UCLouvain

École polytechnique de Louvain

Analyzing energy consumption of smart home devices through network monitoring

Author: **Antoine Tacq**
Supervisors: **Tom Barbette, Ramin Sadre**
Reader: **Nicolas Rybowski**
Academic year 2023–2024
Master [120] in Computer Science

Acknowledgements

First of all, I am extremely grateful to my two supervisors, Tom Barbette and Ramin Sadre, for their guidance over the year. Their feedback and advice significantly contributed to the development of this work.

Special thanks to François De Keersmaeker, who shared his work on updating an existing code, allowing me to use and adapt it for my thesis. I am also grateful to Nicolas Rybowski for agreeing to be my reader and a member of my jury.

Finally, I would like to thank my family for their support throughout my academic journey.

Contents

Chapter 1

Introduction

In today's society, there are more and more connected devices in our homes, ranging from simple devices such as smart light bulbs, to more complex ones such as smart thermostats, fridges, or washing machines. One of the challenges for homeowners today is to clearly identify the energy consumption of all the active devices in their home in order to improve their energy management and, ultimately, reduce their consumption. Unfortunately, most devices cannot monitor their own energy consumption. We know that IoT devices communicate with other devices and that users can remotely activate or deactivate them using applications on smartphones, for example. When these actions occur, the network activity of a device will increase for a short period of time. Therefore, we want to propose a new way to monitor and predict the energy consumption of devices in smart homes. We do this by monitoring their network traffic and identifying events to determine the activity of the devices.

The contribution of this research is to propose an application that recognizes network communication patterns of IoT devices in smart homes and predicts their energy consumption. The first challenge to overcome is to correctly identify the devices on the network without requiring the homeowner to manually label each local IP address with the name of the device. Then, another challenge is to identify events from the devices' network traffic, especially user events, such as when a user turns on a smart plug. For this, we use an existing state-of-the-art approach that identifies user events from the captured network traffic, and we adapted it by developing the necessary tools to detect device activities and user events on a daily basis. The final challenge is to use the known total energy consumption of the smart home and the activities of the devices to accurately determine their actual energy consumption.

In this report, we first introduce the Internet of Things, its various use cases and the one that is important for our research: smart homes. We also explain key concepts from the literature to better understand the subject. Then, we present our first discoveries on IoT devices network traffic and the reasons why we believe it is possible to achieve our objectives. We also describe a methodology that we considered for our work, but that has limitations, which is to identify devices solely based on their network traffic using machine learning classifiers. After that, we describe the design of our solution.

We evaluate the performance of our application in two scenarios. We performed one controlled experiment with three different devices and observed the accuracy to detect user events in the network traffic and to classify them using trained machine learning models. We also summarize the results from a second experiment where we deployed our application in a smart home. Finally, we discuss our proposed solution, its limitations, and limitations of the state-of-the-art approaches that we used in this work.

Chapter 2

Background

In this chapter, we introduce the context of this research as well as different concepts to help the reader to understand the problem we want to solve. We will start by introducing the Internet of Things and smart homes. Then, we will explain device identification, with methodologies proposed by researchers as well as some protocols that are defined and can help to achieve this. Finally, we will review the literature about existing techniques to identify devices behaviors on a local network and also energy monitoring techniques for smart homes.

2.1 Internet of Things

We can describe the term "Internet of Things" (IoT) as all the devices, equipped with sensors or specific software, that collect and exchange information. As outlined by IBM[11], those devices are everywhere and the potential applications of IoT are vast and varied, covering a wide range of industries. To mention a few examples, there is the transportation industry where IoT devices can be used as sensors to monitor vehicle performances. There is also the healthcare industry to manage inventories or track a patient's vital signs. The number of connected IoT devices in the world is rapidly growing. This number was around 15 billion devices in 2023 and could nearly double in the coming years to reach up to 30 billion connected IoT devices in the world in 2030 [20].

2.1.1 Smart homes

In the context of this work, we are interested in smart homes and IoT devices within them. The concept of smart homes, as mentioned by Sovacool et al.[19], is not new and we can find ideas of it back in the 1900s.

However, it is since the 2000s that this notion of smart home has arisen with the goal to make homes more efficient and enjoyable. There are various definitions of smart homes in the literature, but we will use the definition of Balta-Ozkan et al.[4]: *"A smart home is a residence equipped with a communications network, linking sensors, domestic appliances, and devices, that can be remotely monitored, accessed or controlled and which provide services that respond to the needs of its inhabitants"*.

Smart homes encompass a wide range of applications. For example, smart appliances and connected devices can be washing machines or fridges, light bulbs or smart plugs, but also connected energy management systems, sensors or cameras. It is the system as a whole and the connections between the devices that allow to monitor and manage a home easily. In 2023, there were 360 million smart homes worldwide. This number is expected to grow to more than 785 million smart homes by 2028 [21].

2.2 Device identification

Device identification on a local network is a significant research field in IoT and smart homes. It involves the ability to determine the name of a device, its type, or even its vendor name. There can be many reasons for this, such as security or privacy concerns. This is even more important when you don't have access to the complete list of devices in a smart home, especially for an application running autonomously. In such cases, this avoids the need to ask a user to manually provide the list of active devices in their home. In this section, we define the relevant approaches and techniques that can be used in order to identify devices in the local network.

2.2.1 Prediction models

From a security point of view, Bezawada et al.[5] highlighted the need to correctly identify devices on a local network to mitigate vulnerabilities and attacks against these IoT devices. They propose to generate a behavioral profile for each device to identify them. To do this, they first capture network traffic and divide it, for each device, into sessions. Sessions are sequences of packets that have the same source/destination IP addresses and source/destination port numbers, in both directions of the communication flow.

After that, they extract 20 features including 17 packet header features, that are simply 0 and 1 for the presence or absence of a given network protocol. The complete list of features is shown in Table 2.1.

In addition, they consider TCP payload features, such as the TCP payload length, for all the packets in the sessions. After extracting the features, they build feature vectors and with it, they train their machine learning classifier. They evaluated their approach with a dataset that they created. They collected data on a small set of devices representing a significant spectrum of IoT device categories available on the market. They emulated a normal usage of those devices and captured the network traffic. Their results show that they are able to predict with high accuracy (∼95%) the type of a device when presented a new feature vector using their previously trained prediction model.

Layer/Type	**Features**
Link layer	ARP
Network	IP, ICMP, ICMPv6, EAPoL
Transport	TCP, UDP
Application	HTTP, HTTPS, DHCP, BOOTP, SSDP, DNS, MDNS, NTP
IP Options	Padding/Router Alert

Table 2.1: List of protocols used as header features.

Similarly, Miettinen et al.[15] tackle the problem of security vulnerabilities of IoT devices in homes by presenting a solution to identify devices as well as a mitigation strategy to confine traffic of devices identified as vulnerable. Their solution also passively captures network traffic, but here, they focus on the specific timing when a new device start communicating with the gateway (in this case, the router/modem that provides wireless connections to devices on a network). They record n packets during this setup phase and extract 23 features. Most of them are identical to the ones from the research of Bezawada et al.[5] presented earlier as shown in the table 2.1 but including other features such as a destination IP counter. With this, they build feature vectors and then they propose a two-fold identification technique. For each device type, they first train a single classifier that will be able to decide if a new feature vector matches a device or not. Then in case of a tiebreak between two classifiers of two different devices, they use an edit distance-based metric. This metric will compute the distance between the fingerprint of the device to be classified and a subset of fingerprints of those known devices involved in the tiebreak.

To evaluate their approach, they also simulated devices communications in an IoT laboratory environment and manually labeled their collected data to later build their classification models. They used a set of 27 devices representing common IoT devices available on the market.

Their approach has an identification accuracy over 95% for 17 devices and around 50% for the 10 others, because of difficulties to distinguish two similar devices from the same vendor.

2.2.2 Network-based device identification

There are protocols and solutions that are used to allow device discovery and communication on a local network that can be used to identify devices. We describe them in this section.

ARP

To identify devices, a methodology could involve the Address Resolution Protocol (ARP)[7]. This protocol creates a mapping between the IP address of a device and its Media Access Control (MAC) address in order to be able to send IP packets. When it needs to send data to another device, a device will look in its ARP cache to see if there is a MAC address and a corresponding IP address for the destination device. If the information is not available in the cache, the source device will send a broadcast message to request and get the information it needs and store the received answer in its cache for future use.

ARP caching is a method that was implemented to limit the usage of network resources by ARP because the mapping from an IP address to a MAC address occurs on the network for every datagram sent. This caching technique stores network addresses in memory for a period of time, and these addresses can then be reused to minimize the resource utilization. This work being based on the assumption that we have only access to the router of the smart home, the table containing the cache could be accessed in order to get the list of all devices active in the smart home.

