved can extend the device's battery life and reduce data transfer costs. Secondly, Protobuf's language-agnostic nature enables seamless integration with various devices and platforms. Protocol Buffers allow developers to define a common data schema, facilitating smooth data exchange and interoperability between heterogeneous devices, regardless of the programming language used in their implementation. Moreover, Protobuf's support for backward and forward compatibility allows devices to be updated without breaking compatibility with existing data formats, making it easier to maintain and scale IoT systems. Furthermore, security is a paramount concern in IoT. Protobuf can be more resistant to specific attacks as a binary format than text-based serialization formats like JSON (JavaScript Object Notation) or XML (Extensible Markup Language), reducing the risk of injection and tampering vulnerabilities [122].

Binary encoding of SIP messages using Protobuf in the proposed tSIP messaging protocol offers several advantages for efficient and streamlined communication. By leveraging Protobuf in the proposed tSIP, original SIP messages are encoded into a binary format that minimizes size, improves processing speed, and enhances interoperability. Protobuf's binary encoding offers a significant size reduction for tSIP messages compared to their original heavy text-based encoding formats of the original SIP messages. tSIP messages are more compact, requiring less bandwidth for transmission and storage. This size reduction is especially valuable for resource-constrained devices and networks with limited bandwidth, allowing faster message transfer and more efficient resource utilization. In addition to its smaller size, Protobuf also enhances the processing speed of tSIP messages. Due to the efficient binary format, tSIP messages can be deserialized and processed

more quickly. This is particularly important in real-time communication scenarios where speed and responsiveness are crucial, ensuring minimal latency and improved overall performance. Alongside this, Protobuf's schema definition language provides flexibility and extensibility to the proposed tSIP messaging protocol. The schema allows for defining the structure of the original SIP messages and specifying the data types, facilitating interoperability between different systems and programming languages. This flexibility simplifies the integration of Protobuf-encoded tSIP messages into existing devices, applications, and communication frameworks, promoting seamless communication and interoperability across the larger SIP ecosystem. Furthermore, leveraging the backward compatibility of Protobuf-encoded messages, as new features or fields are added to tSIP messages, older versions can still be parsed and processed correctly. This allows for graceful upgrades and backward compatibility between different versions of the proposed tSIP messaging protocol, ensuring smooth transitions and avoiding disruptions in communication.

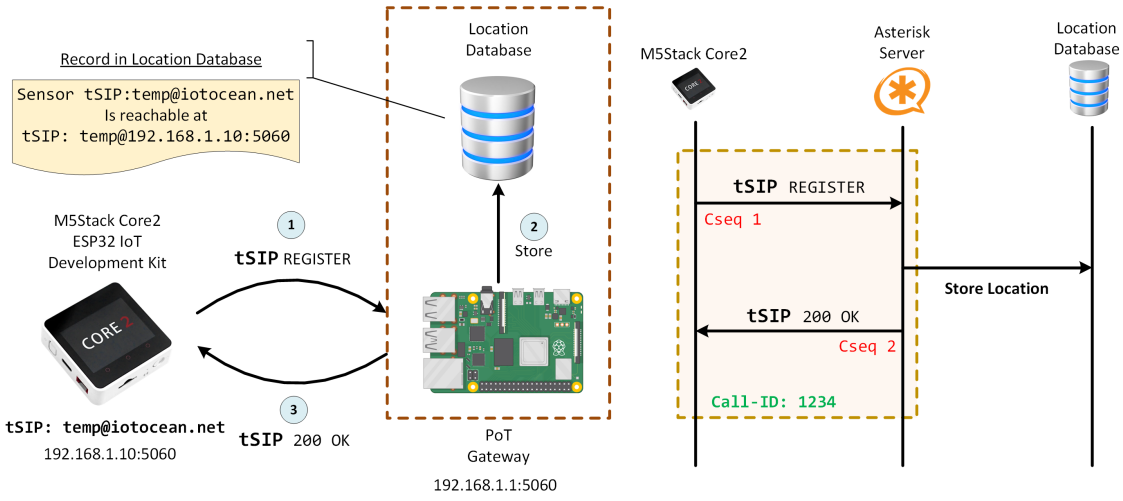

Figure 4.4: tSIP registration.

4.2.2.2 tSIP Schemas with Protocol Buffers

The proposed tSIP schemas, built using Protobuf, offer an ingenious approach to the challenges of exchanging SIP messages with resource-constrained devices in IoT and other constrained environments. Using the Protobuf framework, we define a compact and optimized data structure representing a constrained SIP message version (i.e., tSIP). These tSIP schemas efficiently pack the essential information required for SIP communication while minimizing the message size, reducing bandwidth usage, and conserving valuable resources on constrained devices. Listing 4.1 shows the code snippet of the `.proto` file representing the implementation of `REGISTER` method of the SIP protocol in the proposed tSIP. It encompasses various elements essential for SIP communication with resource-constrained devices. The *Method* `enum` lists the supported SIP methods, with `REGISTER` as the primary device registration method. Additionally, the *Sensor* `enum` represents the capabilities of the device, such as *TEMPERATURE*, *HUMIDITY*, and *PRESSURE*. The message structure itself includes specific fields for various SIP parameters, such as *Cseq* for SIP transaction tracking, *Call_ID* for SIP session tracking, and *Contact_IP* and *Contact_Port* to indicate the device's IPv4 address and listening port number, respectively. Furthermore, the schema includes fields for the *PoT_IP* and *PoT_Port*, which denote the IPv4 address and listening port number of the PoT Gateway. Finally, the *Sensor* field, marked as *repeated*, specifies that the device supports temperature measurements. This well-structured and compact `.proto` file ensures the seamless exchange of SIP `REGISTER` messages between constrained devices and the embedded Asterisk server to the gateway, optimizing data representation and enhancing communication efficiency in resource-constrained environments.

```
1 syntax = "proto3";
2 message tSIP {
3     enum Method // Supported SIP methods
4     {    REGISTER = 0;
```

```
5           INVITE = 1;
6           MESSAGE = 2;
7           ACK = 3;
8           BYE = 4;
9           CANCEL = 5;
10          PUBLISH = 6;
11          NOTIFY = 7;
12          REFER = 8;  }
13      enum Sensor // Sensor capabilities
14      {   TEMPERATURE = 0;
15          HUMIDITY = 1;
16          PRESSURE = 2;   }
17      Method method = 1;          // REGISTER
18      uint32 Cseq = 2;            // SIP transaction tracking
19      unit32 Call_ID = 3;         // SIP Session tracking
20      unit32 Contact_IP = 4;      // Device: IPv4 address
21      unit32 Contact_Port = 5;    // Device: Listening port no.
22      uint32 PoT_IP = 6;          // PoT GW: IPv4
23      unit32 PoT_Port = 7;        // PoT GW: Listening port no.
24      repeated Sensor sensor = 8; // Temperature supported
25  }
```

Listing 4.1: SIP REGISTER method definition in .proto file.

The `REGISTER` method in the proposed tSIP plays a pivotal role in enhancing the usability and efficiency of device location in constrained environments. By leveraging the `REGISTER` method, resource-constrained devices can easily register their presence and location with the embedded Asterisk server in the PoT gateway. Using Protocol Buffers, the compact and optimized data structure defined in the tSIP schema ensures that the `REGISTER` message contains only the essential information, conserving valuable resources on the constrained devices. The usability of the `REGISTER` method lies in its ability to establish a central location database on the PoT gateway, where all registered devices

and their corresponding SIP URIs (Uniform Resource Identifier), IP addresses, and contact details are stored. This directory of registered devices serves as a reference point for efficient call routing and message delivery. When a SIP user wishes to communicate with a specific device to query its sensed information, the PoT gateway can quickly retrieve the device's location information from the database, ensuring seamless and reliable connectivity. Moreover, in resource-constrained environments with limited battery power or intermittent connectivity, the `REGISTER` method allows devices to update their location information periodically or when changes occur, ensuring accurate and up-to-date device tracking. This feature is crucial for IoT applications, where devices may be mobile or deployed in dynamic environments. Figure 4.4 depicts the tSIP `REGISTER` method to allow the PoT gateway (*registrar*) to store the location of the IoT device and its supported sensing capabilities. Therefore, the PoT gateway can use this information to choose the suitable IoT device and use the `MESSAGE` method to ask it for information about what it has sensed to answer the user's question efficiently.

4.2.2.3 tSIP Messages Encoding Using Protobuf

Creating a `.proto` file for defining the structure of selective SIP methods is a fundamental step in leveraging Protocol Buffers in the proposed tSIP messaging in the thesis. The `.proto` file outlines the message formats and data types required to represent the specific SIP methods selected for implementation by tSIP. These methods include `REGISTER`, `INVITE`, `ACK`, `BYE`, and other essential SIP messages. Within the `.proto` file, we define the fields and attributes and their data types, specifying the necessary information for each SIP method. The field names and attributes are not included in the Protobuf data. Instead, every field in a Protobuf message has a distinct number and may be designated as optional or necessary, giving developers the flexibility and extensibility to structure messages optimized for their needs. Once the `.proto` file is created, the next step is to use the *protoc* compiler provided by Protocol Buffers to generate the compiled native wrapper

class of schemas representing the constrained versions of the selective SIP methods in the desired programming language. The Protobuf compiler (*protoc*) natively supports various programming languages (e.g., C++, Python, Java, Go, JavaScript, C#). However, other programming languages are supported by third-party compilers. The *protoc* compiler with the appropriate options and the `.proto` file as input produces language-specific code files that handle the serialization and deserialization of tSIP messages according to the defined structure.

After generating the code, the next phase involves deploying it to the target devices, including the Raspberry Pi gateway and the M5Stack Core2 ESP32 IoT development board. To deploy the code on the Raspberry Pi gateway, we ensure that the necessary dependencies and libraries for the Protocol Buffers are installed on the device. The generated code is integrated into the existing PoT application stack running on the Raspberry Pi, allowing it to handle tSIP messages received from the M5Stack development board efficiently, transform received tSIP messages to their SIP messages counterpart, and pass the transformed tSIP messages to the embedded Asterisk server to the gateway for further processing.

For the M5Stack board, deploying the code involves including the generated code files in the developed firmware project for the board. We ensure the appropriate libraries and configurations are set up to support Protocol Buffers on the ESP32 platform. We use *Nanopb* [123], a small-footprint third-party Protobuf implementation in ANSI C, to generate the native wrapper class for tSIP messages that can be used on microcontrollers (i.e., ESP32). This step is crucial to effectively enable tSIP message processing and serialization on the resource-constrained ESP32 board. After successful deployment to both devices, the Raspberry Pi gateway and the M5Stack board are equipped to communicate using the selective SIP methods implementation by tSIP with the help of Protobuf. tSIP ensures a streamlined, efficient, and standardized data exchange between the devices, optimizing resource usage and enhancing the overall performance of SIP-related appli-

cations in PoT scenarios. The procedures mentioned above for encoding and decoding tSIP messages using the Protobuf are depicted in Figure 4.5.

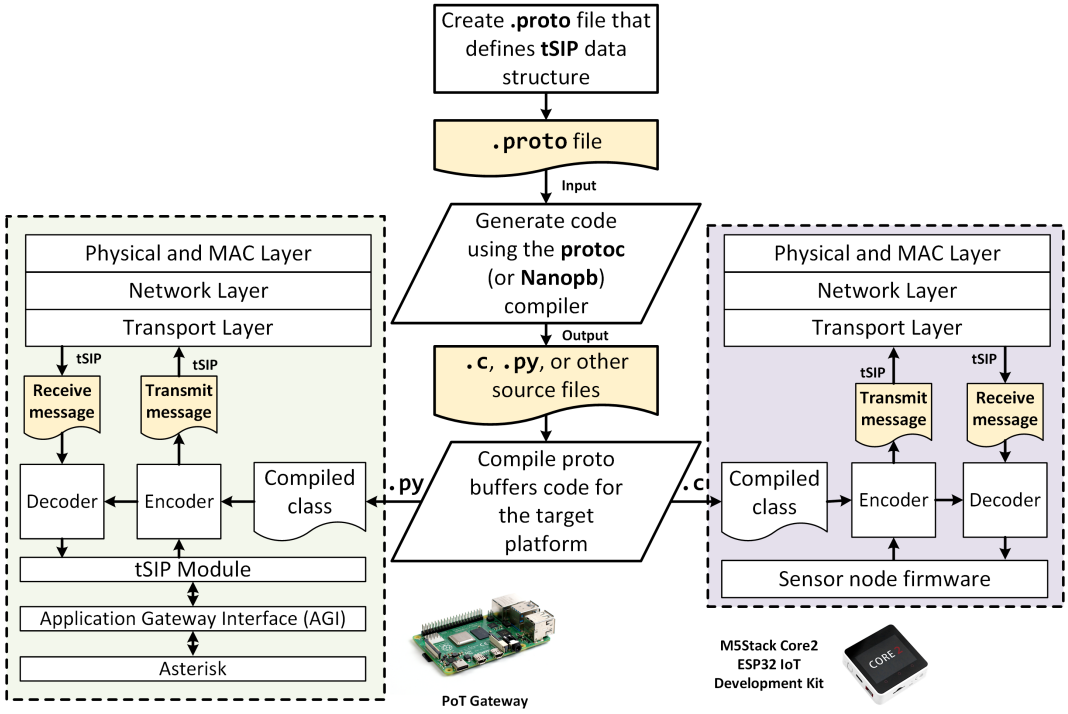

Figure 4.5: Protocol Buffers (Protobuf) utilization for encoding and decoding tSIP messages.

4.2.2.4 Relaying Messages between SIP Servers and Constrained Devices

Pytwinkle serves as a powerful bridge between Asterisk, a full-featured SIP server embedded within the PoT gateway, and the existing communication servers within the premises in typical PoT application scenarios. This Python-based framework facilitates the relay of SIP communication, enabling seamless interaction between the surrounding constrained devices and the existing capable SIP servers within the premises. By acting as an intermediary, *Pytwinkle* takes advantage of Asterisk's extensive capabilities and compatibility with standard SIP protocols while utilizing the tSIP module to efficiently translate and adapt SIP messages received from the communication server for the limited resources and capabilities of the tSIP-enabled devices. This seamless integra-

tion allows constrained devices to participate in complex SIP communication scenarios and act as typical SIP endpoints in the VoIP ecosystem, opening up new possibilities for real-time communication and telephony applications even in environments with restricted computational power and bandwidth constraints. With *Pytwinkle*, we harness the power of Asterisk's rich features while ensuring smooth, reliable, and efficient SIP communication with tiny devices, contributing to advancing innovative PoT applications and low-resource communication solutions.

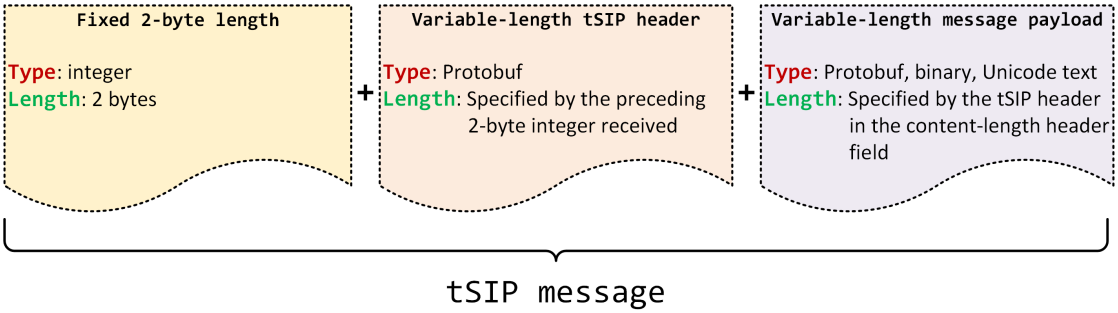

Figure 4.6: tSIP packet structure (tSIP MESSAGE method as an example).

4.2.3 tSIP: Packet Formation

Efficient packet formation in the proposed tSIP is of paramount importance due to the inherent constraints of resource-limited devices and networks. In typical PoT applications utilizing constrained devices, such as those found in IoT devices or embedded systems, available processing power, memory, and bandwidth are often severely limited. As a result, every byte of data transmitted matters, and inefficient packet formation can lead to wastage of precious resources and impact the system's overall performance.

In the proposed tSIP messaging protocol, a fixed-length header contains the size of upcoming packets. This approach optimizes packet delivery in resource-constrained environments. The receiver can predict the exact amount of data to expect in the following packet by including the packet size within a fixed-length header. This predictive capability eliminates the need for the receiver to perform dynamic packet size calculations or

to search for delimiters, both of which would consume additional processing resources. Optimizing packet delivery in this manner brings several advantages:

1. *Reduced Overhead*: Including the packet size in the header eliminates the need for additional metadata or control information to indicate the packet size. This reduction in overhead conserves valuable bandwidth and processing power, particularly for tSIP-enabled devices with strict data transmission constraints.

2. *Streamlined Parsing*: With fixed-length headers, the receiver can efficiently parse incoming packets by directly extracting the size information from the header. This streamlined parsing process results in faster and more deterministic packet processing, making it ideal for resource-constrained devices with limited computational capabilities.

3. *Improved Packet Handling*: Predicting the packet size in advance allows for better memory management on the receiver's side. It enables the allocation of an appropriately sized buffer to accommodate the incoming packet without the need for dynamic memory reallocation, which can be costly in terms of processing time and fragmentation.

4. *Enhanced Reliability*: Fixed-length headers with explicit size information contribute to improved packet validation. The receiver can check whether the expected packet size matches the received data, helping detect and handle potential errors or packet truncations. Also, it simplifies software-defined timeout and retransmission implementations for reliable delivery of the protocol messages if it is not supported by the underlying transport layer (e.g., UDP).

In this context, Figure 4.6 depicts packet formation of instant messaging (IM) implementation by tSIP. The message payload is appended to the `MESSAGE` method header, and a `content-length` attribute is included in the `MESSAGE` method header to determine the end of the header and the beginning of the payload of the message.

4.3 A lightweight and Blockchain-Based Device Registration and Authentication for PoT Applications

4.3.1 Introduction

The growing adoption of VoIP and SIP for communication services has led to an increased focus on securing these communication channels. Traditional registration and authentication mechanisms in SIP systems often rely on centralized servers and complex protocols, leading to more resource utilization on capable devices and scalability issues. Therefore, the deployment complexity of applying traditional registration and authentication mechanisms in SIP systems on resource-constrained devices can be a significant challenge. Resource-constrained devices like IoT devices often have limited processing power, memory, and energy capabilities. Traditional registration and authentication mechanisms typically rely on complex cryptographic protocols and communication with centralized servers, which may need to be better suited for these constrained environments. Implementing such mechanisms on resource-constrained devices can lead to increased processing overhead, longer registration times, and higher energy consumption, potentially compromising the device's overall performance and user experience. Moreover, these mechanisms might require frequent communication with centralized servers, leading to potential network congestion and latency issues, particularly when devices are connected via low-bandwidth networks. As a result, the deployment complexity becomes a significant hindrance, necessitating the exploration of lightweight and efficient alternatives, such as blockchain-based registration and authentication mechanisms, to ensure secure and seamless communication on resource-constrained devices.

In the following subsections, we propose a lightweight decentralized registration and

authentication mechanism based on blockchain technology for smart gadgets in the PoT system to facilitate their association with communication servers. The proposed mechanism provides a secure, trustless, and scalable environment for PoT without requiring high-end communication servers, affecting the existing SIP-based VoIP architecture, or mandating trust in third-party entities. The proposed mechanism uses the Ethereum blockchain and smart contracts to implement a programmatic, immutable access control mechanism to administer the association of IoT devices with the communication server.

