University of Zurich[UZH]

BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

# HomeScout Extension: Investigation of Lightbulb-based User-Profiling and Privacy-Preservation

*Delia Datsomor*
*Zurich, Switzerland*
*Student ID: 19-746-551*

Supervisor: Katharina O.E. Müller, Dr. Bruno Rodrigues
Date of Submission: May 08, 2024

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

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

# Declaration of Independence

I hereby declare that I have composed this work independently and without the use of any aids other than those declared (including generative AI such as ChatGPT). I am aware that I take full responsibility for the scientific character of the submitted text myself, even if AI aids were used and declared (after written confirmation by the supervising professor). All passages taken verbatim or in sense from published or unpublished writings are identified as such. The work has not yet been submitted in the same or similar form or in excerpts as part of another examination.

Zürich, 08. 05. 2024

Signature of student

# Abstract

The number of smart homes, where devices equipped with Bluetooth Low Energy (BLE) and Zigbee like Philips Hue are increasingly being used, is rising. These technologies allow for advanced tracking capabilities, such as BLE devices with AirTags. People have demonstrated Zigbee's promising potential for user tracking in smart homes, but little attention has been given to exploring the extent of information extractable solely from network packets, particularly focusing on Philips Hue packets. Especially the inquiry into whether possessing a single Philips Hue network key enables the decryption of other Philips Hue networks remains largely unexplored. This work presents a real-time scanning and sniffing application designed to identify devices, classify them, and monitor events within a Philips Hue network, facilitating an examination of vulnerabilities in Philips Hue smart homes. This is achieved by conducting an exhaustive analysis of Zigbee communication within several Philips Hue networks and entailing an iterative process of collecting and evaluating Zigbee packets, complemented by the design and implementation of a prototype using inference rules. Results showcase the ability to successfully track different Philips Hue networks using the sniffing application and achieving an accuracy rate of approximately 94% in identifying devices and events within the network presenting certain privacy issues like user profiling concerns.

# Acknowledgments

Grateful for the opportunity to conduct my bachelor's thesis, I wish to express my appreciation to the Communication Systems Research Group (CSG) of the University of Zurich's Department of Informatics. I extend my sincere thanks to my supervisor, Katharina Müller, for her continuous support, granting me the freedom to explore my interests, and providing precise advice, constructive feedback, and generous time. Additionally, I want to express my gratitude to my family for their relentless support and patience throughout this journey, particularly my father, Lawrence Datsomor, whose assistance with the setup and the review of my thesis was invaluable. Lastly, I thank Dimitri Missoh for taking the time to review my work and providing me with valuable suggestions for improvement.

# Contents

**Declaration of Independence** i

**Abstract** iii

**Acknowledgments** v

**1 Introduction** 1

1.1 Motivation . . . 2

1.2 Thesis Goals . . . 2

1.3 Thesis Outline . . . 3

**2 Background** 5

2.1 Zigbee . . . 5

2.2 Philips Hue . . . 11

2.3 Home Assistant . . . 12

**3 Related Work** 15

**4 Design** 17

4.1 Setup . . . 17

4.1.1 Hardware . . . 18

4.1.2 Software . . . 20

4.1.3 Assumption . . . 20

4.1.4 Sniffing operations . . . 20

4.2 Data Collection . . . 21
4.3 Data evaluation . . . 21
4.3.1 Prototype design . . . 22
4.3.2 Home Assistant Integration . . . 23

**5 Implementation** **25**
5.1 Command Extractor . . . 25
5.2 Packet Extractor . . . 26
5.3 Filter . . . 27
5.4 Identifier . . . 28
5.5 Analyzer . . . 29
5.6 Tracker . . . 30
5.7 Runner . . . 31
5.8 Flowcharts . . . 31
5.9 Implementation Discussion . . . 34
5.9.1 Command Extractor . . . 34
5.9.2 Packet Extractor . . . 35
5.9.3 Filter . . . 36
5.9.4 Identifier . . . 37
5.9.5 Analyzer . . . 38
5.9.6 Tracker . . . 43
5.9.7 Runner . . . 43

**6 Results and Evaluation** **45**
6.1 Testing and Outcomes . . . 45
6.1.1 Setup . . . 45
6.1.2 Device Identification Results . . . 46
6.1.3 Event Detection Results . . . 47
6.2 Prototype Evaluation . . . 48

6.2.1 Evaluation Metrics . . . 48
6.2.2 Evaluation Discussion . . . 49
6.2.3 Challenges . . . 49
6.2.4 Emerging Security and Privacy Concerns . . . 50
6.2.5 Reflective Synthesis . . . 52

**7 Conclusions and Future Work** **55**
7.1 Conclusions . . . 55
7.2 Future Work . . . 56

**Abbreviations** **63**

**List of Figures** **64**

**List of Tables** **66**

**List of Listings** **67**

**A Contents of the Repository** **71**
A.1 **README** . . . 71
A.2 **Python Scripts** . . . 73
A.3 JSON Files . . . 73
A.4 Testing . . . 73
A.4.1 Raw Live . . . 73
A.4.2 Raw Passive . . . 73
A.4.3 Summary Tables of Devices and Events . . . 74

# Chapter 1

# Introduction

In recent years, the smart home technology landscape has transformed, driven by the emergence of various wireless communication protocols [1]. Among these, BLE has played a crucial role, in facilitating seamless connectivity and enhancing the functionality of smart devices. However, as the prevalence of BLE-enabled devices continues to rise, so do concerns regarding privacy and security [2].

BLE technology has gained popularity due to its ability to efficiently track personal items, as exemplified by Apple AirTags [3]. This technology offers users a convenient solution for keeping track of items such as keys, wallets, and bags. However, the same tracking capabilities that enhance convenience also raise questions about the potential for misuse, particularly in terms of tracking individuals without their consent [4].

Furthermore, the integration of BLE into various smart home devices has expanded the scope of tracking possibilities. Many smart home appliances and gadgets use BLE for communication and control, contributing to a more interconnected living environment. This integration, while offering numerous benefits in terms of convenience and efficiency, also introduces concerns about privacy and security. The increasing popularity of smart homes [5] also raises concerns about potential invasions of privacy through tracking of user activities.

In parallel, another wireless communication protocol, Zigbee, has emerged as a prominent player in the field of smart home technology. Devices such as Philips Hue smart lighting systems rely on Zigbee for communication between the bridge and individual light devices, offering users greater control and flexibility in lighting solutions [6].

Similar to BLE, Zigbee presents opportunities for tracking within smart home environments, raising additional privacy and security considerations.

As the adoption of smart home technology continues to increase, fueled by the increasing popularity of devices like Philips Hue devices, the potential for tracking extends beyond individual belongings to encompass broader aspects of daily life. The ability to track smart home activities, including lighting schedules, appliance usage, and occupancy patterns, introduces new challenges in terms of privacy protection and security.

As consumers embrace the convenience and connectivity offered by these technologies, it's important to acknowledge the non-technological dimensions, particularly concerning user profiling, which if misused, can potentially contribute to stalking or other forms of harassment.

## 1.1 Motivation

This thesis is driven by a desire to comprehend the nuances of Zigbee communication in a Philips Hue smart home setting [7] unlocking the potential to extract meaningful content from encrypted packets. By scrutinizing the communication protocols within a real Philips Hue smart home, there's an opportunity to gain insights that go beyond technical understanding.

The development of a real-time scanning and sniffing application represents a proactive step towards exploring the finer details of Zigbee communication. This prototype not only aims to decode encrypted packets but also aims to reveal patterns and behaviors encoded within them. Such insights could potentially shed light on user habits, offering a glimpse into the intimate dynamics of smart home interactions.

Furthermore, the integration of Home Assistant into the evaluation process underscores the pursuit of efficiency in Zigbee network scanning.

Ultimately, the exploration of Zigbee communication vulnerabilities within Philips Hue smart homes adds a critical layer of understanding to the larger conversation on smart home security. It raises awareness about the tracking potential inherent within smart home technologies, facilitating necessary discussions about privacy and data protection. But unlike traditional security assessments, the core objective is the seamless integration of scanning and sniffing capabilities into a singular application while also uncovering non-technical privacy concerns such as user profiling. Outlined below are the research questions to be addressed in this thesis.

1. How effectively can a sniffer application capture and analyze Zigbee communication packets in a smart home, especially those related to smart light bulbs?
2. What insights into the packet content can be gained, and how can this information contribute to the tracking of user habits and preferences?
3. What potential security vulnerabilities in Zigbee communication can be identified through packet sniffing?

## 1.2 Thesis Goals

This thesis aims to comprehensively investigate Zigbee communication within a Philips Hue smart home environment. The primary objectives include the analysis of a real

Philips Hue smart home to then transition into the development of a real-time application with scanning and sniffing capabilities, with a focus on smart light bulbs. Additionally, the thesis assesses how Home Assistant might be integrated for Zigbee network scanning efficiency. An examination of Zigbee communication vulnerabilities in smart homes is included as well, which adds to the knowledge of potential security issues. Thus, the main contributions of this work are:

- A comprehensive analysis of Zigbee communication in a Philips Hue smart home setting.
- Developing a real-time scanning and sniffing application.
- Evaluation of Home Assistant's integration for effective scanning of Zigbee networks.
- Evaluation of the real-time scanning and sniffing application leading to an examination of vulnerabilities in Phillips Hue smart homes.

## 1.3 Thesis Outline

Chapter 2 provides a comprehensive background on Zigbee technology, the Philips Hue lighting system, and Home Assistant platforms. This chapter explores the fundamental principles, setting the stage for the subsequent chapters.

Chapter 3 delves into the related work in the field, examining existing research, projects, and literature relevant to the integration of Zigbee, Philips Hue, and Home Assistant. This chapter analyzes previous studies and highlights key insights.

Chapter 4 focuses on the design aspect, detailing the setup procedures, data collection methodologies, and data evaluation techniques employed in the thesis.

Chapter 5 presents the implementation phase of the prototype. This chapter offers a comprehensive description of the implementation process and delves into an implementation discussion.

Chapter 6 evaluates the implemented prototype, analyzing its performance, functionality, and usability. This chapter encompasses testing procedures, and outcome assessment. Furthermore, it presents challenges and emerging security and privacy concerns and presents a reflective framework.

Finally, Chapter 7 initiates a discussion on the findings of the research and outlines potential avenues for future work. This chapter presents the concluding considerations, outlines the limitations of the study, and suggests recommendations for future research and development in the field.

# Chapter 2

# Background

## 2.1 Zigbee

Zigbee is a wireless communication protocol based on the IEEE 802.15.4 standard that was designed for Wireless Personal Area Networks (WPANs) and used in low-powered embedded devices, specifically radio systems. WPANs, which are essential to the landscape of short-range networking, connect personal devices in close proximity from 10 to 100m. Hence, its primary purpose is to enable efficient machine-to-machine communication over short distances. Zigbee notably focuses on low-power, low-data-rate transmission, positioning it as a key technology in the Internet of Things (IoT) [8].

This section's technical descriptions of Zigbee are derived from the Payatu IoT Security Handbook[9], Rudresh's guide on Zigbee basics [8] and [10]. Additional resources are cited accordingly.

### Protocol Stack

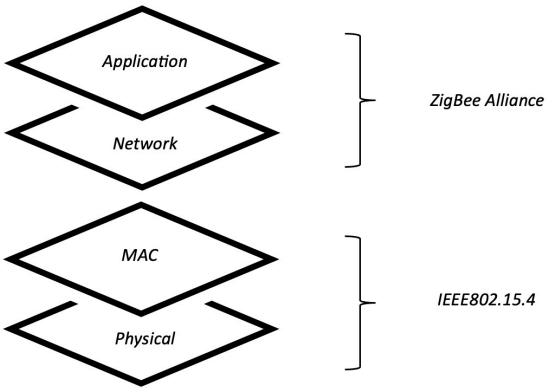


Figure 2.1: Zigbee Protocol Stack

From the bottom up, the Zigbee layer stack is composed of the physical (PHY) layer, the media access control (MAC) layer, the network (NWK) layer, and the application layer. In total, there are four layers. The PHY and MAC layers are defined in the IEEE 802.15.4 standard. The NWK and application layer are defined by the Zigbee alliance. Each layer serves as a data and management layer for the layer above it. Figure 2.1 depicts the layered architecture of the Zigbee network protocol [11].

**Physical Layer** The Zigbee PHY layer serves by providing both PHY data services and management services, while also establishing the interface between the wireless channel and the MAC sublayer. It functions across distinct frequency ranges, employing 16 channels in the high-frequency 2.4 GHz spectrum worldwide. In North America and Australia, it utilizes 30 channels in the 915 MHz band, while in Europe, a single channel is allocated in the 868 MHz band. Additionally, certain Zigbee devices operate on the 784 MHz band in China. The main tasks encompass the control of radio operations, including the enabling and disabling of the radio, the delivery of essential Link Quality Indication (LQI) for received packets, and providing valuable insights into signal quality. Furthermore, Energy Detection (ED) is a central responsibility, involving the evaluation of channel energy to determine signal presence and Clear Channel Assessment (CCA) is performed to verify channel clarity before initiating data transmission.

**Media Access Control Layer** All wireless physical channel access is managed and processed by the MAC layer, which also offers MAC layer data service and administration. Beacon generation and synchronization, support for the establishment and separation of PAN links, and carrier sense multiple access with collision avoidance (CSMA-CA) mechanism implementation are among the main functions of the MAC layer. Additionally, it is responsible for processing and maintaining the guaranteed time slot (GTS) mechanism and ensuring a reliable communication link. Later on, more detail will be provided concerning packets and their structure in the MAC Layer.

**Network Layer** As the protocol stack's basic layer, the NWK layer is largely in charge of network formation and path selection. The Zigbee coordinator's network layer is responsible for creating the network, selecting the network topology (star, tree, or mesh), and assigning network addresses to network devices. The router, a networking device that forwards data packets between computer networks, also assists the coordinator in discovering and managing network routes. More precisely, the layer establishes a service interface connecting the 802.15.4 MAC layer and the application layer. Within this context, the Network Layer Data Entity (NLDE) and the Network Layer Management Entity (NLME), offer the following services; The NLDE is responsible for the generation of the Network level PDU (NPDU) which is a data packet or frame, Topology-specific routing and security while the NLME is occupied with configuring a new device, starting a network and joining, rejoining and leaving a network. Additionally, it plays a crucial role in addressing mechanisms, facilitating tasks such as neighbor discovery, route discovery, reception control, and routing.

**Application Layer** The Zigbee application layer architecture includes the application support layer (APS), Zigbee device object (ZDO), and application objects defined by the manufacturer. The application layer is responsible for keeping the binding database up to date and transmitting messages between bound devices. The APS sub-layer provides an interface between the NWK and application layer through a set of services used by both the manufacturer-defined application objects and the ZDO. Two APS entities that offer the following services are the APS Data Entity (APS-DE) and the APS Management Entity (APS-ME). The APS-DE facilitates data transmission services between application entities and the APS-ME is responsible for security services, binding of devices, and group management. Serving as an intermediary between the application framework and the APS, the ZDO facilitates an interface connecting application objects, the device profile, and the APS. The ZDO holds responsibilities such as initializing the APS, the NWK, and Security Services. Additionally, it gathers configuration information from end applications to determine and execute tasks related to discovery, security management, and network management, as well as binding, node, and group management. The application framework serves as an execution environment for hosting application objects with the capacity for up to 254 distinct objects, each identified by an endpoint address from 1 to 254. Endpoint 0 and Endpoint 255 are designated for the ZDO address and broadcast address, respectively, managed by the Application Support Sub-layer Data Entity - Service Access Point (APSDE-SAP). Endpoints 241 through 254 are reserved by the Zigbee Alliance and require approval for use. Application Profiles establish agreements on messages, their formats, and processing actions, enabling the development of interoperable, distributed applications with entities residing on separate devices. The Zigbee Alliance has released public application profiles for applications like Home Automation, while device manufacturers have the flexibility to define custom profiles suited to their specific applications. Clusters, representing collections of attributes and application messages, are categorized into input and output clusters. A cluster identifier is a unique 16-bit number within the scope of a particular application profile. A detailed depiction of the Zigbee protocol stack can be seen in Figure 2.2.

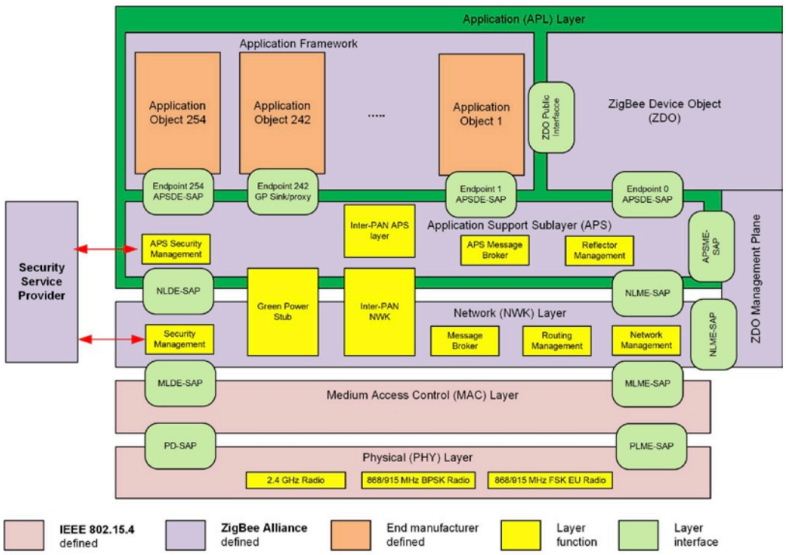


Figure 2.2: Detailed Protocol Stack [12]

### Network Topologies

Zigbee devices are versatile, operating in three distinct modes or node types that define their roles within the network architecture.

The Zigbee Coordinator (ZC) functions as a central node, a Full-Function Device (FFD) responsible for creating, configuring, and managing the Zigbee network. It maintains a list of associated devices and offers services such as association, disassociation, orphan scan, and rejoining. The Zigbee network relies on the constant activity of a Zigbee Coordinator, hence this node cannot enter sleep mode.

In the role of an intermediate FFD, the Zigbee Router (ZR) efficiently relays packets between end devices or between an end device and the coordinator. It becomes the parent for end devices joining the network, facilitating seamless communication.

Zigbee End Devices (ZED) can either be Full-Function Devices (FFD) or Reduced-Function Devices (RFD). These devices, focus on monitoring, data collection, or executing actions based on user commands. With no message routing capabilities, ZEDs communicate solely with their parent node. Typically designed as low-powered, battery-operated devices, ZEDs can enter sleep mode to conserve energy, making them suitable for scenarios requiring minimal power consumption.

The NWK accommodates various topologies, including star, tree, and mesh structures pictured in Figure 2.3.

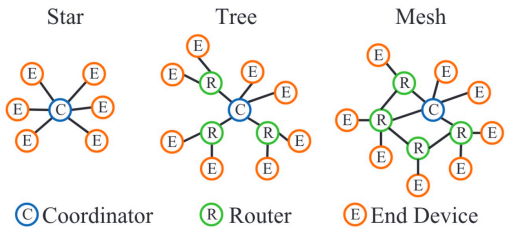


Figure 2.3: Network Topology Types [13]

In a star configuration, a single Zigbee coordinator oversees the entire network, acting as the central controller. End devices communicate directly with the coordinator or other end devices. However, the coordinator can become a bottleneck for message routing, and a failure in the coordinator may lead to a network shutdown.

Tree topologies involve the Zigbee coordinator initiating the network and selecting key parameters, with the network potentially expanding through Zigbee routers. Using a hierarchical routing approach, a router is in charge of transferring control messages and data throughout the network. It might be a child of the coordinator or another router. An end device can be a coordinator or a router's child, and it can only speak with other end devices through a coordinator or a router. If a router fails, it can result in a shutdown of the network segment under the affected router.

In mesh topologies, the Zigbee coordinator is responsible for creation and configuration, and the network can extend using Zigbee routers. Mesh networks facilitate full peer-to-peer communication and are known for their self-healing capability. Even if the coordinator fails, the network persists as end devices communicate with each other and the router. However, mesh networks are complex and introduce messaging overhead in the network.

### Packets

The technical explanations of Zigbee Packets are based on the book "Zigbee wireless networks and transceivers" [14]. Additional resources are cited accordingly.

Information is exchanged among different devices in the form of packets. Figure 2.4 illustrates the typical configuration of a packet. Reusing the 802.15.4 PHY and MAC layers' packet formats, the Zigbee protocol operates on top of them [15]. Packet sniffing in Zigbee typically occurs at the MAC and the PHY layer which is why we will only be focusing on these two layers.

