PacketZapper: A Scalable and Automated Platform for IoT Traffic Collection and Analysis

Mathias Fredrik Hedberg, Jia-Chun Lin[a] and Ming-Chang Lee[b]
Department of Information Security and Communication Technology, Norwegian University of Science and Technology (NTNU), Gjøvik, Norway

Keywords: IoT Traffic Analysis, Automated Traffic Collection, Scalable Data Processing, Smart Homes, IoT.

Abstract: The increasing adoption of IoT devices in home environments has raised significant concerns about security and privacy. Analyzing real IoT traffic is essential for understanding these implications, yet the process poses challenges for researchers, requiring expertise in hardware selection, data collection, storage, and analysis. To address these challenges, we introduce PacketZapper, an automated and scalable platform for IoT traffic collection, processing, and analysis. PacketZapper combines existing open-source tools with custom components to streamline research workflows. It follows a four-stage solution structure—collect, parse, store, and process—ensuring modularity and future extensibility. The platform supports the collection of Zigbee and 433MHz traffic using commercial USB dongles, with the potential to integrate additional IoT protocols. Data is stored in Elasticsearch, enabling efficient querying and exploration, while Apache Airflow automates task orchestration through Directed Acyclic Graphs (DAGs). A case study evaluation demonstrated PacketZapper's capability to infer devices in a smart home and to facilitate effective data exploration. The platform provides a robust foundation for reproducible IoT traffic research, addressing critical gaps in IoT traffic analysis. It offers researchers an extensible, automated, and scalable solution for conducting diverse experiments.

1 INTRODUCTION

Smart devices have become an integral part of modern life, both in our homes and in urban environments. Common household amenities, such as lighting and climate control systems, have increasingly transitioned to the Internet of Things (IoT) ecosystem, offering benefits such as reduced energy consumption, improved productivity, and improved overall health (Karlicek, 2012; Hye Oh et al., 2014).

As IoT devices continue to proliferate, significant research has been conducted to understand potential privacy and security risks (Apthorpe et al., 2017; Lee et al., 2019; Ren et al., 2019; Acar et al., 2020; Gu et al., 2020b; He et al., 2021). For instance, some studies demonstrate that passive analysis of IoT sensor data can enable high-level activity inference, allowing researchers to gain insights into user interactions within a home (Lee et al., 2019). Furthermore, poor implementation of IoT systems can introduce substantial risks to users, including threats such as stalking and harassment (Lopez-Neira et al., 2019).

To address these concerns, researchers have extensively studied the security implications of IoT systems. Many of these studies rely on real-world IoT device traffic, which can be obtained from existing datasets (Cook et al., 2009) or captured directly in controlled laboratory environments (Ren et al., 2019; Hafeez et al., 2020). Existing datasets often lack the specific details needed for research, forcing researchers to capture and process their own IoT traffic.

When researchers attempt to collect their own traffic data, they often face significant challenges. Capturing IoT traffic requires substantial technical knowledge, including expertise in networking protocols, packet analysis, data handling, and familiarity with tools such as Wireshark or specialized IoT monitoring platforms. Moreover, current research practices often lack detailed documentation on how packet data is collected. Many existing research papers provide only a broad overview of their data collection process, leaving out critical details such as the specific tools or configurations used, the environment in which the data were captured and the criteria for selecting devices or networks.(Lee et al., 2019; Acar et al., 2020).

Although the components necessary for a compre-

[a] https://orcid.org/0000-0003-3374-8536
[b] https://orcid.org/0000-0003-2484-4366

Hedberg, M. F., Lin, J.-C. and Lee, M.-C.
PacketZapper: A Scalable and Automated Platform for IoT Traffic Collection and Analysis.
DOI: 10.5220/0013426400003944
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 10th International Conference on Internet of Things, Big Data and Security (IoTBDS 2025)*, pages 370-377
ISBN: 978-989-758-750-4; ISSN: 2184-4976
Proceedings Copyright © 2025 by SCITEPRESS – Science and Technology Publications, Lda.

hensive platform to analyze IoT traffic are available, they are rarely integrated into a cohesive, automated system. Consequently, researchers often rely on manual data collection methods, which are not only error-prone but also inefficient—particularly when using multiple tools and systems. This gap underscores the pressing need for an automated and versatile platform to streamline and enhance the IoT research process.