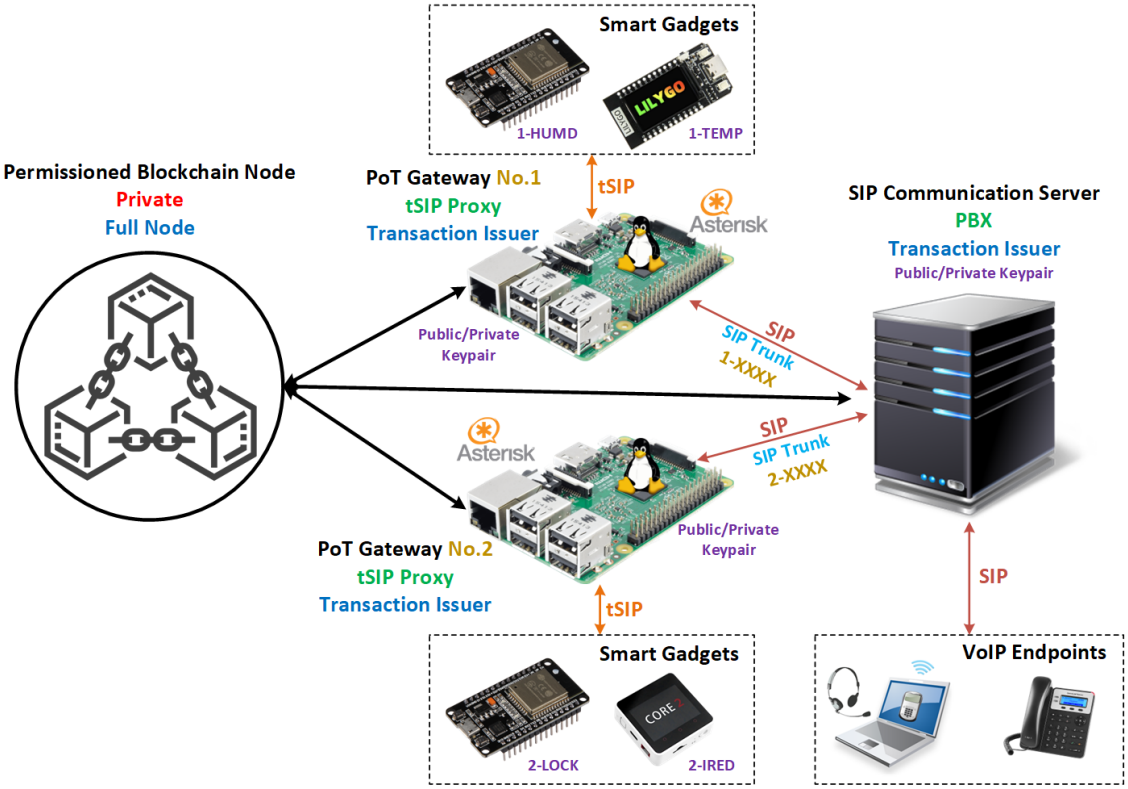

Figure 4.7: Architecture model of the proposed blockchain-based registration and authentication mechanism.

4.3.2 Overview

In this subsection, we overview the proposed solution for registering and authenticating resource-constraint devices to the communication server in PoT applications by harnessing the power of blockchain technology. By combining the principles of blockchain's

decentralization and tamper-resistant properties with a lightweight SIP protocol (i.e., tSIP), we present a secure, efficient, and scalable mechanism for associating smart gadgets with communication servers. This blockchain-based approach offers enhanced trust, data integrity, and privacy, making it an ideal fit for resource-constrained devices and fostering the scalability of the proposed PoT framework. Figure 4.7 overviews the architecture of the proposed mechanism. From a high-level perspective, it consists of the following:

1. ***Full node of private blockchain***: It runs on a local machine using the virtual Ethereum platform and utilizes the Proof-of-Authority (POA) consensus algorithm. POA is a reputation-based consensus mechanism that provides an efficient and high-throughput (i.e., transactions per second (TPS)) solution for private blockchain networks. POA depends on a limited, predefined set of nodes to serve as system moderators for block validation. In the proposed mechanism, POA satisfies the SIP security requirements while enhancing the call setup time and reducing the hardware specification requirements of the blockchain node. Also, the node features smart contract deployment via Solidity that implements functions to add or remove gadgets from the approved device list of a specific PoT gateway during initial device setup. The functions utilize a multidimensional mapping data structure indexed by the public keys of the associated PoT gateways to the PBX server and the unique smart gadget tokens. Likewise, the smart contract implements functions to check the authenticity of associated PoT gateways to the PBX server and the authenticity of gadget association with a specific PoT gateway before proxying tSIP messages between the gadget and the communication server.

2. ***SIP-Based PBX server***: A SIP, Asterisk-based IP-PBX server running on a local machine resembles the communication server in the premise with VoIP phone sets registered to it. The motivation behind the proposed mechanism is to allow

the PBX server to interact with the heterogeneous objects within the premises without affecting its VoIP architecture or impacting its designed capacity. The burden of objects' registration and authentication is delegated to the PoT gateway. The PBX server communicates with the PoT gateway through a mature VoIP technology, namely SIP trunks. Similar to the typical SIP trunk configuration in the VoIP ecosystem, a numbering scheme is defined in the PBX server for each SIP trunk configuration to a specific PoT gateway. This allows the PBX server to route the call to the corresponding PoT gateway based on the callee's number. The proposed system uses the `[1-2]XXXX` numbering scheme for PoT for simplicity. `[1-2]` represents a 1-digit code for the PoT gateway that matches the range of digits in the brackets (in this case, 1 and 2). In the proof-of-concept implementation, we utilized two PoT gateways and assigned a distinct digit to each. The remaining four digits (i.e., `XXXX`) represent the unique extension number of a specific gadget associated with a particular PoT gateway. In the proposed mechanism, the PBX server acts as a transaction issuer on the blockchain. It has a verifiable, self-signed identity (i.e., a private/public key pair) for the blockchain. It neither mandates hosting a full copy of the blockchain nor engaging in computationally intensive consensus algorithms to create new blocks. The PBX server is also responsible for deploying the smart contract to the blockchain during the system startup and issuing the corresponding smart contract functions to add or remove PoT gateways from the list of associated PoT gateways to the PBX server.

3. ***PoT Gateways***: Act as proxies between gadgets and the PBX server to map tSIP messages to their SIP counterparts and vice versa. Like the PBX server, a PoT gateway acts as a transaction issuer on the blockchain. The PBX server uses the gateway's public key to issue the smart contract function that adds the gateway to the list of associated gateways with the PBX server. While setting up the association of a gadget with the PoT gateway for the first time, the PoT gateway

is used to interact with the gadget's developed firmware to generate its unique extension number and store the gateway's public key in the non-volatile storage (NVS) of the gadget. The extension number of the gadget is generated based on the received device attributes to facilitate localizing the object in the system. Also, while registering the object, the PoT gateway invokes the smart contract function to add the unique device identity to the immutable list of associated devices with the gateway. Similarly, the gateway invokes a smart contract function to check the authenticity of the gadget's identity before granting a tSIP message exchange with it.

4. ***Smart gadgets***: Smart gadgets in the proposed system imply devices with embedded programmable software (i.e., firmware) that controls their functionality. These devices typically have resource constraints and require optimized messaging and communication protocols for prolonged battery life. We utilize ESP32-based development boards in the proposed system to simulate smart gadgets. ESP32 is a cost-effective and feature-rich MCU from Espressif Systems. It has integrated Wi-Fi and Bluetooth connectivity, over-the-air (OTA) firmware update capability, and built-in serial communication protocols that fit many IoT applications. Limited resources for smart objects hinder their engagement in blockchain processing. The proposed mechanism leverages the encrypted storage capability of ESP32 to store and transfer gadgets' unique identities and their predefined attributes to the PoT gateway during the registration process.

Table 4.2 summarizes the role of the proposed system components in the blockchain network.

Table 4.2: Role of System Components in the Blockchain Network

System Component	Node Type	Storage	Validator
Blockchain Node	Full Node	Full Blockchain	Yes
PBX Server	Transaction Issuer	None	No
PoT Gateway	Transaction Issuer	None	No
Smart Gadgets	None	None	No

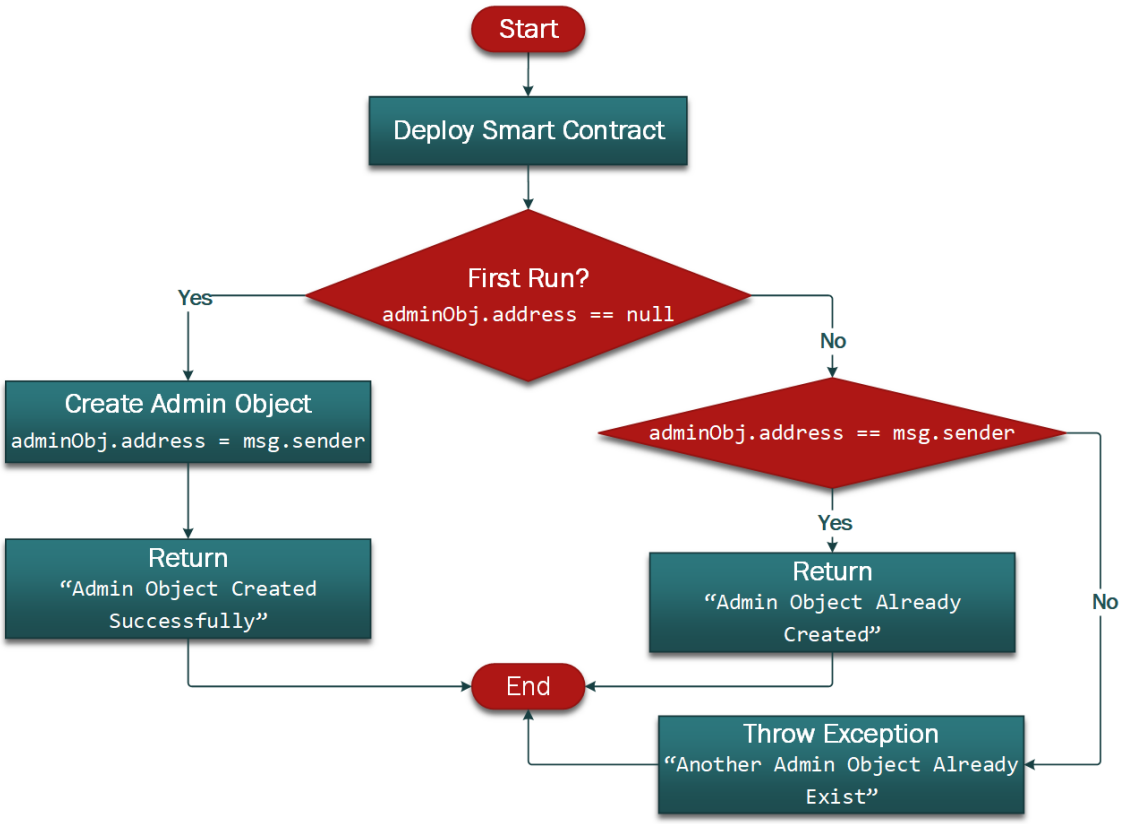

Figure 4.8: Flowchart of smart contract deployment by the designated admin node (i.e., the PBX server).

4.3.3 System Initialization

To initialize the proposed mechanism, we create an asymmetric keypair (i.e., private and public keys) on the PBX server using the Elliptic Curve Digital Signature Algorithm (ECDSA), which resembles the server's self-signed certificate. The generated public key is used to deploy the developed smart contract and assign the PBX server the admin credibility, allowing the PBX server to associate PoT gateways with it. The flowchart in Figure 4.8 depicts the smart contract deployment process during system initialization. It is worth noting that only one component (i.e., the communication server) is assigned admin credibility in the proposed mechanism. The admin node in the proposed mechanism is responsible for creating the unique IDs for the associated gateways, which are then concatenated with their public keys to create their addresses on the blockchain by calculating the hash (SHA-256) of the concatenated string (the public key of the gateway plus its unique ID). This does not overwhelm the PBX server since it has to be done once when registering the gateway with the PBX server for the first time, with all gateway attributes, including its ID, written to the blockchain to avoid impacting the storage capacity of the PBX server.

4.3.4 PoT Gatway Registration

Like the PBX server, each PoT gateway has a unique, verifiable identity used to register and authenticate the gateway on the blockchain. During registration, the gateway sends a registration request to the PBX server along with its public key. The PBX server (i.e., admin node) receives the public key, creates a unique ID for the gateway, encrypts the generated ID with the received public key, and sends the encrypted ID to the gateway. Also, the PBX server appends the ID to the gateway's public key and calculates the hash (SHA-256) of the concatenated string. Upon receiving the encrypted ID from the PBX server, the PoT gateway decrypts it using its private key and sends an acknowledgment

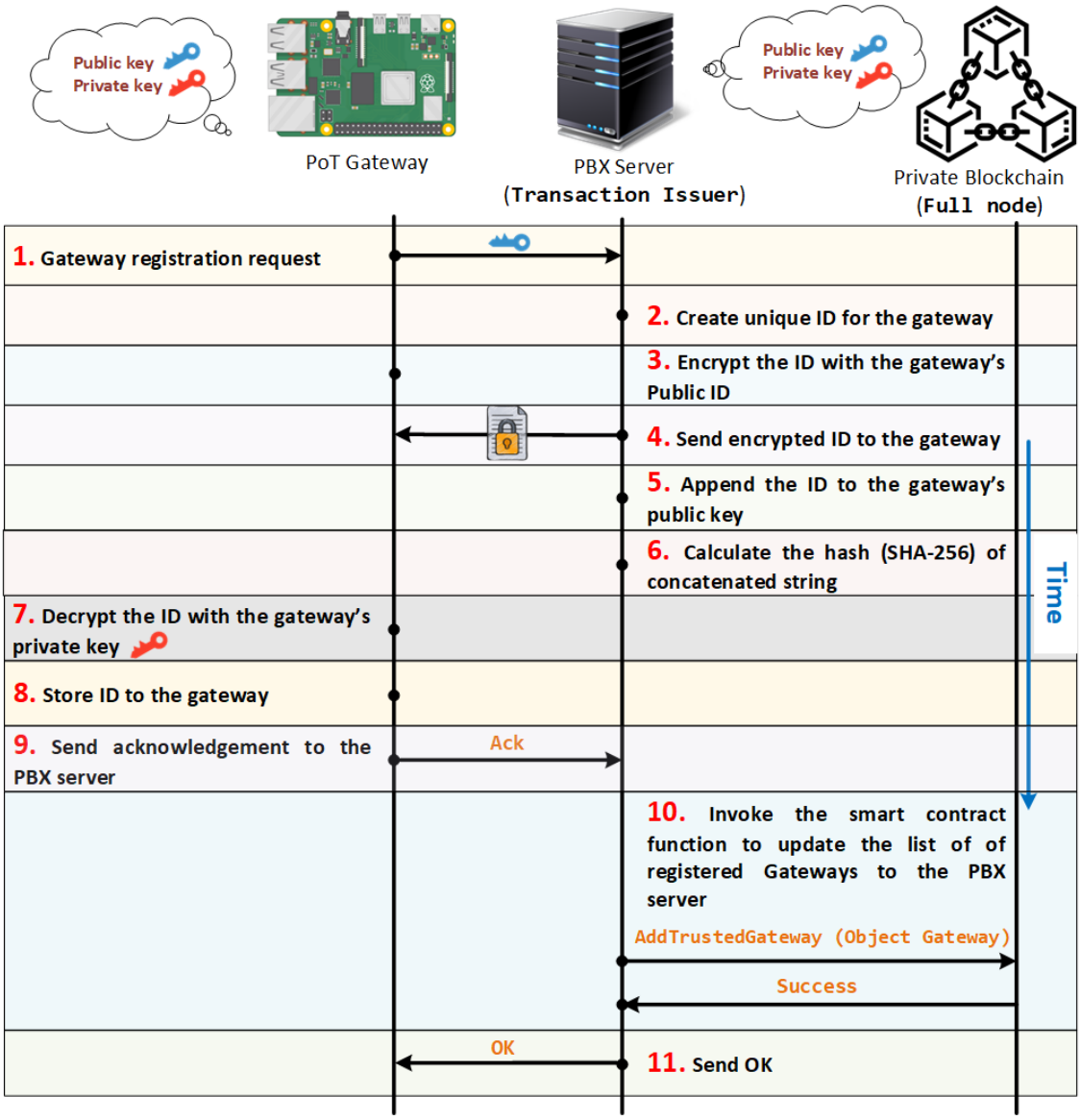

Figure 4.9: PoT Gateway registration with the PBX server.

to the PBX server. The PBX server then invokes a smart contract function to append the gateway to the list of registered gateways on the PBX server using its calculated hash. Once the transaction is successful, the PBX server sends a message to the gateway, indicating the success of the registration process. The PoT registration process is depicted in Figure 4.9.

4.3.5 Gadget Registration

In the proposed system, the non-volatile storage (NVS) of ESP32 is used to store the device-specific data to mitigate impersonation threats. ESP32 supports hardware-level encryption to prevent the retrieval of SPI flash data via physical readouts. In the proposed system, we use two kinds of encryption:

1. *Build-time encryption*: It is used to store the unique device ID, and device attributes that are already known at the compile time.

2. *Run-time encryption*: It stores the gateway's public key when registering the device with the gateway during the device setup. The application generates the key for encryption, where the firmware's data is encrypted and decrypted on the fly by invoking the corresponding ESP-IDF function.

Figure 4.10 illustrates the process of registering a device with the PoT gateway.

4.4 Summary

This chapter presents the implementation details of the proposed Phone of Things (PoT) framework. It begins with a comprehensive feasibility study evaluating embedded Linux single-board computers (SBCs) (i.e., Raspberry Pi) as potential candidates to act as PoT gateways in the proposed framework to handle its ecosystem's requirements. The study assesses factors such as percentage CPU utilization and load average of the operating

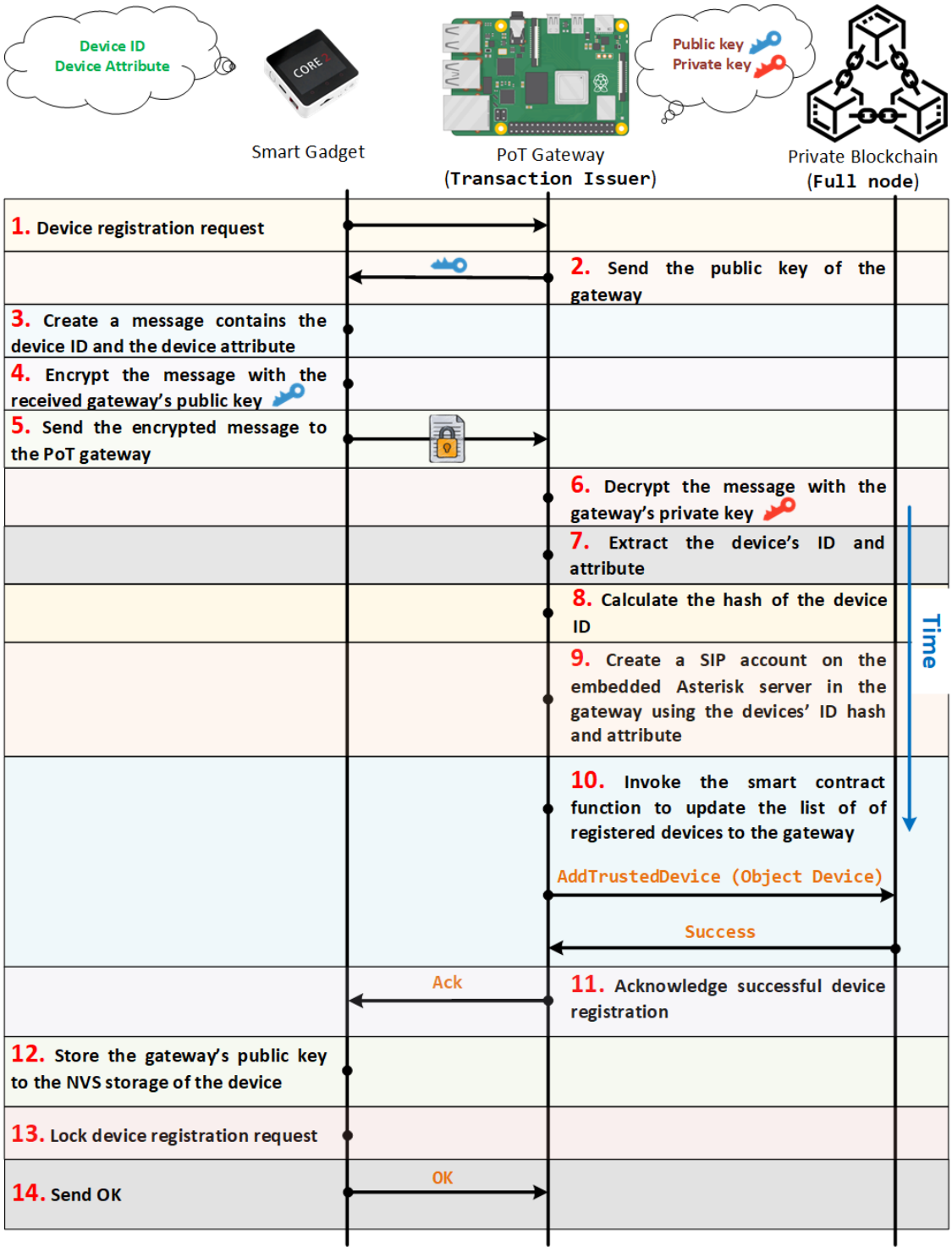

Figure 4.10: Device registration with the PoT gateway.

system running on the boards while generating an increasing number of simultaneous VoIP calls with different configurations (i.e., passthrough and transcoding VoIP calls) for comprehensiveness.

Following the feasibility study, the chapter delves into implementing a lightweight version of the SIP (Session Initiation Protocol) protocol, which we call tSIP, that can be deployed on resource-constrained devices within the PoT framework. This miniaturized SIP protocol aims to reduce the overhead and processing demands of the original SIP protocol while maintaining essential functionalities required for seamless communication between resource-constrained devices and the communication servers. tSIP ensures efficient data exchange and device interactions while ensuring compatibility with the overall PoT architecture.