Zeroconf and mDNS

With devices becoming smaller and more ubiquitous, there was a need in the society to operate in infrastructures and networks without configuration and conventional Unicast DNS servers to look up for DNS resources including host names or DHCP servers to allocate IP addresses. This concept of Zero-configuration networking (Zeroconf), popular with the 'Bonjour' protocol from Apple, facilitates the discovery of services on a local network and communication between devices. In Zeroconf, one key protocol is the protocol multicast Domain Name System (mDNS) that was proposed in the Internet Engineering Task Force (IETF) in RFC 6762[6].

In this protocol, devices are allowed to have link-local multicast DNS host names of the form "single-dns-label.local.", provided that the name is not already in use. These devices can discover and resolve IP addresses and hostnames by broadcasting messages to all other devices, which can then store the information and respond. Not all devices use it, but if available, these host names can be used to identify devices.

Organizationally Unique Identifier

A MAC address is a unique identifier of six octets assigned to each device. The first three octets correspond to the organizationally unique identifier (OUI) that is a 24 bit number that identifies a vendor, manufacturer or organization. This OUI is assigned by the Institute of Electrical and Electronics Engineers (IEEE). IEEE maintains a list[17] that is publicly available of all the identifiers provided to organizations, linked with their addresses and names. By looking at this unique identifier and the corresponding vendor, we can fetch the name of the organization, which provides additional information on the identity of the device.

2.3 Device activity

In smart homes, all the connected devices utilize network protocols such as Wi-Fi, Bluetooth Low Energy (BLE) or Zigbee [18] to exchange information. In this work, we focus on monitoring the Wi-Fi network traffic of devices in smart homes to fetch useful information about their activities. There is a challenge that needs to be addressed in order to obtain the information we need. This challenge is that the network traffic of those devices is encrypted for security reasons.

Researches such as one from Acar et al.[2] showed that even though the traffic is encrypted there is still information that we can use. We can, for example, monitor and extract features from the network traffic such as the mean packet length or the mean inter-arrival time between two packets. With this information, we can train machine learning classifiers, such as a random forest classifier, to predict new device activities. With such methodology, what we can achieve is to identify the states of these devices that we have on our network. For example, if the device is a light bulb, detect whether the device is turned on or off.

One state-of-the-art IoT behavior identification model has been proposed by Hu et al.[10]. In their work, they model a per-device and system-wide behavior of an IoT system, using information inferred from the IP traffic produced by the IoT devices. We will only focus on their per-device approach.

In their research, they consider two device behavior models : periodic models and user-actions models. The periodic models capture recurring behaviors of the IoT devices, such as keep-alive messages. In the other hand, the user-actions models capture behaviors linked to user actions, such as turning on a smart plug or a light bulb.

The first step to infer these 2 models consists in partitioning the network traffic into sequences of packets transmitted to/from an IoT device, also called flows. They define flows as set of chronologically ordered TCP segments or UDP datagrams with the same tuple :

[source IP, source port, destination IP, destination port, transport protocol]

Flows can last for a long time. For this reason, they regroup them into bursts for all consecutive packets in which the interval time between any two consecutive packets is less than 1 second. Then, they annotate each flow burst with the starting time of the flow, the name of the device and the protocol used, combined with 21 features. Because the network traffic is encrypted, these features are related to information from the headers of the encrypted packets but also related to the packets and flow burst sizes. We present some of the features that they use in the Table 2.2 below.

Feature	Descriptoin
meanBytes	Average bytes in a flow
minBytes	The lowest number of bytes in a flow
maxBytes	The highest number of bytes in a flow
medAbsDev	The median absolute deviation of number of bytes in a flow
meanTBP	The average of time differences between consecutive packets
varTBP	The variance of time differences between consecutive packets
network_out_external	The number of packets sent to server
network_in_external	The number of packets received from server
network_in_local	The number of packets received from other device
meanBytes_out_external	Average bytes sent to servers per packet
meanBytes_in_external	Average bytes received from servers per packet

Table 2.2: Examples of features selected to infer device behaviors.

2.3.1 Periodic models

During their analysis of the network traffic generated by devices, they found that most flows with the same destination and protocol appear at regular time intervals. To capture this behavior, they separated every distinct flow burst into distinct traffic groups. Then, for each group they used an unsupervised approach to check if the traffic has periodicity. This approach combines Discrete Fourier Transformation (DTF) and autocorrelation. DTF is used to identify frequencies that carry significant power in spectral density and autocorrelation is then used to validate the presence of these periodic patterns in the groups. The traffic groups that exhibit periodicity are then used as their periodic models.

Once they created their periodic models, they can use them to infer future network traffic and recognize periodic events. To achieve this, they use timers to label periodic traffic with clearly identifiable periods. Then, they also use existing tools such as DBSCAN[8] to group each periodic flow into clusters based on the models they trained.

2.3.2 User-actions models

For user-action models, they train a supervised Random Forest classifier for each IoT device, using the previously mentioned features for each flow burst. This approach requires a dataset of user events where the data in the dataset is correctly labeled with the names of the events. Once trained, the models can predict and classify new unlabeled flows.

2.3.3 System evaluation

They evaluated their approach by building a testbed of 49 IoT devices, which was deployed in a laboratory. They connected the devices to the internet via a gateway that can capture all network traffic. From this testbed, they were able to create a dataset with controlled and uncontrolled experiments. For the uncontrolled part of their dataset, they conducted an experiment where participants could use the testbed as they wanted to. The controlled part is divided into three important categories :

- Activity dataset : data collected from **user events**. They interacted with the IoT devices (e.g., turning on and off a light bulb) and recorded the network traffic generated. Those interactions were repeated many times and labeled so that they could be used as ground-truth to later infer user events.

- Idle dataset : data collected from **non-user events**. As they mentioned, the vast majority of network traffic is, in fact, not related to user actions. To evaluate their model on those events, they captured the network traffic of the IoT devices during periods of inactivity, also called "idle" states (i.e., when no interactions with users occurred).

- Routine dataset : data collected from both user events and non-user events. This part of the dataset is used to simulate a real smart home environment that includes both user events and inactivity periods. For this part, however, they only considered a subset of 18 devices because these devices support action-trigger features that can be used to create sequences of home automation, such as when a camera detects a motion: activate a smart bulb.

They used their idle dataset to evaluate how many non-user flows could be modeled as periodic. They found that 99.8% of the flows in this dataset were exhibiting periodicity. Furthermore, they evaluated the ability of their models to classify flows as periodic events on this dataset. The results showed that their approach could correctly identify 99.2% of the periodic flows as periodic events. Similarly, using the activity dataset, they trained their user-action models and measured an overall classification accuracy of 98.9%.

They analyzed their system in a real-world environment using the whole dataset they created. Using their periodic modeling methodology, they identified 97.8% of events presenting periodicity with a median of 5 periodic models per device. They also observed that there is a correlation between the number of models for a device and its complexity. For example a smart fridge or a smart speaker will have more functionalities and generate more periodic models.

In addition, the user events of the devices were classified with an overall accuracy of 98.9%. The reason for that is that for most of them, these events correspond to easily identifiable network traffic. Nevertheless, they observed that some user events are nearly impossible to classify accurately with their model. Examples of events are ON/OFF events, where the only difference resides in the data of the encrypted payload of the frames. But, because the data is encrypted it is not possible to distinguish them using the Random Forest classifier that they use in their approach.

2.3.4 Comparison with other approaches

In our work, we will focus on the approach presented by Hu et al. [10]. However, various approaches have been proposed prior to their research.

Acar et al.[2] propose a methodology to identify user events using either Random Forest classifier (RF) or k-Nearest Neighbors classifier (KNN). For this, they divide their capture in time intervals of length W and extract feature vectors with the mean packet length, mean inter-arrival time and median absolute deviation of packet size. In terms of results, their RF model has an average of 88% of correctly identified events and 91% for their kNN model. To classify events, their best model is a RF classifier with an average performance measurement of 94% of correctly classified events.

Trimananda et al.[22] propose PingPong, their system to identify user events. They focus on TCP connections in their work and are interested in sequences of packets. They present each individual packet of a network capture to state machines that advance to their next state if the packet matches a previously identified packet sequence. Upon reaching the terminal state of a state machine, the sequence of packets is reported. For their evaluation, they obtain results of at least 97% of accurately identified events for each device.

In the end, the methodologies are different, with completely different features used to train classifiers, but the results are nearly similar. Still, compared to the others, the work of Hu et al. [10] achieves better results. Furthermore, the two other approaches don't have the ability to identify non-user events, and PingPong[22] doesn't support UDP network traffic.

2.4 Energy monitoring

Smart home energy management systems (SHEMS), are systems that monitor the energy consumption of all appliances in a home. Load monitoring also refers to analyzing the energy consumption of appliances. This can be done either through Intrusive Load Monitoring (ILM) or Non-Intrusive Load Monitoring (NILM).

Intrusive load monitoring involves the installation of devices on each appliance in a house to monitor their consumption. For instance, Mtshali et al.[16] propose to implement such smart technologies in a cost-effective manner using smart plugs, which act as intermediaries between the device and the power outlet.

In their research, they propose to connect smart plugs, which have energy monitoring capabilities, to every device in the home and poll each of them for their energy consumption data and store it in a SQL database. Then an application can runs tasks to analyse such data.