In this paper, we introduce PacketZapper, an automated platform designed to simplify the collection and processing of IoT device traffic and enable researchers to address the above-mentioned gaps. Built using a combination of off-the-shelf software components and custom tools, PacketZapper offers a scalable solution to accelerate IoT traffic-based research.

The platform is designed to primarily target traffic from IoT devices and sensors commonly found in smart home environments, with a focus on consumer products like smart light bulbs, motion sensors, and other low-bandwidth devices. Scalability is a key aspect of its design, ensuring it can effectively address potential constraints and bottlenecks in data capture, storage, and processing. The primary contributions of PacketZapper are threefold:

- Proposing a scalable, automated platform for IoT traffic analysis.
- Releasing the platform as open-source[1] for adaptability and extension.
- Facilitating reproducible and efficient IoT research.

The remainder of the paper is structured as follows: Section 2 provides the background, and Section 3 reviews related work. Section 4 introduces the design of PacketZapper, while Sections 5 and 6 present the implementation details and case study analysis, respectively. Finally, Section 7 concludes the paper and suggests directions for future research.

2 BACKGROUND

Since PacketZapper is designed to collect IoT traffic, it is important to have a baseline understanding of the protocols commonly used by IoT devices for communication. While many protocols are available, we focus specifically on the Zigbee protocol and the 433 MHz ISM band due to their widespread adoption in home environments and their critical role in enabling communication for a diverse range of IoT devices.

[1]PacketZapper, https://github.com/PacketZapper/PacketZapper

2.1 Zigbee Protocol

Zigbee is a wireless communication protocol designed to enable low-cost, low-power, and low-data-rate communication between devices (Farahani, 2011). Built on the IEEE 802.15.4 Low-Rate Wireless Personal Area Network (LR-WPAN) standard, Zigbee provides a reliable and efficient way to connect, control, and monitor devices such as light bulbs, smart sockets, and environmental sensors. Zigbee is a key component of popular smart home ecosystems, including Philips Hue and IKEA Trådfri. Its standardized protocol ensures interoperability between devices from different manufacturers.

A Zigbee network must have one coordinator and at least one other device, either a router or an end device. While routers are optional, networks can include multiple end devices. The coordinator is responsible for selecting a PAN ID (both 64-bit and 16-bit) and a communication channel to establish the network. Once the network is formed, the coordinator functions similarly to a router. Both the coordinator and routers can allow other devices to join the network and facilitate data routing. When an end device joins a network through a router or coordinator, it becomes dependent on its ”parent” (the router or coordinator that allowed it to join) for communication. End devices can transmit or receive RF data via their parent. Since end devices often enter low-power sleep modes, the parent must buffer incoming data packets destined for the end device until it wakes up to retrieve the data. This mechanism ensures efficient communication while optimizing power usage for end devices. Zigbee networks can also operate in distributed architectures without a coordinator.

The Zigbee protocol supports 128-bit AES encryption for message authentication and payload confidentiality. However, its implementation has faced criticism. For example, the default link key used for device authentication was publicly leaked, introducing significant security risks (Shafqat et al., 2022). Research also shows that encrypted traffic can still leak substantial information (Acar et al., 2020; Gu et al., 2020a).

Zigbee packets can be captured using inexpensive USB dongles designed for Zigbee networks, such as the TI CC2531. Alternatively, Software Defined Radios (SDRs) paired with GNU Radio modules can decode Zigbee packets (Akestoridis et al., 2020). While SDR-based setups require more expensive hardware and expertise, they provide greater flexibility. Notably, Zigbee traffic can be sniffed without network authentication, as only the data payload is encrypted. Programs like Wireshark can decrypt Zigbee pay-

loads if the necessary encryption keys are provided (Akestoridis et al., 2020).

2.2 The 433MHz ISM Band

The 433 MHz ISM (Industrial, Scientific, and Medical) band[2] is a segment of the radio frequency spectrum, specifically from 433.05 MHz to 434.79 MHz, designated for unlicensed use in industrial, scientific, and medical applications. This allocation is recognized in ITU Region 1, including Europe, Africa, and parts of the Middle East. Devices operating within this band are typically low-power and are used for short-range communications, such as remote controls, wireless sensors, and certain types of RFID systems.