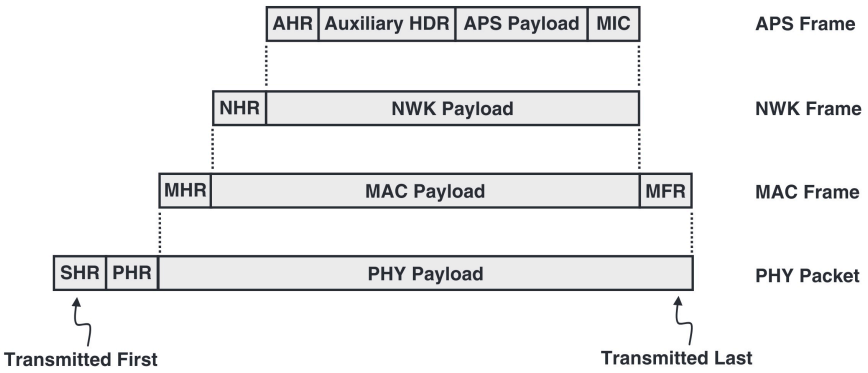


Figure 2.4: Packet Structure [14]

The PHY packet comprises three main elements: the Synchronization Header (SHR), the PHY Header (PHR), and the PHY Payload. The SHR facilitates the receiver in synchronizing and aligning with the incoming bitstream. Within the PHY packet, the PHR carries details about the frame length. The PHY payload, sourced from upper layers, encompasses either data or commands intended for the receiving device.

The MAC frame, sent to other devices as a PHY payload, is divided into three sections: the MAC header (MHR), which contains information such as addressing and security, the MAC payload, which has a variable length size (including zero length), and the MAC footer (MFR), which contains a 16-bit Frame Check Sequence (FCS) for data verification.

Within the framework of IEEE 802.15.4, the standard defines four distinctive MAC frame structures, namely the beacon, data, acknowledge, and MAC command frame depicted in Figure 2.5.

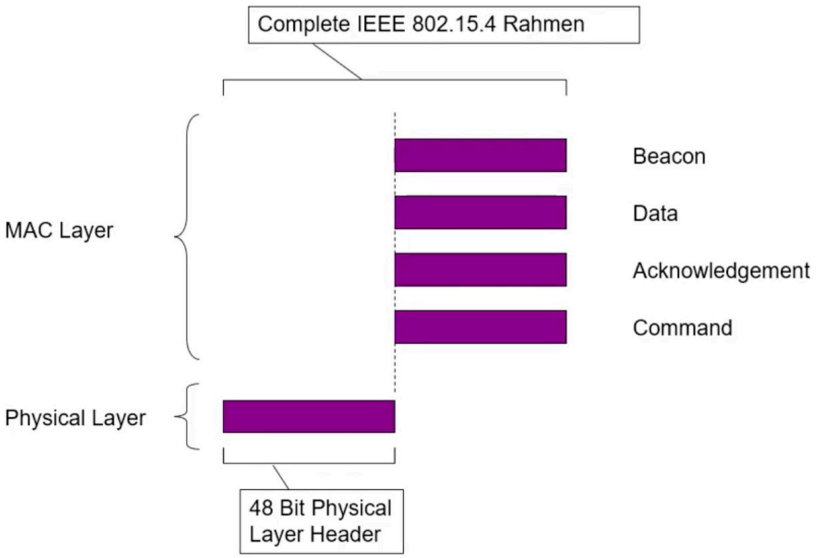


Figure 2.5: MAC Frame [16]

The Beacon Frame assumes its role in the coordination. Coordinators utilize beacon frames to broadcast synchronization signals, aligning the clocks of all devices within the network. For the transmission of actual data, the Data Frame comes into play. Devices leverage data frames to convey information to one another, enabling the seamless exchange of data within the network. Acknowledging successful data reception, the Acknowledge Frame is deployed. When a device successfully receives a data frame, it responds with an acknowledgment frame. This exchange ensures that the sender is informed of the data's secure delivery. Meanwhile, the MAC Command Frame is designated for the transmission of MAC commands. These commands play a central role in network management, addressing, security, and other MAC-layer functions. The MAC command frame serves as the channel through which these crucial commands are communicated, contributing to the effective governance of the network.

### Zigbee Light Link

This subsection is based on the paper "Zigbee light link and its applications" [17] to explain the basics of Zigbee Light Link (ZLL).

ZLL represents a public application profile implemented at the application layer of the Zigbee protocol stack and developed by the Zigbee Alliance to facilitate consumer lighting solutions utilizing the Zigbee PRO wireless network protocol. Operating globally at the 2.4 GHz ISM band, Zigbee PRO supports data rates up to 250 Kbps and incorporates crucial networking mechanisms such as security, mesh routing, and network management. Unlike traditional Zigbee networks, a ZLL system eliminates the need for a coordinator, utilizing a commissioning application called Touchlink for network formation/joining. The ZLL protocol stack is illustrated in Figure 2.6.

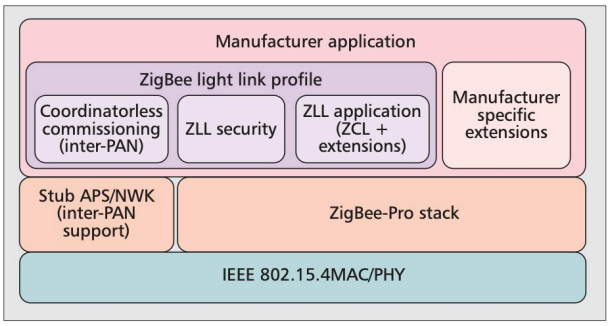


Figure 2.6: ZLL Protocol Stack [17]

Device classifications within ZLL are divided into two main types: light devices and controller devices. Light devices encompass various functionalities such as on/off lights, dimmable lights, color lights, extended color lights, and color temperature lights. Controller devices include a range of tools like light switches, occupancy sensors, remote control units, smartphones, and computing devices. Controller devices can further be categorized into types such as color controllers, color scene controllers, non-color controllers, control bridges or on/off sensors.

## 2.2 Philips Hue

In 2012, Signify [18] unveiled the Philips Hue system, a smart home lighting solution that has gained widespread popularity. Philips Hue consists of interconnected bulbs that users can control through a smartphone or tablet using a Zigbee bridge. This bridge connects to the home router via ethernet. Widely recognized as a leading product in the smart home and Internet of Things space, Philips Hue allows users to manage color-tunable lights from smartphones, web services, or other devices within the system. Significantly, it operates as an open system, allowing other suppliers to add components like smartphone apps, services, light switches, and lamps through standardized or published interfaces [19]. A comprehensive view of the Philips Hue system and its primary components is presented in Figure 2.7.

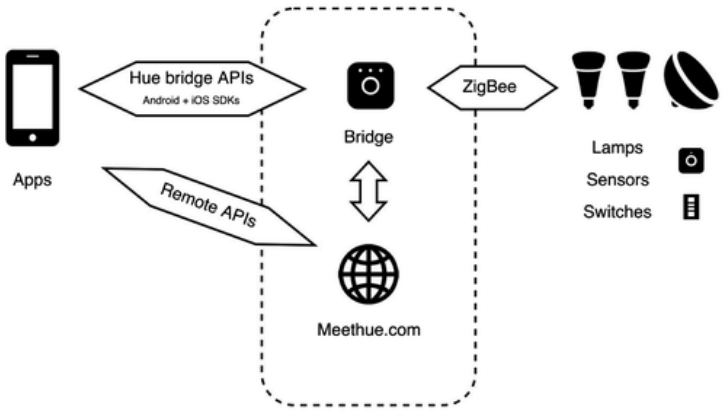


Figure 2.7: Philips Hue Setup [20]

Communication among Philips Hue lamps is facilitated by a standardized Zigbee protocol, enabling integration with Zigbee-based devices such as sensors and light switches. The Philips Hue bridge acting as a coordinator oversees home automation, connecting the Zigbee network to the home IP network and the Internet. On the IP network side, smartphones, web browsers, third-party services, and a Philips Hue portal contribute to the system, all leveraging software and utilizing Philips Hue interfaces or Software Development Kits (SDK) for application development [20].

Philips Hue seamlessly integrates with various other ecosystems, often leveraging standard or open interfaces. The system provides published interfaces for third-party smartphone or tablet apps, third-party services, and browsers to access the Philips Hue system. In essence, services, apps, lights, sensors, and switches designated as Philips Hue or friends of Philips Hue operate using the same mechanisms as third-party apps, third-party services, and standard Zigbee nodes, potentially incorporating some proprietary Philips Hue interfaces or extensions to published interfaces.

Limited details have been released regarding the topology; however, it is worth noting that the light devices possess a Zigbee chip [21]. This notable feature transforms them into routers allowing for a mesh network and seamless installation throughout the home or its surroundings without any compromise in connectivity.

## 2.3 Home Assistant

Home Assistant [22], founded by Paulus Schoutsen, is an open-source platform designed for controlling devices in smart homes or buildings. Maintained by a global community, Home Assistant wants to establish a standard for managing, provisioning, and communicating with a diverse range of devices. As an open-source programmable project, it encourages researchers to actively participate in its development. The platform continually evolves, expanding its functionality through the integration of various dockers, plugins, and devices. Home Assistant embraces existing and widely-used technologies instead of reinventing the wheel, utilizing classic ethernet or local Wi-Fi networks for a functional setup [23].

Home Assistant Core Architecture [24], depicted in Figure 2.8, offers the ability to enhance its functionalities through integrations, each designed for specific domains within the system. These integrations actively engage with events, trigger actions, deliver services, and maintain states. Structured with a core component for fundamental logic and platforms that smoothly interface with other integrations, they are all crafted in Python, leveraging the language's robust capabilities. Home Assistant comes pre-equipped with a diverse array of built-in integrations.

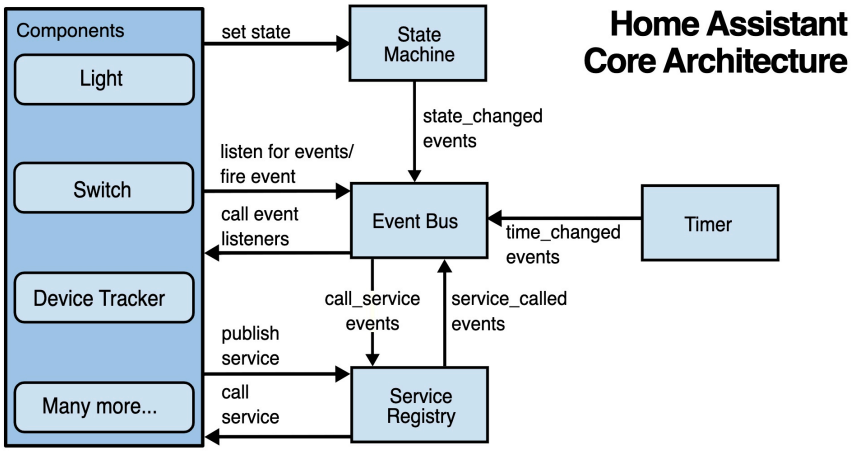


Figure 2.8: Home Assistant Core Architecture

An example of the Philips Hue integration is shown in Figure 2.9. The Philips Hue integration gives the ability to manage and oversee the lights and sensors linked to the Philips Hue bridge. Instead of relying on the Philips Hue bridge, a Zigbee coordinator can be utilized. This enables the use of Zigbee Home Automation (ZHA), facilitating the direct wireless connection of Zigbee-based devices, such as Philips Hue light bulbs, to Home Assistant. [25]

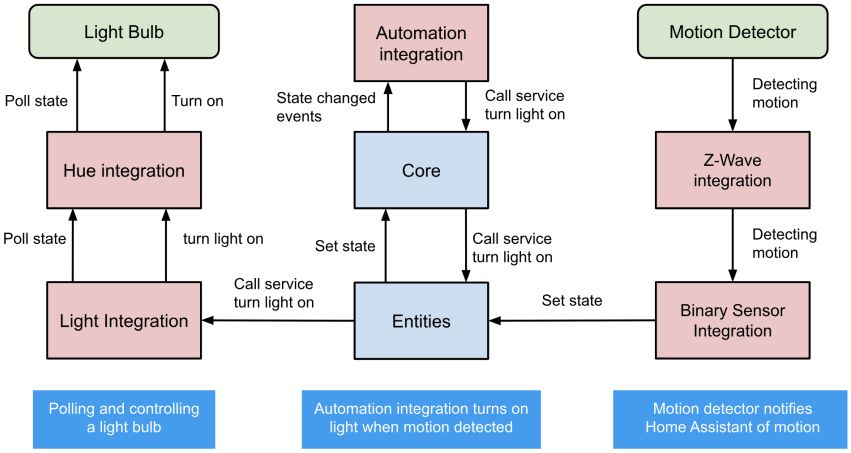


Figure 2.9: Integration Architecture [26]

# Chapter 3

# Related Work

In the past few years, there has been a swift advancement in consumer IoT devices, driven by the demand for smart living among consumers. Nevertheless, smart home devices frequently contain substantial amounts of tangible real-world information that might also interest outsiders and hence open a world for sniffing. An example of a sniffing approach was presented by [27] where they outline a Zigbee protocol packet sniffing recognition method based on software-defined radio, proposing a Zigbee recognition approach using software radio and examining the characteristics of Zigbee protocol messages. Meanwhile, [28] introduced an open-source IoT forensic tool facilitating IoT gateways and Home Automation platforms in conducting live traffic capture and analysis based on IEEE 802.15.4. Another tool that recognizes in-home devices or events within encrypted Zigbee traffic is ZLeaks by [29] working with smart hubs such as Philips Hue. ZLeaks achieves this by deducing a single application layer command in the event's traffic and exploiting the device's periodic reporting pattern and interval. Moreover, [30] which introduces a method to eavesdrop on user privacy by analyzing wireless context, extracting packet sequence features detecting events, and deducing user activities, moods, lifestyle patterns, and the presence of installed IoT devices also sheds light on the user profiling that results from sniffing and monitoring. Also, [31] delved into the extraction of detailed information about users by leveraging diverse everyday devices. The study employs advanced learning techniques to deduce user behaviors, specifically by identifying and locating the use of smart home devices, capitalizing on the exceptional performance of advanced learning across diverse domains.

Conversely, there has also been research aimed at understanding and benefiting from people's behavior, rather than merely observing it. Allahham and Rahman propose a smart monitoring system for campus infrastructure, intending to regulate the opening and closing of doors in numerous halls and potentially integrate with lighting systems and appliances. The OPNET program results reveal variations in the performance of the network topology used within a Zigbee network, providing designers with the flexibility to tailor their networks and select technologies that align with their project requirements [32].

The mere possibility of eavesdropping opens up opportunities for potential attackers. Namely, Jansen demonstrates that an eavesdropping attacker, during the addition of a

device to the Zigbee network, can obtain keying material, enabling them to decrypt and manipulate network traffic which can lead to unauthorized control of devices and the potential for burglars to exploit network activity cues to determine a user's presence at home [33]. Likewise, evaluating real measurements on popular smart home IoT devices, the results indicate that an adversary can achieve over 90% accuracy in identifying the state and actions of targeted devices and users. This was presented by [34] when launching a multi-stage privacy attack on user privacy in smart environments, utilizing advanced machine learning to passively sniff network traffic. A relatively bigger attack was launched by [35] where they outlined a novel threat wherein neighboring IoT devices infect each other with a worm, spreading rapidly over large areas, contingent on the density of compatible IoT devices surpassing a critical mass. The study specifically demonstrates and verifies this infection using Philips Hue smart lamps, illustrating the worm's propagation through their built-in Zigbee wireless connectivity and physical proximity. Comparably, also Wara and Yu have tried themselves on Philips Hue light bulbs where they present a novel replay attack aimed at exposing security vulnerabilities in commercial Zigbee devices, specifically targeting Philips Hue bulbs and Xbee S1 and S2C modules, with two case studies confirming the successful transmission of captured packets despite built-in countermeasures [36].

Transitioning to the broader examination of security vulnerabilities in Zigbee devices, a further study from [37] involving the Philips Hue but now focusing on the security aspect conducts a comprehensive analysis of the ecosystem surrounding the smart bulb to evaluate privacy risks associated with diverse control devices and smartphone applications. The findings reveal that employing varied techniques to switch the bulb ON or OFF significantly influences the actors gathering information on the user's home and the volume of data transmitted to the Internet. As smart home devices frequently contain substantial real-world information, raising concerns about potential information leakage, [38] introduced the ZPA system for analyzing privacy that relies on Zigbee-encrypted traffic mainly to address privacy and security issues in Zigbee-based smart home networks. Furthermore, an interesting approach is the introduction of the ChatterHub system, which addresses privacy concerns in smart-home devices, particularly those utilizing Zigbee or Z-wave and controlled by a centralized smart-home hub within a PAN connected to the Internet. ChatterHub employs passive eavesdropping on encrypted network traffic from the hub and utilizes machine learning techniques to classify events and states of smart-home devices, contributing to enhanced privacy management [39].

# Chapter 4

# Design

This chapter details the design of the thesis, which centers around an experimental process comprising three main phases: setup, data collection, and data evaluation. These phases, presented in Figure 4.1, form a loop where evaluation informs and improves data collection, essentially making it an experimental process. Additionally, the evaluation involves creating a prototype for a live and passive sniffer program analyzing Zigbee traffic.

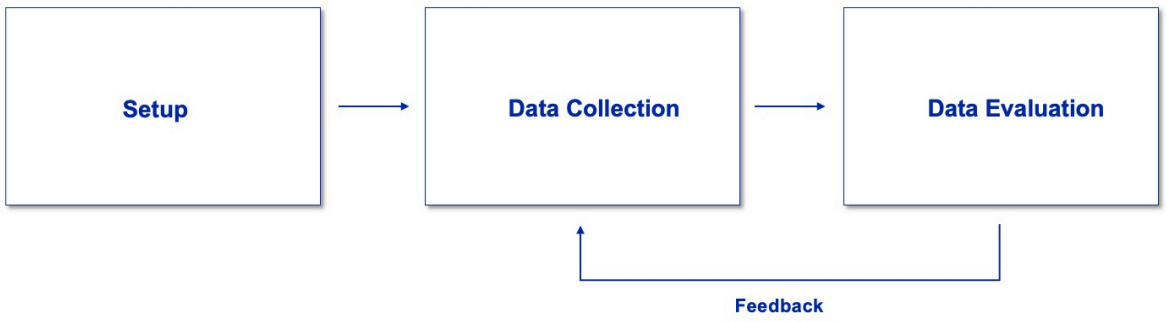


Figure 4.1: Design Phases

The design and implementation are influenced by two key papers. Firstly, the *ZPA* system [38], which analyzes privacy using Zigbee-encrypted traffic. It highlights how easy it is to obtain network keys, enabling the decryption of entire Philips Hue networks and providing insights into user behavior. Secondly, the *Zleaks* tool [29], which identifies devices and events from encrypted Zigbee traffic. This is achieved by analyzing the commands given at the application layer and utilizing device reporting patterns. This demonstrates the usefulness of inference rules in decrypting networks and understanding user behavior.

## 4.1 Setup

The setup comprises both hardware and software components, along with an exploration of the Philips Hue network through packet analysis and an assumption.

### 4.1.1 Hardware

The hardware setup involves utilizing the Nordic Semiconductor nRF52840 Development Kit with its sniffer software flashed onto it, alongside three Philips Hue bridges, and a variety of routers and end devices. Specifically, three distinct setups were utilized for data collection and testing purposes. Each setup operates on different Zigbee channels to minimize interference and simplify management.

Setup 1 encompasses a wide-ranging arrangement including one bridge, about sixteen light devices, and several sensors and switches which are not considered in testing. It mirrors a real-world smart home configuration, covering areas such as the ground floor comprising the living and dining room, kitchen, and corridors and the first floor comprising all bedrooms. Setup 1 operates on the default channel 11, making it the most unpredictable and potentially challenging to work with. A diagram of the setup's topology is shown in Figure 4.2.