The chapter culminates with proposing a blockchain-based registration and authentication mechanism for the PoT framework. Recognizing the need for robust security and decentralization in the PoT ecosystem, the blockchain mechanism leverages distributed ledger technology to securely manage device registration, identities, and authentication processes. By integrating blockchain, the PoT framework gains enhanced data integrity, privacy, and trust among devices, ensuring secure interactions without reliance on central authorities. The proposal highlights the benefits of this blockchain-based approach, offering a tamper-resistant and scalable solution for registration and authentication within the PoT environment, paving the way for a more secure and interconnected network of IoT devices.

Chapter 5

PoT Framework Use Cases

5.1 Introduction

In today's rapidly evolving digital landscape, the convergence of cutting-edge technologies has paved the way for groundbreaking innovations, transforming how we interact with our environment and devices. One such transformative fusion is the integration of Internet of Things (IoT) and Voice over Internet Protocol (VoIP) technologies, giving rise to an unprecedented era of smart and context-aware telephony solutions. In this chapter, we review use case scenarios where the seamless integration of IoT and VoIP through the provision of the proposed multi-tier Phone of Things (PoT) framework would revolutionize telephony solutions for businesses, amplify user engagement with surrounding devices, and redefine mature phone terminologies in a modern and efficient way as we know it.

As the IoT ecosystem continues to expand, connecting an ever-growing array of smart devices and sensors, it has unlocked unparalleled possibilities for data-driven insights and automation. Concurrently, VoIP technology has already made considerable strides in simplifying global communication, enabling voice and multimedia transmission over the Internet. By converging these two powerful technologies, a new paradigm emerges –

one where telephony becomes more than just voice communication but an immersive experience that actively engages users with their interconnected devices and surroundings.

Throughout this chapter, we delve into use case scenarios of the proposed PoT framework, examining how it enhances the functionality and capabilities of telephony systems. We will uncover how the real-time data exchange between IoT devices and VoIP networks through the proposed PoT framework enables seamless interaction with the user's environment, leading to more personalized and context-aware communications. Furthermore, we will explore use case scenarios across industries where this integration could usher in a wave of innovation, revolutionizing customer experiences, improving operational efficiency, and unlocking new business opportunities.

5.2 Context-Aware Telephony Solutions

The context-aware telephony Solution is a remarkable example of the tremendous benefits of the seamless integration of IoT and VoIP technologies. By combining the power of the IoT and VoIP technologies, innovative telephony solutions could revolutionize how we engage with our surroundings during phone calls. With an array of interconnected smart devices and sensors to the PoT gateway in the proposed PoT framework, the system could actively capture real-time data from the user's environment, such as occupancy, ambient noise, lighting conditions, and even temperature. Leveraging this contextual information exchange between the proposed PoT framework and the existing communication servers within the premises, the VoIP network would optimize call quality, automatically adjusting audio settings to ensure crystal-clear communication, even in noisy or challenging environments. Moreover, it could spontaneously forward incoming calls in vacant places to other destinations based on some criteria instead of wastefully ringing extensions in empty places, leveraging the exchange of occupancy sensors' data between the PoT gateway and the communication server. This eradicates the notoriously long waiting times

for phone calls. As a result, the user experiences a heightened sense of presence and immersion as the telephony solution adapts seamlessly to the ever-changing dynamics of their surroundings. This enhances the overall user experience of phone calls, fosters customer satisfaction in business domains, and provides efficient and distraction-free conversations, ultimately reshaping how we perceive and interact with telephony systems in our daily lives.

5.3 Redefine Mature Phone Features in a Modern Way

Integrating IoT and VoIP technologies enables redefining traditional phone features, like call forwarding, in a modern and efficient manner. Leveraging this integration, the modern call forwarding system goes beyond simple call redirection after a static, predetermined number of rings. It now incorporates intelligent decision-making capabilities based on real-time contextual data from the surrounding IoT devices. For instance, when a called person is away from their phone, the system can automatically route calls to nearby extensions or smart speakers, ensuring that important calls are not missed. Furthermore, with VoIP technology, call forwarding becomes highly flexible, allowing users to set up dynamic forwarding rules based on time, location, or even the presence of specific individuals, leveraging the information inferred from RFID card readers, for example. This redefined call forwarding approach enhances user convenience and maximizes accessibility and responsiveness, ensuring that calls are efficiently directed to the most appropriate device, ultimately improving overall communication efficiency in the modern interconnected world.

Interactive Voice Response (IVR) is an automated telephony attendant that allows callers to interact with a computerized voice and make selections using their telephone's touch-tone keypad or voice commands. IVR serves as a virtual front-end for various

organizations, enabling them to efficiently handle a high volume of incoming calls while providing callers with prompt and interactive responses. Through a series of pre-recorded voice prompts and menus, IVR guides callers through different options and routes their calls to the appropriate destination based on their selections. However, in its current practice, IVR is a static configuration that needs to be rebuilt every time a change is required. Moreover, customers usually report frustration regarding option tree changes, confusion, routing calls to the wrong places, or long waiting times to get a response after being routed to the intended representative. The proposed PoT framework integrated with chatbot technology represents a groundbreaking advancement in optimizing IVRs within phone systems. This innovative integration combines the power of IoT and the conversational capabilities of chatbots to revolutionize the traditional IVR experience. By leveraging PoT's connectivity with a wide array of smart devices and sensors, the IVR system gains valuable real-time insights into the called person's environment and preferences. This contextual awareness empowers the chatbot to offer dynamically personalized and contextually relevant responses, transforming the interaction into a natural and seamless conversation. Callers can use natural language to engage with the IVR system, eliminating the need to navigate through cumbersome menus. Moreover, PoT's chatbot integration allows for intelligent call routing and dynamic escalation to human agents when needed, streamlining issue resolution and enhancing overall customer satisfaction. With this powerful fusion, businesses can create more efficient, personalized, and interactive IVR experiences, fostering more robust customer relationships and setting new standards for modern phone systems.

5.4 Location Transparency Call (LTC) System

5.4.1 Background

In today's fast-paced business landscape, missed calls can have significant ramifications, leading to lost opportunities and potential revenue. Businesses typically mitigate the impact of missed calls through voicemail. Voicemail [124] is a mature feature that comes off the shelf with the most unified communication (UC) solutions. A voicemail stores voice messages that the caller can optionally leave to be retrieved later by the person called. The UC platform lets the recipient retrieve voice messages through phone or email. However, statistics reveal that 80% of phone calls to businesses go to voicemails, and the average voicemail response rate is less than 5% [125]. The authors of [126] propose to send an SMS notification to the person missing a call on their desk phone. The authors of [127] interface with an Asterisk system and look for a series of SIP messages that indicate missed calls. They then extract the called phone number from each message, query a database for the corresponding mobile number of the called person, and send a text message if a result is found. IBM holds a patent [128] that utilizes SMS notification to a remote phone device when a missed call occurs. The approaches mentioned above, however, are somehow similar. They do not provide a solution to offer the highest availability, decrease missed call occurrences, and help reduce their corresponding drawbacks. Nevertheless, they provide means to facilitate the retrieval of the missed calls list. Organizations can do better by eliminating the utilization of voicemails in their firms and providing better availability to their employees to mitigate the impact of missed calls in the first place. Moreover, call centers reveal that a voicemail may be seen as a negative encounter for customers and subtracts from their satisfaction. Customers usually need instant answers to their inquiries and are unwilling to wait hours for someone to call back and help [129].

To address this critical challenge, we introduce the Location Transparency Call (LTC) System, a robust use-case scenario that offers a practical solution for businesses. Leveraging the seamless integration of IoT and VoIP in the proposed PoT framework, this innovative system enables business professionals to maintain constant connectivity and accessibility, regardless of their physical location. With PoT's IoT capabilities, the LTC System can intelligently and dynamically route incoming business calls to employees to the nearest extension at their current location within the business premises. The system ensures that calls are efficiently forwarded to the most convenient and available extension through real-time location tracking, effectively mitigating the risk of missed business calls. Whether on the go or working remotely, professionals can stay connected to their clients, colleagues, and opportunities, ensuring enhanced productivity and superior customer service. The LTC System stands as a testament to the transformative potential of PoT, empowering businesses to overcome communication barriers, maximize responsiveness, and seize every opportunity that comes their way.

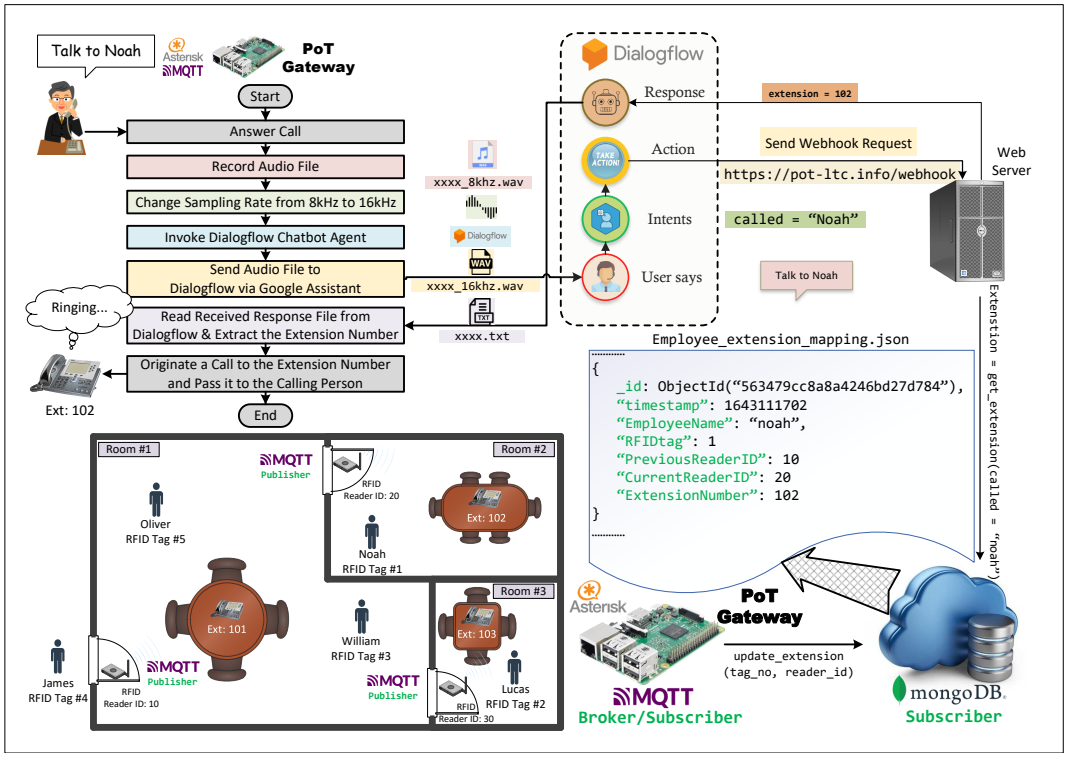

Figure 5.1: Overview of the Location Transparency Call (LTC) system.

5.4.2 Architecture

Figure 5.1 depicts the proposed LTC system. The LTC system is based on the PoT architecture proposed and delineated in the previous chapters and inherits its enabling technologies and modular design philosophy. LTC consists of a PoT gateway, a set of WiFi-enabled RFID door-entry nodes distributed along with the entrances of the rooms within the premise, and assistive cloud-hosted services (namely, MongoDB database and Dialogflow's chatbot agent). The door entry nodes are connected to the PoT gateway through Wi-Fi. The PoT gateway acts as an MQTT broker that relays user logs at door entry nodes (publishers) to the cloud-hosted MongoDB instance (subscriber). User logs include their RFID tag numbers, the ID of the door entry that publishes the log, and the timestamp of the log occurrence. The cloud-hosted database maps the user tracking, based on their RFID tag numbers, to the extension number(s) at their current location, depending on the ID of the door entry node that pushes the log information. Upon dialling a preconfigured hotline extension number, the call is auto-answered by the embedded Asterisk server of the PoT gateway, and it invokes a developed Python script. The script saves the user's utterance as an audio file, sends it to the developed Dialogflow's chatbot agent, and waits for a reply. The chatbot agent recognizes the user's intent, identifies entities in the user's speech, and determines the target employee to be called based on the exact explicit name mentioned or the question asked by the calling person. The chatbot then sends a webhook request to the cloud-hosted database querying the extension number at the current location of the target person to be called. Upon reception of the extension number, the Dialogflow chatbot sends a text file to the PoT gateway that contains the extension number. A Python script reads the received response file from Dialogflow and extracts the extension number. The script then passes the extension number to the Asterisk server running on the PoT gateway to originate a call to that extension number and pass the call to the calling person.

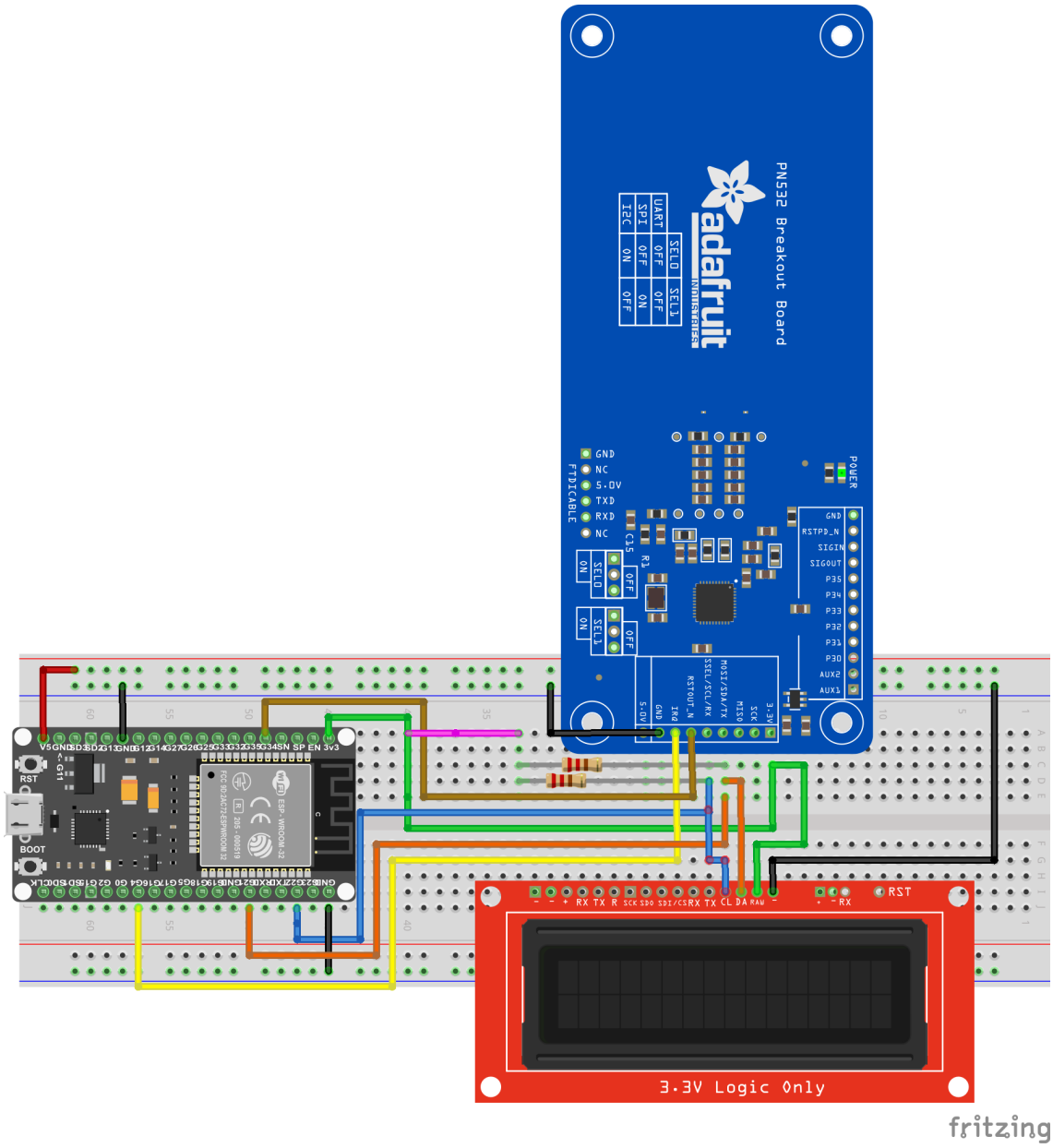

Figure 5.2: The door entry node component of the proposed LTC system (breadboard view).

5.4.3 WiFi-Enabled RFID Door Entry Nodes

Door entry nodes are a vital component of the LTC system, and they are used to track users' current locations while they move inside the enterprise premises. In the proposed LTC model, we use the Radio Frequency Identification (RFID) [130] technology to track the users' mobility while they enter or exit the rooms. The door entry node, depicted in Figure 5.2, mainly consists of an RFID reader and a Wi-Fi module. The door entry node is meant to be simple, cheap, and easy to install and integrate under different circumstances without potential infrastructure modification. The proposed door entry node uses NXP's PN532 Near Field Communication (NFC) controller for RFID functionalities and Espressif's ESP32 Microcontroller Unit (MCU) for the application logic and Wi-Fi connectivity.

When a user carrying an RFID tag passes by a door entry node, the door entry node (MQTT publisher) reads the RFID tag and publishes the tag number and the reader ID to the MQTT broker (the PoT gateway). The web server (MQTT subscriber) subscribes to all users' topics by default, receives the update from the PoT gateway and updates the MongoDB database accordingly. The MongoDB database dynamically maps the nearest extension number to a specific user based on his current location. The nearest extension number is determined by the following: the current door entry checkpoint, the previous door entry checkpoint, and the preconfigured table created at the system startup. The preconfigured table maps the checkpoints' sequence to the location based on the specific floor plan of the premise.

It is worth noting that the emergence of BLE/LoRa tags and beacons also proposes suitable and cost-effective indoor positioning candidates for LTC applications [131]. Moreover, LTC can take advantage of the enterprise's wireless distribution system (WDS) to track people's location without further infrastructure requirements. This is believed to reduce the system cost; since the LTC system would only need the PoT gateway to

function. This also enhances the organizations' workflow; since the system would neither require the employees to tap their RFID tags every time they enter or exit a place nor need expensive long-range RFID readers to be utilized by the door entry checkpoints.

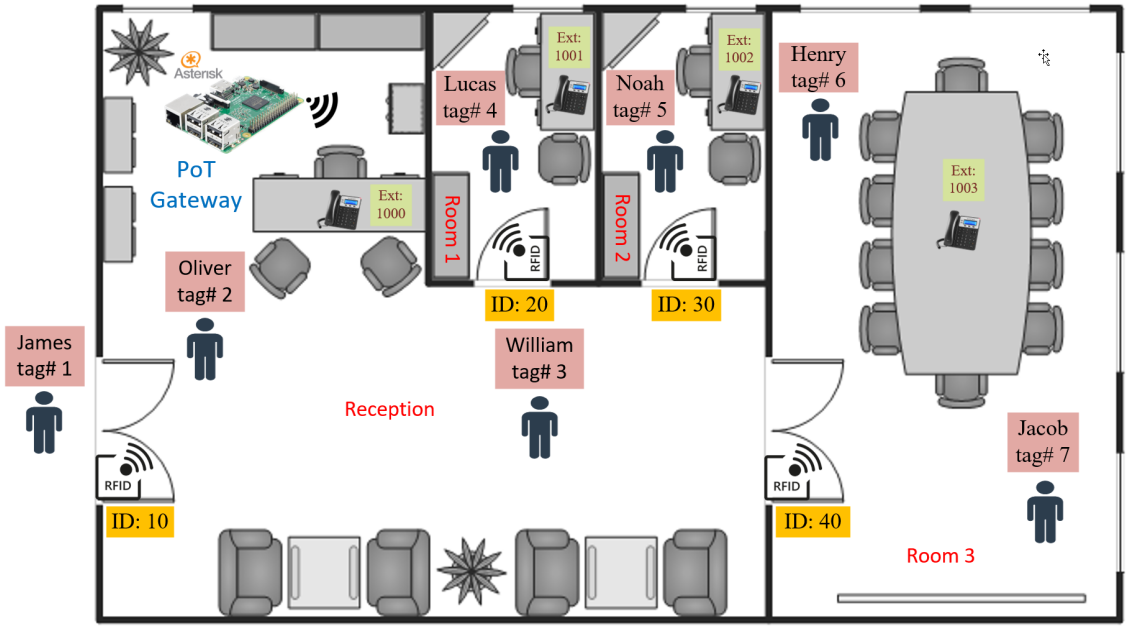

Figure 5.3: Example scenario of entering and exiting premises.