Smart home ecosystems like Telldus and Nexa use the 433MHz ISM band for devices such as weather sensors, smoke detectors, and light bulbs. However, these systems often use proprietary communication protocols, limiting interoperability between brands.

Capturing traffic on the 433MHz band is straightforward due to the simplicity of its modulation techniques. Tools like the Flipper Zero[3] or an SDR paired with rtl_433 software[4] can decode messages. For example, rtl_433 can identify signals from weather stations or TPMS devices, decoding attributes like wind speed or tire pressure. SDR devices based on the Realtek RTL2832U chipset, typically priced around $20, are widely supported by rtl_433 and provide an efficient solution for decoding 433 MHz traffic.

3 RELATED WORK

Numerous projects rely on real IoT traffic for inference and analysis. However, most existing platforms are designed to focus on specific IoT protocols, such as Zigbee, Bluetooth, or TCP/IP, and do not provide a unified approach to traffic collection and processing across multiple protocols.

The authors in (Ren et al., 2019) developed a test infrastructure to automate device testing in controlled IoT labs. Their system remotely controls Android devices and collects TCP/IP traffic passing through the router. However, their work is limited to analyzing network-layer traffic using tools like tcpdump and Wireshark, without addressing wireless protocols. The IoTSpy project (Gu et al., 2020b) takes a different approach, using dongle-based sniffers to capture Zigbee traffic via CC2531 USB dongles. Their focus is on eavesdropping to infer user activities, with Z-Wave mentioned as a potential attack vector but left untested. Both collection and analysis of traffic in this work were performed manually.

The authors of Zigator (Akestoridis et al., 2020) propose a Zigbee-focused platform for analyzing wireless traffic, utilizing SDRs for both passive sniffing and active attacks. The platform is open-source and has been used in subsequent Zigbee security studies (Akestoridis et al., 2022; Akestoridis and Tague, 2021). However, Zigator is limited to Zigbee networks and requires expertise in SDR tools like GNU-Radio. Building on Zigator, Zleaks (Shafqat et al., 2022) uses a CC2531 dongle to passively collect Zigbee traffic and infer user activity but does not employ SDRs. Peek-a-Boo (Acar et al., 2020) collects traffic from multiple IoT sources, including Zigbee, WiFi, and Bluetooth. Although its traffic analysis contributions are significant, the work lacks details about the capture process, and the data processing seems to have been performed manually.

While notable progress have been made in IoT traffic collection and analysis, many platforms rely heavily on manual processes or provide limited automation. PacketZapper distinguishes itself by integrating automated collection and processing capabilities, while also being designed with scalability in mind. Although it currently focuses on specific protocols, its modular design enables future expansion to support IoT network protocols.

4 PacketZapper

This section introduces PacketZapper and its main architecture. PacketZapper follows a four-stage solution structure, as shown in Figure 1. It begins with the Collect stage, where IoT traffic is captured using protocol-specific hardware and software. Next, the Parse stage filters and organizes raw traffic into structured formats. The Store phase then saves the parsed data in a scalable, searchable database for easy retrieval. Finally, the Process stage automates workflows for data analysis, experiment execution, and dynamic adjustment of collection parameters. The sequential flow of data between these stages ensures a streamlined process, while a loop from the Process stage to the Collect stage enables automation. This loop dynamically refines data collection based on the results of ongoing experiments, improving the platform's efficiency and adaptability.

PacketZapper consists of three main components: remote nodes, a centralized backend, and a user en-

[2]ISM radio band, https://en.wikipedia.org/wiki/ISM_radio_band?

[3]Flipper Zero, https://flipperzero.one/

[4]rtl_433, https://github.com/merbanan/rtl_433/

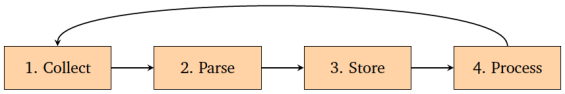

Figure 1: The four-stage solution structure of PacketZapper.

vironment, as illustrated in Figure 2. Remote nodes, located in the target IoT environment, host the Collection Agent, which is a lightweight software module responsible for capturing IoT traffic and performing basic parsing. Remote nodes use protocol-specific hardware, such as a CC2531 USB dongle for collecting Zigbee data, to capture raw IoT traffic. After initial filtering and parsing, the processed data is transmitted to the backend for storage and further analysis.