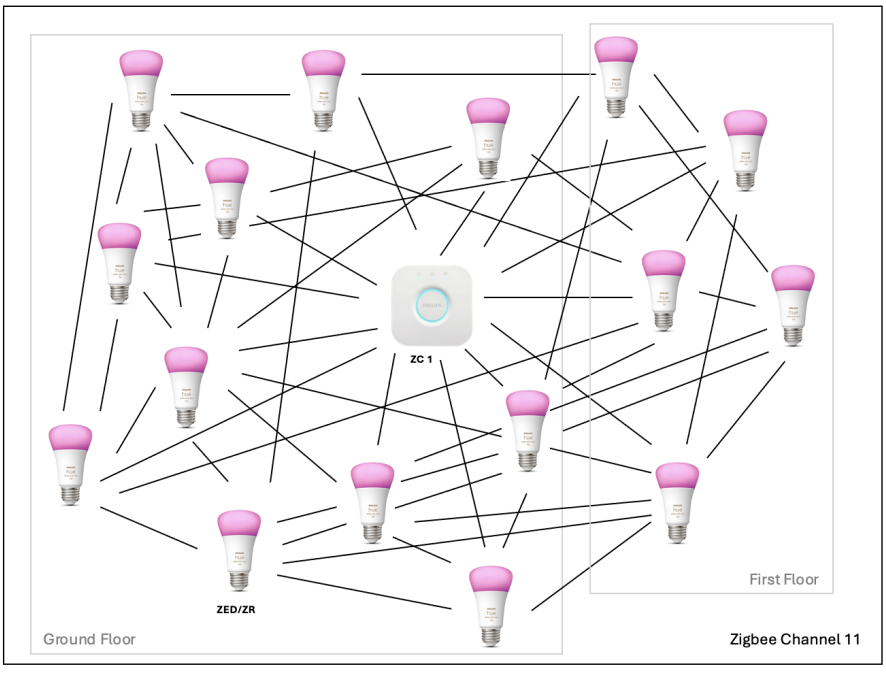


Figure 4.2: Setup 1 Topology

Conversely, Setup 2 seen in Figure 4.3, represents another genuine smart home setup, covering spaces like the basement and laundry room on the same floor. It comprises one bridge, six light devices, and assorted sensors and switches which were also disregarded in testing. Setup 2 operates on channel 20.

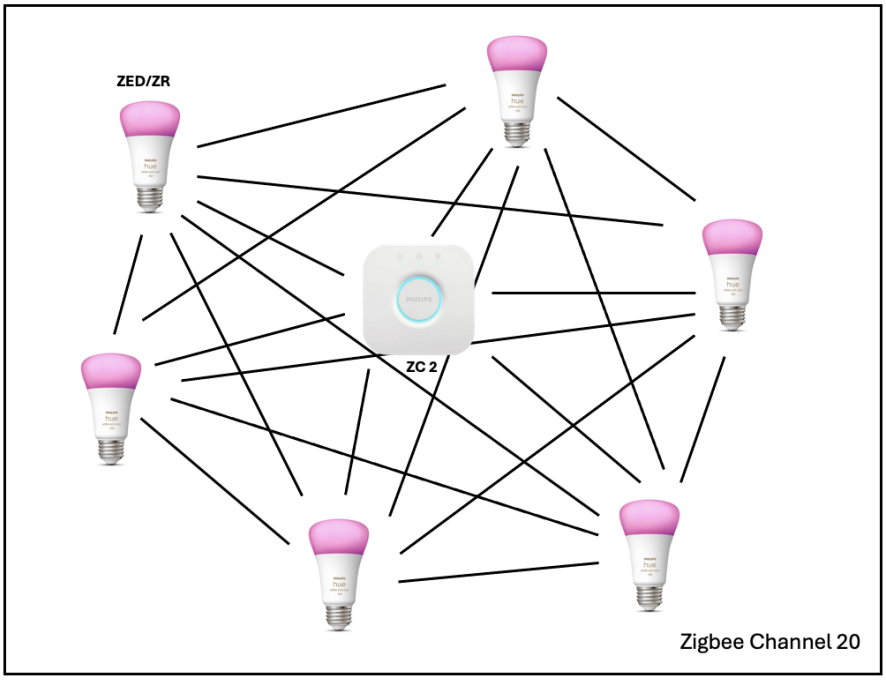


Figure 4.3: Setup 2 Topology

On the other hand, Setup 3, depicted in Figure 4.4 is established as a testing setup, featuring one bridge and two light devices. This arrangement is characterized by its clarity, predictability, and ease of manipulation. Setup 3 utilizes channel 15, chosen for its relative isolation to facilitate easier experimentation.

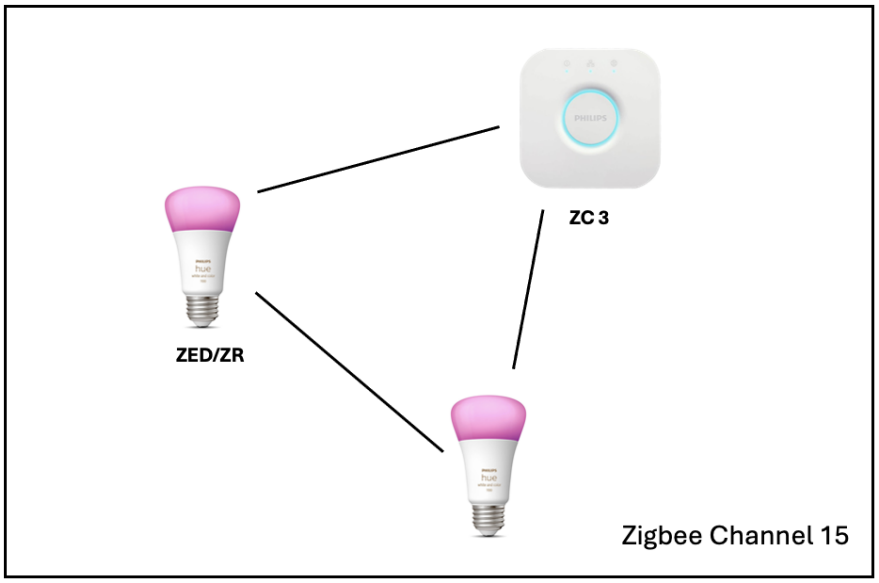


Figure 4.4: Setup 3 Topology

### 4.1.2 Software

The software setup involves the utilization of Wireshark, including Tshark for packet capture and analysis, installed on a MacBook Pro. Additionally, the official Philips Hue app on an iPhone 15 Pro was employed as part of the software infrastructure.

### 4.1.3 Assumption

The setup also featured first-time sniffing to learn about the packets, their structures, and, most crucially, how the Philips Hue network operates in packets. To gain an understanding of their behavior and material in general, sniffing was performed across the several networks and channels to identify commonalities and generalizations. This is also where the assumption originated. Specifically, *possessing a single network key might possibly allow the decryption of other Zigbee Philips Hue networks, regardless of their network keys.* This is what was investigated in the subsequent stages of this thesis.

To examine all of this, the setups' network keys were required to decrypt the networks. This was achieved by using the explanation of the different types of keys, how they work, and instructions in *ZPA*, demonstrating how simple it is to obtain a network key. One merely needs to add a new light device to the Philips Hue network while sniffing to obtain the packet with the transport key and the packet with the network key seen in Figure 4.5, which in Wireshark decrypts every packet when added to the keys.

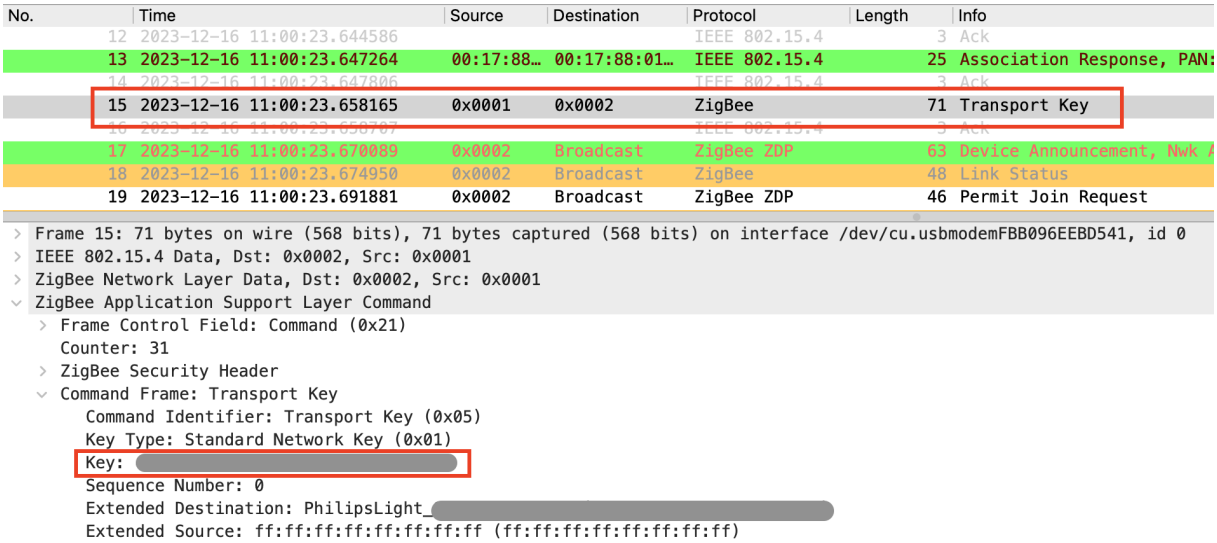


Figure 4.5: Transport Key Packet to gain Network Key

### 4.1.4 Sniffing operations

Another decision made during the setup was which commands to use and where the prototype should be fitted. The final commands chosen were the *on/off* command as

*broadcast* messages and the *level/color control* in *unicast* message format. The *on* and *off* commands were chosen because they are the most commonly used command to turn on and off light devices. The examination and subsequent decision to focus on *broadcast* messages was a time-consuming process which will be discussed extensively in subsequent chapters. The *level* and *color contro*l commands were selected because it is believed they are the fundamental commands that define Philips Hue and make it so popular. Following various considerations, a schedule for later testing the prototype was devised, deciding to test the commands on two lamps in the same room with a three-minute rest period between the commands start program, room on, dim light device 1, dim light device 2, color light device 1, color light device 2, room off and stop program.

The testing of the prototype occurred in all three networks. Twice passively, where Wireshark was used to capture packets and generate a PCAP/PCAPNG file, and twice live, where the identification of devices and their type was tested more rigorously, as explained in the following section.

## 4.2 Data Collection

Data collection began with sniffing while executing various commands in Setup 3 and progressed to Setup 1 over a 15-45-minutes period to determine how much information one could gather in a relatively short period of time. Although it was initially discussed to sniff for 24 hours, it was discovered that Wireshark finds it difficult to sniff for that long. In less than 30 minutes, thousands of packets had already arrived, and many of them contained commands that were not interesting because they are "only" *read attributes* packets and their *responses*. An alternative to Wireshark's long-term sniffing problem could be its integrated Tshark, which is a command-line tool where it is slightly more difficult to analyze content directly but can be used for live tracking because it does not require an interface and can run longer. Sniffing first started in Setup 3 to get a clear picture of the Philips Hue network and to test the Python scripts, which are discussed in detail later. It also helped navigate the entire network, which can be overwhelming. Then sniffing in Setup 1 began to produce a more realistic smart home setting, which can be confusing, chaotic, and unpredictable due to the presence of numerous devices, sensors, and switches and also packets from different networks. After several iterations of the data collection and evaluation loop and several feedbacks from the data evaluation, Setup 2 was brought into activity, which is another realistic smart home setup to finally test the assumption and prototype.

## 4.3 Data evaluation

Data evaluation primarily consisted of analyzing packets and constructing scripts based on the assumption to provide input for a next round of data collection. The first and most important step was to evaluate the packets with Wireshark. Initially, it was simply analyzing what information they held and playing around with the commands until the

first aim was reached: compare de- and encrypted packets and assess what information is displayed when the network is encrypted and which may be used to decrypt it. After analyzing thousands of packets along with *Zleaks*, it was discovered that the frame and data length of packets remain constant for each packet type. For example, the frame and data length of a *unicast on* command packet remained the same no matter the network or Zigbee channel. Transitioning the analysis to focus on packet frame and data length. As certain packet types have the same combination of frame and data length as others, other ways to distinguish them when encrypted were necessary. The idea of inference rules with the inspiration of *Zleaks* came alive. The *unicast off* command has the same frame and data length as a *read attribute* packet where the direction, the source, and the destination, are also the same which is why they are difficult to distinguish. Hence the sequencing of packets becomes crucial in this context. A key observation is that a *read attributes response* packet, originating from the end device and destined for the bridge, consistently follows its counterpart, a *read attributes* packet with opposite directional flow. This establishes an inference rule. Consequently, the packet arrival sequence was carefully examined to establish such rules. However, a significant challenge arises from occasional packet skips which can lead to misidentification during command analysis.

In summary, the components involved in packet analysis comprised packet and data length, packet direction, and packet sequences along with their interconnections. The second part of the data collection included writing scripts and finding ways to decrypt the network. This involved using several packet types and packet directions to identify devices and their types as well as events such as the *on/off* and *level/color control* commands. As a result, a live and passive Zigbee sniffer prototype that can monitor a decrypted Philips Hue network was designed, which is discussed in more detail in further sections.

### 4.3.1 Prototype design

The initial prototype targeted passive decryption of a Philips Hue network by identifying device types and capturing *on/off* and *level/color control* commands. The program comprises various Python scripts, each assigned specific tasks. Python was chosen for its popularity and integrated tools for working with Zigbee packet captures, such as pcap files. Moreover, the fact that platforms like Wireshark, Home Assistant, and *Zleaks* utilize Python further affirmed its suitability. The core scripts that constituted the first approach were the *packet extractor*, the *command extractor*, the *filter*, the *identifier*, and the *analyzer*.

As packets from multiple networks can get mixed up during capture, the *packet extractor* is in charge of extracting the required data from the packets and writing them in a CSV file per PAN ID. Using the comparison of two JSON files from the same network, one encrypted with the network key and the other not, the *command extractor* may extract and map the frame and data length of particular packet types or commands collectively. After that, the *filter* removes any *Zigbee Home Automation* commands, improving the analysis' efficiency. As implied by the name, the *identifier*'s job is to identify devices and their classification. The *analyzer* is in charge of examining the events taking place in the Philips Hue network.

Eventually, however, the prototype changed to serve two purposes: a *tracker* script was built to enable the prototype to track the Philips Hue network in real-time, rather than only passively. Last but not least, the script that keeps everything together and specifies what needs to run when is the *runner*. The upcoming chapter will delve further into the implementation and choices made for each script.

### 4.3.2 Home Assistant Integration

One objective of the thesis was to assess the integration of Home Assistant into the prototype and determine the most effective approach. However, upon scrutinizing the data accessibility through Home Assistant, it became evident that it was not a suitable choice. Home Assistant primarily functions as a home automation tool, emphasizing convenience rather than providing the necessary depth into the protocol layers wished for the prototype's functionalities. Consequently, the decision was made to forego integration, favoring alternative methods such as solely utilizing the nRF board and Wireshark for delving into the packet layer and extracting raw data, which proved more effective in achieving the desired depth.

Although possessing the GitHub repository of Home Assistant, significant changes would have been necessary to align its functionality with the prototype's purpose. Ultimately, the prototype achieved similar functionalities, specifically in tracking Zigbee devices, rendering the integration of Home Assistant unnecessary. Nevertheless, Home Assistant was employed as a control gadget during the testing phase to verify the prototype's device identification accuracy. Given the abundance of Zigbee devices in the household, including those beyond Philips Hue devices and integrated within Home Assistant, it seemed logical to utilize it for verification purposes. However, this posed challenges as well as the prototype operated with network addresses, while Home Assistant concealed them for Philips Hue devices, utilizing MAC addresses instead, highlighting the disparity in depth between the two systems.

# Chapter 5

# Implementation

This chapter will present the implementation choices made for each script, which together represent a comprehensive approach to developing a sniffer application prototype. It will follow the chronological order of implementation, reflecting the iterative nature of the decision-making process, which was informed by the data evaluation outlined in the design phase. The chapter also introduces an implementation discussion, wherein each script is revisited to explain and discuss the process of their implementation. This discussion is kept separate in its own section to prevent any interruptions in understanding the final prototype's implementation flow as it also includes approaches that are no longer present.

## 5.1 Command Extractor

The provided Python script operates under the premise that possessing a Philips Hue network key can potentially decrypt any other Philips Hue network. Its core objective is to extract crucial data from two JSON files originating from the same network capture. One JSON file is decrypted using the network key, while the other remains encrypted. The aim is to correlate commands and frame lengths with their respective data lengths, which are only accessible when encrypted. The script generates three CSV files: *data.csv*, containing packet frame numbers, lengths, and data lengths, *command.csv*, storing frame numbers, lengths, and commands and *command_data.csv*, which combines both by frame number.

The script comprises two main functions. Firstly, *extract_command_from_packets* handles the extraction of commands and their frame lengths, facilitating easier access and reusability through CSV files. These commands encompass various Philips Hue network operations, including *on/off, color/level control, read attributes*, and *route record/request*, each identified by a specific Philips Hue network identifier. The list below demonstrates the commands with their Philips Hue network identifier representing the command.

- on_off = "zbee_zcl_general.onoff.cmd.srv_rx.id"
- color_control = "zbee_zcl_lighting.color_control.cmd.srv_rx.id"

- level_control = ”zbee_zcl_general.level_control.cmd.srv_rx.id”
- read_attributes = ”zbee_zcl.cmd.id”
- route_record = ”Command Frame: Route Record”
- route_request = ”Command Frame: Route Request”

The first four commands were chosen as previously explained in the section sniffing operations 4.1.4. The other 3 commands were extracted for the purpose of identifying device types which will be discussed more in the identifier section 5.4. The *extract_data_from_packets* function focuses on retrieving data lengths from the encrypted JSON file and appending them to the *data* CSV file. By merging the *command* and *data* CSV files based on frame numbers, the script effectively maps command and frame lengths to their corresponding data lengths, yielding the final *data_command* CSV file. This preparatory step lays the groundwork for subsequent processes in identifying and analyzing devices, device types, and network events.

## 5.2 Packet Extractor

The packet extractor serves to streamline the analysis of PCAP or PCAPNG files by condensing the captured data into a CSV format, making it easier to analyze. Its primary objective is to extract essential information from network packets, reducing clutter and facilitating analysis.

The function *extract_packet_data* plays a crucial role in this procedure by identifying the relevant data that has to be extracted, such as:

- Frame time
- Frame length
- Source
- Destination
- Destination PAN
- Frame type
- Radius
- Data length
- Sequence number
- Frame number

These extracted attributes serve various analytical purposes. For instance, the frame time provides temporal context, indicating when events occurred. Meanwhile, frame length, source, destination, and data length aid in device identification and event categorization. The destination PAN plays a crucial role within the *packet extractor* by facilitating the organization of packets based on their network PAN ID. This segmentation ensures that packets are grouped logically according to their respective PANs, enhancing the clarity and manageability of the data. Furthermore, within the *identifier*, both the frame type and radius parameters are utilized. These attributes serve as additional criteria for packet classification and identification purposes. By incorporating these parameters, the *identifier* can more accurately categorize packets, thereby refining the analysis process and providing deeper insights into network activity. The frame number is used for analysis purposes, while the sequence number is used to eliminate duplicates.

The *pcap_to_csv* function initiates by applying a display filter 5.1, focusing on relevant packets while excluding others like Zigbee Device Profile packets. Subsequently, the function traverses each packet, ensuring uniqueness by comparing attributes such as source, destination, length, data length, and sequence number with the subsequent 254 packets to eliminate duplicates. This approach prevents unnecessary duplication removal that could occur if each packet were compared against all others.

```
cap = pyshark.FileCapture(input_pcap, display_filter='zbee_nwk && !(_ws.
    col.protocol == "Zigbee ZDP")')
```

Listing 5.1: Pyshark Display Filter

Once duplicates are mitigated, the script employs *extract_packet_data* to retrieve relevant packet information. The extracted data is then organized into multiple CSV files, each corresponding to a unique PAN address found in the captured data. Following extraction, the *tracking* function determines the file for analysis, functioning as both a live and passive tracker by checking

In summary, the packet extractor streamlines data analysis by condensing captured packets into a structured CSV format, enabling efficient manipulation and analysis.

## 5.3 Filter

The filtering script has a relatively minor role compared to other scripts in the program. Its primary task is to process a CSV file containing extracted data per PAN originating from the *packet extractor* script. The output of this script is a filtered CSV file containing only *ZHA* packets.

The purpose of this filtering process is to exclude any other packets than *ZHA* commands related to basic functionalities such as *on/off, color/level control,* and *read attributes* with *response* commands. By removing other packet types, the analysis of the data becomes smoother, as the inference rules rely on examining consecutive packets, and the presence of other packet types could potentially disrupt the sequence. The filtering script plays an important role in optimizing the data for further analysis, ensuring that only relevant *ZHA* packets are retained, thus facilitating a more accurate and efficient analysis process.