5.4.4 Entering and Exiting a Place

The proposed LTC system uses a single-door entry checkpoint for each room to track the users' mobility. The architecture of the door entry node, depicted in Figure 5.2, depends on the RFID reader, and it does not utilize any other kind of sensors to help identify whether the user is entering or exiting the room. This is meant to simplify the design and the installation of the door entry node. We designed the tracking database's schema to maintain fields that hold the identities of the current door entry checkpoint and the previous door entry checkpoints. Knowing these two identities of door entry checkpoints and the truth table that is manually configured during system startup based on the specific floor plan of the premise, we can determine whether the user is entering or exiting a place. Moreover, we set up a fallback extension for invalid unmapped entries in

the truth table. Invalid entries may be due to system glitches or missed RFID tag readings when the users enter or exit places. The fallback extension number can be configured as the default extension number of the user or as an arbitrary extension number (i.e., the receptionist extension).

Figure 5.3 gives an example scenario that illustrates the leverage of a manually pre-built truth table and a single RFID checkpoint at each door entry to determine the user location. Table 5.1 shows the truth table of the example scenario. The table logically maps the specific floor plan given in the example. It states the location based on the previous (i.e., t−1) and the current (i.e., t) identities of the door entry checkpoints. All users' (t−1) entries are initialized to void/empty upon system startup. This indicates that the user does not exist within the company's premises. This methodology is easy to implement and does not propagate errors in case of missed or invalid reading sequences. Table 5.2 gives examples of valid and invalid entries based on the truth table mentioned above.

5.5 Session Semantic Utilization

The inherent session-based message semantics of the SIP protocol and its constrained implementation in the proposed tSIP enable the development of advanced IoT solutions. This would be advantageous for IoT applications requiring continuous real-time streaming of sensor node data in case of emergencies when critical alarms are triggered. For example, session-based semantics can be utilized in medical applications where tSIP-enabled body sensors can autonomously originate a call to the Public Safety Answering Point (PSAP) and stream their readings to them when the readings exceed their typical values. This makes it easier for doctors to quickly determine how sick a patient is and direct them to the right place to get the necessary care.

Also, in the realm of modern device communication, ensuring seamless and reliable

Table 5.1: Truth table of the example scenario's floor plan.

Reader ID (t−1)	**Reader ID (t)**	**Location**	**Extension No.**	**Remark**
-	10	Reception	1000	Coming
10	10	Reception	1000	Leaving (Fallback)
10	20	Room 1	1001	Entering Room 1
10	30	Room 2	1002	Entering Room 2
10	40	Room 3	1003	Entering Room 3
20	10	Reception	1000	Leaving (Fallback)
20	20	Reception	1000	Entering Reception
20	30	Room 2	1002	Entering Room 2
20	40	Room 3	1003	Entering Room 3
30	10	Reception	1000	Leaving (Fallback)
30	20	Room 1	1001	Entering Room 1
30	30	Reception	1000	Entering Reception
30	40	Room 3	1003	Entering Room 3
40	10	Reception	1000	Leaving (Fallback)
40	20	Room 1	1001	Entering Room 1
40	30	Room 2	1002	Entering Room 2
40	40	Reception	1000	Entering Reception

Table 5.2: Entering and exiting illustration example.

No.	**Name**	**RFID Tag No.**	**Reader ID (t−1)**	**Reader ID (t)**	**Location**	**Extension No.**
1	James	1	10	10	Not Exist	1000 (Fallback)
2	Oliver	2	-	10	Reception	1000
3	William	3	20	20	Reception	1000
4	Lucas	4	10	20	Room 1	1001
5	Noah	5	20	30	Room 2	1002
6	Henry	6	10	40	Room 3	1003
7	Jacob	7	30	40	Room 3	1003
8	Lucas	4	-	20	Invalid	1000 (Fallback)

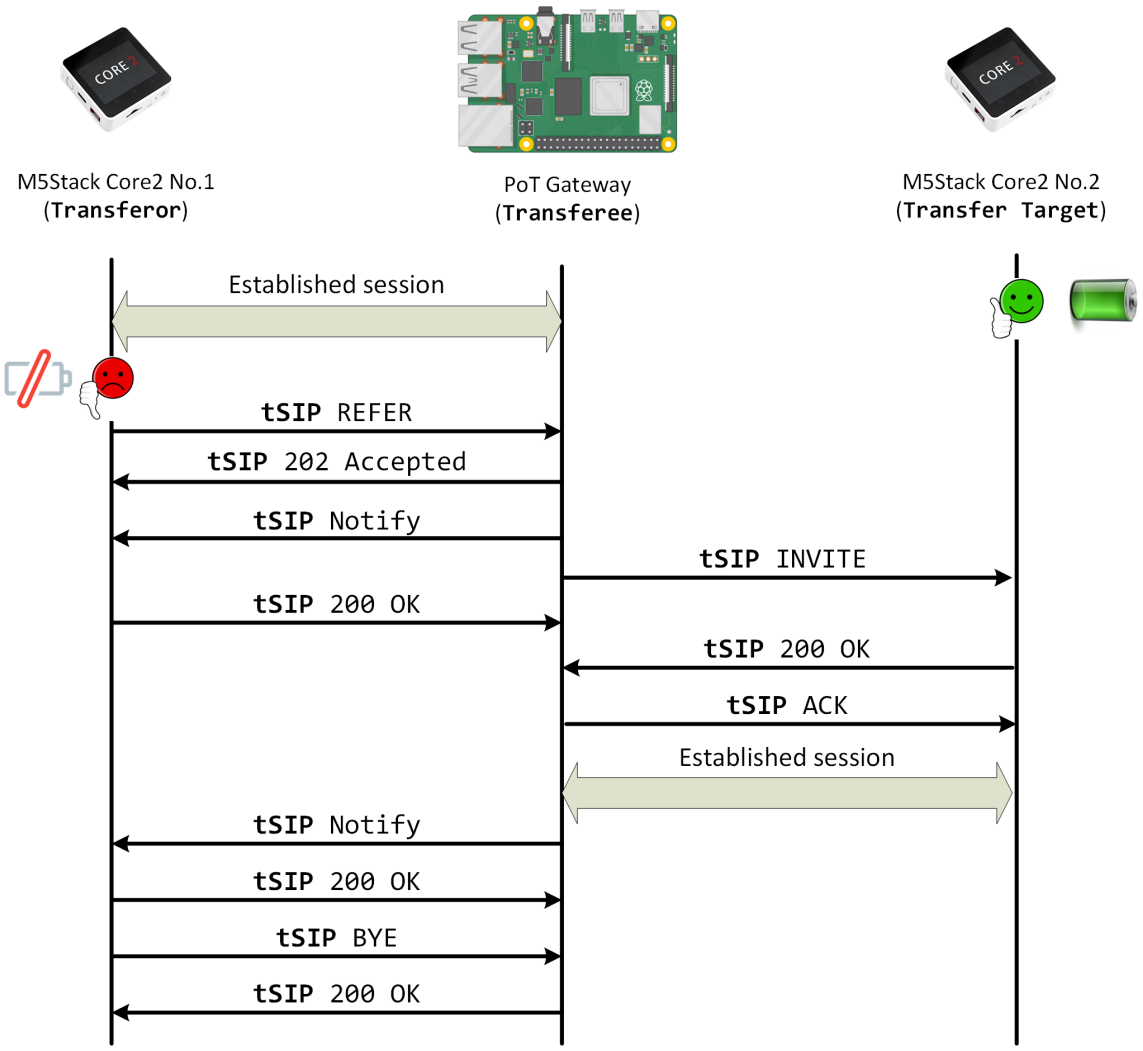

Figure 5.4: An example of tSIP REFER utilization in a session semantics application.

interactions between connected devices is of utmost importance. The proposed resource-constrained version of the SIP protocol (i.e., tSIP) delineated in the previous chapter stands at the forefront of facilitating this communication with remarkable finesse. One of its key strengths lies in its potential adept utilization of session semantics, enabling robust and contextually aware device communication. Through session semantics, the PoT gateway can establish and manage communication sessions that go beyond mere data exchange, encompassing rich context and application-specific information. This allows devices to engage in more meaningful interactions, dynamically adapting to changing network conditions and ensuring uninterrupted communication even in challenging environments. By harnessing session semantics implementation by tSIP, it empowers associated devices with the gateway to communicate intelligently, optimizing resources and enhancing overall reliability, thereby fulfilling robust communication between the gateway and the associated devices.

For example, the tSIP implementation of the `REFER` method allows the tSIP-enabled sensor node (*transferor*) to redirect the PoT gateway (*transferee*), establishing a session with it, to another sensor node (*transfer target*) with similar capabilities if the current sensor node (*transferor*), for example, signifies insufficient residual energy or hardware resources to fulfill its current session involvement with the PoT gateway (*transferee*), as depicted in Figure 5.4. The transferor sensor node sends a constrained `REFER` message to the PoT gateway in this context. Leveraging the location database in the PoT gateway, the PoT gateway searches for an alternative sensor node with similar capabilities and informs the transferor sensor node that it accepts the `REFER` message. The PoT gateway then tries to establish a session with the alternate sensor node (*transfer target*). Once the session is established between the PoT gateway and the alternate sensor node, the PoT gateway notifies the transferor sensor node that the new session has been successfully established with the transfer target. Eventually, the transferor sensor node terminates its established session with the PoT gateway.

5.6 Summary

The chapter delves into diverse use case scenarios of the proposed Phone of Things (PoT) framework, showcasing its immense potential in reshaping the telecommunication and IoT landscapes. First, the chapter explores context-aware telephony solutions. PoT's integration of IoT and VoIP technologies enables seamless interactions with the surrounding devices, leveraging real-time data for more intelligent, personalized, and intuitive user experiences of phone calls. Next, the chapter highlights how PoT redefines mature phone features, such as call forwarding, through its chatbot integration, optimizing Interactive Voice Responses (IVR) and streamlining communication. The chapter further unveils the Location Transparency Call (LTC) system, powered by PoT, which mitigates the impact of missed business calls by intelligently and dynamically routing them to alternate destinations based on the information regarding the location and availability of called parties inferred from the surrounding devices associated with the gateway. Lastly, the chapter delves into implementing session semantics in tSIP, showcasing how this lightweight SIP stack ensures robust and contextually-aware device communication, enabling devices to adapt to changing network conditions intelligently.

Collectively, the chapter demonstrates how the proposed PoT framework empowers a wide range of industries with innovative telephony solutions, ultimately fostering enhanced user experiences, improved operational efficiency, and seamless device communication in the modern interconnected world.

Chapter 6

Experimental Setup and Evaluation

In this chapter, we present the experimental evaluation and results of the Phone of Things (PoT) platform proposed in the thesis that leverages Single Board Computers (SBCs) as gateways to enable seamless integration and communication between the surrounding devices and the communication servers. The core communication framework of the proposed PoT platform is based on a lightweight version of the Session Initiation Protocol (SIP), called tSIP, designed to facilitate efficient message exchange between devices and their corresponding gateways. Additionally, the PoT platform implements a novel blockchain-based registration and authentication mechanism to enhance security and enable a robust association of devices to their respective gateways. The evaluation is conducted in a real-world environment comprising the essential framework's components. This includes SBCs as gateways, devices (i.e., legacy and IoT devices), and assistive optional cloud infrastructure (i.e., OpenVPN server and Google Assistant). We measured key performance metrics of the proposed framework. This includes resource utilization of the embedded Linux SBCs when acting as gateways in the proposed framework. We also evaluate the characteristics of the proposed tSIP messaging protocol based on its implementation in the proposed framework to show the benefits it brings in terms of the resultant message sizes compared to the original SIP messages and CoSIP counter-

parts. Furthermore, we assessed the platform's security by analyzing the efficiency of the blockchain-based registration and authentication mechanism. The experimental results demonstrate that the proposed PoT platform efficiently utilizes SBC resources, optimizing its deployment cost. Moreover, the blockchain mechanism exhibits robust security. It facilitates secure device registration and authentication, making it a promising solution for seamlessly integrating and managing surrounding devices in real-world applications.

6.1 Experiments Setup

The PoT gateway in the proposed framework boasts a modular and robust software stack designed to facilitate the seamless integration of different software components, making it easier to customize the gateway according to specific requirements and use cases. At the core of the software stack lies a scalable and reliable operating system, i.e., Linux, which forms the foundation for all other components. We choose the Raspberry Pi OS (previously known as Raspbian). Raspberry Pi OS is a popular choice as it is optimized for Raspberry Pi hardware. On top of the OS, the gateway utilizes the Asterisk open-source IP-PBX application, empowering it to handle VoIP communication and manage various telephony functionalities efficiently. The gateway employs an OpenVPN client application to create a secure tunnel between the gateway and an external OpenVPN server, promoting system scalability by providing secure communication between distant gateways. Additionally, the software stack incorporates the tSIP implementation scripts developed in Python to map original SIP messages to their constrained tSIP message counterparts and vice versa. The software stack also integrates the essential libraries for blockchain technology, enabling secure and transparent transaction issuing by the gateway to the blockchain network to register and authenticate associated devices to the gateway. Moreover, the software stack of the gateway includes the developed Python scripts to interact with Google Assistant, offering a seamless and user-friendly interface

for device interaction. This comprehensive software stack empowers the POT gateway to efficiently manage data flow, ensure robust security, and provide a seamless user experience.

Nevertheless, to ensure process monitoring and management of the different software components within the PoT's gateway software stack, we utilize *s6-overlay*. s6-overlay is a lightweight and powerful process supervision suite designed to enhance the reliability and stability of running applications, which aligns perfectly with the modular nature of the POT gateway's software stack. By integrating s6-overlay into the POT gateway, each individual software module operates within its own separate and supervised process. This process isolation ensures that if one component encounters an issue or crashes, it will not impact the stability or availability of the entire system. s6-overlay constantly monitors these processes, automatically restarting them if necessary, maintaining continuous operation even in the face of failures. The process monitoring capabilities of s6-overlay are crucial for the environment of the proposed PoT framework, where the POT gateway may interact with numerous devices and handle diverse data streams. The constant supervision of the software modules enables the gateway to maintain optimal performance and quickly recover from any unforeseen issues, thus enhancing the overall reliability of the proposed POT ecosystem. Additionally, s6-overlay allows for efficient resource utilization, as it optimizes memory usage and avoids unnecessary overhead. This is especially advantageous in resource-constrained devices like the PoT gateway, where the efficient management of processes becomes vital. Furthermore, s6-overlay complements the scalability of the PoT framework since s6-overlay can adapt and efficiently handle the increased workload, thanks to its modular approach and process supervision capabilities.

6.2 Experiments Objectives

The proposed PoT framework's primary objectives are to enable seamless communication and interaction between devices and communication servers while ensuring secure and robust device registration and authentication at the gateway level and optimizing its deployment cost, promoting its wide adoption in home automation and other industries. The following experiments aim to validate the performance of the underlying tiny and cost-effective hardware platforms used in the proposed gateway prototypes to prove these objectives. The experiments also assess the performance and efficiency of the constrained tSIP protocol in this context and evaluate the security and scalability of the blockchain-based registration and authentication mechanism. By achieving these objectives, the proposed PoT framework provides a robust and user-friendly solution for managing and interacting with the surrounding devices via the existing communication servers within the premises while maintaining a secure and reliable communication environment.

6.3 PoT Gateway as an IP-PBX Server

6.3.1 Passthrough VoIP Testing

The performance evaluation of the Raspberry Pi boards when processing passthrough VoIP calls is shown in Figures 6.1 and 6.2. The maximum number of simultaneous calls the boards can gracefully handle are contrasted in Figure 6.5. Obviously, the maximum number of simultaneous calls is proportional to the board's hardware specifications concerning the CPU core type, number of cores, and memory size.

The results show that the least powerful Raspberry Pi board family model, namely, Raspberry Pi Zero W, can gracefully handle up to 24 active channels, representing 12 simultaneous calls, after which the load average starts to exceed one. The CPU utilization of the board at this maximum number of simultaneous calls is 94%. On the other hand,

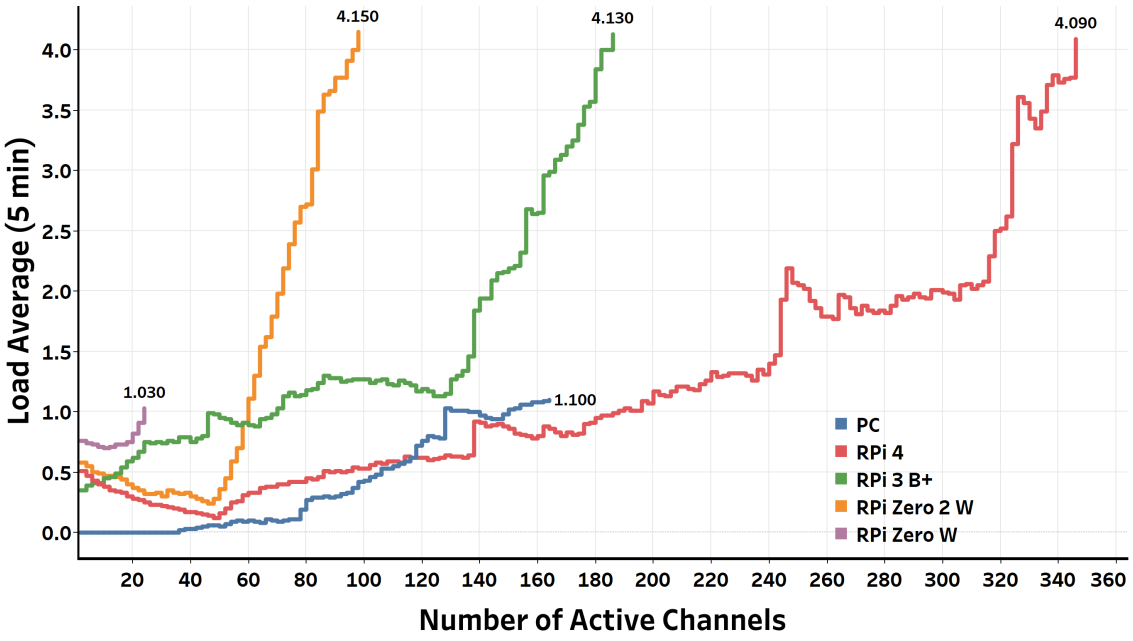

Figure 6.1: Relation between the number of active channels and the operating system load average (passthrough).

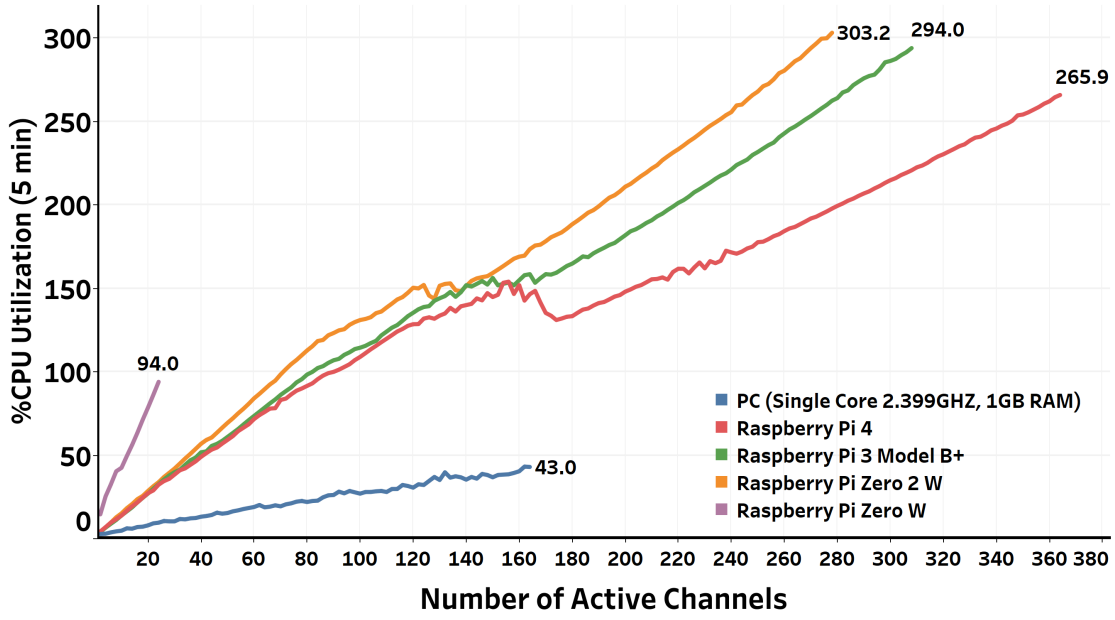

Figure 6.2: Relation between the number of active channels and the CPU utilization (passthrough)

Raspberry Pi 4 B, the current most powerful model of the Raspberry Pi board family, can gracefully handle up to 364 active passthrough channels (182 simultaneous calls) before the five-minute load average starts to exceed four. The CPU utilization of the board at this number of simultaneous passthrough VoIP calls is 266% (66.5% per core). Based on the board's hardware specifications, the maximum number of simultaneous calls for other Raspberry Pi models lies between Raspberry Pi Zero W and Raspberry Pi 4 B.