The backend, hosted on a centralized server, consists of four components:

- Elasticsearch[5]: A highly scalable, distributed search and analytics engine built on top of Apache Lucene, designed to handle large volumes of data. It enables fast and efficient storage, search, and retrieval of IoT traffic data with low latency.
- Kibana[6]: A powerful visualization tool that integrates seamlessly with Elasticsearch, allowing users to explore and analyze stored data through interactive dashboards and visualizations.
- Apache Airflow[7]: An open-source platform for managing batch-oriented workflows and automating complex experimental pipelines. Built on a flexible Python framework, Airflow enables users to define workflows as Directed Acyclic Graphs (DAGs), which outline the sequence of operations for data collection and processing. Airflow's capacity to manage large-scale data workflows plays a central role in enabling PacketZapper's automation capabilities.
- Jupyter Lab[8]: An interactive environment for testing Python code snippets, modifying Airflow DAGs, and prototyping experimental workflows. It provides flexibility for tailoring workflows to specific research needs and allows for iterative experimentation.

The user environment of PacketZapper offers intuitive interfaces for managing and experimenting with the platform. Users interact with PacketZapper through tools like Jupyter Lab and the Airflow UI. The Airflow UI provides a user-friendly interface for managing workflows and monitoring their execution.

[5]Elasticsearch, https://www.elastic.co/elasticsearch
[6]Kibana, https://www.elastic.co/kibana
[7]Apache Airflow, https://airflow.apache.org/
[8]Jupyter Lab, https://jupyter.org/

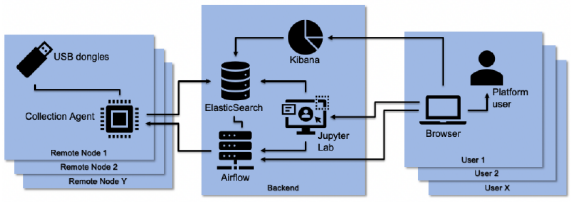

Figure 2: The deployment overview of PacketZapper.

A key strength of PacketZapper lies in its integration of automation and scalability. Airflow orchestrates workflows, allowing users to define experimental pipelines that can trigger specific actions, such as adjusting traffic collection parameters or integrating external systems. PacketZapper's modular design supports horizontal scaling by deploying additional remote nodes, making it adaptable to both small-scale lab setups and large-scale IoT environments.

The combination of Kibana and Jupyter Lab enhances PacketZapper's flexibility and usability. Kibana provides users with an intuitive graphical interface for analyzing IoT traffic, while Jupyter Lab supports customization and iterative experimentation, ensuring PacketZapper remains versatile and aligned with evolving research needs.

By combining a well-defined solution structure, robust architectural design, and industry-standard tools like Elasticsearch, Airflow, Kibana, and Jupyter Lab, PacketZapper effectively addresses the complexities of IoT traffic collection and analysis.

5 IMPLEMENTATION

PacketZapper is packaged by default to run within a Docker runtime environment[9], simplifying the setup process. This environment uses individual Dockerfiles to manage the installation of dependencies required for each component of the platform. Additionally, docker-compose files orchestrate the overall setup by handling networking, storage, and basic system configurations. This approach enables quick and straightforward installation of the platform while offering users the flexibility to customize their deployment. For example, users can scale specific components or integrate new ones as needed.

The use of Docker also makes it transparent how PacketZapper can be installed on bare-metal environments. Users with DevOps experience can adapt the code to run on a variety of hardware configurations, or outsource specific components, such as the Elasticsearch instance, to third-party cloud providers. In

[9]docker, https://www.docker.com/products/container-runtime/

this section, we describe the implementation of each component of PacketZapper.

5.1 Collection Agent

The Collection Agent is responsible for collecting and parsing IoT traffic, using physical hardware like CC2531 USB dongles for Zigbee traffic (via whsniff) and RTL-SDR dongles for 433MHz data (via rtl_433). The agent formats the collected data as JSON strings and sends it to an Elasticsearch database. It runs a lightweight REST API built with the Python FastAPI framework, enabling full remote control and scalability across multiple instances.