## 5.4 Identifier

The *identifier*, as the name implies, is responsible for recognizing devices and categorizing them. It operates by taking a selected PAN CSV file as input and generates device identifications as output. The *process_csv* function initially compiles lists containing various combinations of commands, frames, and data lengths. These combinations serve as reference points for identifying packets during the subsequent comparison process.

The packets under consideration include *route record*, *route request*, and *read attribute response*. Through analysis conducted during the design phase, it was established that *route record* and *request* packets predominantly originate from the Zigbee coordinator. Consequently, these packets serve as key indicators for identifying the coordinator within the network. On the other hand, the *read attribute response* command is commonly associated with Zigbee end devices. This is due to the inherent behavior of such devices, which continuously report their attributes to the coordinator.

The primary function of the *identifier* is to match the frames and data lengths of each packet against the pre-defined combinations. When a match is found, the source of the packet is stored in a list corresponding to a potential device type. This process enables the identification of devices based on their transmitted packets. An example of a code snippet can be seen in Listing 5.2.

```
with open(packet_file, 'r') as pcktfile:
    reader = csv.DictReader(pcktfile)
    for row in reader:
        if row['Source'] != row['Destination']:
            if row['Length'].isdigit() and row['Data length'].isdigit():
                frame_length = int(row['Length'])
                data_length = int(row['Data length'])
                if (frame_length, data_length) in
                    command_data_route_record:
                     if row['Destination'] not in broadcast_addresses:
                         p1_zc_devices.append(row['Destination'])
                     if row['Source'] not in p1_zed_devices and row['
                        Source'] not in broadcast_addresses:
                         p1_zed_devices.append(row['Source'])
            else:
                print('Make sure you removed the network key from
                    Wireshark.')
                break
```

Listing 5.2: Identifier Primary Function

Several improvements have been implemented to enhance the program's reliability. There are two distinct lists for potential coordinators and end devices. These lists are cross-referenced at the end to ensure accurate identification and prevent misidentification. Additionally, the potential coordinators' list undergoes further scrutiny by evaluating which source has generated the most *route records* and *route requests*, as sometimes non-coordinators transmit these packets as well.

Identification of routers does not necessitate comparison with the commands list. Instead, the function directly examines whether the packet is a *link status* packet, a characteristic

of routers, and stores its source. This determination is based on specific conditions: the destination address of the packet being a broadcast address, the frame type being "0x0001", and the radius being "1".

Lastly, the function *store_zc* records the coordinator's network address in a text file, which is subsequently utilized in the *analyzer*. The script organizes the devices into lists according to their respective device types, providing a streamlined approach for further analysis and processing.

## 5.5 Analyzer

The *analyzer*'s unique responsibility is to recognize the *on, off, color,* and *level control* Philips Hue events inside the network. As well as the *identifier* it carefully records the distinct frame and data lengths linked to the particular commands in a list called events. These combinations also serve as reference points for comparison against incoming packets, enabling the *analyzer* to discern whether a packet corresponds to an event. An illustration in Listing 5.3: a *broadcast* packet signaling an *on* command typically exhibits frame and data lengths of "47" and "11" respectively, against which incoming packets are compared to.

```
if event[0] == 'on broadcast' and dst in broadcast_addresses: # 47, 11
    print('User turned light on at ' + row['Time'] + dst)
```

Listing 5.3: Check *On* Packet

*Color control* packets share identical frame and data length combinations with other packet types, such as *read attribute* packets, within an encrypted network. This similarity underscores the necessity for inference rules, as previously discussed where packet sequencing plays a pivotal role. For example, *read attribute* packets always generate a *response*, which requires certain frame and data length sequences that follow it, as seen in Figure 5.1 below.

| | | | | | | |
|---|---|---|---|---|---|---|
| 14027 | 2024-03-17 15:13:01.400405 | 0x1327 | 0x508c | ZigBee HA | 48 | ZCL: Read Attributes, Seq: 214 |
| 14028 | 2024-03-17 15:13:01.400950 | | | IEEE 802.15.4 | 3 | Ack |
| 14029 | 2024-03-17 15:13:01.418558 | 0x508c | 0x1327 | ZigBee HA | 51 | ZCL: Read Attributes Response, Seq: 214 |
| 14030 | 2024-03-17 15:13:01.419102 | | | IEEE 802.15.4 | 3 | Ack |
| 14031 | 2024-03-17 15:13:01.440113 | 0x1327 | 0x508c | ZigBee HA | 48 | ZCL: Read Attributes, Seq: 215 |
| 14032 | 2024-03-17 15:13:01.440659 | | | IEEE 802.15.4 | 3 | Ack |
| 14033 | 2024-03-17 15:13:01.452541 | 0x508c | 0x1327 | ZigBee HA | 51 | ZCL: Read Attributes Response, Seq: 215 |

Figure 5.1: Read Attribute Packet Sequence

In essence, when identifying a potential *color control* packet, the application of inference rules becomes imperative. These rules ascertain whether a subsequent *read attribute response* packet follows. If not, it allows for the distinction of the packet as a *color control* packet. The example in Listing 5.4 demonstrates the code used to examine the subsequent packets following a potential *color control* packet.

```
elif event[0] == 'color control' and src == zc:
    for j in range(1, 11):
        next_row = df.iloc[i + j] if i + j < len(df) else None
```

```
        if next_row is not None and next_row['Length'] == 60 and
           next_row['Data length'] == 25 and next_row['Source'] == dst
           and next_row['Destination'] == src:
            break
        elif next_row is not None and next_row['Length'] == 70 and
           next_row['Data length'] == 35 and next_row['Source'] == dst
           and next_row['Destination'] == src:
            break
        elif next_row is not None and next_row['Length'] == 69 and
           next_row['Data length'] == 34 and next_row['Source'] == dst
           and next_row['Destination'] == src:
            break
        elif next_row is not None and next_row['Length'] == 68 and
           next_row['Data length'] == 33 and next_row['Source'] == dst
           and next_row['Destination'] == src:
            break
        elif next_row is not None and next_row['Length'] == 67 and
           next_row['Data length'] == 32 and next_row['Source'] == dst
           and next_row['Destination'] == src:
            break
        elif next_row is not None and next_row['Length'] == 99 and
           next_row['Data length'] == 64 and next_row['Source'] == dst
           and next_row['Destination'] == src:
            break
    else:
        print('User controlled color of light at ' + row['Time'] + dst)
```

Listing 5.4: Inference Rule for *Color Control* Packet

Furthermore, it's important to consider the direction of packet flow. Specifically, event commands consistently originate from the coordinator, underscoring the significance of storing its network address in the *identifier* for subsequent use. This is again exemplified by the *color control* packet, which may inadvertently be misconstrued as a *ZCL Groups: Get Group Membership response* packet. To discern a *color packet* from the latter, its source must align with the coordinator, while the source of the *ZCL Groups: Get Group Membership response* packet typically stems from an end device. Upon recognizing an event, the program promptly logs the event, its timestamp, and the associated network address.

## 5.6 Tracker

The *tracker* plays a critical role in ensuring the timely execution of appropriate scripts during live network tracking. It operates autonomously, without input or output, relying on the subprocess module to manage tracking and analysis concurrently. By initiating the process with Tshark, it captures an initial set of 255 packets, subsequently analyzing them while continuously capturing more. This 255-packet capture strategy serves several purposes. Firstly, it addresses the time delay incurred by background packet analysis, which could worsen with an increasing number of packets. Secondly, it mitigates the risk of packet duplication, akin to the window setting in packet extraction. Moreover, limiting the number of packets enhances the clarity of analysis, streamlining the analytical process.

## 5.7 Runner

The *run* script serves as the backbone of the prototype, orchestrating the execution of all the scripts seamlessly. It operates independently, neither requiring input nor producing output directly. Upon initiation, it prompts the user to specify the desired mode of operation, live or passive tracking. In live mode, the script starts with a 90-second network sniffing session to capture enough packets for device identification. Subsequently, the user is prompted to select the network for analysis. The script then proceeds to identify devices using the appropriate commands and packet extraction techniques, providing the user with a comprehensive list of devices detected. Following network selection, the *filter* is applied, and live tracking commences as described in the *tracker*. Conversely, in passive mode, the initial 90-second sniffing phase is omitted. Instead, the script analyzes the PCAP/PCAPNG file provided by the user. It follows a sequence of command execution, packet extraction, filtering, device identification, and analysis.

Finally, the *run* script concludes by clearing all generated CSV and text files or their contents, ensuring a clean slate for subsequent analyses. The following diagrams below illustrate the entire process in short.

## 5.8 Flowcharts

The execution of the live mode, shown in Figure 5.2 begins with the initiation of *run.py*. In the initial step, the user opts for live mode, providing the interface number of the sniffer as input through *tshark_command.txt* to facilitate the 90-second capturing. Following this, the *command extractor* processes the two JSON files, yielding three distinct CSV outputs: *command.csv*, *data.csv*, and *command_data.csv*. Subsequently, the *packet extractor* module operates on *capture.pcap* from the 90-second capture, extracting packet data into a formatted CSV file named *pan_ID.csv*. The *identifier* then analyzes the packet data, producing *zc.txt* and storing the coordinator as output. Then the *filter* is employed on the *pan.csv* selected by the user which yields *filtered_pan.csv* as the output. Then the *tracker* commences also taking the interface number as input through *tshark_command.txt*. The *packet extractor* and *filter* are run again but with the input from each live capture. The *analyzer* then finally, utilizes *command_data.csv* and *filtered_pan.csv* as input and analyzes them printing out the events when finished.

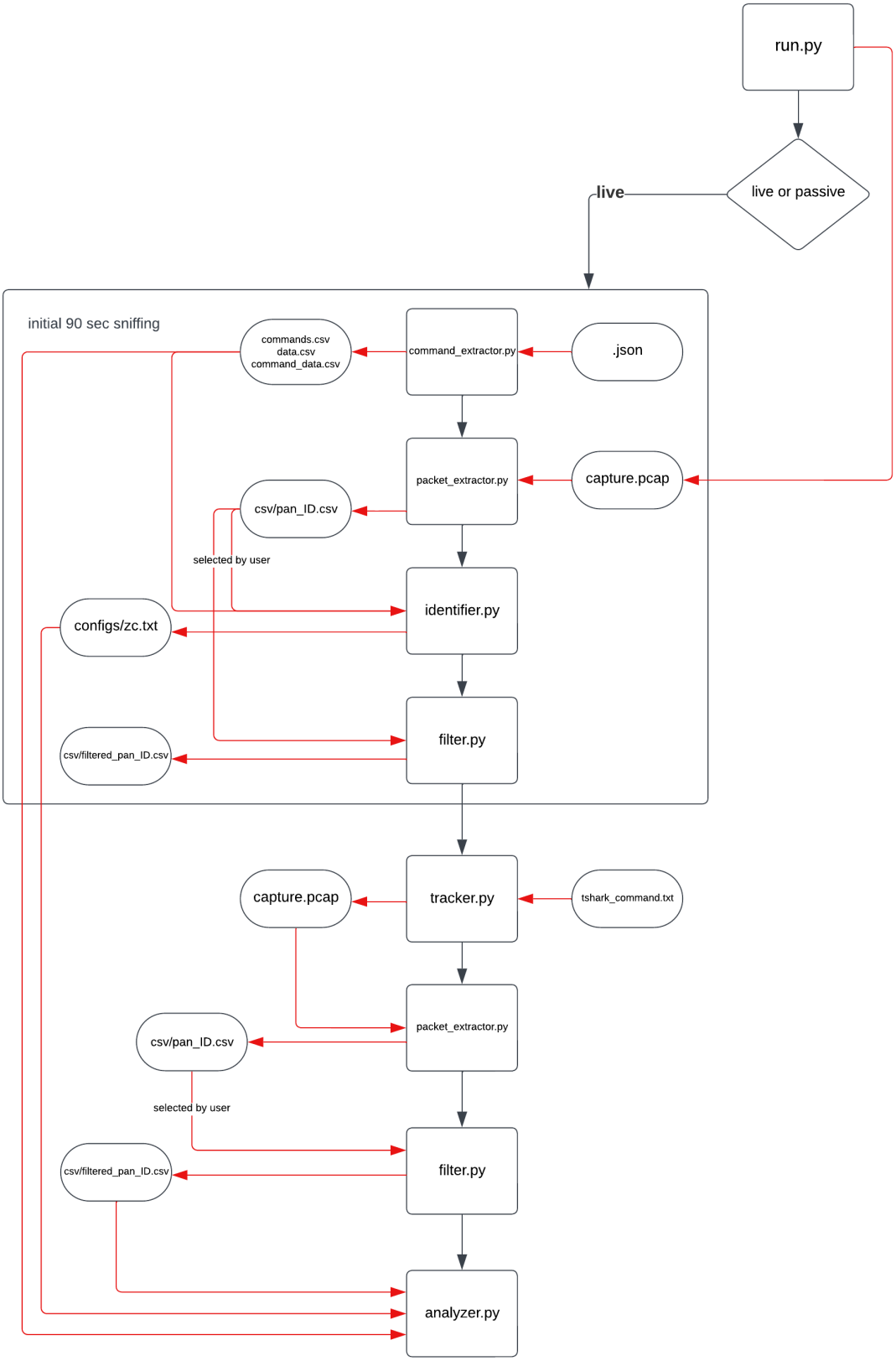


Figure 5.2: Flowchart Live Mode

The execution in passive mode is depicted in Figure 5.3 and begins with the initiation of *run.py* where the user now opts for passive mode. Following this, the *command extractor* processes the two JSON files, also yielding three distinct CSV outputs. Then, the *packet extractor* operates on the PCAP file *.pcap* from the PCAP folder extracting the packet data into CSV files per PAN. The *identifier* module then analyzes the packet data, producing the *zc.txt* and storing the coordinator. Then the *filter* takes the *pan.csv* selected by the user which gives *filtered_pan.csv* as the output. Finally, the *analyzer* utilizes *command_data.csv* and f*iltered_pan.csv* as input and analyzes them printing out the events.

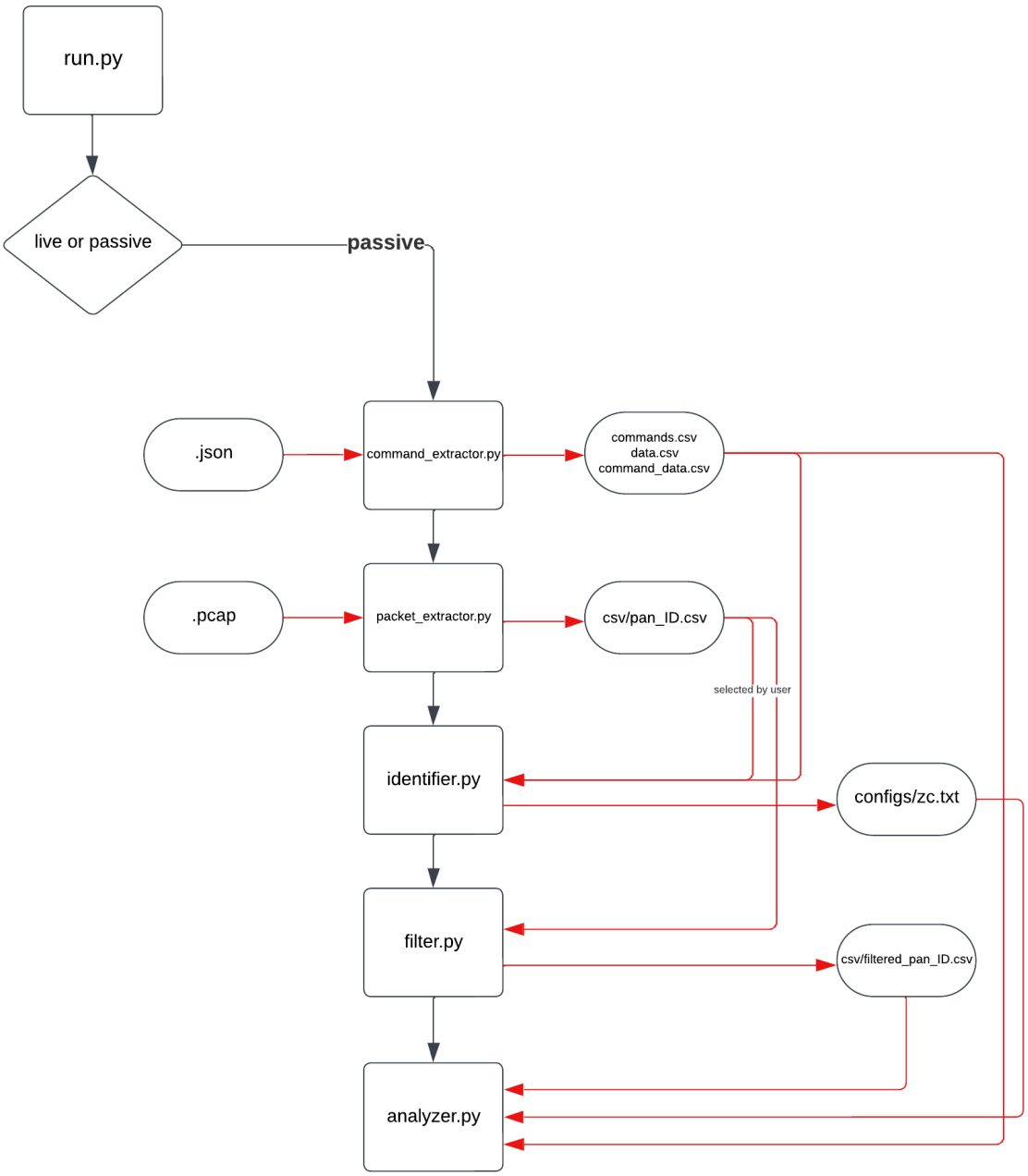


Figure 5.3: Flowchart Passive Mode

## 5.9 Implementation Discussion

In this section, the strategies employed during the implementation phase will be discussed and evaluated. To maintain a smooth narrative flow, each script will be reviewed in the same order as they were described in the sections above. This will allow the thought process and the reasoning behind each decision to be delved into and its effectiveness assessed.

### 5.9.1 Command Extractor

Utilizing two JSON files facilitates a comparative analysis between decrypted and encrypted networks, offering valuable insights into the information still accessible post-decryption. This first step stemmed from the fundamental assumption that possessing a single network key opens up the potential to decrypt any other network. Opting for JSON files provides ease of handling, particularly in exporting a network capture from Wireshark and mapping, given the potential variability in packet attributes such as frame and data length within the network. The code allows for flexibility, ensuring accurate attribute mapping even if packet attributes like frame or data length would fluctuate. However, it's essential to acknowledge the limitation that the evaluation is constrained to only the packet types captured in the JSON files, potentially omitting certain packet types and compromising identification accuracy in the subsequent scripts.

To verify precision in attribute extraction, the script relies on Philips Hue identifiers, ensuring the extraction of data and frame length from the correct packet type. Mapping them by frame number guarantees an accurate association between data lengths and corresponding frame lengths. The Zigbee NWK and Zigbee Cluster Library (ZCL) layers serve as primary sources of decrypted information, facilitating the extraction of relevant data to generate a CSV file containing frame numbers, frame lengths, and command names. The ZCL is a cluster located in the application layer as described in the background 2.1. Importantly, the code remains adaptable and capable of extracting data from various JSON files independently if the user ever wishes to exchange them. However, if the JSON files don't contain all the necessary packet types that serve as reference points for identification, the prototype could malfunction. It is also worth noting that any changes in Philips Hue-specific information would necessitate updating the code, even while such instances are uncommon.

In encrypted networks, only the Zigbee NWK layer is accessible. The process remains the same as when extracting the commands, involving the extraction of frame and data length. Notably, there are no limitations imposed by Philips Hue-specific information, enhancing the versatility of this function.

The output consists of three CSV files, enabling direct inspection which was helpful during the data evaluation process. This transparency is particularly beneficial for new users of the prototype, providing insight into intermediate steps and therefore making it valuable, both in ongoing project development and for future reference. Comparisons with the system of the *Zleaks* paper, which lacked intermediary results when the program did

not work, underscore the value of this approach. Additionally, examining the commands captured in the *command_data.csv* file offers insights into the functionality of the scripts, highlighting any missed critical commands, for example, an *off* command, and guiding further refinement.

The objective of locating event commands was fulfilled as they were present in the CSV file. It's plausible that there may be other combinations of frame and data lengths of the four chosen events not captured, but this is highly improbable given the extensive analysis of millions of packets and prototype testing. Utilizing CSV files and data frames streamlines data manipulation, enhancing efficiency throughout the analysis.