It is worth noting that Raspberry Pi Zero 2 W, the newest member of the Raspberry Pi boards family that launched in October 2021, has comparable performance to Raspberry Pi 3 B+. Raspberry Pi Zero 2 W comes at the same form factor as Raspberry Pi Zero W with a neglectable price increase ($19 CAD for Raspberry Pi Zero 2 W compared to $15 CAD for Raspberry Pi Zero W). However, Raspberry Pi Zero 2 W comes with a quad-core ARM Cortex A53 processor, unlike the single-core ARM11 that comes with Raspberry Pi Zero W. The results show that Raspberry Pi Zero 2 W can gracefully serve up to 278 active channels, whereas Raspberry Pi 3 B+ can gracefully serve 308 active channels. Therefore, Raspberry Pi Zero 2 W can ideally fit PoT applications where physical size and power consumption are the primary concerns.

Except for Raspberry Pi Zero W, the performance of the Raspberry Pi boards used in the evaluation exceeded the VPS instance's performance, whose specifications are listed in Table 3.1. Raspberry Pi Zero 2 W, for instance, can support an approximately 70% higher number of active channels than the number of active channels supported by the VPS. It is also worth noting that the boards can still serve more VoIP calls if we switch the threshold to the 15-minute load average instead of the 5-minute load average or raise the threshold limit slightly beyond four. However, when the load average exceeds four, some VoIP call processes will be queued, waiting to be served by the operating system. This queuing affects the soft real-time constraints of VoIP, degrading the quality measurements of the established VoIP calls.

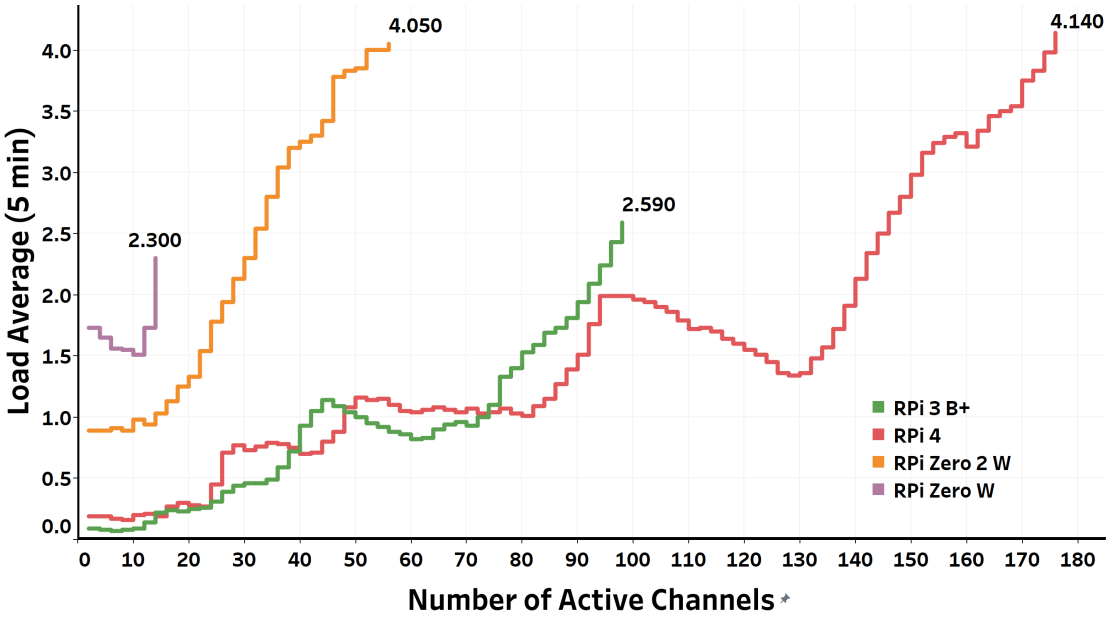

Figure 6.3: Relation between the number of active channels and the operating system load average (transcoding).

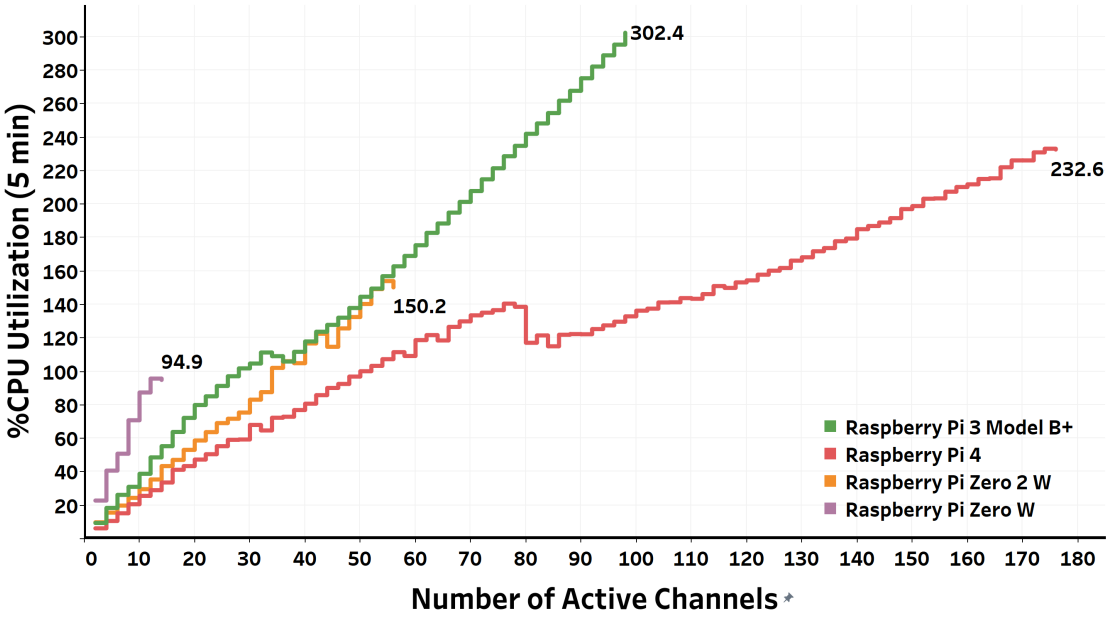

Figure 6.4: Relation between the number of active channels and the CPU utilization (transcoding).

6.3.2 Transcoding VoIP Testing

The simultaneous transcoding VoIP call capacity testing results that the Raspberry Pi boards can safely process are shown in Figures 6.3 and 6.4. In contrast to passthrough VoIP, transcoded VoIP is a process-intensive task that consumes many resources. Therefore, it is obviously expected that a smaller number of simultaneous transcoding VoIP calls can be afforded than simultaneous passthrough VoIP calls using the same platform. After using the same thresholds imposed on the testing algorithm when performing the passthrough testing, the results showed that the Raspberry Pi 4 B board, for example, can gracefully handle up to 176 active channels (88 simultaneous calls) before outpacing the thresholds. This represents 48% of passthrough VoIP calls the same board can afford. The ratio between the number of graceful transcoding VoIP calls to the number of graceful passthrough VoIP calls is proportional to the board's processing power. The higher the board's processing power, the higher the ratio of transcoded to passthrough VoIP calls the board can gracefully afford. Concerning the VoIP call quality metrics of established calls throughout the testing procedure, the results from Wireshark, as summarized in Table 6.1, show values of jitter and packet loss that all fall within the acceptable standardized limits of VoIP call quality metrics.

6.3.3 Passthrough vs. Transcoding VoIP Calls

It is worth noting that the number of maximum simultaneous transcoded VoIP calls represents an extreme, where every originated call needs transcoding, of what actually happens in real situations. In most real cases, transcoding is more likely avoided by enforcing particular audio codecs to be utilized by the communicating parties. Transcoding only happens when interconnecting with third-party providers, e.g., Internet Telephony Service Providers (ITSP), that operate other audio codecs to what we have already utilized. Therefore, the maximum number of simultaneous VoIP calls the Raspberry Pi

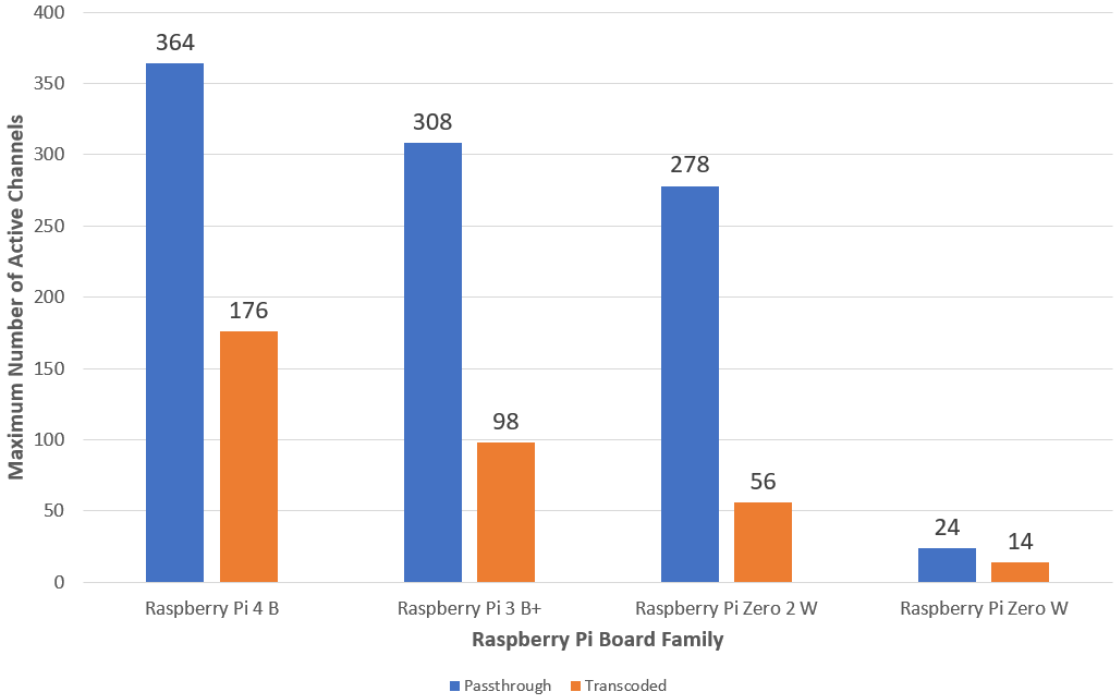

Figure 6.5: Contrasting the maximum number of simultaneous active channels that different Raspberry Pi board families can gracefully serve.

Table 6.1: VoIP call quality measurements during passthrough and transcoded VoIP call testings for different Raspberry Pi boards.

	RPi 4 B		**RPi 3 B+**		**RPi Z 2 W**		**RPi Z W**	
	Passthrough	**Trans.**	**Passthrough**	**Trans.**	**Passthrough**	**Trans.**	**Passthrough**	**Trans.**
Forward (Phone-to-RPi)								
Max jitter (milliseconds)	9.35	10.49	11.87	10.97	12.05	13.85	15.33	16.02
Mean jitter (milliseconds)	7.08	6.45	8.09	6.88	9.96	10.13	11.58	13.69
RTP Packets	132,486	115,715	111,460	65,414	101,623	37,211	8688	5058
Expected	132,486	115,715	111,460	65,414	101,623	37,211	8688	5058
Packet loss (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Reverse (RPi-to-Phone)								
Max jitter (milliseconds)	5.07	5.98	5.90	8.43	8.85	9.32	10.02	20.36
Mean jitter (milliseconds)	0.65	0.68	0.70	0.73	0.98	0.85	0.90	3.5
RTP Packets	132,450	115,624	111,365	65,369	101,577	37,188	8610	5049
Expected	132,450	115,624	111,365	65,369	101,577	37,188	8610	5049
Packet loss (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

boards gracefully serve lies somewhere between the maximum number of passthrough VoIP calls and the maximum number of transcoding VoIP calls the specific platform can serve. The exact number of simultaneous VoIP calls depends on the application that defines the expected setup of the established VoIP calls.

6.3.4 Conclusion

The feasibility study of employing Raspberry Pi boards as suitable PoT gateway candidates to handle an adequate number of simultaneous calls for a PoT application demonstrates promising results. The Raspberry Pi's versatility and low cost make it an attractive choice for creating PoT gateways with VoIP capabilities. By integrating Asterisk, the Raspberry Pi can effectively manage multiple concurrent calls while connecting various devices within the PoT ecosystem. However, it is crucial to consider the specific requirements of the PoT application and the expected call volume. A single Raspberry Pi may suffice for smaller-scale PoT deployments, like in home automation. However, for more extensive and demanding PoT networks, multiple or a cluster of Raspberry Pi boards might be necessary to ensure optimal call quality and performance. Despite its inherent processing power and memory limitations, the Raspberry Pi's ability to handle simultaneous calls, combined with its GPIO pins for connecting sensors and actuators, makes it an attractive candidate for implementing cost-effective and adaptable PoT gateways.

6.4 Integration with Chatbot Agents

We deployed and tested chatbot integration in the proposed framework to leverage chatbot agents' sophisticated natural language processing and AI algorithms capabilities, such as Google Assistant and Amazon Alexa. This integration enables the provision of innovative telephony solutions that help enhance user experience with phone calls, as delineated in Chapter 5. We tested integrating the gateway with Google Assistant,

empowering users with new possibilities for querying and monitoring their surroundings, making virtual assistants more inclusive, accessible, and valuable in diverse scenarios. Also, phone call queries can be useful in areas with limited internet access or during network outages.

6.4.1 Google Assistant: Intents, Actions, and Fulfillment

Google Assistant's workflow incorporates the interaction of actions, intents, and fulfillment, the basic building blocks of Google Assistant, to provide an interactive conversational user experience. The following summarizes these elements:

1. *Intents*: Intents represent the user's intentions or requests expressed in natural language. When users interact with Google Assistant, their voice query or typed message is processed to identify the intent behind it. Dialogflow, a natural language understanding (NLU) platform developed by Google, plays a crucial role in determining the intent. Dialogflow examines the user's input, recognizes the intent, and extracts relevant entities (i.e., parameters) from the user's query, enabling the Assistant to understand the user's request accurately.

2. *Actions*: Actions are the logical representations of an application or service's various capabilities and functionalities built for Google Assistant. Each intent corresponds to a specific action or set of actions. For example, by asking for a person's name in the proposed location transparency call (LTC) system proposed in Chapter 5, the corresponding action queries a database about the nearest extension number to the called person's current location within the premises, which is dynamically mapped by the LTC system utilizing the information inferred from the RFID sensor nodes within the premises. Therefore, actions are designed to handle different intents and provide appropriate responses based on the user's query.

3. *Fulfillment*: Fulfillment is the backend service or code that processes the user's

intent and generates a response based on the action associated with that intent. It is responsible for fulfilling the user's request with dynamic and personalized content. The fulfillment service can interact with databases, APIs, or other external services to gather the necessary information to respond accurately to the user's query. In the case of the LTC system example mentioned earlier, the fulfillment is the text file sent by Google Assistant to the system containing the extension number to reach the person called.

6.4.2 PoT Gateway and Google Assistant: The Integration Workflow

The workflow of the integration of Google Assistant with the PoT gateway works as follows:

1. *Triggering the Assistant*: The user initiates a phone call by dialling a specific phone number on the embedded Asterisk server to the gateway, which is associated with integrating the gateway and Google Assistant. The embedded Asterisk server automatically invokes a developed Python script upon receiving a call on this specific extension, leveraging the power of Asterisk's AGI (Asterisk Gateway Interface) feature. We apply the necessary configurations in the `extensions.conf` file of the Asterisk system to define the dialplan and specify the extension context, extension number, and the AGI command to execute the Python script.

2. *Establishing a connection with Google Assistant*: Leveraging Google's Python SDK, we install the necessary packages, including the Google Assistant library and its dependencies on the gateway. We set up a Google Cloud Project and enabled the Google Assistant API, obtaining the necessary credentials for authentication. With the credentials in place, the developed Python script creates an instance of Google Assistant. The Assistant SDK handles audio input/output and communication

with the Google Assistant service. Through this connection, the script can send user utterances or text queries to Google Assistant and receive responses, allowing integration of the power of Google Assistant into the gateway.

3. *Sending user queries to Google Assistant*: The developed Python script interfaces to the established VoIP channel with the user. It stores user utterances as an audio file. The sampling rate of the audio file is 8kHz, the default sampling rate for phone calls. The script then oversamples the audio file into the required format for communication with the Assistant API (i.e., 16kHz) and sends the processed audio data as a voice query to the Google Assistant service, which will process the user's utterance and generate a response.

4. *Taking actions and generating responses*: We developed a prototyping Dialogflow chatbot application for the proposed framework. The bot is connected to Google Assistant, allowing users to interact with the bot through Google Assistant. When a user initiates a conversation with Google Assistant through the gateway and addresses the chatbot, Google Assistant processes the user's voice query and sends it to the Dialogflow chatbot application for analysis. The chatbot application interprets the user's intent, entities, and context based on the trained data to understand the user's request accurately. Based on this analysis, the chatbot takes actions like fetching data from databases, calling APIs, interacting with other services, or generating a custom response tailored to the user's query.

5. *Playing back responses*: The generated responses by Google Assistant are fetched by the developed Python script, downsampled from 16kHz to 8KHz by the script and played back to the user through the established VoIP channel, completing the conversation loop.

6.4.3 Voice Activity Detection (VAD)

Voice Activity Detection (VAD) is a fundamental audio processing technique for detecting and distinguishing speech segments from silence or background noise in audio stream applications. The primary objective of VAD is to identify user utterances, allowing applications to focus on processing and analyzing only the relevant speech potions, thereby optimizing computational resources compared to other approaches like continuous listening and fixed-time window. VAD improves the user experience in the proposed integration between the gateway and Google Assistant, as it ensures that audio file processing by the gateway is triggered only when user utterance is detected, optimizing resource utilization of the board.

We integrate this functionality into the developed script using the WebRTC VAD library for Python (i.e., `webrtcvad`). First, we install the necessary Python packages and the `webrtcvad` library. We read the audio data from the real-time audio stream of the established VoIP channel with the user. The VAD algorithm segments the audio into frames and classifies each frame as either containing speech or not. Analyzing the classification results lets us detect voice activity and identify speech regions within the audio, reducing the amount of audio data that needs to be processed and transmitted to Google Assistant.

The following Python pseudo-code provides a general outline of the aforementioned steps to interface the PoT gateway to Google Assistant. For simplicity, the pseudo-code utilizes the Asterisk Manager Interface (AMI) to answer the call, record user utterances while waiting for silence, send the recorded audio file to Google Assistant, wait for a response, and then play back the response to the caller. The Asterisk Manager Interface (AMI) is a set of commands and events that interact with the Asterisk system. It allows external applications or scripts to programmatically control and monitor various aspects of the Asterisk server.

```
import time
from asterisk.ami import AMIClient, SimpleAction
import webrtcvad             # VAD library

### Function to apply voice activity detection to audio data
def apply_vad(audio_data, vad):
    is_speech = vad.is_speech(audio_data, sample_rate=8000)
    return is_speech
### Function to record audio from the channel using AMI with VAD
def record_audio_with_vad(client, channel, filename):
    vad = webrtcvad.Vad()
    vad.set_mode(2)          # Set VAD aggressiveness (0-3)
    response = client.send_action(SimpleAction(
        "MixMonitor",
        Channel=channel,
        File=filename,
        Format="wav"
    ))
    if response.get("Response") == "Success":
        print(f"Recording started on channel {channel}.")
    else:
        print(f"Failed to start recording on channel {channel}.")
    # Loop for recording until silence is detected
    while True:
        response = client.send_action(SimpleAction(
            "GetChannel",
            Channel=channel,
            Variable="SPEECH(dB)"
        ))
```

```
30          rms = float(response.get("Value", "0"))
31          # Read audio data from the channel
32          audio_data = read_audio_data_from_channel(channel)
33          # Apply VAD to check if there's speech in the audio data
34          is_speech = apply_vad(audio_data, vad)
35          if not is_speech:
36              break          # Stop recording if silence or no speech is detected
37          time.sleep(0.1)    # Wait for a short interval before checking again
38      # Stop the recording
39      response = client.send_action(SimpleAction(
40          "MixMonitorStop",
41          Channel=channel
42      ))
43      if response.get("Response") == "Success":
44          print(f"Recording stopped on channel {channel}.")
45      else:
46          print(f"Failed to stop recording on channel {channel}.")
47  def main():
48      # Set parameters and connect to Asterisk AMI
49      asterisk_host = "localhost"
50      asterisk_user = "username"
51      asterisk_pass = "password"
52      channel = "SIP/1001"   # Preconfigured SIP channel for Google Assistant
            integration
53      output_file = "user_utterance.wav"
54      silence_threshold = -40 # Silence threshold as needed based on the
            specific environment
55      # Connect to Asterisk AMI
56      client = connect_to_ami(asterisk_host, asterisk_user, asterisk_pass)
```

```
    # Answer the incoming call on the specified channel
    answer_call(client, channel)
    # Record the user's utterance and save it to a file using VAD
    record_audio_with_vad(client, channel, output_file)
    print(f"User's utterance recorded and saved to {output_file}")
    # Oversampling recorded audio file
    ...
    # Send converted audio file to Google Assistant
    ...
    # Wait to receive a response from Google Assistant
    ...
    # Downsample the received audio file of the response
    ...
    # Playback the response to the caller
    response_audio = "response_audio.wav"
    playback_audio(client, channel, response_audio)
    print("Response playback completed.")
    # Disconnect from Asterisk AMI
    client.close()

if __name__ == "__main__":
    main()
```

Listing 6.1: Python pseudo-code snippet for integrating Google Assistant with the PoT gateway.