Han et al. [9] propose a SHEMS architecture based on ZigBee focusing both on the consumption and generation of energy. To monitor the consumption, they also use an ILM approach and install an Energy Measurement and Communication Unit (EMCU) in each outlet to measure the energy consumption of the appliances and send it to a home server using ZigBee. In their research, they also analyze the energy generation from solar or wind power systems in the house and schedule the appliances with the home server to reduce the energy consumption. At any time, a user can access and browse the information related to each appliance using their web application connected to the home server.

While these approaches will correctly monitor the energy consumption of the appliances in a home, we can argue that they are not convenient to setup. Indeed, they require effort from the homeowner to buy and place such devices for every appliance in the house in order to effectively monitor their energy consumption.

Alternatively, a non-intrusive load monitoring approach (or energy disaggregation) determines the energy consumption of each appliance in a house from a single source (e.g., the total energy consumption of the house) using advanced algorithms. Kelly and Knottenbelt [14] investigate the use of neural networks to perform energy disaggregation. Neural networks are directed graphs where the nodes are artificial neurons and the edges allow information to pass from one neuron to another. One architecture that they use is a denoising autoencoder neural network with 6 layers. Its task is to attempt to recover the power demand of a target appliance from the background 'noise' that is produced by other appliances. An important consideration is that neural networks require a lot of training data because of the number of trainable parameters. Nevertheless, they tested their approach using an existing source dataset and were able to determine the energy consumption of all appliances with accuracy score of 93%, highlighting the good performance of this model.

Chapter 3

Motivations and considerations

Nowadays, every home and smart home have a customer premises equipment (CPE), or modem, provided by an internet service provider (ISP) that acts as a wireless access point that is connected to the internet and to which smart devices are connected. In the literature, we found no solution that was using the network communications of IoT devices to try to predict their actual energy consumption. However, this is an interesting source of information and we felt that is was worth trying to utilize it.

On the other hand, there are existing home management solutions, such as Home Assistant. Home Assistant is an open-source home automation software that allows to control all of the devices in a smart home. Home Assistant provides tools to manage your home and allows you to control your devices from a dashboard but also to monitor their activity. Unfortunately, if a device doesn't have built-in energy monitoring features, it is not possible for Home Assistant to track its energy consumption. Therefore, we decided to imagine this work as a service that could be deployed by ISPs as a software application at the CPE level that could monitor the energy consumption of each device and provide advice to users.

For this work, we identified three key steps to consider when thinking about creating an application that can monitor the energy consumption of devices in a smart home. First, we must identify which devices are active on the network. Then, having the list of devices we have to be able to identify their states to finally be able to predict their energy utilization. In this chapter, we will explain the first experiment that we conducted that motivated our research and a methodology that we considered for device identification.

3.1 First observations

We conducted a small experiment at the beginning of this research to observe and understand how devices exchange data. Figure 3.1 illustrates the setup used for this experiment. A Raspberry Pi is configured as a wireless access point, with smart devices connected to it, and is running Home Assistant.

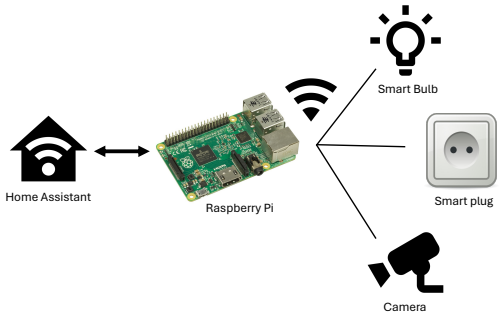

Figure 3.1: Initial experiment setup.

We connected the following device types to the Raspberry Pi : a smart bulb, a camera and two smart plugs, all connected to the Raspberry Pi using Wi-Fi. With the Raspberry Pi, we captured the network traffic of the devices and interacted with them via Home Assistant during the capture to turn them on and off. As we mentioned, Home Assistant is a powerful application that a person can use to interact with and monitor their devices. The application also uses a database that stores activity histories and other useful information, which can be easily accessed. Using this database of Home Assistant, we retrieved the names and timestamps of the actual events and plotted the results as you can see in Figure 3.2.

From this, we can observe that for the smart plug, when an event that turns on or off occurs, there is a temporary increase in the amount of exchanged bytes. For the camera, it is when the device is turned on that the device will send more data because it records videos. Apart from that, it seems that when no event occurs the network traffic looks periodic and different for each device. The results of this experiment reinforced our beliefs that it was certainly possible to deduce the different states of each device in a smart home by monitoring their network traffic.

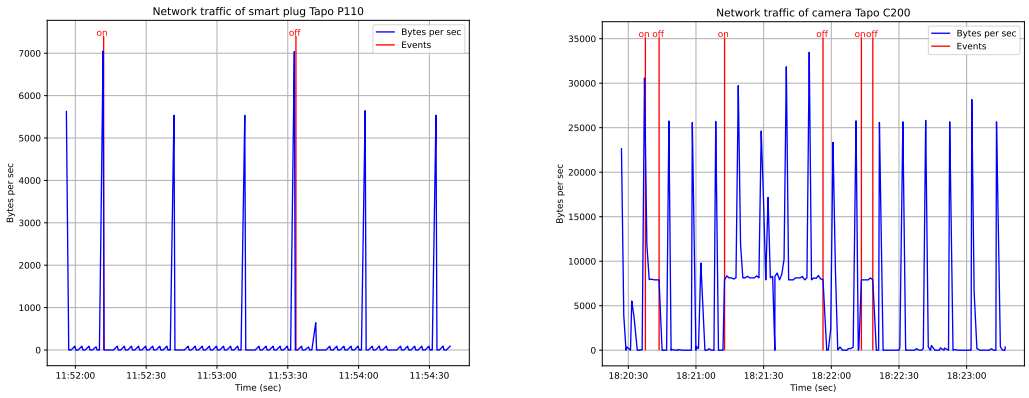

Figure 3.2: Network traffic of two devices captured for the experiment.

3.2 Identification based on prediction models

In the literature, we found a substantial number of articles published within the last 7 years that presented solutions to identify devices on a home network using machine learning classifiers, but mostly for security concerns. We presented two examples of such solutions in Section 2.2.1. The majority of models were described as having an accuracy of at least 93% to predict and identify devices, which seemed promising for our study. Following this, we tried to replicate the methodology presented in the research of Bezawada et al.[5] to assess its accuracy and capabilities, with the only difference being that we divided the network traffic packets by groups of 10 seconds instead of the proposed decomposition in "sessions", for simplicity. Furthermore, we couldn't get access to the dataset that they created to conduct their research so we had to find other available datasets.

To address the dataset issue, we used the YourThings dataset[1] that was made publicly available by Alrawi et al.[3] in relation to their article on the security of home-based devices where they systematize the literature to understand attack techniques against IoT devices and their mitigations and they propose a methodology to study those devices. For the dataset, they generated network traffic for 65 devices during 13 days and uploaded the corresponding packet capture (.pcap) files on a website. Alongside the PCAP files, they added a CSV file that maps device names to their local IP address, an important element to correctly label the packets and train our model. Due to the size of the dataset we only used the first day of network traffic and divided it into a training and testing set. The results of our model showed a prediction accuracy of 90% on new unseen data, which is close to other models from the literature.

Despite the results, we decided not to use this methodology of training machine learning classifiers to identify devices. The reason is that in order to be able to train a model, we need data, and specifically network traffic captures of many devices. We also need data of a large number of devices, sufficient enough to be used in real life scenarios. The problem is that it is not an easy task to find such publicly available datasets, if any. On top of that, the dataset needs to be labeled with the network traffic files mapped in one way or another to a device and its name. One limitation of the approaches proposed in the literature is that they cannot be used outside of laboratories with more than a small set of known devices. They don't provide tools to collect data of unseen devices in homes along with additional information to assist the homeowner to label the collected data. Following this idea, one solution for our research might have been to :

- Create a framework to pre-train a prediction model for popular devices in a controlled environment, such as a lab. This model could then be used to identify these devices in any home where they are present.
- Ask users to provide a sample of data for our dataset if the model encounters an unknown device. This would allow us to retrain our model so that it can later identify the previously unknown device.

Unfortunately, this solution would increase the complexity of the application we want to develop so we didn't consider it and preferred to find an alternative approach.

Chapter 4

Design

Now that the background of our work is set, we will describe the design choices that we made to create an application that provides the tools to establish the link between the network activity of devices in a smart home and their energy consumption. We will first present an overview of our application and its key steps. Then, we will delve deeper in the description of our approach to explain the decisions we made.

4.1 Design overview

As we mentioned, our objective is to infer the energy consumption of IoT devices in smart homes. To achieve it, we identified three important phases that we must achieve in our application, as shown in Figure 4.1.

4.1.1 Initialization

The first phase is the initialization, or setup phase. Here, we collect the necessary data to create the environment in which our application will run. The initial task involves establishing the list of active devices in the smart home with the participation of the homeowner and using available information. After the identification process, we capture the network traffic of these devices for a sufficient period, long enough to capture the behaviors of each device. Eventually, after the capture, we generate the idle models that will be used later to predict user activities.