### 5.9.2 Packet Extractor

At first, the script presented itself differently, requiring significant iterative improvement. Numerous iterations were conducted regarding the extraction of data. Initially, time, frame length, data length, source, and destination were clearly defined as the primary focus was on identifying key data to discern devices and events along with their timestamps. It was noted earlier that sniffing commenced during Setup 3, isolated on channel 15, capturing packets solely from one network. However, upon transitioning to other setups, it became evident that capturing packets from various networks necessitated the inclusion of destination PAN. Additionally, frame type and radius were utilized for device identification. Frame numbers were incorporated to facilitate data evaluation, aiding in the comparison and analysis of packets within Wireshark.

Initially, the *pcap_to_csv* function solely extracted data from packets and organized it into CSV files per PAN ID. However, numerous extraneous packets, including those not classified as Zigbee packets, were extracted. To streamline analysis, the pyshark filter 5.1 was applied to eliminate duplicates. Attempts were made to employ sequence numbers for duplicate removal but encountered complications due to the presence of two different sequence numbers, one from the Zigbee NWK layer and the other from the ZCL layer. While the ZCL layer sequence number was only present in decrypted network packets, the Zigbee NWK layer sequence number was employed for duplicate removal when the network was encrypted. Additional key data, including source, destination, frame length, and data length, were included to ensure accurate identification of duplicate packets. However, merely relying on this key data proved insufficient due to the sequence number cycling from 0 to 255 before restarting. To address this, a window size of 254 packets was implemented, wherein each packet was compared only to the preceding 254 packets, minimizing duplication. While this approach significantly reduced duplication, the script still exhibits a limitation wherein packets with sequence number shifts, possibly due to millisecond delays, may not be identified as duplicates. The potential impact of this issue on inference rules will be addressed in subsequent discussions.

The *tracking* function was implemented following the decision to incorporate live tracking functionality, necessitating a function to determine whether to utilize live tracking capture or passive capture. However, this function isn't entirely reliable, as it relies on checking for file existence to determine which tracking mode to use. If *capture.pcap* exists (generated from live tracking mode), the script extracts packets from it until the process completes

or is stopped. If the file doesn't exist, the script searches for a *.pcap* file in the PCAP folder and uses the first one it finds. One limitation arises if the corresponding file for the selected tracking mode doesn't exist, or if the user deviates from the instructions in the *README.md*, such as placing multiple files in the PCAP folder, which doesn't affect the program but may cause confusion if expecting an analysis of a different file. The prioritization is given to checking for *capture.pcap* because there might be a tendency for users to forget to remove the PCAP file from the folder for passive tracking, which could result in the unintended utilization of the PCAP file for live tracking. Although the automatic deletion of the PCAP file after each run was considered, it was ultimately opted against, as users may wish to analyze the specific PCAP file multiple times, and difficulties were encountered in locating the deleted file on the MacBook for reintegration into the folder. Alternative methods for selecting the file were experimented with, such as direct user input or global variables, but challenges arose, particularly due to the subprocess module interruptions when importing from different scripts. The utilization of CSV files persisted, primarily due to the significant facilitation of iterative analysis and evaluation during the design stage, especially with frame numbers.

### 5.9.3 Filter

The code of the *filter* was initially hosted within the *packet extractor* with the aim of manipulating all packets in a single script. The concept of the *filter* script emerged during the analysis of *ZHA* packets, where there was a necessity for a clear overview of these packets to analyze their sequencing patterns.

One notable pattern observed, which has been displayed in Figure 5.1, was the *read attribute response* packet consistently following a *read attributes* packet, wherein the source and destination are exchanged. However, these packets were not always perfectly consecutive due to interruptions by other packet types what can be seen in Figure 5.4.

| | | | | | | |
|---|---|---|---|---|---|---|
| 227 | 2024-03-17 15:01:25.946240 | 0x1327 | 0x30c2 | ZigBee HA | 48 | ZCL: Read Attributes, Seq: 246 |
| 228 | 2024-03-17 15:01:25.946784 | | | IEEE 802.15.4 | 3 | Ack |
| 229 | 2024-03-17 15:01:25.957710 | 0x30c2 | 0x1327 | ZigBee | 53 | Route Record, Dst: 0x1327 |
| 230 | 2024-03-17 15:01:25.958254 | | | IEEE 802.15.4 | 3 | Ack |
| 231 | 2024-03-17 15:01:25.988679 | 0x86e9 | 0x0000 | IEEE 802.15.4 | 10 | Data Request |
| 232 | 2024-03-17 15:01:25.989221 | | | IEEE 802.15.4 | 3 | Ack |
| 233 | 2024-03-17 15:01:26.004150 | 0x86e9 | 0x0000 | ZigBee | 51 | Data, Dst: 0x0000, Src: 0x86e9 |
| 234 | 2024-03-17 15:01:26.004692 | | | IEEE 802.15.4 | 3 | Ack |
| 235 | 2024-03-17 15:01:26.023283 | 0x1327 | Broadcast | ZigBee | 49 | Route Request, Dst: 0xf980, Src: 0x1327 |
| 236 | 2024-03-17 15:01:26.025570 | 0x1327 | Broadcast | ZigBee | 49 | Route Request, Dst: 0xf980, Src: 0x1327 |
| 237 | 2024-03-17 15:01:26.041317 | 0x30c2 | 0x1327 | ZigBee HA | 51 | ZCL: Read Attributes Response, Seq: 246 |
| 238 | 2024-03-17 15:01:26.041861 | | | IEEE 802.15.4 | 3 | Ack |
| 239 | 2024-03-17 15:01:26.043978 | 0x1327 | Broadcast | ZigBee | 49 | Route Request, Dst: 0xf980, Src: 0x1327 |
| 240 | 2024-03-17 15:01:26.079213 | 0x1327 | Broadcast | ZigBee | 49 | Route Request, Dst: 0xf980, Src: 0x1327 |
| 241 | 2024-03-17 15:01:26.093892 | 0x1327 | 0x30c2 | ZigBee HA | 48 | ZCL: Read Attributes, Seq: 247 |
| 242 | 2024-03-17 15:01:26.094436 | | | IEEE 802.15.4 | 3 | Ack |
| 243 | 2024-03-17 15:01:26.098353 | 0x1327 | Broadcast | ZigBee | 49 | Route Request, Dst: 0xf980, Src: 0x1327 |
| 244 | 2024-03-17 15:01:26.103930 | 0x30c2 | 0x1327 | ZigBee HA | 51 | ZCL: Read Attributes Response, Seq: 247 |

Figure 5.4: Interrupted Read Attribute Sequence

Therefore, the *filter* was developed to specifically extract and filter out these packets, resulting in a CSV file containing only filtered *ZHA* commands. This decision was informed by the observation that packets with a data length under 10 were consistently unrelated, following thorough packet analysis.

The filtered CSV file serves the purpose of event detection but not device identification, as other packet types are also utilized for the latter. Consequently, the *packet extractor*

is responsible for preparing the CSV file for device identification, while the *filter* is tasked with refining the file for event analysis. This separation of tasks led to the creation of the *filter* as a standalone script. Additionally, the *filter* script requires access to the CSV file generated from the *packet extractor*. Embedding the *filter* function within the *packet extractor* script would potentially hinder its ability to access the necessary CSV files, especially when processing multiple PANs simultaneously.

Once again, the output is a CSV file, aiding in the analytical process by facilitating comparison with packet data in Wireshark.

### 5.9.4 Identifier

The initial concept behind the *identifier* was to establish reference points by combining frame and data lengths for specific packet types, enabling the identification of device types. Consequently, the program first generates lists of these reference points, pairing frame and data lengths with their corresponding commands.

The first step involved identifying packet types presenting distinctive characteristics of various device types. Attempts were made to identify the bridge using *read attribute* packets, typically sent from the bridge to end devices. However, this approach proved ineffective as all other devices besides the bridge also send a lot of these packets as forwarding router devices. After further analysis, *route* packets, particularly *route requests* mainly sent by bridges, were considered. Despite this, it wasn't adequate enough to distinguish the bridge. Therefore, multiple conditions were employed to narrow down the options, with one idea being to check if the source of a *route request* packet matches the destination of a *read attributes response*, demonstrated in Figure 5.5.


Figure 5.5: Coordinator Detection Condition

An alternative approach could have been to use *read attributes* packets for bridge identification but the decision was made not to utilize them. This decision stemmed from the fact, as previously explained, that when encrypted, it appears identical to a *unicast off* packet. This similarity would have potentially disrupted the event analysis process, prompting it to exclude it from consideration.

Unfortunately, the *read attribute response* packet also shares similar packet types, particularly the *color control* packet. However, distinguishing between these packets is aided by their message types: *unicast* and *broadcast*. This easier distinction was the reason for choosing for the *read attribute response* packet over *read attribute* packets. However, even with these refinements, it wasn't sufficient to determine the bridge once and for all. Hence, the concept of multiple possible device type lists and additional packet types for reference points was introduced. This involved creating two lists for potential coordinators and two for potential end devices, along with the inclusion of the packet-type *route record*. Consequently, the program iterates through packets, identifying *route record* packets and

adding their destinations to the first potential coordinator list and sources to the first potential end device list. Similarly, for *route requests*, only the source is added to the second potential coordinator list, and the same applies to *read attribute responses*, where the source is added to the second potential end device list.

This approach proved effective, as the intersection of these lists largely facilitated accurate device identification. However, during implementation, it was discovered that other devices besides the bridge also send *route record* and *request* packets. To address this, the script was modified to identify the device that sends most of these packets, which consistently led to the correct identification of the coordinator, as confirmed through testing.

The routers were easier to identify, as the program directly checks a *link status* packet sent by routers. The only issue arises if the capture fails to recognize a *link status* packet for every router device, in which case it will not identify the device as a router. However, as explained in the background chapter 2, every Philips Hue light device has an integrated chip allowing them to act as routers as well. Therefore, even if the *identifier* does not recognize all routers, one can deduce that any device in the ZED list is also a router if the identification of ZEDs is accurate.

Since the Philips Hue bridge also functions as a router, it might be mistakenly identified as an end device because it forwards *read attribute* packets. Thus, it needs to be removed from the list of end devices at the end of the process.

Due to the subprocess module interrupting the order of the scripts, the coordinator was not stored in a global variable and imported into any necessary script. Instead, it is stored in the *zc.txt* file. Although this approach requires a few more lines of code, it proved helpful for consistently verifying if the correct device was identified as the bridge. Additionally, it can provide useful insights for new users. The outcomes of testing the identifier will be delved into at a later time.

### 5.9.5 Analyzer

As outlined in the implementation 5, the *analyzer* operates similarly to the *identifier*, focusing on events involving commands such as *on, off, color*, and *level control*. Rather than solely depicting the code's functionality, the comprehensive evolution of the script and the decision-making process at this important juncture of the project is delved into. First, the script starts with a list of reference points comprising command, frame, and data length tuples. The original plan was to detect additional events, including *on/off* in both *unicast* and *broadcast, color* and *level control in uni-* and *broadcast*, and potentially the *recall scene* event. Subsequently, the reasoning behind limiting the testing to *on/off* in *broadcast* messages, and *color* and *level control* in *unicast* messages is explained. Opting for *broadcast* packets for *on/off* commands is accompanied by the Figure 5.6 below. Any newly added light must be associated with a room within the network. During the initial phase of sniffing, the predominance of *broadcast* commands directed towards lights, mixed with occasional instances of *unicast* commands, was observed.

It wasn't until later that the pattern was grasped, with the understanding that the choice between *broadcast* and *unicast* often depended on whether the command was made by activating an entire room or specifically targeting an individual light device. Commands were routinely verified by isolating individual lamp activation, ensuring that the corresponding *unicast* command was categorically applied to the targeted light. As data analysis progressed and the *analyzer* script development commenced, it became clear that a *unicast off* command shared identical frame and data lengths with a *read attributes command*, and similarly matched in direction, making a decrypted *unicast off* packet indistinguishable from a *read attributes* packet. Consequently, upon scrutinizing the order of events, it became apparent that a *read attributes response* must follow the *read attributes packet*, which does not occur with the *off* packet. Therefore, an inference rule was devised based on these findings to eliminate false event identifications. Given the abundance of packets, a small code to identify packets with identical frame and data lengths was designed, attempting to differentiate them by analyzing what sets them apart. A small extract of the result from the code is shown in Table 5.1.

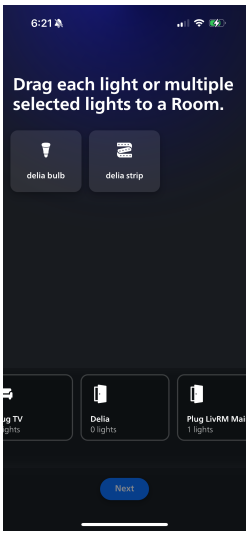


Figure 5.6: Adding New Lights

| Frame Length | Info | Data Length | Command |
|---|---|---|---|
| 48 | ZCL: Read Attributes | 13 | Off |
| 48 | ZCL Scenes: Get Scene Membership | 13 | Off |
| 51 | ZCL: Read Attributes Response | 16 | Level Control Broadcast |
| 51 | ZCL Scenes: Get Scene Membership Response | 16 | Read Attributes Response |
| 67 | ZCL Scenes: Get Scene Membership Response | 32 | Read Attributes Response |
| 63 | ZCL Scenes: Get Scene Membership Response | 28 | Read Attributes Response |
| 52 | ZCL: Read Attributes | 17 | Read Attributes Response |
| 52 | ZCL Groups: Get Group Membership Response | 17 | Read Attributes Response |
| 47 | ZCL Groups: Get Group Membership | 12 | On Broadcast |
| 47 | ZCL OnOff: On | 12 | On Broadcast |

Table 5.1: Packets with identical Frame and Data Lengths Comparison

New rules and patterns were attempted to be identified to distinguish between the *off* packet and the *read attributes* packet. This involved examining the sequence of packet types that typically precede or follow each type and their directional flow.

```
elif event[0] == 'off': # row['Length'] == 48 and row['Data length'] =
    13
     src = row['Source']
     dst = row['Destination']
     seqno = row['Sequence number']
     prev_row = df.iloc[i - 1] if i > 0 else None
     next_row = df.iloc[i + 1] if i < len(df) - 1 else None
     next_row2 = df.iloc[i + 2] if i < len(df) - 2 else None
     if prev_row is not None and prev_row['Length'] == 50 and prev_row['
        Data length'] == 15 and prev_row['Source'] == dst and prev_row['
        Destination'] == src and prev_row['Sequence number'] != seqno:
         break
     elif prev_row is not None and prev_row['Length'] == 51 and prev_row[
        'Data length'] == 16 and prev_row['Source'] == dst and prev_row['
        Destination'] == src:
         break
     elif next_row is not None and (next_row['Length'] == 48 and next_row
        ['Data length'] == 13 and next_row['Source'] == src and next_row[
        'Destination'] == dst) and (next_row2['Length'] == 51 and
        next_row2['Data length'] == 16 and next_row2['Source'] == dst and
         next_row2['Destination'] == src):
         break
     elif next_row is not None and next_row['Length'] == 51 and next_row[
        'Data length'] == 16 and next_row['Source'] == dst and next_row['
        Destination'] == src:
         break
     elif next_row is not None and next_row['Length'] == 60 and next_row[
        'Data length'] == 25 and next_row['Source'] == src and next_row['
        Destination'] == dst:
         break
     elif next_row is not None and next_row['Length'] == 60 and next_row[
        'Data length'] == 25 and next_row['Source'] == dst and next_row['
        Destination'] == src:
         break
     elif next_row is not None and next_row['Length'] == 66 and next_row[
        'Data length'] == 31 and next_row['Source'] == dst and next_row['
        Destination'] == src:
         break
     elif next_row is not None and next_row['Length'] == 67 and next_row[
        'Data length'] == 32 and next_row['Source'] == src and next_row['
        Destination'] == dst:
         break
     else:
         print('User turned light off at ' + row['Time'] + dst)
```

Listing 5.5: Attempted Inference Rule for *Off* Packets

The approach, displayed in Listing 5.5, was refined until encountering packets like the *ZCL Scenes: Get Scene Membership* packet, which posed a challenge as they appeared indistinguishable. At this juncture, a pause was made to reassess the frequency of *unicast off* packets. Recalling the initial confusion about *unicast* message transmission when sniffing began, it was realized that from a user's standpoint, it's often more efficient to

control an entire room's lighting, be it through a switch, sensor, or app, rather than individually operating each lamp. Given that these actions are *broadcast* commands, the focus of the analysis was set on those events. In Setup 1, an actual home environment, one could observe a significantly higher frequency of these *broadcast* commands compared to *unicast on/off* packets.

Figures 5.7 and 5.8 are illustrations in the app of the preference for turning on the entire room rather than targeting specific lamps within it. Also sensors and switches are linked to a room which makes them transmitt *broadcast* messages when used.

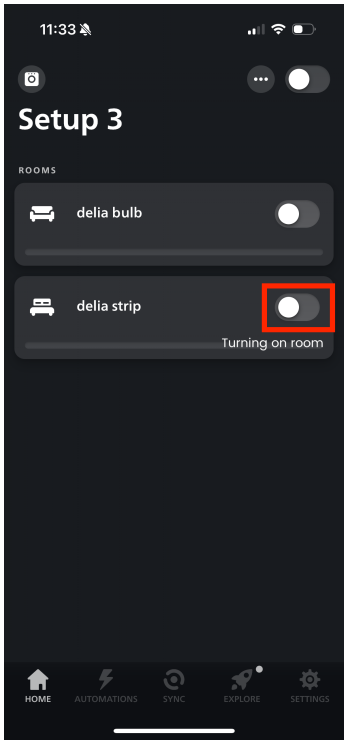


Figure 5.7: Room ON

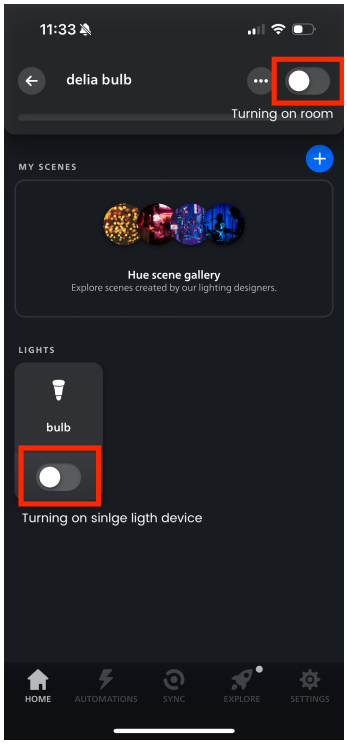


Figure 5.8: Light ON

The decision on *level* and *color control* in *unicast* form is also related to the process in the Philips Hue app. Color or saturation adjustment is needed to be accessed through the specific light device. No other method has been discovered during the thesis for issuing these commands in *broadcast* form, which is why they were decided against. The *recall scene* was not selected because the aim was to concentrate on the four core commands that define the essence of Philips Hue and contribute to its popularity. However, exploring *recall scenes* could be considered a subsequent step in this project's evolution. Upon determining the events for analysis, after reviewing the overlay CSV, it was noticed that other packets shared similar identification features. Most were distinguishable by analyzing their direction, origin (coordinator or *broadcast*), or packet type. Only the *unicast color control* packet revealed a similar conflict as the *unicast off* packet; other packet types appeared identical upon decryption. To address this, an inference rule was devised leveraging packet order, sequence patterns, and direction, eliminating most misidentifications by cross-referencing the source with the Philips Hue bridge's network address identified in the *identifier*.

However, one type of misidentification remained unresolved: instances where the packet sequence was disrupted by skipped packets. As mentioned earlier, skipped packets lack a *response*, creating the illusion of an event occurrence. Coding the inference rules posed a challenge as Wireshark displays sequence numbers from the ZCL layer 5.9 and matches *responses* to originating packets, providing a logical basis to identify non-events.

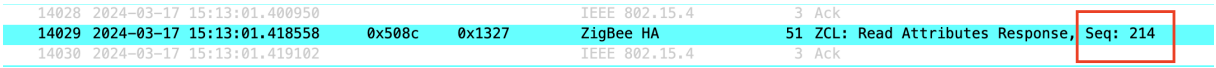


Figure 5.9: ZCL Sequence Number

Unfortunately, this approach is not feasible since the decrypted network only displays the sequence number of the NWK layer and no corresponding *responses,* as depicted in Figure 5.10. The sequence number of the NWK layer was attempted to be utilized. Nevertheless, it was discovered that these numbers are set individually by each device's packet sequence, rather than across all packets.