The agent is implemented in Python 3.9 and is packaged to run within a Docker environment for easy deployment on various hardware, from Raspberry Pi (ARM-based) to x86 servers. While the default setup runs in Docker, the agent can also be executed directly on the host system. The code is modular and designed for extensibility, making it straightforward to add support for additional dongles or protocols. However, due to reliance on UNIX pipes, Windows-only parsing software is currently unsupported.

The Collection Agent runs in Docker, isolating it from the host system, including USB devices. While USB passthrough works on Linux systems by running the container in privileged mode and exposing the USB bus, it is not fully supported on macOS due to Docker's virtualization limitations. macOS users must run the agent directly on the host.

To function, the agent requires the Elasticsearch endpoint and connectivity to the Airflow service. The default Docker configuration handles this out of the box, but remote deployments may require additional network setup, such as creating an encrypted tunnel.

Data is sent to Elasticsearch in batches of 10–50 items to avoid overloading the server with HTTP requests. While currently hard-coded for each protocol, dynamically adjusting the batch size would improve performance, particularly in low-traffic scenarios.

The agent exposes a REST API on port 8000, with endpoints for starting, stopping, and checking the status of sniffing processes. HTTP POST handles state-changing (start/stop), while GET retrieves status info. The API also includes auto-generated OpenAPI documentation for easy testing and debugging.

5.1.1 Zigbee Support

Zigbee support was implemented using TI CC2531 USB dongles, an inexpensive and widely available hardware option. The dongle requires flashing with alternate sniffing firmware, available from TI, using external hardware like a CC-debugger[10]. Projects such as Zigbee2MQTT[11] provide detailed flashing instructions for various operating systems. The open-source whsniff software[12] is used with the CC2531 dongle to sniff and decode Zigbee traffic into PCAP format. The output is piped through tshark to convert the data into JSON format for ingestion into Elasticsearch. While the encrypted Zigbee payload remains undeciphered, the headers across the OSI packet structure are preserved and readable.

5.1.2 433MHz Support

433MHz message decoding was implemented using the Nooelec NESDR SMART dongle and the open-source rtl_433 software, which utilizes generic SDR receivers (e.g., Realtek RTL2832U chipset) to decode signals from devices such as temperature sensors, garage door openers, and remote controls.

During testing, rtl_433 detected nearby environmental sensors transmitting temperature and humidity data. Adding the -F json flag enables live JSON export to STDOUT for Elasticsearch ingestion. The software supports various command-line options for filtering signals, and users can pass extra arguments via the Collection Agent API using the EXTRA_ARGS variable in a POST request.

5.2 Elasticsearch

PacketZapper uses Elasticsearch as the primary storage for traffic data, chosen for its dynamic data mapping and advanced query capabilities. It enables complex searches, such as calculating average Zigbee packet size over 10-minute intervals or grouping data by source/destination. No fixed JSON structure is enforced, allowing flexibility to add new protocols as long as Elasticsearch can process them. An index template ensures proper data interpretation, storing all traffic in the "packetzapper" index. Users can filter by the sniffer key to distinguish protocols, e.g., Zigbee traffic labeled as whsniff.

5.3 Kibana for Data Visualizations

PacketZapper includes Kibana to simplify the exploration and visualization of data stored in Elasticsearch. Kibana offers a user-friendly interface for creating powerful visualizations, combining them into dashboards, and enabling users to identify trends and

[10] CC-debugger, https://www.ti.com/tool/CC-DEBUGGER

[11] Zigbee2MQTT, https://www.zigbee2mqtt.io/

[12] whsniff, https://github.com/homewsn/whsniff

gain insights through advanced filtering and aggregation. Its built-in search functionality supports intricate queries, such as fuzzy matching and range searches. Users can export filters or visualizations as DSL queries and run them directly against Elasticsearch via scripts for task automation.

5.4 Airflow for Task Automation

PacketZapper uses Apache Airflow to automate pipelines, from data collection to inference calculation, via DAG files defining task sequences. It runs Airflow in Docker, based on the official image, allowing users to add dependencies like Python Elasticsearch packages. The setup, deployed with docker-compose.yml, includes an internal bridge network for seamless communication between Airflow, Jupyter, the Collection Agent, and Elasticsearch. It also integrates internal services like PostgreSQL and Redis. The DAGs folder is bind-mounted, enabling modifications from both the host OS and Jupyter.