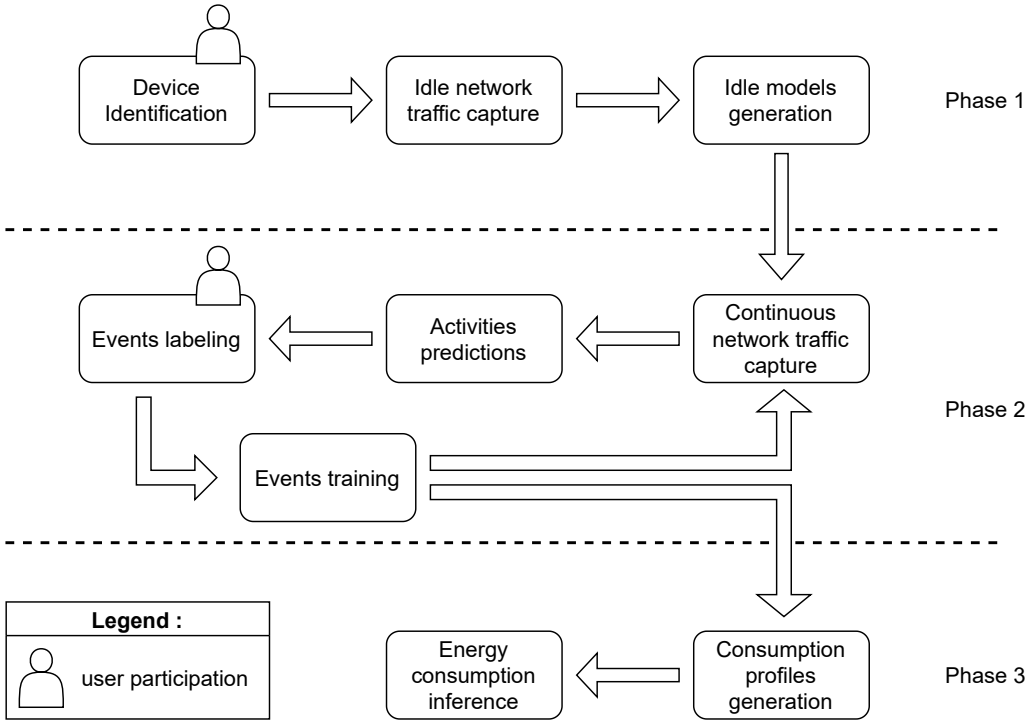

Figure 4.1: Overview of the workflow of our application.

4.1.2 Activities detection

Once the initialization phase is done, we enter the second phase of the workflow, which is to identify events in the network traffic that are triggered by user activities. These activities lead to devices changing their states, and our objective is to accurately identify such changes.

For this reason, we capture the network traffic of the devices registered in our application. Using the models generated in phase 1, we can already analyze the captured data to identify events. We then send the list of detected events to the user to ask for labeling, so that we can train a model to recognize and classify them. By doing so, we create an iterative process from scratch, where we ask a user to actively participate to allow the system to eventually create models that accurately classify events that are detected.

4.1.3 Consumption profiles creation

The final phase consists of creating energy consumption profiles for each device present in the smart home. Similarly to the previous phase, this is an iterative process that improves the profiles over time. We use the list of states that we generated previously combined with the total energy consumption of the house. Because we know the specific times the changes in states occurred for the devices, we can infer the consumption related to a specific event by computing the difference between the total consumption of the house before and after the event.

4.2 Device identification in the network

Device identification in the context of this work is an important first phase to accomplish. The objective here is to be able to filter the network traffic captured at the home modem and map each local IP address to the corresponding device (e.g., map the IP address 192.168.1.15 to a Tapo P110 smart plug).

4.2.1 Identification with human interaction

The approach that we have chosen relies on a set of tools and network protocols that are available in combination with the cooperation of the homeowner to map each local IP address to its device. In this approach, the entry point is to query the information stored in the ARP cache introduced in subsection 2.2.2. The result of this query contains a list, as shown in Table 4.1, with all the available devices.

Combined with an organizationally unique identifier lookup to fetch the vendor name and Zeroconf to get the link-local hostname, we can create a complete inventory of devices connected in the smart home with as much information as possible. Once the inventory is completed, the next phase is to display it to the user that will then be able to enter names for each of his devices based on the information available and his knowledge of his home. Finally, a mapping between the device names given by the user and their IP addresses will be stored.

IP Address	MAC Address	Interface
192.168.0.5	68:c6:3a:e6:7c:de	wlan0
192.168.0.8	9c:53:22:a2:63:f0	wlan0

Table 4.1: Example of ARP Table.

DHCP

Dynamic Host Configuration Protocol, or DHCP, is a network protocol used to attribute IP addresses to hosts in a local network so that they can communicate with other hosts using IP. While this protocol is not used for device discovery like mDNS, we observed that it might help us to identify devices. For this reason, we describe it here. In this protocol, devices that want to connect to the DHCP server (the modem in case of smart homes) will broadcast a discovery request to get an answer with configuration information for the network they are in. The DHCP server will receive this request and answer with an offer containing an IP address and a time period called a lease, for which this configuration will be valid. The device that created the initial discovery request will send a request to confirm that it accepts the given IP address and this message will be acknowledged by the server.

This is this last request, sent by the client, that is interesting for our work. In this request, the host has the possibility to provide different options[12]. In particular, the option with code 12 : Host Name Option, that allows the device to specify its hostname. This information may or may not be useful and vary with the configuration made by the vendor of a device and may also not be supported at all. Nevertheless, if this option is used and depending on the operating system used, we observed that this host name can be present in the ARP cache table when requested with option "**-a**" to try to resolve hostnames. We then add this hostname as supplementary information to facilitate the identification of devices on the network.

4.3 Devices states

In this section we will describe the design choices regarding the identification of the different states of IoT devices. This is the most critical part of our work, because it determines how accurately we can observe activities in the smart home and influences the predictions we can make concerning the energy consumption of these IoT devices. We decided to use the the system created by Hu et al.[10] that we presented in section 2.3.

The decision to use this approach comes from the promising results of their research with their ability to accurately find user events in captured network traffic. The Python code that they used for their work was made publicly available, and we could access the controlled part of their dataset upon request. This accessibility motivated us to continue in this direction.

Moreover, an updated version of the Python code, with correction of errors, was done by François De Keersmaeker, who created a public GitHub repository[13]. The system, developed by the researchers, can be viewed as a well-defined pipeline. It first decodes the captured network traffic present in the dataset and creates flows that are then used to extract feature vectors. After that, the pipeline separates the data into periods and preprocesses it to later train its periodic models for the idle part and the user-action models for the activities. In the end, the system uses the generated models to infer events on the routine part of the dataset.

In order to use this system in a real-world application, we had to make some changes and adapt it. We focused on recreating the environment in which their pipeline was used and collecting the data required to use it properly. In a few words, we divided the pipeline into three key steps. The first one, that we will call "*Idle models*", generates the periodic models necessary to identify network traffic related to non-user activities (e.g., heartbeats or keep-alive messages to distant servers). The second one, "*Activities predictions*", processes the network traffic of the IoT devices that is captured continuously by the application to identify possible user events. The third one, "*Events training*", trains the machine learning classifiers to predict new unseen events that the system may encounter.

4.3.1 Idle models

The first time that the application is launched, no data is available to model the behaviors of the IoT devices. Consequently, the initial step towards modeling these behaviors is to capture the network traffic of the devices. The idle dataset of the research upon which our work is based was created with captured network traffic of five days for each device.

Unfortunately, it is not reasonable in a real-world setting to capture network traffic for devices that must remain inactive during such a long period. This is something that could be done in a lab for a small set of devices such as popular devices, but it cannot be done for all devices available on the market. So, we opted to capture idle traffic for three hours at first, then changed to six hours, which is a reasonable period of inactivity that would still allow the system to observe the majority of the behaviors of a device. We will explain the reasons for these changes in the evaluation chapter. Then, the captured data is filtered based on the IP addresses of the devices and sent to train and generate the periodic models.

4.3.2 Activities predictions

Similarly to the preceding step, this phase requires to capture network traffic. However, unlike the previous phase, there is no need for the devices to remain inactive and we expect the user to use them in a typical manner. The objective is to identify user events that are not periodic or related to usual traffic observed in the idle dataset. The application will receive these captures and use the periodic models previously created to identify events and filter out traffic that has been tagged as periodic.

4.3.3 Events training

Now that the periodic models are used and that potential user events have been identified, we still need to predict the types/names of events that have occurred previously and those that will occur in the future. The activity part of the dataset constructed by the researchers contained more than 30 manually labeled events for all types of events supported by each of the devices. Unfortunately for us, there is no label on the events that we detect and no public dataset that could be used to train our machine learning models. We then rely on the participation of the user once more. We provide the detected events to the user and ask them to label them. With this information, we can train our classifiers. Initially, all events are unknown, so the user can only label them using activity histories from their home management apps. Eventually, the models can predict most events once they have sufficient data available. This is an iterative process where we continuously retrain the classifiers with newly labeled events provided by the user, thus improving the ability to identify and predict new events over time.