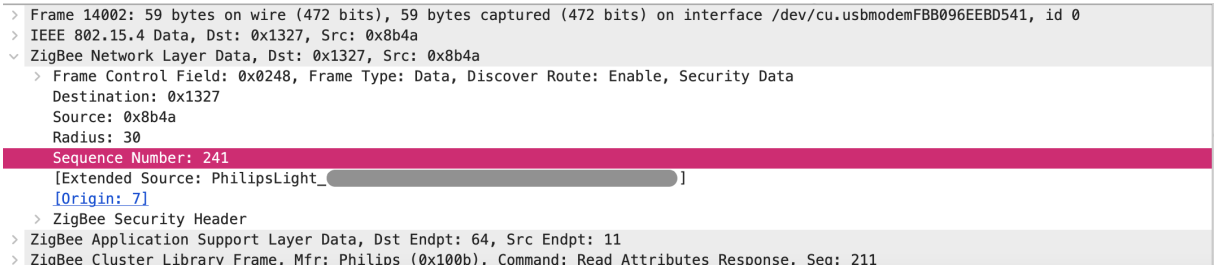


Figure 5.10: NWK Sequence Number

The task of identifying packets belonging together is significantly complicated by this observation. In an effort to resolve this, experiments were conducted to calculate the differences between each device and the bridge to determine which *response* packet corresponds to which device. However, even in this approach, sequence numbers were found to be skipped intermittently or switched in packet order. Again also the cyclic identity would make the calculation process a lot harder. Figures 5.11 and 5.12 illustrate the different ways of how the sequence numbers are setup in the different layers.

Thus, the script sequentially examines subsequent packets and compares them to specific frame and data lengths that are expected. Utilizing ZCL sequence numbers would have provided a more dependable method, as the current approach relies on analyzing millions of packets, potentially overlooking certain patterns.

Identified events are then promptly printed as output along with the respective network address to ensure they pertain to the correct devices.

Testing outcomes of the *analyzer*, will be elaborated in the subsequent chapter.

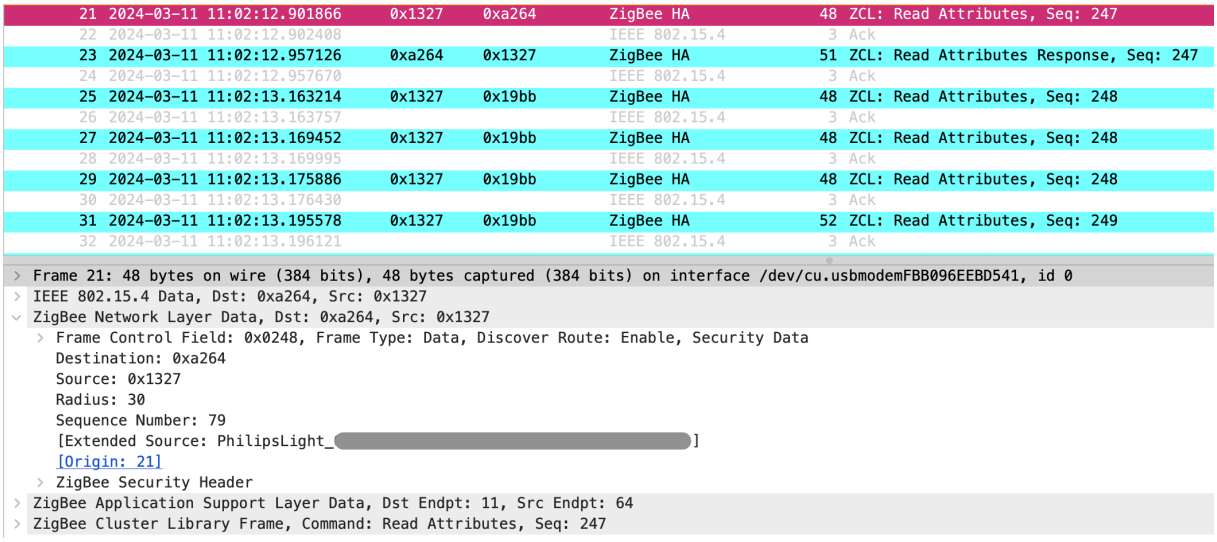


Figure 5.11: Packet with NWK Sequence Number 79 and ZCL 247

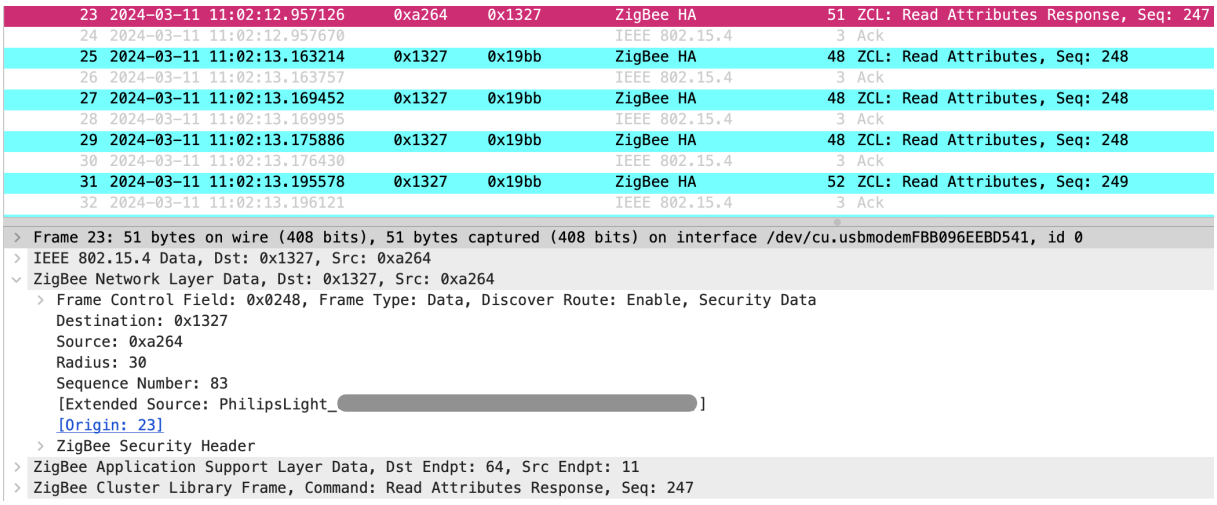


Figure 5.12: Packet with NWK Sequence Number 83 and ZCL 247

### 5.9.6 Tracker

The *tracker* seamlessly manages multiple threads with the subprocess module: one for sniffing and capturing packets, and another for analyzing them in the background. However, a limitation arises when importing between files: the subprocess module disrupts the flow by running imported scripts. For instance, importing the *coordinator*'s network address from the *identifier* into the *analyzer* script causes interruptions, leading to the *identifier* running in a loop despite it having been processed earlier.

### 5.9.7 Runner

Before deciding to utilize the script as both a live and passive tracker, it was only employed for organizing scripts and executing commands to determine which script should be active.

Following this decision, modifications were necessary to ask the user for the desired mode of operation and execute scripts accordingly. Opting for live mode initiates a 90-second sniffing process before tracking, as packet captures may encompass multiple networks. Given the challenge of identifying the correct network's PAN ID and corresponding CSV file for analysis, a brief sniffing and device identification process was introduced to present the user with network details, allowing for estimation and confirmation of their network. If incorrect, the user can stop the program and retry. Alternatively, running the *identifier* for each PAN CSV file was considered, but this approach proved overly complex and could affect program functionality. A 90-second duration was chosen to balance user wait time and the efficiency of packet capture for identification, as shorter durations sometimes yielded insufficient data. Although a 90-second wait carries some risk, exceeding this timeframe could be regarded as troublesome for some. Passive mode operates similarly but skips the 90-second sniffing period. Subsequently, the script executes the appropriate subprocess commands before concluding by clearing all utilized files or their contents.

# Chapter 6

# Results and Evaluation

In the next chapter, testing outcomes will be discussed before an overall evaluation of the prototype is given. The chapter also explores the challenges that came up and the vulnerabilities of the Philips Hue network. Lastly, it provides a small reflective framework.

## 6.1 Testing and Outcomes

This section will delve into the testing phase, examining the outcomes and evaluating the results obtained. This evaluation clarify the strategies employed, highlighting successes and areas for improvement.

### 6.1.1 Setup

Testing was conducted across the 3 setups 4.1.1 to evaluate the assumption of decrypting different networks with one specific network key. These setups varied in size, Zigbee channel, and network configuration as explained in the design chapter 4. The testing schedule 6.1 included the specific actions below with 3-minute intervals:

| | |
|---|---|
| Minute 0 | Start |
| Minute 3 | Room on |
| Minute 6 | Dim light device 1 |
| Minute 9 | Dim light device 2 |
| Minute 12 | Color light device 1 |
| Minute 15 | Color light device 2 |
| Minute 18 | Room off |
| Minute 21 | Stop |

Table 6.1: Testing Schedule

During analysis, commands were issued to two devices per setup. Events from devices outside this scope were deemed undefined outcomes due to the challenge of tracking devices across different floors and rooms, as well as potential interference from family members.

The *identifier* was tested with a total of 27 devices, with varying numbers from each Setup. 3 from Setup 3, 7 from Setup 2, and 17 from Setup 1. Any other devices that were detected in the house, like sensors and switches, were excluded from consideration.

Each setup was tested live twice and passively twice hence 2 rounds of testing per setup. Round 1 involved executing commands, while round 2 testing involved only monitoring the network without issuing commands to see if the prototype would identify commands that were not given.

Additionally, the *identifier* was tested live three times more per setup due to its higher fragility; sniffing for 90 seconds yields a smaller packet amount compared to the 21-minute testing session hence less data to identify the devices from.

Testing was conducted under optimal conditions. Meaning if a command failed to produce a physical response, the test was restarted to maintain the integrity of the schedule. Efforts were made to ensure all lights were reachable, though occasional disruptions could occur during sniffing.

### 6.1.2 Device Identification Results

| Round | Live/Passive | Coordinators | End Devices | Routers |
|---|---|---|---|---|
| 1 | Live | 3 | 23 | 26 |
| 2 | Live | 3 | 22 | 25 |
| 3 | Live | 3 | 21 | 21 |
| 4 | Live | 3 | 23 | 25 |
| 5 | Live | 3 | 22 | 26 |
| 1 | Passive | 3 | 24 | 25 |
| 2 | Passive | 3 | 24 | 26 |

Table 6.2: Device Detection Outcomes

The detection in the ideal case of all coordinators indicates that the approach of identifying coordinators is very reliable, achieving a 100% success rate. Additionally, the detection of 159 out of 168 ZED devices, approximately 95%, showcases a robust performance. However, it's important to note that this success rate may be affected by the occasional unreachability of light devices during testing, which could result in the non-transmission of packets. Thus, while the prototype appears promising, definitive conclusions cannot be drawn about how reliable the code is.

Similarly, the detection of 174 out of 189 ZR devices, around 92%, demonstrates good results. However, there are still questions about whether the discrepancies are caused by potential flaws in the prototype or sporadic unreachability of light devices during testing, similar to the challenges faced with the ZED devices. This question is challenging to

investigate as the unreachability of devices was not investigated because of the difficulty in influencing them.

The weakest results were observed in the detection of routers. This is likely attributed to the requirement in the approach for every device to send a *link status* packet during testing for successful detection. Factors such as intermittent internet connection loss, or device range limitations may render routers temporarily unreachable, impacting detection rates. Although routers had the most straightforward conditions to identify them, they got the weakest results. But again as the coordinator and ZEDs are also routers, they could be deduced from their lists.

In comparing live and passive detection methods, the live mode achieved a detection rate of 249 out of 270, approximately 92%, while the passive mode detected 105 out of 108 devices, around 97%. Although the difference is minimal, passive detection outperformed live detection, possibly due to the longer duration of sniffing, allowing for a larger number of packets to be captured and more data to be analyzed. Consequently, it's advisable to conduct more extensive testing in live mode to further refine device identification. A short overview of the device detection outcomes is presented in Table 6.2.

### 6.1.3 Event Detection Results

| Live/Passive | Round | Correct detected events / non-events |
|---|---|---|
| Live | 1 | 17 |
| Live | 2 | 18 |
| Passive | 1 | 18 |
| Passive | 2 | 18 |

Table 6.3: Event Detection Outcomes

The event detection results show promise, with a detection rate of 97% in round 1, where 35 out of 36 events were detected, and a perfect non-detection rate in round 2, with 36 out of 36 non-events not detected. This leads to an overall accuracy of 98% for both detection and non-detection. There were a few undefined occurrences observed, particularly in Setup 1, which is the largest setup and likely to have more complex interactions. It's uncertain whether these undefined occurrences are false identifications or were induced by household activities.

Regarding the comparison between live and passive modes, there was minimal difference observed. One event in live mode was not detected, while all events were detected in passive mode. The occurrence of this missed event may be attributed to the brief pause that occurs after capturing 255 packets in live mode, during which the system writes the file and resumes capturing. Commands issued during these milliseconds could potentially be overlooked also presenting a quite big limitation.

Passive mode detected more events that shouldn't have been detected, particularly *color control* events. This may be linked to the identification of packets with similar frames, data lengths, and directions, originating from the bridge. Despite efforts to differentiate

these packets as explained in a section above, no definitive rules were found to entirely distinguish *color control* packets from all other packet types with the same frame and data length combination, likely due to skipped packets. Nevertheless, since the results were not overwhelmed with skipped packets, it's deemed acceptable to tolerate these discrepancies. These factors may also contribute to the occurrence of undefined events. A short summary of the event detection outcomes is displayed in Table 6.3.

## 6.2 Prototype Evaluation

This section aims to give an overall conclusive evaluation of the prototype as a whole based on the evaluation metrics of precision, recall, and accuracy. The testing resulted in 389 True Positives (TP), 3 False Positives (FP), 25 False Negatives (FN) and 36 True Negatives.

### 6.2.1 Evaluation Metrics

The evaluation metrics for the prototype's performance are as follows:

**Recall** focuses on the prototype's ability to find the proportion of true positive predictions among all actual positive instances. The calculated recall value is about 94%, indicating a high proportion of relevant cases being identified.

$$\frac{TP}{TP + FN} = \frac{389}{389 + 25} \approx 0.940$$

**Precision** focuses on the prototype's ability to not label a negative sample as positive. The precision value is around 99%, suggesting that the prototype has a high precision in correctly identifying positive samples.

$$\frac{TP}{TP + FP} = \frac{389}{389 + 3} \approx 0.992$$

**Accuracy** provides an overall assessment of the prototype's correctness, considering both true positive and true negative predictions. The accuracy value is rounded 94%, indicating a high level of correctness in the prototype's predictions.

$$\frac{TP + TN}{TP + FP + FN + TN} = \frac{389 + 36}{389 + 3 + 25 + 36} \approx 0.938$$

### 6.2.2 Evaluation Discussion

Overall, the metrics demonstrate good performance of the prototype for live as well as passive tracking. However, its reliability is contingent upon certain conditions being met. These conditions include lights being mostly reachable, proper functioning of Philips Hue commands, and sufficient packets being captured. While the prototype was designed to employ a holistic and iterative approach based on packet analysis and inference rules, it still has limitations. For instance, its effectiveness heavily relies on these conditions being fulfilled. It's worth noting that alternative approaches, such as a machine learning approach, would face similar challenges if these conditions are not met.

Additionally, human analysis and trial-and-error methods are incorporated into the prototype's approach, which introduces the possibility of errors or overlooking patterns. Furthermore, errors may also arise from issues with Python and its modules or the nRF board's setup.

Another limitation arises from the interdependence among scripts and their sequential interactions. An illustrative scenario is the *identifier* script, responsible for storing the coordinator in a text file, a prerequisite for the proper functioning of the *analyzer*. If this crucial step is omitted, the *analyzer*'s performance will be compromised. However, this interdependence can also be viewed as a benefit. Despite relying on each other, the scripts are designed to be executable individually as long as all required input files are present. This flexibility is advantageous for the iterative process and further exploration. For instance, having access to the filtered PAN packets CSV file, the *command_data.csv*, and the *zc.txt* file from a previous run enables running the *analyzer* independently. This capability streamlines the iterative improvement of individual scripts, facilitating more efficient development and experimentation.

Even as a prototype, it not only fulfills its primary objectives admirably but also shows significant promise. Its iterative methodology and script structure contribute to its transparency, making it particularly inviting for newcomers and individuals who want to contribute to the project's advancement.

Overall, the prototype demonstrates good performance during testing when all conditions are met. However, it is highly dependent on several factors and requires multiple dependencies to function optimally. Therefore, ongoing attention to these conditions and potential sources of error is important for maintaining the prototype's effectiveness.

### 6.2.3 Challenges

The main challenges encountered during the thesis included initially setting up the nRF board, where the wrong port was used for attempting to sniff. As elaborated in previous chapters, deciding which events should be considered and analyzed was another difficulty. Obtaining the network key for the network on Setup 1 proved challenging due to the large increase in packet communication resulted in missing them in the flood. The difficulty of finding an inference rule for *unicast off* packets, as explained in the evaluation chapter 6, was initially pursued but later discarded. Similarly, identifying an exclusive rule to ignore

misidentified *color control* packets was a challenge that was eventually accepted. Testing the prototype proved problematic as experiments were based on optimal conditions that weren't always achievable. For instance, scheduling the light to turn on after 3 minutes sometimes failed, disrupting the timeline. Additionally, the lights' unreachability in Setup 3, due to channel changes, necessitated multiple attempts and patience to overcome. Minor challenges were also issues with Wireshark and Python where unexpectedly, frame and data length were being stored as floats instead of integers. Despite Wireshark exporting them as integers and the use of the *int()* function as a precaution, the problem persisted, but thus indicating that the program wasn't compatible with floats. Furthermore, issues were encountered with Apple saving PCAP files as *.DS_Store* files, which resulted in the prototype being unable to recognize them as capture-type files and rendered them unanalyzable. Another challenge arose when errors were thrown because the network key was still stored in Wireshark. In such cases, the packets will lack a data length parameter, causing the program to throw an error in the *identifier*. This problem is addressed using an if-else loop, as shown in Listing 5.2. An if-else loop is chosen over a try-catch clause to inform the user about the specific error and to prompt them to remove the network key.

### 6.2.4 Emerging Security and Privacy Concerns

The development and testing of the prototype have demonstrated the feasibility and potential implications of both live and passive tracking within a Philips Hue smart home. The successful implementation of tracking functionalities underscores the importance of understanding the nuanced relationship between network architecture and data transmission particularly in the context of IoT devices.

Through the prototype, it became evident that live and passive tracking of Philips Hue smart lighting systems can indeed be achieved. This capability was elaborated and realized through the relatively straightforward approach, the analysis of network packets related to Philips Hue communication, and the formulation of rules based on their sequences. While the development and testing of the prototype show promising results, they also entail additional consequences. The approach demonstrated that by utilizing an nRF board, Wireshark, and the network key, a substantial data breach could be orchestrated, exposing several security and privacy vulnerabilities inherent in the Philips Hue ecosystem. The implications of live and passive tracking of Philips Hue smart lighting systems extend beyond technical feasibility, delving into profound ethical and privacy concerns within the realm of IoT devices.

One of the main network and security concerns that emerge from this whole project is the confirmed assumption that possessing a single network key might possibly allow the decryption of other Zigbee Philips Hue networks, regardless of their network keys. This arises from the consistent frame and data lengths designated for various packet types, regardless of the network or Zigbee channel. This presents the risk that any Philips Hue user could potentially decrypt another user's network with the necessary setup.

The prototype's analysis also brings attention to other potential network vulnerabilities within the smart home environment. For instance, during the prototype analysis, gaining access to the Philips Hue network key demonstrated how easily anyone could penetrate

the network and decrypt its communications. Furthermore, the prototype's ability to analyze Zigbee packets introduces the risk of data interception, potentially compromising sensitive information transmitted between the smart lighting devices and the bridge. This intercepted data includes device network addresses and even user commands, posing a threat to the confidentiality and integrity of the Philips Hue smart lighting system's communications. The sensitive information and transmitted data could then be shared with third parties without their explicit consent which could lead to unauthorized access or misuse of users' personal information by third-party entities, raising concerns about data privacy and security.

Moreover, exploiting vulnerabilities in the smart lightning's security could grant malicious actors control over the smart lighting system. Examining the decrypted network's JSON file can yield extensive packet details, facilitating the creation of replay attacks and unauthorized network access, as tested out by [36], which opens the door for security threats if lights are manipulated.

The prototype also introduces the possibility of greater malicious activity. In the wrong hands, it could be used to orchestrate harmful actions, such as denial-of-service attacks by flooding the network with commands, worms [35] or disrupting the functionality of other devices on the network. Such malicious activities could compromise the stability and security of the entire smart home ecosystem not only the Philips Hue network.