6.4.4 Evaluating the Integration with Google Assistant

To evaluate the integration with Google Assistant in the PoT gateway, we measure the response time from the end of the user query until the user receives the response from Google Assistant. We use timestamps in the developed Python code to insert markers at key points, such as when the user query is processed and sent to Google Assistant and when the response from Google Assistant is received, processed, and starts playing back to the user. We use the `time` module in Python to calculate the timestamps as shown in Listing 6.2. By using timestamps, we can measure the time it takes for each specific part of the process.

Typically, the average response time from sending a query to Google Assistant until receiving a response can vary depending on several factors, including the network speed, processing power of the board, the complexity of the query, and the overall load on Google Assistant servers. Therefore, response times fluctuate due to network conditions and server loads.

```
import time

# ... (Code to record user query)
# Record the start time when the PoT gateway starts processing the user query
start_time = time.time()
# ... (Code for processing user query and send it to Google Assistant)
# Record the time when the query is sent to Google Assistant
start_time_response = time.time()
# ... (Waiting for response...)
# ... (Code for processing the response received from Google Assistant)
# Record the time when the response starts playing back to the user
end_time_response = time.time()
# ... (Code for playing back the response received from Google Assistant)
```

```
# Calculate the total response time
total_response_time = end_time_response - start_time
query_processing_time = start_time_response - start_time
response_time = end_time_response - start_time_response
print("Total Response Time:", total_response_time, "seconds")
print("Query Processing Time:", query_processing_time, "seconds")
print("Response Time:", response_time, "seconds")
```

Listing 6.2: Python pseudo-code snippet for measuring the response time when integrating the PoT gateway to Google Assistant.

We evaluate Google Assistant integration with the PoT gateway using different Raspberry Pi boards. We measure the time it takes for a query to be sent to Google Assistant and for the response to be received. We write a Python script that runs on each Raspberry Pi board to perform the test. The script initiates the communication with Google Assistant, places time stamps on the key points, as illustrated in Listing 6.2, sends the predefined query, records the response time, and then calculates and stores the results. We execute the test script on each Raspberry Pi board multiple times with different queries to accurately capture the average response time, as some queries may have different complexities and response times. The Python pseudo code of the test script is shown in Listing 6.3. We collect the results from all Raspberry Pi boards and Calculate the average response time for each board based on the multiple test runs.

```
import time
from google_assistant_api import send_query_to_google_assistant

def measure_response_time(query):
    start_time = time.time()
    response = send_query_to_google_assistant(query)
```

```
    end_time = time.time()

    response_time = end_time - start_time
    return response_time

def main():
    queries = ["What_is_the_weather_like_today.wav",
               "Tell_me_a_joke.wav",
               "What_is_the_square_root_of_64.wav"]

    for query in queries:
        response_time = measure_response_time(query)
        print(f"Query: {query} | Response Time: {response_time:.2f} seconds")

if __name__ == "__main__":
    main()
```

Listing 6.3: Python pseudo-code snippet for the test script to measure the response time when interfaces with Google Assistant.

Table 6.2: Response time of different Raspberry Pi boards when interfaced with Google Assistant.

	Raspberry Pi 4	**Raspberry Pi 3 B+**	**Raspberry Pi Zero 2 W**
Response Time	3.44	4.13	4.67

Table 6.2 shows the average response time obtained from running the test script on different Raspberry Pi boards. The differences in response times are attributed to variations in the different Raspberry Pi models' processing power. However, the results for the response times are generally suitable for PoT applications. The Raspberry Pi

4, with an average response time of 3.44 seconds, demonstrates the best response time among other Raspberry Pi boards. While slightly slower than the Raspberry Pi 4 model, the Raspberry Pi 3 and the Raspberry Pi Zero 2 W still manage an average response time of 4.13 and 4.67 seconds, respectively. These results indicate that these boards can handle voice interactions with Google Assistant within an acceptable response time range. For most PoT applications, these response times are well within acceptable bounds, allowing users to interact with the devices and the telephony system within the premises and receive timely responses without significant delays. The Raspberry Pi 4 would be preferred if the application requires faster response times. On the other hand, if the application has more relaxed response time requirements, the Raspberry Pi 3 or Zero 2 W might be sufficient, potentially allowing for cost-effective solutions in specific scenarios.

6.5 tSIP Protocol Evaluation

This section evaluates the proposed tSIP messaging protocol for PoT applications. The evaluation ensures seamless and efficient communication between PoT devices and the gateway in the proposed framework. We evaluate the size of selective tSIP messages with their counterparts in the literature, namely the SIP and CoSIP messages. The following subsections cover the evaluation workflow of the proposed tSIP messages.

6.5.1 Serialization and Deserialization of tSIP Messages

Serializing and deserializing tSIP messages is crucial when working with Protocol Buffers (Protobuf). Serialization refers to the process of converting structured tSIP data, represented by a .proto file, into a compact binary format that can be efficiently transmitted over networks or stored in files. This serialized binary data is platform-agnostic, making exchanging information between different components of the proposed framework written in different programming languages easy. Deserialization, on the other hand, is the

reverse process of serialization. It involves taking the binary data received and reconstructing it into its original structured form, which is the protocol buffer message. This allows applications to process the data and extract meaningful information efficiently.

Listing 6.4 shows the Python pseudo-code to implement the `.proto` file of tSIP presented in Section 4.2.2.2. We import the necessary modules and the auto-generated protocol buffer classes for the **.proto** file we get after running the `protoc`.

The `serialize_tSIP_message` function creates an instance of the tSIP message, sets the values for its fields, and serializes it to bytes. The `deserialize_tSIP_message` function takes the serialized data, deserializes it, and processes the retrieved fields.

```
# Pseudo-code for creating and serializing the tSIP message
function serialize_tSIP_message():
    // Create an instance of the tSIP message
    tsip_msg = tSIP()

    // Set the values for the fields
    tsip_msg.method = tSIP.REGISTER
    tsip_msg.Cseq = 1
    tsip_msg.Call_ID = 12345
    tsip_msg.Contact_IP = 192168001 // IP address of the device
    tsip_msg.Contact_Port = 5060  // Port number the device is listening to
    tsip_msg.PoT_IP = 192168002   // PoT GW IP address
    tsip_msg.PoT_Port = 5070      // PoT GW port number
    tsip_msg.sensor.extend([tSIP.TEMPERATURE, tSIP.HUMIDITY])

    // Serialize the message to bytes
    serialized_data = tsip_msg.SerializeToString()

    return serialized_data
```

```
20
21 # Pseudo-code for deserializing the tSIP message
22 function deserialize_tSIP_message(serialized_data):
23     // Create an empty instance of the tSIP message
24     tsip_msg = tSIP()
25
26     // Parse the serialized data into the tSIP message instance
27     tsip_msg.ParseFromString(serialized_data)
28
29     // Access the values of the fields for further process
30     print("Method:", tsip_msg.method)
31     print("Cseq:", tsip_msg.Cseq)
32     print("Call_ID:", tsip_msg.Call_ID)
33     print("Contact_IP:", tsip_msg.Contact_IP)
34     print("Contact_Port:", tsip_msg.Contact_Port)
35     print("PoT_IP:", tsip_msg.PoT_IP)
36     print("PoT_Port:", tsip_msg.PoT_Port)
37     print("Sensor:", tsip_msg.sensor)
```

Listing 6.4: Python pseudo-code snippet for serializing and deserializing tSIP messages.

6.5.2 tSIP Packet Capture

To capture the tSIP messages and calculate their sizes, we use Wireshark, a free network protocol analyzer, to capture the UDP packets from the network interface card (NIC) of the host machine used in the test. Also, we use the `Netcat` (`nc`) command in Linux to set up a UDP server on the host machine for receiving UDP packets through the network. To create the tSIP messages for evaluation purposes, we first used CloudShark to get shared packet captures of SIP messages. We encode their equivalent tSIP messages following

the steps delineated in Section 4.2. We use the `Nanopb` Protobuf compiler to create the compiled class bindings of the `.proto` file in the C language. This class is then imported to the M5Stack development board and is used by a developed sketch to encode the tSIP messages with the header field values equal to their corresponding ones in the shared SIP messages we get from CloudShark. We leveraged the auto-generated functions in the compiled C class to get the encoded tSIP messages and their sizes. The sketch then creates the corresponding tSIP packets for different session messages as delineated in Section 4.2.3. The sketch sends the packets to the UDP server, and their sizes (including encapsulations by the underlying layers of the UDP/IP protocol stack) are compared against their corresponding SIP messages. We also looked at how well tSIP and CoSIP compressed SIP messages. The results are shown in Table 6.3.

Table 6.3: Comparison between tSIP, CoSIP, and SIP message sizes for different message types used during session establishment and tear down.

Message type	**tSIP (bytes)**	**SIP (bytes)**	**tSIP Compression Ratio**	**CoSIP Compression Ratio**
REGISTER	69	460	0.150	0.451
INVITE	156	740	0.211	0.537
100 TRYING	105	503	0.209	0.505
180 RINGING	107	504	0.212	0.465
200 OK	112	514	0.218	0.577
ACK	98	485	0.202	0.595
BYE	110	546	0.201	0.592

Fig 6.6 depicts the size of SIP, CoSIP, and tSIP messages. tSIP achieves the highest compression ratio of SIP messages compared to the compression ratio of CoSIP. The resulting message size of tSIP is almost half the size of its corresponding CoSIP. This implies better bandwidth utilization and shorter transmit times, which also optimizes power consumption for tSIP-enabled constrained IoT devices. Furthermore, given Protobuf's portability, widespread community support, and the availability of Protobuf compilers in various programming languages, tSIP is far more suitable for deployment in heterogeneous applications. The sketch used to encode tSIP messages and make their packets takes up 15 KB of program storage space and 1 KB of dynamic memory. This means it

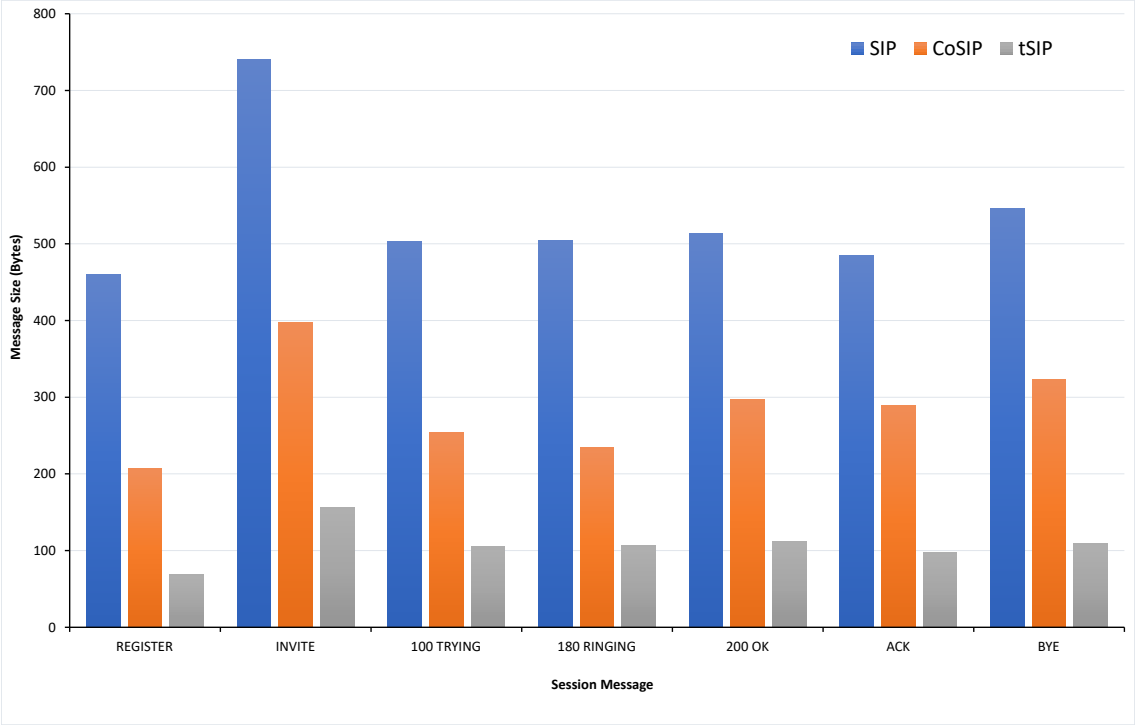

Figure 6.6: Message size in bytes per message type for SIP, CoSIP, and tSIP.

can be used on embedded devices with less memory and storage space.

6.6 Evaluation of the Registration and Authentication Mechanism

This section evaluates the registration and authentication mechanism designed for the proposed PoT framework. Through the evaluation, we seek to analyze this novel approach's performance, security, and usability aspects, ultimately determining its suitability for integration into the envisioned PoT framework. This evaluation will offer valuable insights and empirical evidence to inform the design decisions and highlight the potential benefits of adopting the new registration and authentication mechanism for securing the PoT ecosystem.

6.6.1 Implementations

6.6.1.1 The PBX server

As a proof of concept, we Implement a bare-metal Asterisk installation on a local Linux machine using virtualization techniques. The virtual machine of the PBX server features 1GB of RAM, 25GB of storage, and a single-core 2.4GHz CPU. Asterisk, an open-source PBX software, resembles the communication server within the premises for managing voice, video, and messaging services within the development environment for the evaluation. We utilize Asterisk 18 IP-PBX software installed on top of Ubuntu Server 18.04. The installation process involves setting up the Linux environment, installing prerequisite libraries and dependencies, and compiling Asterisk from the source code. Once installed, we configure extensions, SIP trunks, and dial plans, enabling call routing and communication features between the PBX server, SIP extensions, and the prototyping PoT gateways in the development environment as depicted in Figure 4.7. the PBX server is configured with two SIP trunks that connect the PBX server to the PoT gateways. We need not consider the PBX server's performance since the proposed mechanism delegates the registration and authentication of the PoT devices to the blockchain, avoiding impacting the PBX server's performance or affecting its design capacity. A single SIP trunk configuration to each PoT gateway abstracts the communication with each set of devices to their corresponding gateway. The performance evaluation of the PBX server is shown in Figure 6.1 and Figure 6.2 using the methodology presented in Section 4.1.6.

6.6.1.2 The blockchain node

The full node of the private blockchain in the proposed system is implemented on a Linux docker image running on a local Linux machine, and acts exactly as the public Ethereum blockchain. We install the Geth application on top of the Linux image, which is the standard distribution of Ethereum written in the Go programming language. We utilize

the PoA consensus algorithm, elevate the authority of the node to create new blocks, set up the PoA genesis block, and define the network's consensus rules and permissions. The utilized local Linux machine for the private blockchain node deployment features Intel Core i5 CPU@3.20GHz, 16GB RAM, and 500GB SSD HDD. The computation overhead (i.e., % CPU utilization) of the private blockchain node is around 41%.

We utilize a private blockchain with a reduced mining consensus algorithm, single network node, and hence reduced competition for block creation. Therefore, latency (the time taken for a transaction to be processed, confirmed, and made available on the blockchain network) and throughput (the number of transactions the network can process per second (TPS)) are negligible in our test environment. Alongside, it is worth noting that in the proposed mechanism, block creation is performed when issuing a transaction to register a PoT gateway or device for the first time. However, this would change if we utilize a public blockchain network like Ethereum.

6.6.1.3 The Gas Price

In blockchain networks, gas price is crucial in governing transaction costs and prioritizing transactions for block inclusion. Gas represents the computational effort required to execute a transaction or smart contract function. Every operation within a blockchain network consumes a specific amount of gas, and the gas price determines the fee users must pay for each unit of gas consumed. Setting a higher gas price incentivizes miners or validators to prioritize a transaction, as they are more likely to include it in a block and receive the associated transaction fees. Conversely, lower gas prices might lead to delayed transaction processing or even exclusion from blocks during times of network congestion. Gas prices are essential for striking a balance between transaction priority and network efficiency, and they allow blockchain networks to allocate computational resources fairly and efficiently while preventing resource abuse.

We use *REMIX*, an Ethereum IDE for contract development, to develop the smart

contract for the proposed registration and authentication mechanism. If applying the proposed mechanism to the public Ethereum network, the most expensive part will be deploying the smart contract during system initialization. The deployment cost depends on the memory utilization and the acquired computation resources in the developed smart contract functions. Approximately 0.000550 ETH will be spent to deploy the smart contract of the proposed mechanism, which works out to USD 1.03 at the average Ether price of $1,879.16 USD on July 29, 2023. Comparatively, it costs around 0.000097 ETH, or about $0.18 USD per transaction issuance, to register a new PoT gateway or device.

6.6.1.4 The PoT gateway

The PoT gateways are implemented using Raspberry Pi 3 Model B+ boards running at 1.2GHz and 1GB of RAM. We develop Python scripts to facilitate the interaction with the blockchain and the smart gadgets associated with the gateway. The Python script that interacts with the blockchain utilizes the Web3 Python library that facilitates the interaction with the Ethereum blockchain by providing APIs to issue a transaction, reading block data, and interfacing with smart contracts. The PoT gateway plays the role of a transaction issuer in the proposed mechanism, with no block storage requirements or block validation commitments, even if applied to the public Ethereum network. The impact of issuing a transaction to register a new device or reading block data to authenticate a device before granting its data exchange with the PBX server is negligible, and it has a slight impact on the performance evaluation study or the Raspberry Pi when acting as a PoT gateway conducted in Section 6.3.

6.6.1.5 The smart gadgets

Since the heterogeneity of smart objects and their different capabilities considering processing and storage constraints, the proposed mechanism does not impact the resource

limits of embedded devices, promoting their engagement in PoT. In the prototype of the proposed mechanism, we utilize a tiny storage capacity to store the predefined device ID (32 bits), attributes (100 Bytes (JSON)), and the public key of the associated PoT gateway (256 bits). We utilize the SPI flash encryption capability of ESP32 to securely save the device's ID and attributes, mitigating the impersonation threats against the device.

6.6.2 Security Analysis

The proposed mechanism ensures that it meets the CIA triad for tSIP communication: confidentiality, integrity, and availability.

6.6.2.1 Confidentiality

It ensures that messages are securely transmitted from source to destination, preventing unauthorized access attempts to eavesdrop on the transmitted information. In the proposed mechanism, confidentiality is achieved through the utilization of asymmetric cryptography for tSIP registration and authentication messages between the gadget and the PoT gateway on the one hand and the utilization of blockchain technology, which heavily depends on cryptography, to maintain the list of authenticated gadgets at the gateway on the other hand.

6.6.2.2 Integrity

It implies consistency and trustworthiness of information through its life cycle. The proposed system meets integrity by utilizing blockchain technology, which keeps an immutable ledger of transactions that is very hard to alter. Also, integrity is achieved by encrypting the NVS flash partition, which contains identifiable information about the gadget, to mitigate the impersonation threats.

6.6.2.3 Availability

It means the readiness of data for authorized parties when requesting it. Availability is achieved in the proposed mechanism through blockchain technology's inherent decentralization nature, where all transactions happening over the blockchain network are replicated and updated by each participating full node. This means that if one or some of the participating full nodes go offline, the system still functions, ensuring high availability.

6.7 Summary

This chapter presents a comprehensive implementation and evaluation of the proposed Phone of Things (PoT) framework in the thesis. The chapter begins by providing an in-depth overview of the proposed PoT framework's experimental setup, highlighting its software stack's architecture. The chapter then goes through a thorough performance evaluation of SBCs (i.e., Raspberry Pi) as suitable candidates to act as PoT gateways in the proposed framework. The evaluation shows that Raspberry Pi boards can gracefully handle an adequate number of simultaneous VoIP call with different configuration scenarios (i.e., passthrough and transcoding). This promotes the utilization of these tiny and cost-effective boards as Gateways in the proposed framework, optimizing its deployment cost.

To facilitate the implementation of seamless and advanced interaction with the surrounding devices, the chapter reviews the integration of the gateway with Google Assistant. This promotes seamless accessibility to the devices through intuitive voice commands via phone calls and provides innovative telephony solutions as well. The chapter goes through the integration workflow and evaluates the response time of different Raspberry Pi boards while handling user queries and Google Assistant responses. The boards demonstrate good response time for PoT applications.