4.4 Energy profiles

Now that we have identified devices and that we can identify their states based on their network traffic, the final phase involves the creation of energy consumption profiles. To this end, we utilize the total energy consumption of the house, as it represents the most reliable source of information we can have regarding the actual energy consumption of devices. The methodology is to compute, for each event, the difference between the total consumption before and after the event occurred. Similarly to the previous process, this methodology is incremental. This implies that the energy profiles will be enhanced over time as the number of events detected increases. By continuously analyzing these changes in energy consumption, we can create profiles that effectively correspond to the devices' energy consumption within the smart home.

4.5 Implementation

In this section, we outline the architecture and implementation of our application, which involves developing a web application that can be deployed on a home router. It is composed of a backend developed with Flask, and a frontend built with Angular. The code of our application is available in this GitHub repository.

The frontend is developed with Angular. This is an open-source framework, based on TypeScript, that has the necessary tools to build a simple web application. To create it, we used the default build command using Node.js to start from a template project and we divided it into components, a feature of angular that allows easy development and separation of the key elements of the application. Additionally, we use the HttpClient library to create http requests targeted to the backend of our application. We create one http request per piece of information that we need. For example: a POST request to "`http://127.0.0.1:5000/api/submit_device_names`" to register the names given by the user for each device, or a GET request to "`http://127.0.0.1:5000/api/get_device_list`" to get the list of devices available in the database of the application.

For the backend of our application we decided to use Flask, a lightweight framework in Python that allows users to easily develop web applications. In relation with our frontend, we create the corresponding responses for each API HTTP request in our backend. For eaxmple: a function linked to the flask route "`@app.route('/api/get_device_list', methods=['GET'])`", that queries our database and returns a JSON object with the list of devices. To store our list of devices, their events and consumption profiles, we needed a database. We decided to use the Flask-SQLAlchemy extension to create a simple SQLite database and generated 4 tables that we describe in Table 4.2.

Alongside the Python framework, we built all the components and tools to establish the logic of our system. We use, for example, the Python subprocess library to run commands on the machine. We combine the `tcpdump` command to capture network traffic and the `timeout` command from GNU coreutils to specify the durations of the captures. We use other Python libraries that we need in order to build our models, such as scikit-learn, pandas, numpy, statsmodels or scipy. We can also mention the Python APScheduler tool that we use to create jobs. In particular, one job is repeated with intervals of 4 hours to capture network traffic (to create PCAP files of manageable sizes). We have a job that is executed every day at 8:00 and 16:00, that analyses the PCAP files and launches bash scripts that predict events. We chose these specific times because this task requires significant computing power and does not need to be repeated more frequently.

Devices Table	
Column	**Description**
id	Integer
device_name	Name of the device
ip_address	IP address of the device
mac_address	MAC address of the device
idle_done	Boolean value set to true if the capture of idle network traffic is done

Events Table	
Column	**Description**
id	Integer
device_name	Name of the device
is_classified	Boolean : true if the event was validated by the user
event_name	Event name, either via model prediction or via user
timestamp	Timestamp of the event

Consumption Table	
Column	**Description**
id	Integer
device_name	Name of the device
active_power	Power in Watts when the device is active
idle_power	Power in Watts when the device is idle

NetworkInterface Table	
Column	**Description**
id	Integer
interface_name	Interface where to capture network traffic
can_capture	Boolean value true if the setup phase is complete for at least 1 device
router_ip	IP address of the router, to filter it out of each capture file

Table 4.2: Tables in our SQLite database.

4.5.1 Interface

Our application interface is functional and simple. The backend and its components that run in the background are complex, for this reason we decided to develop a simple web interface for the user. We present illustrations of this interface in Figures 4.2 and 4.3.

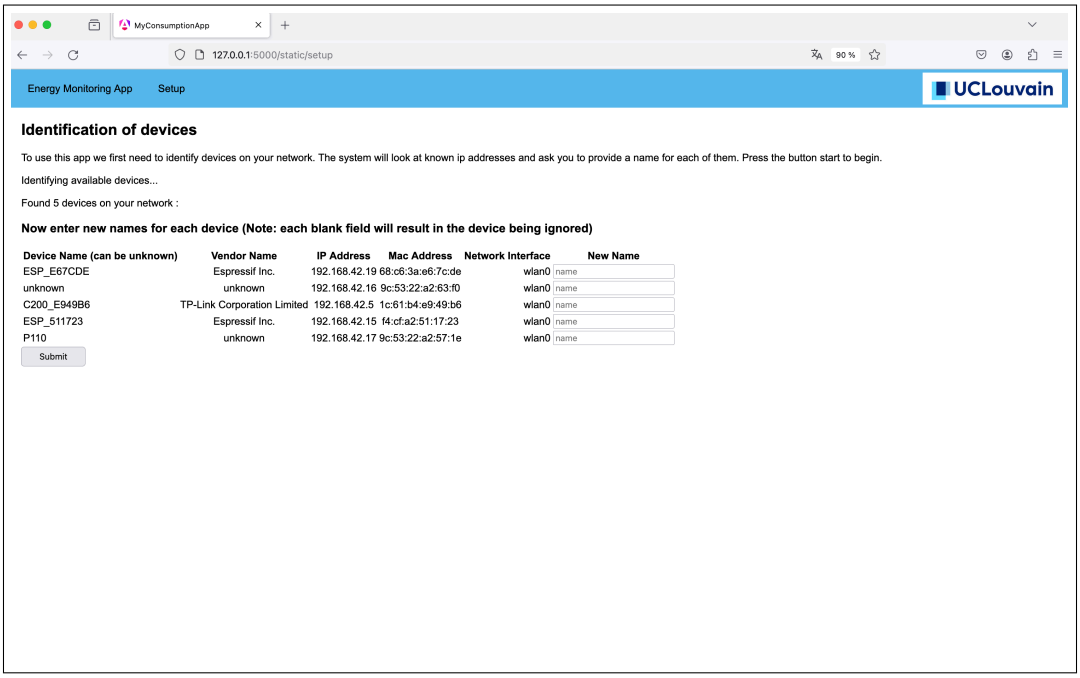

Figure 4.2: Screenshot of the setup page with the device identification phase.

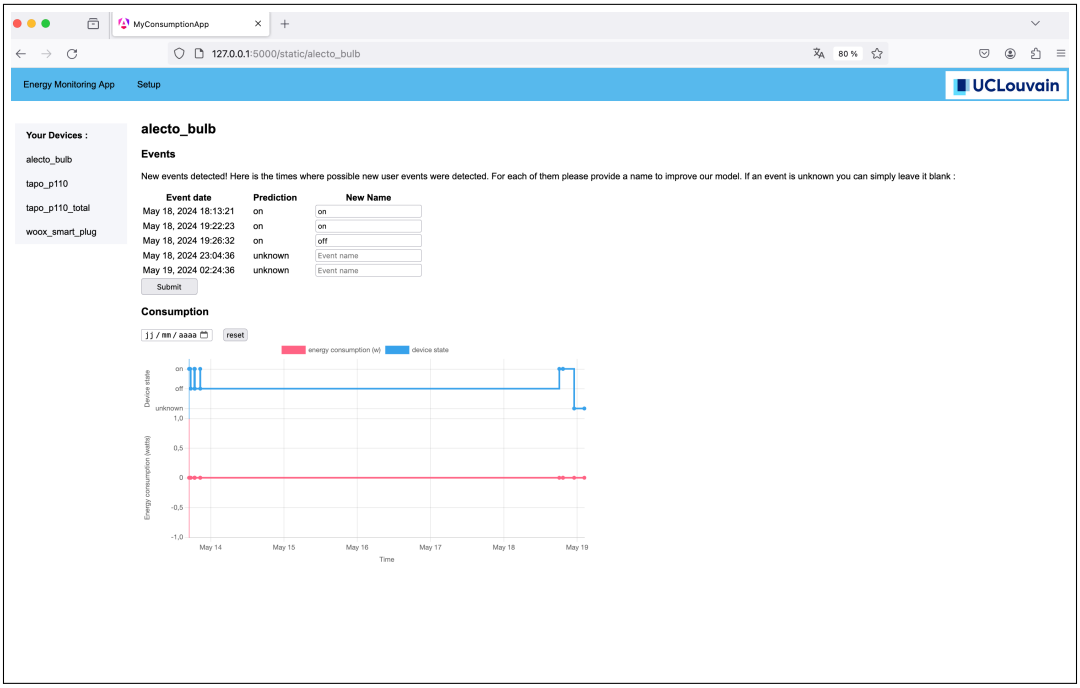

Figure 4.3: Screenshot of the page of a device with new events detected but no consumption profiles.