It is clear that by analyzing events like *on/off*, *color changes*, and *brightness adjustments*, the prototype inadvertently tracks users' activities within their homes. This data can reveal sensitive information about users' habits, routines, and occupancy patterns, violating their privacy and autonomy within their living spaces. This analysis of users' interactions with their smart lighting system enables the potential for behavioral profiling.[40] By observing patterns in users' preferences, habits, and routines, detailed profiles of individual users can be constructed from the information. This information could be exploited for targeted advertising or other purposes without users' consent. Patterns in users' lighting usage inferred by the capture of the prototype may inadvertently reveal sensitive information about users' daily lives, such as their presence at home, sleep patterns, or even moods or activities based on the colors and brightness levels of their lights. The inferred patterns of users' behavior and occupancy within their homes could pose physical risks, such as burglary. If malicious actors gain access to detailed information about users' routines and when they are typically away from home based on their lighting usage patterns, this could facilitate targeted break-ins or unauthorized entries, compromising users' physical security and safety.

While the positive outcomes of prototype testing are evident, there is also a notable downside concerning security and privacy vulnerabilities. Beyond technical security risks, the implementation of the prototype revealed additional risks associated with using the Philips Hue network and the extent of information that could be extracted solely from packet traffic in a smart home environment.

### 6.2.5 Reflective Synthesis

As elaborated in the design chapter 4, inspiration was drawn from two papers. The objective of this section is to establish a reflective framework taking a step back and highlighting small differences that also underscore the research gap, which was investigated in this work. A compact overview displaying the most significant differences is presented in Table 6.4.

The difference to *ZPA* is quite significant. This paper was actually used as inspiration to understand how the network keys work and to explore how easily Philips Hue network keys can be gained, aiding in the analysis of the network in the iterative process. A machine learning approach is employed by them to decrypt the network, thus accounting for the big difference from this approach.

With *Zleaks*, one might perceive big similarities between their work and this thesis, as both employ inference rules to decrypt Zigbee networks. However, the primary and most significant distinction lies in *Zleaks*' broader focus on Zigbee devices overall, including Philips Hue but not exclusively, unlike the sole emphasis on Philips Hue within this thesis. Another notable difference lies in their decryption methodology, which relies on hard-coded data and frame lengths in their code. Conversely, this approach utilizes JSON files to extract these lengths dynamically, ensuring adaptability in case of changes. This decision stemmed from encountering issues when analyzing their hard-coded lengths and their program being ineffective when trying it out. Although it's uncertain whether this was the sole cause of failure, the JSON-based method provides flexibility. One could argue that employing two JSON files introduces a potential drawback, as not all packet types may be included, affecting identification accuracy since they serve as reference points. However, the adaptable approach was deemed preferable despite this, as the issue arises only when JSON files lacking certain packet types are substituted. Moreover, employing an intermediate step of saving packet type combinations alongside frame and data lengths in a CSV file could reduce this concern by highlighting missing packet types, if necessary.

| | ZPA System [38] | Zleaks [29] | Thesis Prototype |
|---|---|---|---|
| Machine Learning | ✓ | × | × |
| Inference Rules | × | ✓ | ✓ |
| Periodic Reporting Patterns | × | ✓ | × |
| Dynamic System | ✓ | × | ✓ |
| Philips Hue | × | ✓ | ✓ |
| Other Manufacturers | ✓ | ✓ | × |
| In-Depth Analysis | × | × | ✓ |
| Broad Analysis | ✓ | ✓ | × |
| Network Key Extraction | ✓ | × | ✓ |
| Privacy Risks | ✓ | ✓ | ✓ |
| Live Tracking | × | × | ✓ |

Table 6.4: Short Comparison of the Three Works

The distinction between this work to these two and other related works is the lack of exploration into decrypting multiple networks with a single network key. Typically, the

focus lies on demonstrating the decryption of an encrypted network and highlighting privacy concerns. However, the potential of using a single network key to decrypt any other network remains largely unexplored which could indicate that any Philips Hue user may already have easier access to decrypting other networks. This also underscores the research gap.

# Chapter 7

# Conclusions and Future Work

## 7.1 Conclusions

Most objectives of the thesis were successfully accomplished. The foundation for a comprehensive analysis of the Philips Hue network was laid through an extensive background chapter 2, providing essential theoretical insights into the network setup. Furthermore, obtaining the network key facilitated easier packet analysis, enabling multiple iterations of data collection and evaluation as outlined in the design chapter 4. This approach enabled the successful development of a real-time scanning and sniffing application, leveraging insights obtained from Philips Hue network packets. Through iterative refinement detailed in the design chapter 4, the application's scripts, and inference rules were systematically improved. An assessment was conducted regarding the integration of Home Assistant, but the integration was not possible due to insufficient depth of network information regarding the network layers and packets, which were not obtained as desired. Furthermore, the prototype underwent successful evaluation during the testing phase, yielding positive results from the evaluation process.

Key findings of the thesis include the development of an effective real-time scanning and sniffer prototype capable of capturing and analyzing Zigbee communication packets within a smart home environment, particularly those associated with smart light bulbs. The application successfully intercepted and analyzed packets from the Philips Hue smart home system, extracting valuable insights and information. Additionally, a significant discovery is the validation of the assumption that having a network key enables the decryption of any other Philips Hue network. Moreover, it demonstrated the ability to discern device types and identify various events occurring within a Philips Hue Zigbee network. The thesis also uncovered potential security and privacy risks, for example regarding user profiling, inherent in Zigbee communication within the Philips Hue network. These findings not only align with the goals of the thesis but also address the research questions posed.

In relation to research question 1 the thesis explored the efficiency of a sniffer application in capturing and analyzing Zigbee communication packets within a smart home, particularly focusing on interactions with smart light bulbs. Packet capture proved quite effective, facilitated by the prototype's integration with tools like Wireshark, Tshark, and the

nRF board. Additionally, specifying capture parameters enhanced the process's efficiency. However, analyzing the data presented challenges, including latency issues, perhaps sub-optimal speed, and occasional packet loss during live monitoring, potentially impacting effectiveness. Insights gained from packet content included critical information such as frame and data length, source and destination addresses, and pan ID, contributing to tracking user habits and preferences. Although more information could be gained, these insights suffice for constructing user-profiles and identifying device events. This shows what insights into the packet content could be gained contributing to the tracking of user habits and preferences and answering research question 2. Regarding research question 3 about security vulnerabilities, the prototype showed that significant risks such as network vulnerabilities for example data interception which can potentially expose sensitive information could exist. Additionally, the thesis underscored the potential for malicious actors to exploit captured data, not only to gain unauthorized access to lighting systems but also to infer user habits, raising concerns about privacy and physical security, including burglary risks.

There were small modifications during the execution of the thesis. The initial focus on real-time tracking of BLE devices within a smart home gradually shifted towards a more specific target: Philips Hue smart lighting systems as a real smart home presented as the perfect experimental setup for this. This narrowed scope allowed for deeper exploration, particularly highlighting the advantages of Zigbee communication over BLE within the Philips Hue ecosystem. Despite the shift, the core objective remained centered around real-time tracking within smart homes, though with clearer boundaries and a more refined approach.

As for timeline modifications during execution, there were minimal changes. The process integrated the implementation of the prototype seamlessly within a continuous loop of data collection and evaluation instead of its own process. This iterative approach ensured that the design and implementation phases were closely connected, facilitating a dynamic flow where each stage informed the other. Consequently, the overall structure and goals of the project remained largely unchanged, with emphasis on adaptability and refinement throughout the execution process.

## 7.2 Future Work

Looking into the future, the next phase of development could evolve the existing prototype into a fully functional application equipped with both a user-friendly interface (UI) and a robust backend infrastructure. Central to this could also be the optimization of performance to ensure seamless operation without interruptions, such as the minor pause observed after capturing 255 packets and improved analysis speed.

Expanding the scope of command recognition may enhance tracking functionalities. This includes identifying additional commands, such as *scene recall*, and extending the prototype to encompass all *unicast* and *broadcast* commands. Furthermore, advanced functionalities like new event detections, such as Philips Hue automation, and incorporating habit tracking mechanisms to recognize usage patterns could enrich the prototype's capabilities

and give more insight into how much information can be extracted from packet traffic. Comparing the amount of data obtainable through BLE packets versus Zigbee packets could also offer valuable insights. Examining whether simply switching communication protocols could serve as a security measure or potentially decrease security and increase privacy concerns would be particularly intriguing.

Recognizing the dynamic nature of smart home ecosystems, efforts could be directed towards integrating devices from diverse brands beyond Philips Hue devices or Zigbee compatibility which may lead to monitoring of the entire smart home environment similar to *Zleaks* or Home Assistant. Incorporating support for additional messaging protocols, such as BLE, alongside Zigbee, may further enhance interoperability and device compatibility. Mapping BLE data to Zigbee standards might enable seamless integration of BLE devices into the ecosystem, unlocking new possibilities for device interaction and data inference.

However, if there may be a next development process, ethical considerations should remain paramount.

# Bibliography


[1] A. A. Zaidan, B. B. Zaidan, M. Y. Qahtan, O. S. Albahri, A. S. Albahri, M. Alaa, F. M. Jumaah, M. Talal, K. L. Tan, W. L. Shir, and C. K. Lim, "A survey on communication components for IoT-based technologies in smart homes," *Telecommunication Systems*, Vol. 69, No. 1, pp. 1–25, Sep. 2018. [Online]: https://doi.org/10.1007/s11235-018-0430-8

[2] A. K. Ray and A. Bagwari, "Study of smart home communication protocol's and security & privacy aspects," *2017 7th International Conference on Communication Systems and Network Technologies (CSNT)*. Nagpur, IEEE, Nov. 2017, pp. 240–245. [Online]: https://ieeexplore.ieee.org/document/8418545/

[3] "AirTag - Apple," [Accessed: 2024-04-07]. [Online]: https://www.apple.com/airtag/

[4] N. Shafqat, N. Gerzon, M. V. Nortwick, V. Sun, A. Mislove, and A. Ranganathan, "Track You: A Deep Dive into Safety Alerts for Apple AirTags," *Proceedings on Privacy Enhancing Technologies*, 2023. [Online]: https://petsymposium.org/popets/2023/popets-2023-0102.php

[5] O. Ayan and B. Turkay, "IoT-Based Energy Efficiency in Smart Homes by Smart Lighting Solutions," *2020 21st International Symposium on Electrical Apparatus & Technologies (SIELA)*, Jun. 2020, pp. 1–5. [Online]: https://ieeexplore.ieee.org/abstract/document/9167065

[6] "Zigbee 3.0 support in Hue ecosystem," [Accessed: 2024-04-07]. [Online]: https://developers.meethue.com/zigbee-3-0-support-in-hue-ecosystem/

[7] "Smart Lighting - Philips Hue," [Accessed: 2023-12-04]. [Online]: https://www.philips-hue.com/en-us

[8] V. Rudresh, "ZigBee Security: Basics (Part 1)," Nov. 2017, [Accessed: 2023-11-22]. [Online]: https://research.kudelskisecurity.com/2017/11/01/zigbee-security-basics-part-1/

[9] Dattatray, "IoT Security - Part 5 (ZigBee Protocol - 101)," Jun. 2020, [Accessed: 2023-11-19]. [Online]: https://payatu.com/wp-content/uploads/2022/12/c5.pdf

[10] T. Zhou, C. Yang, H. Chen, Y. Han, W. Bao, and Q. Cheng, "Performance research on ZigBee wireless sensor network self-organizing network for 220 kV four-circuit transmission lines on the same tower," *Sustainable Energy*

*Technologies and Assessments*, Vol. 53, p. 102302, Oct. 2022. [Online]: https://linkinghub.elsevier.com/retrieve/pii/S221313882200354X

[11] S. Long and F. Miao, "Research on ZigBee wireless communication technology and its application," *2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*. Chengdu, China, IEEE, Dec. 2019, pp. 1830–1834. [Online]: https://ieeexplore.ieee.org/document/8997928/

[12] "ZigBee Specification," 2017, [Accessed: 2023-12-04]. [Online]: https://zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf

[13] A. Li, J. Li, D. Han, Y. Zhang, T. Li, T. Zhu, and Y. Zhang, "PhyAuth: Physical-Layer message authentication for ZigBee networks," *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA, USENIX Association, Aug. 2023, pp. 1–18. [Online]: https://www.usenix.org/conference/usenixsecurity23/presentation/li-ang

[14] S. Farahani, *ZigBee Wireless Networks and Transceivers*, ser. Addison-Wesley Wireless Communications Series. Newnes/Elsevier, 2008. [Online]: https://books.google.ch/books?id=0m6zlAEACAAJ

[15] "Zigbee addressing and packet structure," [Accessed: 2023-11-23]. [Online]: https://learning.oreilly.com/library/view/internet-of-things/9781788470599/4b61c16d-3cf6-4d5e-a4f2-8688779f5d76.xhtml

[16] Mario Neugebauer, "ZigBee IEEE 802.15.4 - Message Structure," May 2021, [Accessed: 2023-11-23]. [Online]: https://www.youtube.com/watch?v=3-PRH-a2bjs

[17] Jianfeng Wang, "Zigbee light link and its applicationss," *IEEE Wireless Communications*, Vol. 20, No. 4, pp. 6–7, Aug. 2013. [Online]: http://ieeexplore.ieee.org/document/6590043/

[18] "Signify," [Accessed: 2023-12-03]. [Online]: https://www.signify.com/global

[19] P. Samadi Khah, "Eventing in the hue system," EngD Thesis, Oct. 2018, PDEng thesis.

[20] S. Hilbolling, H. Berends, F. Deken, and P. Tuertscher, "Sustaining Complement Quality for Digital Product Platforms: A Case Study of the Philips Hue Ecosystem," *Journal of Product Innovation Management*, Vol. 38, No. 1, pp. 21–48, 2021, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/jpim.12555. [Online]: https://onlinelibrary.wiley.com/doi/abs/10.1111/jpim.12555

[21] "So einfach funktioniert Philips Hue," [Accessed: 2023-12-04]. [Online]: https://www.philips-hue.com/de-ch/explore-hue/how-it-works

[22] "Home Assistant," [Accessed: 2023-12-04]. [Online]: https://www.home-assistant.io/

[23] B. K. Akhmetzhanov, O. A. Gazizuly, Z. Nurlan, and N. Zhakiyev, "Integration of a Video Surveillance System Into a Smart Home Using the Home Assistant Platform," *2022 International Conference on Smart Information Systems and*

*Technologies (SIST)*. Nur-Sultan, Kazakhstan, IEEE, Apr. 2022, pp. 1–5. [Online]: https://ieeexplore.ieee.org/document/9945718/

[24] "Core Architecture Home Assistant Developer Docs," May 2021, [Accessed: 2023-11-23]. [Online]: https://developers.home-assistant.io/docs/architecture/core

[25] "Zigbee Home Automation," [Accessed: 2023-11-29]. [Online]: https://www.home-assistant.io/integrations/zha/

[26] "Home Assistant Philips Hue Integration," [Accessed: 2023-11-29]. [Online]: https://www.home-assistant.io/integrations/hue/

[27] K. Cheng, Y. Deng, L. Zhang, X. Cui, J. Chen, and W. Luo, "Research on ZigBee Device Recognition Based on Software Defined Radio," *Journal of Physics: Conference Series*, Vol. 2290, No. 1, p. 012040, Jun. 2022, publisher: IOP Publishing. [Online]: https://dx.doi.org/10.1088/1742-6596/2290/1/012040

[28] A. Boiano, A. E. C. Redondi, and M. Cesana, "IoTScent: Enhancing Forensic Capabilities in Internet of Things Gateways," Oct. 2023, arXiv:2310.03401 [cs]. [Online]: http://arxiv.org/abs/2310.03401

[29] N. Shafqat, D. J. Dubois, D. Choffnes, A. Schulman, D. Bharadia, and A. Ranganathan, "ZLeaks: Passive Inference Attacks on Zigbee Based Smart Homes," *Applied Cryptography and Network Security*, ser. Lecture Notes in Computer Science, G. Ateniese and D. Venturi, Eds. Cham, Springer International Publishing, 2022, pp. 105–125.

[30] T. Gu, Z. Fang, A. Abhishek, and P. Mohapatra, "IoTSpy: Uncovering Human Privacy Leakage in IoT Networks via Mining Wireless Context," *2020 IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications*, Aug. 2020, pp. 1–7, iSSN: 2166-9589. [Online]: https://ieeexplore.ieee.org/abstract/document/9217236?casa_token=rw7A7UO35LUAAAAA:kOsfxeqo0sGje-We8nCITdPyVY26wFStPN3xnlTXh16WbWPJEB6iSusGVhYswMcz6fqo0894k

[31] X. Guo, J. Quan, J. Hou, H. Zhou, X. He, and T. He, "Accurately Identify and Localize Commodity Devices from Encrypted Smart Home Traffic," *2022 18th International Conference on Mobility, Sensing and Networking (MSN)*, Dec. 2022, pp. 663–670. [Online]: https://ieeexplore.ieee.org/abstract/document/10076752

[32] A. A. Allahham and M. A. Rahman, "A smart monitoring system for campus using ZigBee wireless sensor networks," *International Journal of Software Engineering and Computer Systems*, Vol. 4, pp. 1–14, 02 2018. [Online]: https://core.ac.uk/reader/159195000

[33] L. J. A. Jansen, "Assessing smart home security : a Zigbee case study," Jan. 2022, publisher: University of Twente. [Online]: https://essay.utwente.nl/89274/

[34] A. Acar, H. Fereidooni, T. Abera, A. K. Sikder, M. Miettinen, H. Aksu, M. Conti, A.-R. Sadeghi, and S. Uluagac, "Peek-a-boo: i see your smart home activities, even encrypted!" *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, ser. WiSec '20. New York, NY,

USA, Association for Computing Machinery, Jul. 2020, pp. 207–218. [Online]: https://dl.acm.org/doi/10.1145/3395351.3399421

[35] E. Ronen, A. Shamir, A.-O. Weingarten, and C. OFlynn, "IoT Goes Nuclear: Creating a ZigBee Chain Reaction," *2017 IEEE Symposium on Security and Privacy (SP)*. San Jose, CA, USA, IEEE, May 2017, pp. 195–212. [Online]: http://ieeexplore.ieee.org/document/7958578/

[36] M. S. Wara and Q. Yu, "New Replay Attacks on ZigBee Devices for Internet-of-Things (IoT) Applications," *2020 IEEE International Conference on Embedded Software and Systems (ICESS)*. Shanghai, China, IEEE, Dec. 2020, pp. 1–6. [Online]: https://ieeexplore.ieee.org/document/9301593/

[37] M. Thiery, V. Roca, and A. Legout, "Privacy implications of switching ON a light bulb in the IoT world." [Online]: https://inria.hal.science/hal-02196544

[38] R. Li, W. Zhang, L. Wu, Y. Tang, and X. Xie, "ZPA: A Smart Home Privacy Analysis System Based on ZigBee Encrypted Traffic," *Wireless Communications and Mobile Computing*, Vol. 2023, p. e6731783, Jan. 2023, publisher: Hindawi. [Online]: https://www.hindawi.com/journals/wcmc/2023/6731783/

[39] O. Setayeshfar, K. Subramani, X. Yuan, R. Dey, D. Hong, I. K. Kim, and K. H. Lee, "Privacy invasion via smart-home hub in personal area networks," *Pervasive and Mobile Computing*, Vol. 85, p. 101675, Sep. 2022. [Online]: https://www.sciencedirect.com/science/article/pii/S1574119222000955

[40] Y. Wan, K. Xu, F. Wang, and G. Xue, "IoTMosaic: Inferring User Activities from IoT Network Traffic in Smart Homes," *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*. London, United Kingdom, IEEE, May 2022, pp. 370–379. [Online]: https://ieeexplore.ieee.org/document/9796908/