The chapter also evaluates the proposed tSIP messaging protocol. It starts by review-

ing the serialization and deserialization process of tSIP by abstracting the implementation details of tSIP through pseudo-code examples. It also highlights tSIP packet capture by the testing server for evaluation purposes. The results show that tSIP achieves smaller packet sizes than SIP and COSIP counterparts, implying faster transmit time, prolonged battery life for devices, and efficient network bandwidth utilization. Also, the code footprint of tSIP is small, promoting its deployment to a wide range of embedded devices.

Lastly, the chapter reviews a novel blockchain-based registration and authentication mechanism for devices in PoT. It reviews the implementation of different components of the proposed mechanism and evaluates the performance of different aspects of the proposed mechanism.

Chapter 7

Conclusion and Future Work

IoT (Internet of Things) and VoIP (Voice over Internet Protocol) technologies have yet to be fully integrated, despite the potential benefits of combining these two powerful communication paradigms. While both IoT and VoIP have independently made significant advancements, the convergence of these technologies faces some challenges. IoT devices, often designed with limited resources considering processing, memory, and power consumption, mandating lightweight communication protocols to transmit data efficiently. On the other hand, VoIP demands capable devices featuring real-time and reliable communication, which is hard to achieve on resource-constrained IoT devices. Integrating the two technologies requires addressing issues like security, bandwidth optimization, and interoperability between diverse IoT devices and VoIP platforms.

Therefore, we propose the Phone of Things (PoT) framework in this thesis. The proposed poT holds tremendous promise as a modular platform based on open-source technologies to seamlessly integrate IoT and VoIP technologies, bridging the gap between the two domains and enabling innovative communication solutions. PoT leverages the widespread adoption of VoIP technologies and the advanced capabilities of the communication servers to act as central hubs for IoT devices, facilitating easy control and monitoring through typical phone calls and intuitive voice commands. By integrating

VoIP functionality directly into the proposed PoT framework and providing a constrained version of the SIP protocol, we envision rendering the surrounding devices into typical SIP endpoints in the VoIP ecosystem. Therefore, users can seamlessly interact with the surrounding IoT devices and engage in real-time voice communication with the devices through the ubiquitous phone assets (i.e., phone sets, softphones) connected to the existing communication servers within the premises. Besides, this integration opens up new possibilities for enhanced user experiences of phone calls, such as context-aware telephony solutions and dynamic IVRs. As technology evolves, PoT's potential to serve as a comprehensive platform for IoT and VoIP integration may pave the way for exciting advancements in both domains and further enrich the landscape of connected devices and communication technologies.

7.1 Conclusions

The proposed PoT framework represents an innovative approach to seamlessly integrate IoT and VoIP technologies, taking advantage of a miniaturized SIP protocol version (i.e., tSIP) and blockchain technology for efficient registration and authentication mechanism. Combining the proposed tSIP protocol and blockchain allows for a robust, secure communication infrastructure tailored to resource-constrained IoT devices in typical PoT framework applications. With tSIP, the proposed PoT framework ensures that data exchange between the surrounding devices and the PoT gateways remains efficient even on devices with limited resources. By leveraging blockchain technology, PoT provides a decentralized and tamper-proof mechanism for device registration and authentication in the PoT ecosystem, enhancing the overall security and privacy of the proposed framework. The blockchain-based mechanism simplifies device onboarding and enables seamless communication between devices, eliminating the need for centralized servers and reducing potential points of failure. As a result, the proposed PoT framework offers a scalable,

resilient, and secure platform that fosters seamless integration between IoT devices and VoIP services, opening up new avenues for advanced voice-controlled IoT applications and intelligent telephony solutions.

Chapter 2 provides an in-depth analysis of the current state of research and development in several key areas relevant to the proposed PoT framework. Firstly, it examines the integration of Asterisk in IoT applications, exploring how this open-source IP-PBX communication software can enhance IoT frameworks' connectivity and communication capabilities. Next, the chapter delves into the analysis of standard IoT messaging protocols and their characteristics, assessing their suitability for efficient data exchange in IoT ecosystems. Special attention is given to the SIP protocol and its integration into IoT, investigating the challenges and opportunities it presents for enabling efficient message exchange paradigms in IoT applications. Finally, the chapter explores blockchain technology as a viable secure SIP registration and authentication solution. It reviews recent blockchain-based approaches in the literature for ensuring the integrity and privacy of SIP-based communications. By synthesizing and critically evaluating the existing literature in these areas, the chapter lays the foundation for developing the proposed PoT framework that integrates Asterisk, a constrained version of the SIP protocol, and blockchain technology to enable secure and efficient bridging between IoT and VoIP in the proposed PoT framework.

Chapter 3 outlines the essential building blocks of the proposed PoT platform, focusing on the gateway and potential devices. The PoT gateway is designed as a versatile and powerful component with multiple functionalities to achieve its designated tasks. It serves as an IP-PBX server, enabling seamless VoIP communication between IoT devices and users. Additionally, the gateway functions as an OpenVPN client, ensuring secure and private tunnels between distant gateways. It also acts as a Wi-Fi access point, facilitating easy and convenient connectivity for IoT devices. Moreover, the gateway takes on the role of an MQTT broker, facilitating efficient and lightweight event-driven messaging

for data exchange. In a further enhancement, the chapter discusses the integration of the gateway with a chatbot agent, empowering it with interactive capabilities. The chatbot integration enables the gateway to communicate with users through natural language processing, thereby enabling device monitoring and control through typical phone calls and intuitive voice commands, enhancing the overall user experience. Additionally, the chapter highlights the incorporation of tiny chatbot capability into the gateway, ensuring the gateway's efficient handling of basic voice commands and decoupling its dependence on the Internet. The chapter also emphasizes the gateway's integration with powerline communication technology. The gateway can leverage existing electrical wiring to establish a reliable communication network for legacy and mains-powered devices by incorporating a powerline communication modem. This integration provides a cost-effective solution for extending the connectivity options for the surrounding devices within the premises.

Chapter 4 presents a thorough feasibility study on using single-board computers (SBCs) as suitable candidates for acting as PoT gateways in the proposed PoT framework. The study assesses the performance of various SBCs with different capabilities when handling increasing passthrough and transcoding VoIP calls. The study aims to promote tiny and cost-effective embedded Linux platforms as suitable PoT gateways candidates. It provides insights into the maximum number of simultaneous VoIP calls that different boards can gracefully handle without exhausting their resources. This helps designers to optimize their PoT design decisions by selecting the appropriate boards for PoT gateways that fit their specific application needs. Additionally, the chapter delves into the details of the proposed tiny version of the SIP protocol (i.e., tSIP) explicitly tailored for PoT applications. It provides an in-depth review of the proposed tSIP binary encoding and packet formation, ensuring efficient and lightweight communication for resource-constrained IoT devices. Moreover, the chapter outlines the proposed blockchain-based registration and authentication mechanism for PoT applications. The blockchain solution offers a decentralized and secure approach to device registration and authentication

in PoT applications, enhancing the overall security and privacy of the PoT ecosystem. Combining the performance evaluation of SBCs, the customized SIP protocol, and the blockchain-based security mechanism, the chapter lays the foundation for a robust and practical implementation of the PoT framework, enabling seamless integration of IoT and VoIP technologies with enhanced communication capabilities.

Chapter 5 explores a range of use-case scenarios based on the proposed PoT framework. It reviews various context-aware telephony solutions utilizing the integration between IoT and VoIP technologies in the proposed PoT framework. It also discusses how the PoT framework redefines mature phone technologies in a modern context. The chapter examines how PoT can address context-specific communication needs, enhancing user experiences through intelligent call routing based on contextual information. By leveraging the PoT framework, businesses can implement the proposed Location Transparency Call (LTC) systems that mitigate missed business calls by intelligently redirecting calls to available and relevant extensions, ensuring seamless communication regardless of the user's physical location. Furthermore, the chapter delves into session semantic communication for devices in PoT, highlighting how the framework enables devices to communicate with the gateway in a meaningful and contextually relevant manner. These use case scenarios demonstrate the versatility and applicability of the proposed PoT framework, showcasing its potential to revolutionize traditional phone technologies and unlock innovative communication solutions tailored to the specific needs of users and businesses in today's interconnected world.

Chapter 6 presents a comprehensive implementation and evaluation of the proposed PoT framework. It provides an in-depth overview of the framework's experimental setup and software stack architecture. The chapter evaluates Raspberry Pi's (SBCs) performance as PoT gateways, demonstrating their ability to handle simultaneous VoIP calls with various configuration scenarios. Integrating the gateway with Google Assistant promotes seamless device interaction, providing innovative telephony solutions. The in-

tegration workflow is evaluated, and the average response time of different Raspberry Pi boards is shown to be suitable for PoT applications. The tSIP messaging protocol is evaluated, showing smaller packet sizes than their corresponding SIP and CoSIP counterparts, resulting in faster transmit time, longer battery life, and efficient network bandwidth utilization. Also, the code footprint of tSIP is small, promoting its deployment to a wide range of embedded devices. Finally, the chapter reviews the proposed blockchain-based registration and authentication mechanism for devices in PoT, evaluating its implementation and performance. The analysis demonstrates that the proposed mechanism complies with the security standards of the SIP protocol and fits the constraints of embedded smart objects.

7.2 Possible Future Directions

The future directions for the proposed PoT framework in the thesis hold immense potential for advancing communication technologies and IoT applications. One prominent direction is continuously refining and optimizing the PoT gateway's capabilities to support a broader range of IoT devices and communication protocols. This includes enhancing the gateway's capacity to handle more complex IoT workflows and exploring the integration of building automation system (BAS) protocols [132]. Integrating BAS protocols like BACnet (Building Automation and Control Network) [133], LonWorks [134], KNX (Konex) [135], and DALI (Digital Addressable Lighting Interface) [136] into the PoT framework holds significant potential for creating a comprehensive and interconnected smart environment. By incorporating these protocols, the proposed PoT framework can seamlessly communicate with various building automation devices and systems, enabling coherent and centralized control and monitoring of building infrastructure. BACnet, a widely adopted standard, allows for interoperability among different BAS components, making it possible for the PoT framework to interact with HVAC systems, lighting con-

trols, and other building automation devices. LonWorks enables communication between devices over a Local Operating Network (LON). It is known for its robustness and flexibility in creating networked systems. KNX is a global standard for home and building automation that allows different devices and systems to communicate and interact with each other. DALI is a protocol specifically designed for controlling and dimming building lighting systems. Integrating these protocols, in turn, empowers users to control and manage building automation systems remotely via the phone network (PSTN and VoIP) through voice commands, enabling energy efficiency, cost savings, and improved occupant comfort. As the IoT and building automation domains continue to evolve, this integration can revolutionize how we interact with smart buildings, making them more user-friendly, adaptive, and environmentally sustainable.

Another promising direction for the proposed PoT framework is the exploration of AI and machine learning algorithms to enhance the context-aware telephony capabilities of PoT further. By leveraging AI, the framework can dynamically adapt call routing and device interaction based on user behaviour and preferences, providing a more personalized and seamless communication experience. Additionally, future developments may focus on expanding the PoT ecosystem through collaborations with industry partners and open-source communities, fostering innovation and creating a broader range of compatible devices and applications. Moreover, research efforts may be directed toward optimizing the proposed blockchain-based registration and authentication mechanism to improve scalability and efficiency while ensuring robust security for an expanding network of IoT devices.

Also, utilizing open-source frameworks for building tailor-made Linux-based operating systems, such as Yocto and Buildroot [137], for the gateways in the proposed framework offers numerous potential benefits for PoT deployments. Yocto Linux provides a powerful and flexible build system, enabling the creation of custom Linux distributions with specific configurations and packages tailored to the requirements of the PoT gateway. By

leveraging Yocto's build system capabilities, we can optimize the operating system for the hardware of the PoT gateway, ensuring efficient resource utilization and performance. Additionally, creating a tailor-made operating system allows for the inclusion of only essential components, minimizing the attack surface and enhancing security. Moreover, this approach facilitates easy integration of the PoT gateway into existing IoT infrastructures, ensuring seamless communication with various IoT devices and cloud platforms. This adaptability ensures that the PoT gateway can effectively interact with a wide range of IoT devices, making it highly versatile and future-proof.

Bibliography

[1] Rafiullah Khan, Sarmad Ullah Khan, Rifaqat Zaheer, and Shahid Khan. Future internet: the internet of things architecture, possible applications and key challenges. In *2012 10th international conference on frontiers of information technology*, pages 257–260. IEEE, 2012. ISBN 0769549276.

[2] Khalid Elgazzar, Haytham Khalil, Taghreed Alghamdi, Ahmed Badr, Ghadeer Abdelkader, Abdelrahman Elewah, and Rajkumar Buyya. Revisiting the internet of things: New trends, opportunities and grand challenges. *Frontiers in the Internet of Things*, 1, 2022. ISSN 2813-3110. doi: 10.3389/friot.2022.1073780. URL `https://www.frontiersin.org/articles/10.3389/friot.2022.1073780`.

[3] Qian Zhu, Ruicong Wang, Qi Chen, Yan Liu, and Weijun Qin. Iot gateway: Bridgingwireless sensor networks into internet of things. In *2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, pages 347–352. Ieee, 2010. ISBN 1424497191.

[4] Chrispin Gray, Robert Ayre, Kerry Hinton, and Rodney S Tucker. Power consumption of iot access network technologies. In *2015 IEEE International Conference on Communication Workshop (ICCW)*, pages 2818–2823. IEEE, 2015. ISBN 1467363057.

[5] William Wong. Iot security breaches: 4 real-world exam-

ples, 2021. URL `https://conosco.com/industry-insights/blog/iot-security-breaches-4-real-world-examples`.

[6] Piet De Vaere and Adrian Perrig. Liam: An architectural framework for decentralized iot networks. In *2019 IEEE 16th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pages 416–427. IEEE, 2019. ISBN 1728146011.

[7] Stylianos Karapantazis and Fotini-Niovi Pavlidou. Voip: A comprehensive survey on a promising technology. *Computer Networks*, 53(12):2050–2090, 2009. ISSN 1389-1286.

[8] techienerd. A brief history of voip: How voice over ip changed communication, June 2023. URL `https://voipinsights.com/history-of-voip-how-voice-over-ip-changed-communication/`.

[9] Jason Hoffman. Voip adoption statistics for 2019 & beyond, 2019. URL `https://wisdomplexus.com/blogs/voip-adoption-statistics-2019-beyond`.

[10] The future of voip is in the iot, 2019. URL `https://www.idtexpress.com/blog/the-future-of-voip-is-in-the-iot/`.

[11] Pablo Montoro and Eduardo Casilari. A comparative study of voip standards with asterisk. In *2009 Fourth International Conference on Digital Telecommunications*, pages 1–6. IEEE, 2009. ISBN 0769536956.

[12] Asterisk: Open source communications software, 2023. URL `https://www.asterisk.org/`.

[13] Jim Van Meggelen, Leif Madsen, and Jared Smith. *Asterisk: The future of telephony.* " O'Reilly Media, Inc.", 2007. ISBN 0596510489.

[14] Jonathan Rosenberg, Henning Schulzrinne, Gonzalo Camarillo, Alan Johnston, Jon

Peterson, Robert Sparks, Mark Handley, and Eve Schooler. Sip: session initiation protocol. Report 2070-1721, 2002.

[15] Asterisk — the open-source pbx, 2022. URL `https://www.asterisk.org/`.

[16] Alan B Johnston. *SIP: understanding the session initiation protocol*. Artech House, 2015. ISBN 1608078647.

[17] Simone Cirani, Marco Picone, and Luca Veltri. Cosip: a constrained session initiation protocol for the internet of things. In *European Conference on Service-Oriented and Cloud Computing*, pages 13–24. Springer, 2013.

[18] Muhammad Salek Ali, Massimo Vecchio, Miguel Pincheira, Koustabh Dolui, Fabio Antonelli, and Mubashir Husain Rehmani. Applications of blockchains in the internet of things: A comprehensive survey. *IEEE Communications surveys and tutorials*, 21(2):1676–1717, 2019. ISSN 1553-877X. doi: 10.1109/COMST.2018.2886932.

[19] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review*, page 21260, 2008.

[20] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Weili Chen, Xiangping Chen, Jian Weng, and Muhammad Imran. An overview on smart contracts: Challenges, advances and platforms. *Future Generation Computer Systems*, 105:475–491, 2020. ISSN 0167-739X.

[21] Bengt Ahlgren, Markus Hidell, and Edith C-H Ngai. Internet of things for smart cities: Interoperability and open data. *IEEE Internet Computing*, 20(6):52–56, 2016. ISSN 1089-7801.

[22] Sylvain Kubler, Jérémy Robert, Ahmed Hefnawy, Kary Främling, Chantal Cherifi, and Abdelaziz Bouras. Open iot ecosystem for sporting event management. *IEEE Access*, 5:7064–7079, 2017. ISSN 2169-3536.

[23] Jérémy Robert, Sylvain Kubler, Niklas Kolbe, Alessandro Cerioni, Emmanuel Gastaud, and Kary Främling. Open iot ecosystem for enhanced interoperability in smart cities—example of métropole de lyon. *Sensors*, 17(12):2849, 2017. ISSN 1424-8220.

[24] Salah A Alabady, Fadi Al-Turjman, and Sadia Din. A novel security model for cooperative virtual networks in the iot era. *International Journal of Parallel Programming*, 48:280–295, 2020. ISSN 0885-7458.

[25] Ramon Sanchez-Iborra and Antonio F Skarmeta. Tinyml-enabled frugal smart objects: Challenges and opportunities. *IEEE Circuits and Systems Magazine*, 20 (3):4–18, 2020. ISSN 1531-636X.

[26] Ramachandra Gurunath, Mohit Agarwal, Abhrajeet Nandi, and Debabrata Samanta. An overview: security issue in iot network. In *2018 2nd international conference on I-SMAC (IoT in social, Mobile, analytics and cloud)(I-SMAC) I-SMAC (IoT in social, Mobile, analytics and cloud)(I-SMAC), 2018 2nd international conference on*, pages 104–107. IEEE, 2018. ISBN 1538614421.

[27] Mazen Juma, Azza Abdel Monem, and Khaled Shaalan. Hybrid end-to-end vpn security approach for smart iot objects. *Journal of Network and Computer Applications*, 158:102598, 2020. ISSN 1084-8045.

[28] Cary Stothart, Ainsley Mitchum, and Courtney Yehnert. The attentional cost of receiving a cell phone notification. *Journal of experimental psychology: human perception and performance*, 41(4):893, 2015. ISSN 1939-1277.

[29] Eclipse iot - leading open source community for iot innovation, 2023. URL `https://iot.eclipse.org/`.

[30] Johannes Kristan, Paolo Azzoni, Lukas Römer, Sven Erik Jeroschewski, and Elisa Londero. Evolving the ecosystem: Eclipse arrowhead integrates eclipse iot. In

NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium, pages 1–6. IEEE, 2022. ISBN 1665406011.

[31] Thingsboard - open-source iot platform, 2023. URL `https://thingsboard.io/`.

[32] openhab, 2023. URL `https://www.openhabfoundation.org/`.

[33] Home assistant, 2023. URL `https://www.home-assistant.io/`.

[34] Kaa — enterprise iot platform with free plan, 2023. URL `https://www.kaaiot.com/`.

[35] The things network, 2023. URL `https://www.thethingsnetwork.org/`.

[36] Node-red — low-code programming for event-driven applications, 2023. URL `https://nodered.org/`.

[37] JA Herndon and FH Tendick. A time division switch for an electronic private branch exchange. *IEEE Transactions on Communication and Electronics*, 83(73): 338–345, 1964. ISSN 0536-1532.

[38] Bilal Muhammad Khan, Muhammad Fahad, Rabia Bilal, and Ali Hanzala Khan. Performance analysis of raspberry pi 3 ip pbx based on asterisk. *Electronics*, 11 (20):3313, 2022. ISSN 2079-9292.

[39] Sandeep Sonaskar and Shubhangi Giripunje. Voice over intranet based private branch exchange system design. In *2011 3rd International Conference on Electronics Computer Technology*, volume 6, pages 287–291. IEEE, 2011. ISBN 1424486793.

[40] Mark Spencer. Introduction to the asterisk open source pbx. *Linux Support Services, Inc*, 2002.

[41] Gary C Kessler and Peter V Southwick. *ISDN concepts, facilities, and services*. McGraw-Hill, Inc., 1996. ISBN 0070342490.

[42] Ghannam Aljabari. Integrating voip systems with the internet of things. The 4th Palestinian International Conference on Computer and Information, 2015.

[43] Foteini Andriopoulou, Theofanis Orphanoudakis, and Tasos Dagiuklas. Iota: Iot automated sip-based emergency call triggering system for general ehealth purposes. In *2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 362–369. IEEE, 2017. ISBN 1538638398.

[44] Danilo De Freitas Melo, Epaminondas De Souza Lage, Anderson Vagner Rocha, and Braz De Jesus Cardoso. Improving the consumption and water heating efficiency in smart buildings. In *2017 13th International conference and expo on emerging technologies for a smarter world (CEWIT)*, pages 1–6. IEEE, 2017. ISBN 1538622750.