Chapter 5

Evaluation

To evaluate the performances of our approach, we decided to test it in different environments. For our first experiment, we created a controlled environment where we interacted with simple devices to understand the system's capabilities. These devices are smart bulbs, smart plugs or cameras that are connected via Wi-Fi to a Raspberry Pi. The second experiment is a real-world evaluation. We deployed our application in a smart home equipped with IoT devices to evaluate the system's behavior in such a scenario with 4 new devices that we didn't use in the first experiment.

5.1 Experimental setup

The experimental setup that we used is similar to the one introduced in chapter 3, as shown in Figure 3.1, except that we adapted it to correspond as much as possible to a real world application. The devices available were :

- Two TP-Link Tapo P110 plugs that can be activated and deactivated remotely and that have energy consumption monitoring capabilities.
- One Woox smart plug R5024, that can also be activated remotely but without the monitoring capabilities.
- One Alecto smart-bulb 10. With this lightbulb, the user has the possibility to change the color or the brightness as well as turn it on or off.
- One TP-Link Tapo C200 camera.

We were able to use all of these devices except the camera due to limitations in processing power offered by the Raspberry Pi that we used as our wireless access point.

Our application requires dealing with multiple PCAP files with a size around 1 to 3 Mo for a capture duration of 4 to 6 hours. However, the camera generates much more network traffic because it captures videos. This results in PCAP files of up to 100Mo for the same duration, and apparently the Raspberry Ri doesn't have the power to process such files and crashes. For this reason we decided to remove it from our testbed.

To simulate the total energy consumption of a smart home, we connected a power strip to a P110 plug, which we will call P110_total. This smart plug is always on and is only used to monitor the consumption. The remaining 3 devices were plugged directly into the power strip and we connected simple desk lamps to the two smart plugs to simulate possible real life applications. We conducted this experiment during 5 days. First, we analyzed the number of events detected by the system, compared to the actual number of events. Then, we measured the accuracy of the system to classify events. At the end, we measured the accuracy of the consumption profiles generated.

5.1.1 Events detection

As we already mentioned, the ability to detect user events is the most critical step in our application. This part alone influences the behavior of the system and its performance in the next steps. Therefore, we interacted with the 3 devices via our Home Assistant dashboard multiple times and monitored the response of the system a few hours later, after it completed its scheduled event identification task.

We only interacted with the devices by turning them on and off. It it possible to change the color and the brightness of the Alecto smart bulb, but only the brightness is integrated in Home Assistant. In preliminary tests, we tried to change the brightness but our application was not able to detect such event so we decided to put it aside. In the following, we compare the the results obtained from our system and our Home Assistant activity history. We identified the following metrics for our analysis :

- True positive : number of events that are recognized as such by the system and are actually present as user events in Home Assistant activity history.
- False positive : number of events that are detected by the system but that are not user events.
- False negative : number of user events that were not recognized by the system.

For the true negative, this number represents the number of non-user events that were correctly identified and filtered out by the system. Examples of such events are heartbeats or keep-alive messages. We will not mention this number here because it is not relevant for our analysis. In fact, this number is much higher than the others because the vast majority of events in the network traffic of a device are expected to be periodic and should be filtered by the system.

Device	**True positive**	**False positive**	**False negative**	**True negative**
Day 1				
Alecto_bulb	9	19	1	N/A
Woox_smart_plug	8	5	0	N/A
Tapo_P110	0	47	0	N/A
Day 2				
Alecto_bulb	3	0	2	N/A
Woox_smart_plug	5	10	0	N/A
Tapo_P110	3	0	1	N/A
Day 3				
Alecto_bulb	8	0	0	N/A
Woox_smart_plug	4	6	1	N/A
Tapo_P110	5	32	0	N/A
Day 4				
Alecto_bulb	2	2	1	N/A
Woox_smart_plug	2	8	0	N/A
Tapo_P110	4	36	1	N/A
Day 5				
Alecto_bulb	3	0	1	N/A
Woox_smart_plug	4	1	0	N/A
Tapo_P110	4	0	0	N/A
Total				
Alecto_bulb	25	21	5	N/A
Woox_smart_plug	23	30	1	N/A
Tapo_P110	16	116	2	N/A

Table 5.1: Events detection results.

As we can observe in Table 5.1, there are some expected and unexpected results. First, we can say that if we only focus on the true positive and false negative, the results show that the model accurately identify at least 85% of the events for two of the three devices.

We observe that the Woox_smart_plug is the device that performed the best at identifying user events, with an accuracy of 95.8% and only one event not recognized out of 24. On the other hand, The application was able to retrieve 25 of the 30 user events for the Alecto_bulb and the Tapo_P110 got 16 of the 18 user events.

False positive analysis

Looking at the false positive results, we can observe that the number of events detected by the system is significant. For the Alecto_bulb, 45.6% of identified events are false positives. Similarly, the Woox_smart_plug has 56.6% and the Tapo_P110 has 87.8% of non-user related events. We can explain these numbers by the instability of the network during the experiment. We analyzed the PCAP files of the devices using Wireshark to understand where these unknown events were coming from.

The device Tapo_P110, for example, has the highest number of false positive events in the four hours network traffic capture of day 1. In this capture, there are 2392 retransmitted TCP packets out of a total of 5773 packets for the entire capture. This alone highlights the instability of the network, which is a factor that we can not control. It is also one reason why the true positive and false negative values are equal to 0 for this device on day 1 because we simply could not interact with it.

A few TCP retransmission packets from time to time are not a real concern for our approach because such packets are discarded. Unfortunately, this amount of retransmissions highlights the difficulty for these devices to maintain stable TCP connections with other devices. Even if retransmissions are discarded, they introduce delay in packet transmissions, deviating from the normal behavior. As a result, the periodic models generated by analyzing the behaviors of the devices in the setup phase are useless because the network traffic is completely altered in an unstable network. Network issues, such as TCP packets retransmissions or duplicated acknowledgments are highly correlated to the number of false positive events. For the experiment of day 3 for example, we had a zero false positive for the Woox_smart_plug and only four for the Tapo_P110 when we interacted with the devices in the morning.

However, we let our application and the devices run idle in the afternoon, and all our devices became unavailable on our Home Assistant dashboard a few minutes apart from each other. After reviewing the PCAP files, we confirmed the same signs of instability in the network as previously. On day 5, there is only one false positive event detected by our application. In fact, no devices suffered from network instability issues that day.

5.1.2 Events classification

To evaluate the accuracy of our event classification models, we measure the number of events that are correctly classified. When detecting new events, the application will show them to the user, with a prediction if it is available, and ask for classification. Because the model require a minimum number of events, we only analyze the results of days 3, 4 and 5.

It is important to mention that even though the number of false positive events is high, none of these events were predicted as actual user events by our application. For this reason, we only focus on the prediction of true user events. The number of correctly classified events corresponds to the number of events that are ON (or OFF) and are classified as ON (or OFF). Similarly, incorrectly classified events are those that are classified as ON when they are actually OFF, or classified as OFF when they are actually ON. Finally, marked as unknown means that the models couldn't label the events. Table 5.2 summarizes the results.

Device	**Correctly classified**	**Incorrectly classified**	**Marked as unknown**
Day 3			
Alecto_bulb	3	0	5
Woox_smart_plug	3	1	0
Tapo_P110	1	4	0
Day 4			
Alecto_bulb	1	1	0
Woox_smart_plug	1	1	0
Tapo_P110	0	1	3
Day 5			
Alecto_bulb	1	0	2
Woox_smart_plug	3	1	0
Tapo_P110	3	0	1

Table 5.2: Events classification results.

The number of events used to train the models is limited, thus it is difficult to make conclusions. We can only observe that, because the data is limited, the models fail at classifying user events for the Alecto bulb. Specifically, the models can only classify ON events, but not OFF events. Hu et al.[10] mentioned that for some devices, it is impossible to accurately classify events. If this conclusion applies to the Alecto bulb, the only difference between ON and OFF events would lie in the payload of the packets. For the other two devices however, the classification models seem to improve over time, with 75% prediction accuracy for the Tapo_P110 on day 5 compared to 20% for day 3. Unfortunately, more data would be needed to confirm this.

5.1.3 Consumption profiles

For the energy consumption profiles that we generated, we could rely on the data of the Tapo smart plug that we used to simulate the total consumption of a house. From the total number of ON and OFF events for each device after the four days, only a subset of 8 values per device were valid.

Since the difference between the consumption before and after an event was too small in most cases, these values were discarded by our model to avoid noise. The reason is that the Tapo smart plug that we used is sending consumption data every 30 seconds by default, which can lead to missing precision. A problem that is avoided with real smart meters, which have much more precise monitoring capabilities. Nevertheless, the profiles generated in our application have an active power consumption of :

- 5.29 watts (W) for the Alecto smart bulb.
- 4.74 W for the Tapo P110 plug.
- 24.52 W for the Woox smart plug.

We can manually compare the correctness of these values using the tool of Home Assistant that displays both the activity history of the devices and the total electricity consumption of the house (i.e. The P110_total in our case), as you can see in Figure 5.1. As expected, the profiles generated by our approach effectively correspond to the actual consumption of the devices, with an error margin of ~ 0.3 W. Combined with a device state detection module that accurately detects and classify user events, this approach can successfully be used to predict the future consumption of devices.