# Abbreviations

| | |
|---|---|
| APS | Application Support Layer |
| APS-DE | Application Support Layer Data Entity |
| APS-ME | Application Support Layer Management Entity |
| APSDE-SAP | Application Support Sub Data Entity - Service Access Point |
| BLE | Bluetooth Low Energy |
| CCA | Clear Channel Assessment |
| CSMA-CA | Carrier Sense Multiple Access with Collision Avoidance |
| ED | Energy Detection |
| FCS | Frame Check Sequence |
| FFD | Full-Functioning Device |
| FN | False Negative |
| FP | False Positive |
| GTS | Guaranteed Time Slot |
| IoT | Internet of Things |
| LR-WPAN | Low Rate Wireless Personal Area Networks |
| LQI | Link Quality Indication |
| MAC | Media Access Control |
| MFR | MAC Footer |
| MHR | MAC header |
| NLDE | Network Layer Data Entity |
| NLME | Network Layer Management Entity |
| NPDU | Network Power Distribution Unit |
| NWK | Network Layer |
| PAN | Personal Area Network |
| PHY | Physical Layer |
| PHR | PHY Header |
| PDU | Power Distribution Unit |
| RFD | Reduced-Function Device |
| SDK | Software Development Kit |
| SHR | Synchronization Header |
| TN | True Negative |
| TP | True Positive |
| UI | User Interface |
| WPAN | Wireless Personal Area Network |
| ZC | Zigbee Coordinator |
| ZCL | Zigbee Cluster Library |

| | |
|---|---|
| ZDP | Zigbee Device Profile |
| ZDO | Zigbee Device Object |
| ZED | Zigbee End Device |
| ZHA | Zigbee Home Automation |
| ZLL | Zigbee Light Link |
| ZR | Zigbee Router |

# List of Figures


2.1 Zigbee Protocol Stack . . . . . 5

2.2 Detailed Protocol Stack [12] . . . . . 7

2.3 Network Topology Types [13] . . . . . 8

2.4 Packet Structure [14] . . . . . 9

2.5 MAC Frame [16] . . . . . 10

2.6 ZLL Protocol Stack [17] . . . . . 11

2.7 Philips Hue Setup [20] . . . . . 11

2.8 Home Assistant Core Architecture . . . . . 13

2.9 Integration Architecture [26] . . . . . 13

4.1 Design Phases . . . . . 17

4.2 Setup 1 Topology . . . . . 18

4.3 Setup 2 Topology . . . . . 19

4.4 Setup 3 Topology . . . . . 19

4.5 Transport Key Packet to gain Network Key . . . . . 20

5.1 Read Attribute Packet Sequence . . . . . 29

5.2 Flowchart Live Mode . . . . . 32

5.3 Flowchart Passive Mode . . . . . 33

5.4 Interrupted Read Attribute Sequence . . . . . 36

5.5 Coordinator Detection Condition . . . . . 37

5.6 Adding New Lights . . . . . 39

5.7 Room ON . . . . . 41

5.8 Light ON . . . . . 41

5.9 ZCL Sequence Number . . . . . 42

5.10 NWK Sequence Number . . . . . 42

5.11 Packet with NWK Sequence Number 79 and ZCL 247 . . . . . 43

5.12 Packet with NWK Sequence Number 83 and ZCL 247 . . . . . 43

A.1 Device Detection Summary . . . . . 74

A.2 Event Detection Round 1 Summary . . . . . 75

A.3 Event Detection Round 2 Summary . . . . . 76

# List of Tables

5.1 Packets with identical Frame and Data Lengths Comparison . . . . . 39

6.1 Testing Schedule . . . . . 45

6.2 Device Detection Outcomes . . . . . 46

6.3 Event Detection Outcomes . . . . . 47

6.4 Short Comparison of the Three Works . . . . . 52

# Listings

5.1 Pyshark Display Filter . . . . . 27
5.2 Identifier Primary Function . . . . . 28
5.3 Check *On* Packet . . . . . 29
5.4 Inference Rule for *Color Control* Packet . . . . . 29
5.5 Attempted Inference Rule for *Off* Packets . . . . . 40

# Appendix A

# Contents of the Repository

The code repository contains the following content:

## A.1 README

**Philips Hue Zigbee Network Sniffer** This project aims to analyze Philips Hue Zigbee networks. It explores the assumption that obtaining one network key can potentially decrypt any other Hue network. The sniffer identifies devices, their types, and events. These events include on/off and level/color control commands. The identifier leverages different packet types and their directions, considering source and destination information. The analyzer relies on designated frame and data length, as well as packet sequences, to analyze events.

**Prerequisites**

1. *Python 3.x*: Ensure Python is installed on your system. Otherwise download and install Python from the official website: Python.org.
2. *Wireshark*: Install Wireshark from Wireshark.org, which includes `Tshark`. If you use a package manager, `Tshark` will be installed automatically along with Wireshark.
3. *IDE* of your choice. It is beneficial to use an IDE to run the program as it is easier to set up, manipulate and run the scripts all in one.

**Getting Started**

1. Configure nRF Board: Configure your nRF board for Zigbee communication. This project used the Nordic Semiconductor nRF52840 Development Kit. All documentation can be found on Nordic Semiconductor Infocenter.
    - Flash firmware on the nRF board.

- Set up Wireshark for Zigbee. Download and follow the User Guide PDF.

2. Check Interface Number:
   - Connect the nRF board to your device where Wireshark/Tshark is running on.
   - Determine the interface number associated by using the command `{path_to_tshark} -D` in a command line interface.
3. Configure Tshark Command File:
   - Add the path to `Tshark` to the *first* line of the `tshark_command.txt` text file.
   - Add the interface number to the *second* line in `tshark_command.txt`.
     Example:

```
1 /Applications/Wireshark.app/Contents/MacOS/tshark
2 20
```

4. Add directory path: Add your path to the project in the tracker.py `os.chdir('{path_to_project}')`
5. Create the CSV folder When first time using the program, create a CSV folder in the root directory so that the CSV files can be stored there.
6. (Change zigbee channel):
   - If your hue bridge is on another Zigbee channel than the default (11), change the channel on Wireshark's UI.
   - Go to capture then options and change to the preferred channel through the settings icon next to your sniffer interface.

**Usage**

Execute the run script to start capturing and analyzing Zigbee network traffic. Make sure no network key is saved in Wireshark. The program only works for encrypted networks.

1. Live: Choose live mode when asked.
2. Passive:
   - Create a `PCAP` folder and import the pcap/pcapng file that you want to analyze into that folder. (Note: Only keep one file in the folder as the program will only analyze one.)
   - Choose passive mode when asked.

**Disclaimer**

This project is for educational and research purposes only. Ensure you have proper authorization before analyzing any Zigbee networks.

## A.2 Python Scripts

https://github.com/nyc6/BA-Homescout/tree/main

## A.3 JSON Files

https://github.com/nyc6/BA-Homescout/tree/main/data

## A.4 Testing

### A.4.1 Raw Live

https://github.com/nyc6/BA-Homescout/blob/main/raw

### A.4.2 Raw Passive

https://github.com/nyc6/BA-Homescout/tree/main/raw

https://github.com/nyc6/BA-Homescout/blob/main/raw

### A.4.3 Summary Tables of Devices and Events

| | Round 1 | Round 2 | Round 3 | Round 4 | Round 5 | | Round 1 | Round 2 |
|---|---|---|---|---|---|---|---|---|
| | Live | Live | Live | Live | Live | | Passive | Passive |
| | | | | | | | | |
| **Network address ZC / ZR** | **ZC / ZR** | **ZC / ZR** | **ZC / ZR** | **ZC / ZR** | **ZC / ZR** | | **ZC / ZR** | **ZC / ZR** |
| 0x1327 | TP / TP | TP / TP | TP / TP | TP / TP | TP / TP | | TP / TP | TP / TP |
| 0x2bc7 | TP / TP | TP / TP | TP / TP | TP / TP | TP / TP | | TP / TP | TP / TP |
| 0x0001 | TP / TP | TP / TP | TP / TP | TP / TP | TP / TP | | TP / TP | TP / TP |
| | | | | | | | | |
| **Network address ZED/ ZR** | **ZED / ZR** | **ZED / ZR** | **ZED / ZR** | **ZED / ZR** | **ZED / ZR** | | **ZED / ZR** | **ZED / ZR** |
| 0x508c | TP / TP | TP / TP | TP / TP | TP / TP | TP / TP | | TP / TP | TP / TP |
| 0x18ef | TP / TP | TP / TP | TP / TP | TP / TP | TP / TP | | TP / TP | TP / TP |
| 0x6897 | TP / TP | TP / TP | TP / TP | TP / TP | TP / TP | | TP / TP | TP / TP |
| 0x2a10 | TP / TP | TP / TP | TP / TP | TP / TP | TP / TP | | TP / TP | TP / TP |
| 0x3270 | TP / TP | TP / TP | FN / FN | TP / TP | TP / TP | | TP / TP | TP / TP |
| 0xc0cb | TP / TP | FN / TP | TP / TP | TP / TP | TP / TP | | TP / TP | TP / TP |
| 0x67c3 | TP / TP | TP / FN | TP / TP | TP / TP | TP / TP | | TP / FN | TP / TP |
| 0x52a7 | TP / TP | TP / TP | TP / FN | TP / TP | TP / TP | | TP / TP | TP / TP |
| 0xdcab | TP / TP | TP / TP | TP / TP | TP / TP | TP / TP | | TP / TP | TP / TP |
| 0x51d4 | TP / TP | TP / TP | TP / TP | TP / TP | TP / TP | | TP / TP | TP / TP |
| 0xc818 | TP / TP | TP / TP | TP / TP | TP / TP | TP / TP | | TP / TP | TP / TP |
| 0x88e8 | FN / FN | FN / FN | FN / FN | FN / FN | TP / FN | | TP / FN | TP / FN |
| 0x0006 | TP / TP | TP / TP | TP / TP | TP / TP | TP / TP | | TP / TP | TP / TP |
| 0x1e85 | TP / TP | TP / TP | TP / FN | TP / TP | TP / TP | | TP / TP | TP / TP |
| 0x8ddf | TP / TP | TP / TP | TP / TP | TP / TP | TP / TP | | TP / TP | TP / TP |
| 0x4f02 | TP / TP | TP / TP | TP / TP | TP / TP | TP / TP | | TP / TP | TP / TP |
| 0x6ab3 | TP / TP | TP / TP | TP / TP | TP / TP | TP / TP | | TP / FN | TP / TP |
| 0x831b | TP / TP | TP / TP | TP / FN | TP / TP | TP / TP | | TP / TP | TP / TP |
| 0x2496 | TP / TP | TP / TP | TP / TP | TP / TP | TP / TP | | TP / TP | TP / TP |
| 0x8b4a | TP / TP | TP / TP | TP / TP | TP / TP | TP / TP | | TP / TP | TP / TP |
| 0x30c2 | TP / TP | TP / TP | TP / FN | TP / TP | TP / TP | | TP / TP | TP / TP |
| 0x1995 | TP / TP | TP / TP | TP / TP | TP / FN | TP / TP | | TP / TP | TP / TP |
| 0x2ecd | TP / TP | TP / TP | FN / TP | TP / TP | FN / TP | | TP / TP | TP / TP |
| 0x49df | TP / TP | TP / TP | TP / TP | TP / TP | FN / TP | | TP / TP | TP / TP |

Figure A.1: Device Detection Summary

| Round 1 | | |
|---|---|---|
| | | |
| | **Schedule** | **Outcome** |
| **Setup 1 Live** | **15:54** | **Start** |
| | 15:57 Room on | User turned light on at Mar 11, 2024 15:57:02.431899000 CET 0xfffd |
| | 16:00 Change Saturation ZED 1 | User controlled saturation of light at Mar 11, 2024 16:00:03.131944000 CET 0x508c |
| | 16:03 Change Saturation ZED 2 | User controlled saturation of light at Mar 11, 2024 16:03:02.017325000 CET 0x8ddf |
| | | User controlled color of light at Mar 11, 2024 16:05:07.889990000 CET 0x2a10 |
| | | User controlled color of light at Mar 11, 2024 16:05:11.402664000 CET 0x51d4 |
| | | User controlled color of light at Mar 11, 2024 16:06:01.920226000 CET0x52a7 |
| | | User controlled color of light at Mar 11, 2024 16:05:31.802397000 CET0xc818 |
| | 16:06 Change Color ZED 1 | User controlled color of light at Mar 11, 2024 16:06:02.1839728000 CET 0x508c |
| | | User controlled color of light at Mar 11, 2024 16:06:07.303198000 CET 0x18ef |
| | 16:09 Change Color ZED 2 | User controlled color of light at Mar 11, 2024 16:09:02.2092544000 CET 0x8ddf |
| | 16:12 Room off | User turned light off at Mar 11, 2024 16:12:02.886945000 CET 0xfffd |
| | **16:15** | **Stop** |
| | | |
| **Setup 2 Live** | **20:33** | **Start** |
| | 20:36 Room on | User turned light on at Mar 11, 2024 20:36:02.444159000 CET 0xfffd |
| | 20:39 Change Saturation ZED 1 | User controlled saturation of light at Mar 11, 2024 20:39:08.623043000 CET 0x1995 |
| | | User controlled color of light at Mar 11, 2024 20:40:39.413127000 CET0xdbe7 |
| | | User controlled color of light at Mar 11, 2024 20:41:34.219766000 CET0x3c2b |
| | 20:42 Change Saturation ZED 2 | User controlled saturation of light at Mar 11, 2024 20:42:06.002742000 CET 0x831b |
| | 20:45 Change Color ZED 1 | User controlled color of light at Mar 11, 2024 20:45:02.460575000 CET0x1995 |
| | | User turned light off at Mar 11, 2024 20:48:17.312109000 CET0xfffd |
| | 20:48 Change Color ZED 2 | User controlled color of light at Mar 11, 2024 20:49:03.369231000 CET0x831b |
| | 20:51 Room off | User turned light off at Mar 11, 2024 20:51:08.057320000 CET 0xfffd |
| | **20:53** | **Stop** |
| | | |
| | | |
| **Setup 3 Live** | **17:53** | **Start** |
| | 17:56 Room on | User turned light on at Mar 13, 2024 17:56:29.670453000 CET 0xfffd |
| | 17:59 Change Saturation ZED 1 | User controlled saturation of light at Mar 13, 2024 17:59:16.182916000 CET 0x2ecd |
| | 18:02 Change Saturation ZED 2 | User controlled saturation of light at Mar 13, 2024 18:02:16.280485000 CET 0x49df |
| | 18:05 Change Color ZED 1 | User controlled color of light at Mar 13, 2024 18:05:31.088916000 CET 0x2ecd |
| | 18:08 Change Color ZED 2 | |
| | 18:11 Room off | User turned light off at Mar 13, 2024 18:11:27.262318000 CET 0xfffd |
| | **18:14** | **Stop** |
| | | |
| **Setup 1 Passive** | **16:33** | **Start** |
| | 16:36 Room on | User turned light on at Mar 11, 2024 16:36:02.436834000 CET 0xfffd |
| | 16:39 Change Saturation ZED 1 | User controlled saturation of light at Mar 11, 2024 16:39:01.982564000 CET 0x508c |
| | 16:42 Change Saturation ZED 2 | User controlled saturation of light at Mar 11, 2024 16:42:01.593252000 CET 0x8ddf |
| | 16:45 Change Color ZED 1 | User controlled color of light at Mar 11, 2024 16:45:01.187659000 CET 0x508c |
| | 16:48 Change Color ZED 2 | User controlled color of light at Mar 11, 2024 16:48:06.806189000 CET 0x8ddf |
| | 16:51 Room off | User turned light off at Mar 11, 2024 16:51:01.905245000 CET 0xfffd |
| | **16:54** | **Stop** |
| | | |
| | | |
| | | |
| | | |
| | | |
| **Setup 2 Passive** | **19:05** | **Start** |
| | | User turned light off at Mar 11, 2024 19:07:09.297732000 CET 0xfffd |
| | 19:08 Room on | User turned light on at Mar 11, 2024 19:08:07.956965000 CET 0xfffd |
| | 19:11 Change Saturation ZED 1 | User controlled saturation of light at Mar 11, 2024 19:11:02.255053000 CET 0x1995 |
| | 19:14 Change Saturation ZED 2 | User controlled saturation of light at Mar 11, 2024 19:14:01.894444000 CET 0x831b |
| | | User controlled color of light at Mar 11, 2024 19:15:49.592143000 CET 0x6ab3 |
| | 19:17 Change Color ZED 1 | User controlled color of light at Mar 11, 2024 19:17:01.054682000 CET 0x1995 |
| | 19:20 Change Color ZED 2 | User controlled color of light at Mar 11, 2024 19:20:04.945487000 CET 0x831b |
| | | User turned light off at Mar 11, 2024 19:21:13.916713000 CET 0xfffd |
| | 19:23 Room off | User turned light off at Mar 11, 2024 19:23:05.349314000 CET 0xfffd |
| | | User turned light off at Mar 11, 2024 19:24:35.838192000 CET 0xfffd |
| | **19:26** | **Stop** |
| | | |
| **Setup 3 Passive** | **18:30** | **Start** |
| | 18:33 Room on | User turned light on at Mar 13, 2024 18:33:03.093025000 CET 0xfffd |
| | 18:36 Change Saturation ZED 1 | User controlled saturation of light at Mar 13, 2024 18:36:10.350111000 CET 0x2ecd |
| | 18:39 Change Saturation ZED 2 | User controlled saturation of light at Mar 13, 2024 18:39:02.199639000 CET 0x49df |
| | 18:42 Change Color ZED 1 | User controlled color of light at Mar 13, 2024 18:42:11.038416000 CET 0x2ecd |
| | 18:45 Change Color ZED 2 | User controlled color of light at Mar 13, 2024 18:45:09.308161000 CET 0x49df |
| | 18:48 Room off | User turned light off at Mar 13, 2024 18:48:07.790282000 CET 0xfffd |
| | | User controlled color of light at Mar 13, 2024 18:48:31.221261000 CET 0x49df |
| | **18:51** | **Stop** |

Figure A.2: Event Detection Round 1 Summary

| **Round 2** | | |
|---|---|---|
| | | |
| | **Schedule** | **Outcome** |
| **Setup 1 Live** | **15:33** | **Start** |
| | | User controlled color of light at Mar 17, 2024 15:35:33.306449000 CET0xa264 |
| | | User controlled color of light at Mar 17, 2024 15:35:54.329851000 CET0x67c3 |
| | 15:36 | |
| | 15:39 | |
| | 15:42 | |
| | 15:45 | |
| | 15:48 | |
| | | User controlled color of light at Mar 17, 2024 15:50:18.557108000 CET0xc0cb |
| | 15:51 | |
| | | User controlled color of light at Mar 17, 2024 15:51:32.094359000 CET0x2a10 |
| | **15:54** | **Stop** |
| | | |
| | | |
| **Setup 2 Live** | **16:45** | **Start** |
| | 16:48 | |
| | | User controlled color of light at Mar 17, 2024 16:49:20.124291000 CET0xdbe7 |
| | 16:51 | |
| | 16:54 | |
| | | User controlled color of light at Mar 17, 2024 16:57:12.526758000 CET0xa535 |
| | | User controlled color of light at Mar 17, 2024 16:58:16.903288000 CET0x6ab3 |
| | 16:57 | |
| | 17:00 | |
| | 17:03 | |
| | **17:06** | **Stop** |
| | | |
| | | |
| **Setup 3 Live** | **14:00** | **Start** |
| | 14:03 | |
| | 14:06 | |
| | 14:09 | |
| | 14:12 | |
| | 14:15 | |
| | 14:18 | |
| | **14:21** | **Stop** |
| | | |
| **Setup 1 Passive** | **15:01** | **Start** |
| | 15:04 | |
| | | User controlled color of light at Mar 17, 2024 15:05:14.624026000 CET0xc818 |
| | | User controlled color of light at Mar 17, 2024 15:05:53.431734000 CET0x2496 |
| | 15:07 | |
| | 15:10 | |
| | 15:13 | |
| | 15:16 | |
| | | User controlled color of light at Mar 17, 2024 15:19:45.555600000 CET0x3206 |
| | 15:19 | |
| | | User controlled color of light at Mar 17, 2024 15:21:40.597662000 CET0x88e8 |
| | **15:22** | **Stop** |
| | | |
| **Setup 2 Passive** | **16:02** | **Start** |
| | 16:05 | |
| | 16:08 | |
| | 16:11 | |
| | | User controlled color of light at Mar 17, 2024 16:13:43.591511000 CET0x6ab3 |
| | 16:14 | |
| | 16:17 | |
| | 16:20 | |
| | **16:23** | **Stop** |
| | | |
| | | |
| | | |
| | | |
| **Setup 3 Passive** | **13:15** | **Start** |
| | 13:18 | |
| | | User controlled color of light at Mar 17, 2024 13:18:20.212248000 CET0x49df |
| | | User controlled color of light at Mar 17, 2024 13:18:36.471395000 CET0x2ecd |
| | 13:21 | |
| | 13:24 | |
| | 13:27 | |
| | 13:30 | |
| | 13:33 | |
| | **13:34** | **Stop** |

Figure A.3: Event Detection Round 2 Summary