[45] Juan Pablo Berrío López and Yury Montoya Pérez. Integration of asterisk ip-pbx with esp32 embedded system for remote code execution. *Multidisciplinary Digital Publishing Institute Proceedings*, 21(1):38, 2019. ISSN 2504-3900.

[46] Leonel Hernandez and Maria Ospina. Scheme and creation of a prototype for the supervision of lights and electronic devices with a pbx, using a wlan solution based on iot. In *2019 IEEE Colombian Conference on Communications and Computing (COLCOM)*, pages 1–6. IEEE, 2019. ISBN 1728135036.

[47] Jirawat Sangkong and Machigar Ongtang. Smart voip postbox with confirmation receipt using iot technology. In *Proceedings of the 2017 International Conference on Industrial Design Engineering*, pages 71–75, 2017.

[48] Karen Rose, Scott Eldridge, and Lyman Chapin. The internet of things: An overview. *The internet society (ISOC)*, 80:1–50, 2015.

[49] KV Shibu. *Introduction to embedded systems*. Tata McGraw-Hill Education, 2009. ISBN 007014589X.

[50] Hao Chen, Xueqin Jia, and Heng Li. A brief introduction to iot gateway. In *IET international conference on communication technology and application (ICCTA 2011)*, pages 610–613. IET, 2011. ISBN 184919470X.

[51] Charles Severance. Eben upton: Raspberry pi. *Computer*, 46(10):14–16, 2013. ISSN 0018-9162.

[52] Gerald Coley. Beaglebone black system reference manual. *Texas Instruments, Dallas*, 5:2013, 2013.

[53] Neven Nikolov, Ognyan Nakov, and Daniela Gotseva. Operating systems for iot devices. In *2021 56th International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST)*, pages 41–44. IEEE, 2021. ISBN 1665428872.

[54] Jonathan A Ariza and Heyson Baez. Understanding the role of single-board computers in engineering and computer science education: A systematic literature review. *Computer Applications in Engineering Education*, 30(1):304–329, 2022. ISSN 1061-3773.

[55] Jolle W Jolles. Broad-scale applications of the raspberry pi: A review and guide for biologists. *Methods in Ecology and Evolution*, 12(9):1562–1579, 2021. ISSN 2041-210X.

[56] William Shotts. *The Linux command line: a complete introduction*. No Starch Press, 2019. ISBN 1593279531.

[57] Martin Becker and Samarjit Chakraborty. A valgrind tool to compute the working set of a software process. *arXiv preprint arXiv:1902.11028*, 2019.

[58] A. Goucher and T. Riley. *Beautiful Testing: Leading Professionals Reveal How*

They Improve Software. O'Reilly Media, 2009. ISBN 9781449388683. URL `https://books.google.ca/books?id=qAv8p4piP2cC`.

[59] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE communications surveys & tutorials*, 17(4):2347–2376, 2015. ISSN 1553-877X.

[60] Eyhab Al-Masri, Karan Raj Kalyanam, John Batts, Jonathan Kim, Sharanjit Singh, Tammy Vo, and Charlotte Yan. Investigating messaging protocols for the internet of things (iot). *IEEE Access*, 8:94880–94911, 2020. ISSN 2169-3536.

[61] Nitin Naik. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. In *2017 IEEE international systems engineering symposium (ISSE)*, pages 1–7. IEEE, 2017. ISBN 1538634031.

[62] Selma Dilek, Kerem Irgan, Metehan Guzel, Suat Ozdemir, Sebnem Baydere, and Chalermpol Charnsripinyo. Qos-aware iot networks and protocols: A comprehensive survey. *International Journal of Communication Systems*, 35(10):e5156, 2022. ISSN 1074-5351.

[63] Mehar Ullah, Syed Rameez Ullah Kakakhel, Tomi Westerlund, Annika Wolff, Dick Carrillo, Juha Plosila, and Pedro HJ Nardelli. Iot protocol selection for smart grid applications: Merging qualitative and quantitative metrics. In *2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO)*, pages 993–998. IEEE, 2020. ISBN 9532330992.

[64] Dipa Soni and Ashwin Makwana. A survey on mqtt: a protocol of internet of things (iot). In *International conference on telecommunication, power analysis and computing techniques (ICTPACT-2017)*, volume 20, pages 173–177, 2017.

[65] Carsten Bormann, Angelo P Castellani, and Zach Shelby. Coap: An application protocol for billions of tiny internet nodes. *IEEE Internet Computing*, 16(2):62–67, 2012. ISSN 1089-7801.

[66] Parul Datta and Bhisham Sharma. A survey on iot architectures, protocols, security and smart city based applications. In *2017 8th International conference on computing, communication and networking technologies (ICCCNT)*, pages 1–5. IEEE, 2017. ISBN 1509030387.

[67] Jorge E Luzuriaga, Juan Carlos Cano, Carlos Calafate, Pietro Manzoni, Miguel Perez, and Pablo Boronat. Handling mobility in iot applications using the mqtt protocol. In *2015 Internet Technologies and Applications (ITA)*, pages 245–250. IEEE, 2015. ISBN 1467395579.

[68] Jevgenijus Toldinas, Borisas Lozinskis, Edgaras Baranauskas, and Algirdas Dobrovolskis. Mqtt quality of service versus energy consumption. In *2019 23rd International Conference Electronics*, pages 1–4. IEEE, 2019. ISBN 1728122090.

[69] Cenk Gündoğan, Christian Amsüss, Thomas C Schmidt, and Matthias Wählisch. Toward a restful information-centric web of things: A deeper look at data orientation in coap. In *Proceedings of the 7th ACM Conference on Information-Centric Networking*, pages 77–88, 2020.

[70] Tetsuya Yokotani and Yuya Sasaki. Comparison with http and mqtt on required network resources for iot. In *2016 international conference on control, electronics, renewable energy and communications (ICCEREC)*, pages 1–6. IEEE, 2016. ISBN 150900744X.

[71] Nurzhan Nurseitov, Michael Paulson, Randall Reynolds, and Clemente Izurieta. Comparison of json and xml data interchange formats: a case study. *Caine*, 9: 157–162, 2009.

[72] I-Fen Yang, Yi-Chun Lin, Shun-Ren Yang, and Phone Lin. The implementation of a sip-based service platform for 5g iot applications. In *2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)*, pages 1–6. IEEE, 2021. ISBN 1728189640.

[73] Rosario G Garroppo, Loris Gazzarrini, Stefano Giordano, Michele Pagano, and Luca Tavanti. A sip-based home gateway for domotics systems: From the architecture to the prototype. In *International Conference on Computer Networks*, pages 344–359. Springer, 2016.

[74] Foteini Andriopoulou, Anastatios Fanariotis, and Theofanis Orphanoudakis. Seek: Sip-based emergency embedded framework supports elderly and disabled to perform emergency calls. In *2018 21st Euromicro Conference on Digital System Design (DSD)*, pages 442–449. IEEE, 2018. ISBN 1538673770.

[75] Mike Botts and Alexandre Robin. Opengis® sensor model language (sensorml) implementation specification. version 1.0. 0. 2007.

[76] Abdulkadir Karaagac, Pieterjan Camerlynck, Pieter Crombez, and Jeroen Hoebeke. Viot: Voice over internet of things. In *2020 Global Internet of Things Summit (GIoTS)*, pages 1–6. IEEE, 2020. ISBN 1728167280.

[77] Dylan Yaga, Peter Mell, Nik Roby, and Karen Scarfone. Blockchain technology overview. *arXiv preprint arXiv:1906.11078*, 2019.

[78] Joe Abou Jaoude and Raafat George Saade. Blockchain applications–usage in different domains. *Ieee Access*, 7:45360–45381, 2019. ISSN 2169-3536.

[79] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. Blockchain challenges and opportunities: A survey. *International journal of web and grid services*, 14(4):352–375, 2018. ISSN 1741-1106.

[80] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[81] Dominique Guegan. Public blockchain versus private blockhain. 2017.

[82] Du Mingxiao, Ma Xiaofeng, Zhang Zhe, Wang Xiangwei, and Chen Qijun. A review on consensus algorithm of blockchain. In *2017 IEEE international conference on systems, man, and cybernetics (SMC)*, pages 2567–2572. IEEE, 2017. ISBN 1538616459.

[83] Omar Dib, Kei-Leo Brousmiche, Antoine Durand, Eric Thea, and Elyes Ben Hamida. Consortium blockchains: Overview, applications and challenges. *Int. J. Adv. Telecommun*, 11(1):51–64, 2018.

[84] Fan Yang, Wei Zhou, QingQing Wu, Rui Long, Neal N Xiong, and Meiqi Zhou. Delegated proof of stake with downgrade: A secure and efficient blockchain consensus algorithm with downgrade mechanism. *IEEE Access*, 7:118541–118555, 2019. ISSN 2169-3536.

[85] Oluwakayode Onireti, Lei Zhang, and Muhammad Ali Imran. On the viable area of wireless practical byzantine fault tolerance (pbft) blockchain networks. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2019. ISBN 1728109620.

[86] Dongyan Huang, Xiaoli Ma, and Shengli Zhang. Performance analysis of the raft consensus algorithm for private blockchains. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 50(1):172–181, 2019. ISSN 2168-2216.

[87] Shuai Wang, Yong Yuan, Xiao Wang, Juanjuan Li, Rui Qin, and Fei-Yue Wang. An overview of smart contract: architecture, applications, and future trends. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 108–113. IEEE, 2018. ISBN 1538644525.

[88] Chris Dannen. *Introducing Ethereum and solidity*, volume 1. Springer, 2017.

[89] M. Abubakar, Z. Jaroucheh, A. Al Dubai, and B. Buchanan. Blockchain-based authentication and registration mechanism for sip-based voip systems. In *2021 5th Cyber Security in Networking Conference (CSNet)*, pages 63–70, 2021. ISBN 2768-0029. doi: 10.1109/CSNet52717.2021.9614646.

[90] Andi Xu, Mi Li, Xin Huang, Nian Xue, Jie Zhang, and Qiankun Sheng. A blockchain based micro payment system for smart devices. *Signature*, 256(4936): 115, 2016.

[91] Yifeng Tian, Zheng Lu, Peter Adriaens, R Edward Minchin, Alastair Caithness, and Junghoon Woo. Finance infrastructure through blockchain-based tokenization. *Frontiers of Engineering Management*, 7:485–499, 2020. ISSN 2095-7513.

[92] Osama Younes and Umar Albalawi. Securing session initiation protocol. *Sensors*, 22(23):9103, 2022. ISSN 1424-8220.

[93] Dimitris Geneiatakis, Tasos Dagiuklas, Georgios Kambourakis, Costas Lambrinoudakis, Stefanos Gritzalis, Karlovassi Sven Ehlert, and Dorgham Sisalem. Survey of security vulnerabilities in session initiation protocol. *IEEE Communications Surveys & Tutorials*, 8(3):68–81, 2006. ISSN 1553-877X.

[94] Mahdi Nikooghadam and Haleh Amintoosi. A secure and robust elliptic curve cryptography-based mutual authentication scheme for session initiation protocol. *Security and privacy*, 3(1), 2020. ISSN 2475-6725. doi: 10.1002/spy2.92.

[95] Mahdi Nikooghadam and Haleh Amintoosi. Perfect forward secrecy via an ecc-based authentication scheme for sip in voip. *The Journal of supercomputing*, 76(4): 3086–3104, 2020. ISSN 0920-8542. doi: 10.1007/s11227-019-03086-z.

[96] E. F. Kfoury and D. J. Khoury. Secure end-to-end volte based on ethereum blockchain. In *2018 41st International Conference on Telecommunications and Signal Processing (TSP)*, pages 1–5, 2018. doi: 10.1109/TSP.2018.8441204.

[97] Elie F. Kfoury and David J. Khoury. Secure end-to-end voip system based on ethereum blockchain. *J. Commun.*, 13:450–455, 2018.

[98] A. Febro, H. Xiao, and J. Spring. Sipchain: Sip defense cluster with blockchain. In *2019 Principles, Systems and Applications of IP Telecommunications (IPTComm)*, pages 1–8, 2019. doi: 10.1109/IPTCOMM.2019.8920874.

[99] Mustafa Kara, Hisham RJ Merzeh, Muhammed Ali Aydın, and Hasan Hüseyin Balık. Voipchain: A decentralized identity authentication in voice over ip using blockchain. *Computer Communications*, 198:247–261, 2023. ISSN 0140-3664.

[100] Maria Rita Palattella, Nicola Accettura, Xavier Vilajosana, Thomas Watteyne, Luigi Alfredo Grieco, Gennaro Boggia, and Mischa Dohler. Standardized protocol stack for the internet of (important) things. *IEEE communications surveys & tutorials*, 15(3):1389–1406, 2012. ISSN 1553-877X.

[101] John Sonnenberg. Serial communications rs232, rs485, rs422. *Technical brief, Raveon Technologies Corp*, 2018.

[102] Raspberry Pi. Raspberry pi 3 model b. *online].(https://www. raspberrypi. org*, 2015.

[103] Gerry Howser and Gerry Howser. Raspberry pi operating system. *Computer Networks and the Internet: A Hands-On Approach*, pages 119–149, 2020. ISSN 3030344959.

[104] Janne Magnusson. Sip trunking benefits and best practices. *Ingate Systems whitepaper*, 2006.

[105] Muhammad Iqbal and Imam Riadi. Analysis of security virtual private network (vpn) using openvpn. *International Journal of Cyber-Security and Digital Forensics*, 8(1):58–65, 2019.

[106] Josip Balen, Denis Vajak, and Khaled Salah. Comparative performance evaluation of popular virtual private servers. *Journal of Internet Technology*, 21(2):343–356, 2020. ISSN 2079-4029.

[107] Myung-Ki Shin, Ki-Hyuk Nam, and Hyoung-Jun Kim. Software-defined networking (sdn): A reference architecture and open apis. In *2012 International Conference on ICT Convergence (ICTC)*, pages 360–361. IEEE, 2012. ISBN 1467348287.

[108] Wu-Jeng Li, Chiaming Yen, You-Sheng Lin, Shu-Chu Tung, and ShihMiao Huang. Justiot internet of things based on the firebase real-time database. In *2018 IEEE International Conference on Smart Manufacturing, Industrial & Logistics Engineering (SMILE)*, pages 43–47. IEEE, 2018. ISBN 1538631830.

[109] Daniel Kant, Andreas Johannsen, and Reiner Creutzburg. Analysis of iot security risks based on the exposure of the mqtt protocol. *Electronic Imaging*, 2021(3): 96–1–96–8, 2021. ISSN 2470-1173.

[110] Navin Sabharwal, Amit Agrawal, Navin Sabharwal, and Amit Agrawal. Introduction to google dialogflow. *Cognitive virtual assistants using google dialogflow: develop complex cognitive bots using the google dialogflow platform*, pages 13–54, 2020. ISSN 1484257405.

[111] Nir Simionovich. *Asterisk gateway interface 1.4 and 1.6 programming*. Packt Publishing Ltd, 2009. ISBN 1847194478.

[112] Niovi Pavlidou, AJ Han Vinck, Javad Yazdani, and Bahram Honary. Power line communications: state of the art and future trends. *IEEE Communications magazine*, 41(4):34–40, 2003. ISSN 0163-6804.

[113] Ag2130 - silvertel — power over ethernet modules — telecom modules, 2023. URL `https://silvertel.com/ag2130/`.

[114] Shaik Javeed Hussain, Samiullah Khan, Raza Hasan, and Shaik Asif Hussain. Design and implementation of animal activity monitoring system using ti sensor tag. In *Cognitive Informatics and Soft Computing: Proceeding of CISC 2019*, pages 167–175. Springer, 2019. ISBN 981151450X.

[115] Gabriel Gaspar, Peter Fabo, Michal Kuba, Juraj Dudak, and Eduard Nemlaha. Micropython as a development platform for iot applications. In *Intelligent Algorithms in Software Engineering: Proceedings of the 9th Computer Science On-line Conference 2020, Volume 1 9*, pages 388–394. Springer, 2020. ISBN 3030519643.

[116] Kyungtae Kim and Young-June Choi. Performance comparison of various voip codecs in wireless environments. In *Proceedings of the 5th International Conference on Ubiquitous Information Management and Communication*, pages 1–10, 2011.

[117] Bur Goode. Voice over internet protocol (voip). *Proceedings of the IEEE*, 90(9): 1495–1517, 2002. ISSN 0018-9219.

[118] Alan Clark. Common voip metrics. In *Workshop on End-to-End Quality of Service. What is it? How do we get it*, volume 1, page 3, 2003.

[119] Tim Szigeti and Christina Hattingh. Quality of service design overview. *Cisco, San Jose, CA, Dec*, pages 1–34, 2004.

[120] Amandeep Kaur, Srinidhi Ayyagari, Manasi Mishra, and Rachit Thukral. A literature review on device-to-device data exchange formats for iot applications. *JOURNAL OF INTELLIGENT SYSTEMS AND COMPUTING*, 1(1):1–10, 2020. ISSN 2976-8098.

[121] Srdan Popic, Drazen Pezer, Bojan Mrazovac, and Nikola Teslic. Performance evaluation of using protocol buffers in the internet of things communication. In *2016 International Conference on Smart Systems and Technologies (SST)*, pages 261–265. IEEE, 2016. ISBN 1509037209.

[122] Kazuaki Maeda. Performance evaluation of object serialization libraries in xml, json and binary formats. In *2012 Second International Conference on Digital Information and Communication Technology and it's Applications (DICTAP)*, pages 177–182. IEEE, 2012. ISBN 1467307343.

[123] Amrit Kumar Biswal and OBADA AL MALLAH. Analytical assessment of binary data serialization techniques in iot context (evaluating protocol buffers, flat buffers, message pack, and bson for sensor nodes). 2019.

[124] Steve Whittaker, Julia Hirschberg, Brian Amento, Litza Stark, Michiel Bacchiani, Philip Isenhour, Larry Stead, Gary Zamchick, and Aaron Rosenberg. Scanmail: a voicemail interface that makes speech browsable, readable and searchable. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 275–282, 2002.

[125] Sales. Cold call sales voicemail scripts that get callbacks, 2020. URL `https://pipeline.zoominfo.com/sales/cold-sales-voicemails`.

[126] James P Dupuis and Michael C Stinson. Sms notifications for missed calls expanding mobility for tdm environments. In *Proceedings of the 3rd annual conference on Research in information technology*, pages 71–74, 2014.

[127] Irham Nurhalim and David Gunawan. Pstn voip application support system design using mobile short message service (sms): Case study of pstn voip missed call notification to mobile phone by sms. In *Proceedings of the 2011 International Con-*

ference on Electrical Engineering and Informatics, pages 1–4. IEEE, 2011. ISBN 1457707527.

[128] Michael N Aberethy Jr, Travis M Grigsby, Michael A Paolini, and Lakshmi Potluri. Missed call integration with voicemail and granular access to voicemail, 2013.

[129] Understanding the impact of voicemail, 2015. URL `https://www.answer365.ca/blog/102-understanding-the-impact-of-voicemail.html`.

[130] Ron Weinstein. Rfid: a technical overview and its application to the enterprise. *IT professional*, 7(3):27–33, 2005. ISSN 1520-9202.

[131] Kyungki Kim, Sining Li, Milad Heydariaan, Nour Smaoui, Omprakash Gnawali, Wonho Suh, Min Jae Suh, and Jung In Kim. Feasibility of lora for smart home indoor localization. *Applied Sciences*, 11(1):415, 2021. ISSN 2076-3417.

[132] Pedro Domingues, Paulo Carreira, Renato Vieira, and Wolfgang Kastner. Building automation systems: Concepts and technology review. *Computer Standards & Interfaces*, 45:1–12, 2016. ISSN 0920-5489.

[133] Larry K Haakenstad. The open protocol standard for computerized building systems: Bacnet. In *Proceedings of the 1999 IEEE International Conference on Control Applications (Cat. No. 99CH36328)*, volume 2, pages 1585–1590. IEEE, 1999. ISBN 078035446X.

[134] Byoung-Hee Kim, Kwang-Hyun Cho, and Kyoung-Sup Park. Towards lonworks technology and its applications to automation. In *Proceedings KORUS 2000. The 4th Korea-Russia International Symposium On Science and Technology*, volume 2, pages 197–202. IEEE, 2000. ISBN 0780364864.

[135] Michele Ruta, Floriano Scioscia, Giuseppe Loseto, and Eugenio Di Sciascio. Knx: A worldwide standard protocol for home and building automation: State of the

art and perspectives. *Industrial Communication Technology Handbook*, pages 58–1–58–19, 2017.

[136] PF Hein. Dali-a digital addressable lighting interface for lighting electronics. In *Conference Record of the 2001 IEEE Industry Applications Conference. 36th IAS Annual Meeting (Cat. No. 01CH37248)*, volume 2, pages 901–905. IEEE, 2001. ISBN 0780371143.

[137] Tuomo Perä. Comparison of custom embedded linux build systems: Yocto and buildroot. 2022.