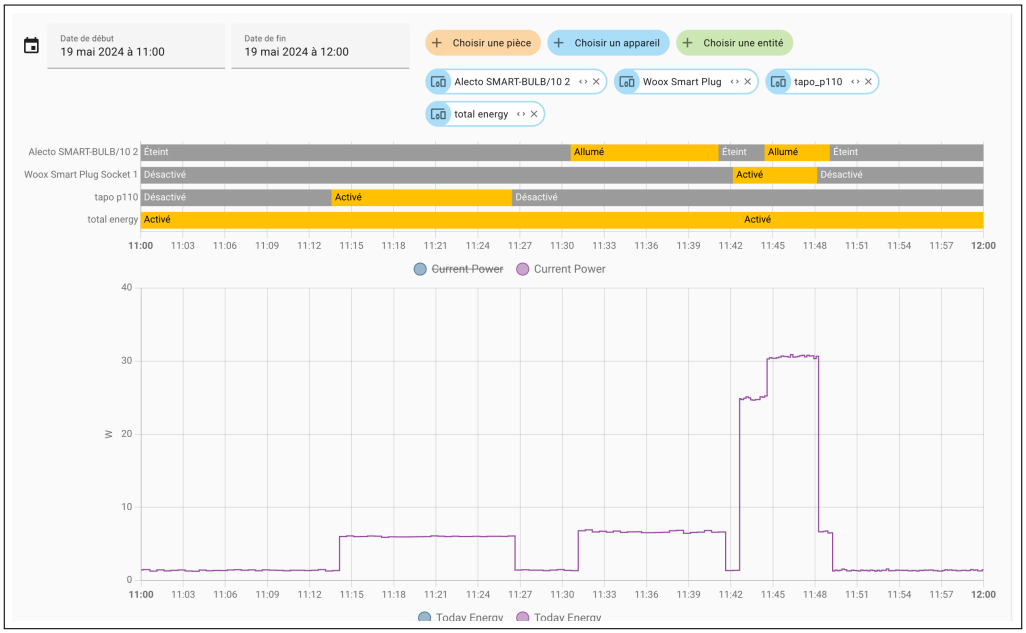

Figure 5.1: Screenshot of the devices activity in Home Assistant.

5.2 Smart home evaluation

For the smart home evaluation, we conducted an experiment where we had access to a real smart home and used our application for 4 days there. In this experiment, we analyzed the behavior of our system with two TP-Link smart plugs HS110 as well as two connected washing machines, a Miele WCF370 and a Miele TCC570. Due to challenges encountered in the activity prediction phase where the application detects events and asks the user to label them, we did not analyze other devices and we did not go further than that in the evaluation of our application in the smart home.

5.2.1 Washing machines

For the setup phase, we initially set the duration to three hours to capture the network traffic of the devices while they are idle. This duration was decided based on our observations and allowed us to effectively model the behaviors of the devices we tested. However, we didn't take into account the possibility of having devices with more complex behaviors and that require much longer network captures. Consequently, our application was detecting false user events every hour at the same time for these devices. We then doubled this duration to six hours for the setup phase to solve the problem.

During the 4 days, we could observe some events that were detected by our application. As shown in Figure 5.2, these events showed some periodicity by starting at a given hour, lasting between 1 hour and 30 minutes and 2 hours, and then stopping. It was actually because the washing machines were active and therefore sending data, possibly to give updates on their cycle status.

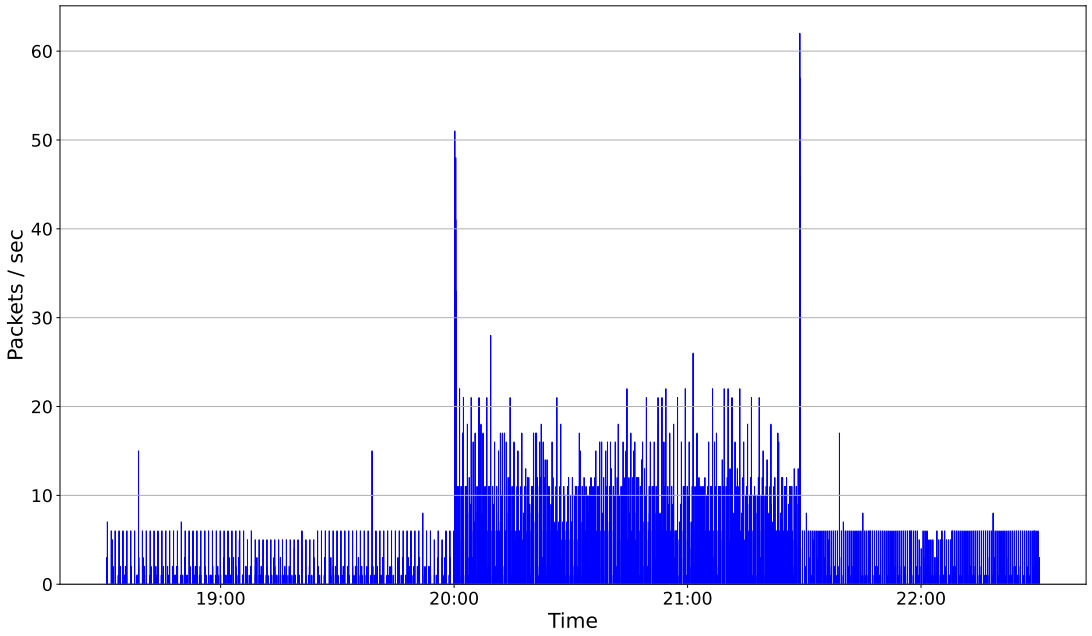

Figure 5.2: Network traffic of the Miele TCC570.

Our application detected 92 false positives for the Miele WCF370 out of a total of 1142 events. Based on our analysis, the remaining 1050 events are divided into two distinct time periods, where the machine was active. For example, 424 of these events occured between 10:00 and 12:15 on day 2 of our experiment. For the Miele TCC570, we identified 14 false positive events out of a total of 922 events. Similar to the previous machine, the remaining 908 events correspond to two distinct activities of the machine. Specifically, 489 events were identified between 20:00 and 21:30 on day 2, corresponding to a single activity, as depicted in Figure 5.2.

While this was not expected, it highlights the limitations of our system, and in particular the one we use from Hu et al[10]. We didn't consider the possibility of detecting events where devices send data periodically while they are active, rather than only when they change their state.

5.2.2 Smart plugs

For the smart plugs, we obtained completely different results. On one of the two plugs, we observed no events with our application. While this is surprising, the reason behind it is that the user only uses this smart plug for its energy consumption monitoring capabilities and does not activate or deactivate it remotely. Unfortunately, the energy consumption is stored in the payload of the packets so it is impossible for the system to determine events solely based on the payload.

On the other hand, no events detected also means that there were no false positive events either. For our first experiment, the real issue with our event detection approach was the network instability. This instability was changing the behaviors of the devices because they couldn't effectively communicate with others. Here, the network of the smart home seemed to be much more stable. We confirmed this by analyzing the PCAP files from the network captures and observed no signs of connectivity issues such as TCP packet retransmissions. As a result, it is expected to observe the absence of events if none were triggered by a user during our evaluation period.

For the second smart plug, we were only able to identify one ON and one OFF event, along with four false positive events. Similarly to the other smart plug, this device is mostly used for its electricity monitoring feature, to measure the consumption of the television connected to it.

The two user events that the application recognized were the only two events triggered by the user. For the false positive events that occurred, they all happened simultaneously, 1 second apart from each other, as the power consumption of the smart plug increased due to the activation of the television. Unfortunately we cannot confirm in any way the relation between the events that our system detected and the activation of the television connected to the smart plug.

Chapter 6

Discussion and limitations

In this chapter, we discuss the drawbacks and limitations of our application and improvements that could enhance the system. We first discuss networking issues and the problem of noise in network communications. Then, we describe the limitations of the system due to the complexity of some appliances. Finally, we talk about possible problems with our current approach to generate energy consumption profiles.

6.1 Network traffic

There are several factors that can negatively affect the performance of our solution, one of which is the network stability itself. In this section, we will discuss problems that we encountered in our evaluation and other limitations we haven't observe but that may arise.

6.1.1 Unstable networks

One objective of our research was to develop a solution that could predict accurately the power consumption of each device in a smart home. To do so, we utilized the information available in the network traffic of connected devices. When the network is stable, with no data loss and perfect communications between devices, the periodic models that we create are used effectively to filter the routine network traffic and extract user events. As a result, we expect the system to identify a very small percentage of false user events.

As we mentioned in Section 5.1.1, we observed that our event detection approach produced a large number of false positive user events. This unexpected behavior was due to the instability of the network.

To limit the number of false positive events, one solution would be to use additional information, such as the total energy consumption of the house, to determine if the consumption has changed at a given time or if the event should be ignored.