5.5 JupyterLab

PacketZapper includes a JupyterLab instance for remote system management and prototyping collection pipelines before integrating them into Airflow DAGs. JupyterLab has access to all configured services and supports collaboration, allowing multiple users to connect. It also provides a command line for installing Python packages and dependencies.

The Airflow /dags folder is mounted in JupyterLab, and installing the apache-airflow package via pip enables IDE code completion for DAG development. While JupyterLab is useful for prototyping, full system management (e.g., managing Docker containers) must be done on the host. On Linux, passing the Docker socket could allow such commands from JupyterLab, but this was avoided to maintain platform-agnostic compatibility.

6 CASE STUDY ANALYSIS

To evaluate PacketZapper's capabilities in automating IoT traffic collection and inference tasks, a lab environment was set up in a residential apartment, and a case study was conducted using commonly used IoT devices. The environment was designed to simulate a smart home installation while remaining non-RF-isolated, meaning it also captured signals from neighboring smart devices like power meters, weather stations, and other Zigbee networks. This setup reflects realistic conditions where wireless signals overlap, introducing challenges similar to those faced in real-world environments. An overview of the lab setup is shown in Figure 3.

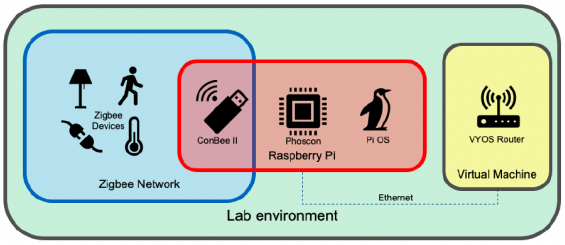

Figure 3: The lab environment for evaluating PacketZapper.

The lab environment included a VyOS virtual router connected to an isolated subnet, configured with NAT, DHCP, DNS, and basic firewall rules. A Raspberry Pi 3B running Raspberry Pi OS Lite was connected via Ethernet, equipped with a ConBee II Zigbee dongle and running the Phoscon and deCONZ applications. The Zigbee network included a mix of devices: one smart socket, two environment sensors, and one smart light (see Table 1). The deCONZ applications provided detailed insight into the Zigbee network, such as MAC addresses, device types, and link quality, which were used to confirm the device configurations.

Table 1: List of devices connected to the Zigbee network.

Device	Function	MAC Address
ConBee II	Zigbee gateway node (Zigbee coordinator)	00:21:2e:ff:ff:06:0b:e4
Hue Bloom	Desk lamp (Zigbee router)	00:17:88:01:0c:67:e6:27
Hue Smart Plug	Smart socket relay switch (Zigbee router)	00:17:88:01:0b:e1:19:53
Hue Motion sensor	Environment sensor (end device)	00:17:88:01:0b:d0:aa:cf
Aquara temperature	Environment sensor (end device)	00:15:8d:00:05:44:ee:f6

The lab setup also ensured that a potential threat actor would have no physical access to devices or equipment, forcing all inference tasks to rely on wireless signals alone. This aligns with realistic smart home threat models.

For this case study, Apache Airflow was extensively used to automate all tasks, while JupyterLab facilitated the creation of Airflow DAGs and debugging of the code used within them. A simple DAG was created to automate the inference pipeline (as shown in Figure 4), consisting of the following tasks:

1. pz_online: Verifies that the Collection Agent is online and responsive.
2. pz_start: Starts Zigbee sniffing.
3. pz_sleep: Waits for 5 minutes to collect sufficient data.

4. pz_stop: Stops Zigbee packet capture.
5. zleaks_infer: Runs inference tasks and posts results back to Elasticsearch.

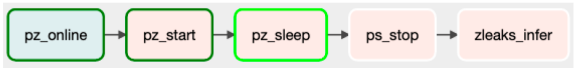

Figure 4: Screenshot of the DAG graph view for the inference pipeline.

This pipeline is simple yet fully automated and can be extended to perform additional tasks. In this case study, we configured it to run only when manually triggered. The Collection Agent Python client library was used to interact with the API in a Pythonic manner. The zleaks_infer task contains the logic for performing inference operations, including identifying the number of devices on the network, such as the Zigbee coordinator, routers, and end devices. The task executes these inferences and posts the results to Elasticsearch, where users can view them.