Fortunately, both the setup part where we capture the network traffic of devices while they are idle, and the identification of true user events were not impacted. Still, this problem of unstable network raises questions about the impact that this could have if the setup phase was also corrupted. In the worst case, the whole network traffic could be identified as user events, but we found no solutions to mitigate this problem.

6.1.2 Device updates

In our approach, we rely on a setup phase in which we capture the behavior of the devices only once in order to build our models. While this is not an issue in the short term, we may expect vendors to deploy firmware updates of their devices from time to time. If these updates change the communication patterns, protocols used, or other information, this could result in obsolete models. These models would no longer be able to correctly identify events.

One solution to this problem could be to develop a system that pre-trains behavioral models for popular devices in a lab and updates those models whenever device behavior changes due to firmware updates. Similarly, another solution is to periodically capture the network traffic and rebuild the idle system. This would allow the system to be updated frequently and resolve changes due to firmware updates. An example would be to start capturing between 1:00 and 7:00 for all devices once a month, or once every two weeks. To do this, someone would have to make sure that all devices are actually idle during the capture.

6.2 Appliances complexity

In this section, we discuss the limitations of our application to identify user events for complex devices as well as the difficulties to classify them correctly for some devices.

6.2.1 Events detection complexity

The results obtained in our experiments are quite satisfying when it comes to identifying user events for simple devices such as smart bulbs or smart plugs (and without taking the number of false positive events into account). These devices only have ON and OFF events that are composed of a single burst of consecutive packets exchanged with another device. When it comes to more complex ones such as washing machines, we observed that the task was more challenging.

Here, the washing machines are sending multiple bursts of packets at intervals of several dozens of seconds. Our approach, in its current form, is not able to identify these multiple events as a single event because of limitations. We do not have solutions for it and leave it for future work. However, an idea could be to combine all these events and develop a model capable to identify such patterns in the network traffic.

6.2.2 Events classification difficulties

For some devices, it is impossible to distinguish between user events because the difference is in the payload of the packets. While we haven't developed a solution for this, we do have one idea on how to address it, as we have access to the total power consumption of the smart home, which could be useful in this context.

In case of a tie, or when an ON and an OFF classification model have the same probability of classifying an event, we could use the available total consumption. To determine whether the event is ON or OFF, a solution could be to calculate the difference between the consumption before and after the event. If this value is positive, it means that the consumption increased and the device was turned on. If the value is negative, it means that the consumption decreased and the device was turned off.

6.3 Consumption profiles

The approach we chose to generate the consumption profiles is based on the total power consumption of the house. According to our experiments, this method effectively generates accurate profiles, but it has some limitations, which we will discuss here.

6.3.1 Devices used in groups

We can consider the behavior of our approach if it encounters multiple devices that are used simultaneously through home automation. Using various tools or systems, someone could set up a button that they press on their app to turn on multiple smart bulbs at the same time, for example. If these devices are only used through this mechanism, the total energy consumption would be the sum of the energy consumption of each smart bulb. However, since our system only generates consumption profiles and identifies events for each device, the profiles we generate may be inaccurate and correspond to the combined consumption of all the smart bulbs involved.

6.3.2 Energy meters precision

The precision of the energy meters used is also a limitation for our approach. If we consider the smart plug that we used to simulate the total power consumption of a house, this device is transmitting data every 30 seconds. Consequently, if two or more devices are activated within this 30 seconds time frame, the calculated value for the consumption related to the events may be incorrect. Therefore, the choice of smart meter and its transmission rate is important, to ensure the smooth running of our application and to avoid noise in the data.

Chapter 7

Conclusion

With the increasing number of smart homes and IoT devices within them, it is becoming important to monitor the energy consumption of all these connected devices to better understand and manage them accordingly. Unfortunately, most devices lack the ability to monitor their own energy consumption.

In this thesis, we propose a new approach where we analyze the network traffic of each device in a house and try to predict the states of these devices and their consumption. Our approach consists of three phases. The first one is the initialization phase. This phase consists in identifying the list of devices active in the network and capturing their network traffic while they are idle to establish their communication patterns. In the next phase, our application captures the network communications of all devices and identifies user events, such as turning on a light bulb. These events are then labeled by the owner of the house using his home activity history to train machine learning classifiers, which can then attempt to classify new events. In the final phase, the total energy consumption of the house is used to generate consumption profiles for the devices.

Our solution is efficient at identifying user events and generating the consumption profiles for simple devices such as smart plugs or smart bulbs. Unfortunately, our approach suffers from weaknesses, highlighting the limitations of the current state-of-the-art approaches. The stability of the network has a huge impact on our event detection approach, and it is also impossible for our application to detect complex events for devices such as washing machines. However, there are potential solutions to improve it, such as using the total energy consumption of the house or combining our approach with energy disaggregation techniques.

Bibliography

[1] Yourthings iot dataset, 2018. https://yourthings.info/data/.

[2] Abbas Acar, Hossein Fereidooni, Tigist Abera, Amit Kumar Sikder, Markus Miettinen, Hidayet Aksu, Mauro Conti, Ahmad-Reza Sadeghi, and Selcuk Uluagac. 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, 2020.

[3] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. Sok: Security evaluation of home-based iot deployments. In *2019 IEEE symposium on security and privacy (sp)*, pages 1362–1380. IEEE, 2019.

[4] Nazmiye Balta-Ozkan, Benjamin Boteler, and Oscar Amerighi. European smart home market development: Public views on technical and economic aspects across the united kingdom, germany and italy. *Energy Research & Social Science*, 3:65–77, 2014.

[5] Bruhadeshwar Bezawada, Maalvika Bachani, Jordan Peterson, Hossein Shirazi, Indrakshi Ray, and Indrajit Ray. Behavioral fingerprinting of iot devices. In *Proceedings of the 2018 workshop on attacks and solutions in hardware security*, pages 41–50, 2018.

[6] Stuart Cheshire and Marc Krochmal. Multicast DNS. RFC 6762, February 2013.

[7] Cisco. Address resolution protocol, 2024. https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipaddr_arp/configuration/15-s/arp-15-s-book/Configuring-Address-Resolution-Protocol.html#GUID-5F461B65-6268-453B-98FB-E751E74B460B.

[8] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

[9] Jinsoo Han, Chang-Sic Choi, Wan-Ki Park, Ilwoo Lee, and Sang-Ha Kim. Smart home energy management system including renewable energy based on zigbee and plc. *IEEE Transactions on Consumer Electronics*, 60(2):198–202, 2014.

[10] Tianrui Hu, Daniel J Dubois, and David Choffnes. Behaviot: Measuring smart home iot behavior using network-inferred behavior models. In *Proceedings of the 2023 ACM on Internet Measurement Conference*, pages 421–436, 2023.

[11] IBM. What is the internet of things (iot)? `https://www.ibm.com/topics/internet-of-things`.

[12] IETF. Dhcp options and bootp vendor extensions, March 1997. `https://www.ietf.org/rfc/rfc2132.txt`.

[13] François De Keersmaeker. Github repository : fork of the code for the imc23 paper dubbed behaviot. `https://github.com/fdekeers/BehavIoT`.

[14] Jack Kelly and William Knottenbelt. Neural nilm: Deep neural networks applied to energy disaggregation. In *Proceedings of the 2nd ACM international conference on embedded systems for energy-efficient built environments*, pages 55–64, 2015.

[15] Markus Miettinen, Samuel Marchal, Ibbad Hafeez, N Asokan, Ahmad-Reza Sadeghi, and Sasu Tarkoma. Iot sentinel: Automated device-type identification for security enforcement in iot. In *2017 IEEE 37th international conference on distributed computing systems (ICDCS)*, pages 2177–2184. IEEE, 2017.

[16] Progress Mtshali and Freedom Khubia. A smart home energy management system using smart plugs. In *2019 Conference on Information Communications Technology and Society (ICTAS)*, pages 1–5. IEEE, 2019.

[17] Institute of Electrical and Electronics Engineers (IEEE). Organizationally unique identifier public listing. `https://standards-oui.ieee.org/`.

[18] Stanislav Safaric and Kresimir Malaric. Zigbee wireless standard. In *Proceedings ELMAR 2006*, pages 259–262. IEEE, 2006.

[19] Benjamin K Sovacool and Dylan D Furszyfer Del Rio. Smart home technologies in europe: A critical review of concepts, benefits, risks and policies. *Renewable and sustainable energy reviews*, 120:109663, 2020.

[20] Statista. Number of internet of things (iot) connected devices worldwide from 2019 to 2023, with forecasts from 2022 to 2030. `https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/`).

[21] Statista. Number of users of smart homes worldwide from 2019 to 2028. https://www.statista.com/forecasts/887613/number-of-smart-homes-in-the-smart-home-market-in-the-world).

[22] Rahmadi Trimananda, Janus Varmarken, Athina Markopoulou, and Brian Demsky. Packet-level signatures for smart home devices. In *Network and Distributed Systems Security (NDSS) Symposium*, volume 2020, 2020.

Use of generative AI

ChatGPT was consulted during the writing of this thesis report to check and improve the wording of sentences but did not contribute to the generation of new content for this work.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**