To identify the number of Zigbee routers, denoted by ZR, we applied the formula described in (Shafqat et al., 2022) on broadcast packets with a Zigbee radius value of 1, as shown in Equation 1:

$$\mathrm{ZR} = \frac{\text{packet payload length} - 2}{3} \quad (1)$$

To implement this in Airflow, we first validated the approach in Kibana by filtering packets with a destination address of 0xffff and a network radius of 1. After verifying the Zigbee payload length formula, we implemented the logic in Python and integrated it into the zleaks_infer task in Airflow.

Implementing functionality to identify the number of active Zigbee end devices on the network proved challenging, as the methodologies described in previous works (Akestoridis et al., 2020; Shafqat et al., 2022) were vague. To address this, we created a Kibana dashboard to explore the collected data and identify potential methods for implementation in Airflow. Figure 5 shows a screenshot of this workflow, where the dashboard presents graphical representations of traffic grouped by different source addresses and message types.

Through the process of analyzing data in Kibana and referencing the Zigbee and LR-WPAN specifications (ZigBee, 2015; LR-WPANs, 2011), we discovered that end devices that do not perform routing could be identified by searching for data requests containing the WPAN long address instead of the short address, as suggested in (Akestoridis et al., 2020; Shafqat et al., 2022). However, we also observed that certain devices, like the Hue Motion Sensor, despite being battery-powered, behaved as Zigbee routers by

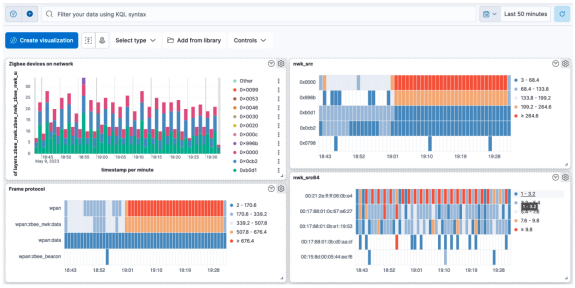

Figure 5: Example usage of PacketZapper for capturing and analyzing Zigbee traffic.

relaying messages. Consequently, it did not appear as an end device using our newly developed technique. Ultimately, we were only able to identify the Aqara temperature sensor as an end device, indicating that the methodology described in (Shafqat et al., 2022) may be inaccurate, though our process brought us close to the correct value.

To identify the Zigbee coordinator by its MAC address, we filtered for packets with the Zigbee source address of 0x0000 and retrieved the most recent 64-bit source MAC address. We integrated all this functionality into the zleaks_infer task and triggered the Airflow DAG using the web interface. The progress was monitored, and results were viewed in Kibana. The inference results are summarized in Table 2.

Table 2: Inference results compared to true values.

Inferred item	True value	Inference Result
Device count	5	5
Zigbee routers	2	2
End devices	2	1
Coordinator MAC	00:21:2e:ff:ff:06:0b:e4	00:21:2e:ff:ff:06:0b:e4

As shown in the table, we successfully inferred several network attributes using passive methods. However, we were unable to determine the correct number of end devices, either using the technique described in (Shafqat et al., 2022) or through our own developed methods. This discrepancy stems from the limitations of the inference techniques rather than any shortcomings of the platform itself. Overall, this case study demonstrates that the platform is a suitable choice for conducting inference experiments. It effectively showcased its capabilities for task automation, as well as for developing and debugging methods.

7 CONCLUSIONS AND FUTURE WORKS

In this paper, we introduced PacketZapper, an automated and scalable platform for IoT traffic collec-

tion, processing, and analysis, streamlining research workflows from traffic capture to inference tasks. Its design is based on a four-stage solution structure, combining reusable software components with future extensibility. Currently, the platform supports Zigbee and 433MHz traffic collection through horizontally scalable Collection Agents, with Elasticsearch managing data storage and Apache Airflow enabling workflow automation.

The case study demonstrated PacketZapper's effectiveness in automating tasks and supporting data exploration, successfully addressing the challenges of IoT traffic collection and automation. While the platform presents a learning curve, particularly with Python, Elasticsearch, and Airflow, it offers significant flexibility and advanced functionality for researchers. Future enhancements, such as support for additional protocols like Bluetooth and Z-Wave, will further extend its applicability and relevance in IoT traffic research.

ACKNOWLEDGEMENT

This work received funding from the Research Council of Norway through the SFI Norwegian Centre for Cybersecurity in Critical Sectors (NORCICS), project no. 310105.

REFERENCES

Acar, A., Fereidooni, H., Abera, T., Sikder, A. K., Miettinen, M., Aksu, H., Conti, M., Sadeghi, A.-R., and Uluagac, S. (2020). Peek-a-boo: I see your smart home activities, even encrypted! In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 207–218.

Akestoridis, D.-G., Harishankar, M., Weber, M., and Tague, P. (2020). Zigator: Analyzing the security of zigbee-enabled smart homes. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 77–88.

Akestoridis, D.-G., Sekar, V., and Tague, P. (2022). On the Security of Thread Networks: Experimentation with OpenThread-Enabled Devices. In *Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 233–244.

Akestoridis, D.-G. and Tague, P. (2021). HiveGuard: A Network Security Monitoring Architecture for Zigbee Networks. In *2021 IEEE Conference on Communications and Network Security (CNS)*, pages 209–217. IEEE.

Apthorpe, N., Reisman, D., Sundaresan, S., Narayanan, A., and Feamster, N. (2017). Spying on the smart home: Privacy attacks and defenses on encrypted iot traffic. *arXiv preprint arXiv:1708.05044*.

Cook, D., Schmitter-Edgecombe, M., Crandall, A., Sanders, C., and Thomas, B. (2009). Collecting and disseminating smart home sensor data in the CASAS project. In *Proceedings of the CHI workshop on developing shared home behavior datasets to advance HCI and ubiquitous computing research*, pages 1–7. IEEE.

Farahani, S. (2011). *ZigBee wireless networks and transceivers*. newnes.

Gu, T., Fang, Z., Abhishek, A., Fu, H., Hu, P., and Mohapatra, P. (2020a). IoTGaze: IoT security enforcement via wireless context analysis. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 884–893. IEEE.

Gu, T., Fang, Z., Abhishek, A., and Mohapatra, P. (2020b). Iotspy: Uncovering human privacy leakage in iot networks via mining wireless context. In *2020 IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications*, pages 1–7. IEEE.

Hafeez, I., Antikainen, M., Ding, A. Y., and Tarkoma, S. (2020). IoT-KEEPER: Detecting malicious IoT network activity using online traffic analysis at the edge. *IEEE Transactions on Network and Service Management*, 17(1):45–59.

He, X., Yang, Y., Zhou, W., Wang, W., Liu, P., and Zhang, Y. (2021). Fingerprinting mainstream iot platforms using traffic analysis. *IEEE Internet of Things Journal*, 9(3):2083–2093.

Hye Oh, J., Ji Yang, S., and Rag Do, Y. (2014). Healthy, natural, efficient and tunable lighting: four-package white leds for optimizing the circadian effect, color quality and vision performance. *Light: Science & Applications*, 3(2):e141–e141.

Karlicek, R. F. (2012). Smart lighting-beyond simple illumination. In *2012 IEEE Photonics Society Summer Topical Meeting Series*, pages 147–148. IEEE.

Lee, M.-C., Lin, J.-C., and Owe, O. (2019). PDS: Deduce elder privacy from smart homes. *Internet of Things*, 7:100072.

Lopez-Neira, I., Patel, T., Parkin, S., Danezis, G., and Tanczer, L. (2019). 'internet of things': How abuse is getting smarter.

LR-WPANs (2011). IEEE Standard for Local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs). https://ieeexplore.ieee.org/document/6012487. [Online; accessed 02-February-2025].

Ren, J., Dubois, D. J., Choffnes, D., Mandalari, A. M., Kolcun, R., and Haddadi, H. (2019). Information exposure from consumer iot devices: A multidimensional, network-informed measurement approach. In *Proceedings of the Internet Measurement Conference*, pages 267–279.

Shafqat, N., Dubois, D. J., Choffnes, D., Schulman, A., Bharadia, D., and Ranganathan, A. (2022). Zleaks: Passive inference attacks on Zigbee based smart homes. In *International Conference on Applied Cryptography and Network Security*, pages 105–125. Springer.

ZigBee (2015). ZigBee Specification. https://zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf. [Online; accessed 02-February-2025].