Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Helene Skjelsbæk Knudsen

Private Information Inference Based on Network Traffic Patterns of Air Quality Monitors

Master's thesis in MISEB
Supervisor: Jia-Chun Lin
June 2023

NTNU
Norwegian University of
Science and Technology

Helene Skjelsbæk Knudsen

Private Information Inference Based on Network Traffic Patterns of Air Quality Monitors

Master's thesis in MISEB
Supervisor: Jia-Chun Lin
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

Private Information Inference Based on Network Traffic Patterns of Air Quality Monitors

Helene Skjelsbæk Knudsen

01.06.2023

Title: Private Information Inference Based on Network Traffic Patterns of Air Quality Monitors

Student: Helene Skjelsbæk Knudsen

Problem Description:
The Internet of Things is becoming more widespread and available for everyone. Home appliances that normally were not connected to Internet of Things systems are being given functionality by adding sensors and technology. The added functionality can be used to communicate with other devices, servers in the cloud or carry out automated tasks based on own decisions. Manufactures of these devices are constantly adding functionality and new devices to keep up with the increasing demand from users wanting to create smart homes. Turning on the heat at home from another location through an application or having the doorlock send notifications when it is opened or locked are examples of functionality that makes up the Internet of Things.

Air quality monitors are a type of Internet of Things devices that monitor the indoor climate in it's installed environment. The devices can communicate over different communication protocols to applications on a user's phone or the vendors cloud services. In addition, their functionality varies as they are equipped with different sensors, giving the user different values of how good or bad their indoor climate is. The monitors can communicate with other devices and be integrated in a smart home environment, such as fans or heaters. However, having devices monitoring our home 24/7 arises security issues. Private information about user behaviour can be attractive to different parties, from companies wanting to know how to target advertises to malicious actors that want to misuse the information.

This thesis will investigate several different air quality monitors to see how and what private information can be gathered and inferred from carrying out a passive network eavesdropping attack. As the market contains a wide variety of air quality monitor devices, devices from different vendors should be chosen to carry out the attack and discover what kind of private information in a smart home environment can be collected.

Supervisor: Jia-Chun Lin, NTNU

Abstract

The emerging use of IoT devices in homes arises security concerns whether or not these devices expose private information about users and their environment. Air quality monitors are a type of IoT devices which are always on to monitor the indoor air quality of the environment. Even though air quality monitors have been included in several tests within security, there are research gaps in comparing different devices to each other and conducting test cases specifically designed to see if there are differences in the level of inference on the air quality monitors. This master thesis carries out a passive network eavesdropping attack on three different air quality monitors to investigate if the network traffic pattern changes during a triggered event and if these patterns can be used to infer user activities. The devices communicate over Wi-Fi and are manufactured from different vendors. The research first presents a baseline capture of the devices to learn their traffic patterns when events are not triggered. Then four different test cases were tested on the devices to see if it is possible to infer private information by looking at the corresponding network patterns. The test cases designed in this thesis are cooking, showering, window open during night and weekends at home or gone. The results showed that only one of the air quality monitors expose private information in one of the test cases. For this device it is possible to infer whether a user is home or gone by looking at bytes sent and received and the differences in packet sizes. This private information can be misused by an adversary to know when a user is home or not just by looking at the Wi-Fi traffic sent to and from that device. This research demonstrates that air quality monitors communicating over the same protocol, in this case Wi-Fi, have different traffic patterns and expose private information differently. It also motivates to continue the research on air quality monitors, as these have not been as popular to test as other IoT devices.

Sammendrag

Den økende bruken av IoT enheter i hjem skaper sikkerhetsutfordringer om disse enhetene avslører privat informasjon om brukere og hjemmemiljøet. Luftkvalitetsmålere er en type IoT enheter som alltid er påskrudd for å måle luftkvaliteten innendørs. Selv om luftkvalitetsmålere har vært inkludert i flere sikkerhetstester, er det mangler i forskningen på å sammenligne ulike enheter med hverandre og å utføre tester som er spesifikt designet for å se om det er forskjeller i hvor mye privat informasjon enhetene avslører. Denne masteroppgaven utfører et passivt nettverksavlytnings angrep mot tre ulike luftkvalitetsmålere for å undersøke om mønstre på nettverkstrafikken endrer seg når en test utføres og om dette kan brukes for å hente ut privat informasjon for å fange opp brukeraktivitet. Alle enhetene kommuniserer over Wi-Fi og er laget av ulike produsenter. Oppgaven presenterer først standard trafikk fra enhetene for å se på mønstre på nettverkstrafikken når det ikke blir utført spesifikke tester i miljøet. Deretter ble fire ulike tester utført i miljøet til enhetene for å se om det var mulig å hente ut informasjon om hva som foregikk i miljøet. De fire testene som ble utført er *lage mat, dusje, ha vindu åpent på natten og helg hjemme eller borte*. Resultatene viste at kun en av luftkvalitetsmålerne avslører privat informasjon fra en av testene utført. Fra trafikken på denne enheten er det mulig å se om en bruker er hjemme eller ikke ved å se på bytes sendt og mottatt og forskjeller i pakkestørrelse. Denne informasjonen kan bli misbrukt av en ondsinnet aktør ved å vite om brukeren er hjemme kun ved å se på trafikken sendt over Wi-Fi, til og fra enheten. Testene viser også at ulike luftkvalitetsmålere som kommuniserer med samme protokoll, i denne sammenheng Wi-Fi, har ulike mønstre på nettverkstrafikken og avslører privat informasjon ulikt. Dette motiverer også til å fortsette testingen på luftkvalitetsmålere, siden disse ikke har vært like populære å teste som andre IoT enheter.

Preface

This document serves as my Master Thesis in Information Security and concludes my 3-year degree at the Norwegian University of Science and Technology, NTNU, department Gjøvik.

I would like to thank my supervisor, Jia-Chun Lin, for being a great support and challenging me to learn new things during my research and writing process. I would also like to thank everyone in our IoT-group: Benjamin Ulsmåg, Mathias Hedberg, Kevin Nordnes, Lloyd Nicolay Gustavsson, Aafan Ahmad Toor, Ming Chang Lee and Ernst Gunnar Gran for sharing their knowledge and being discussion partners througout this project.

At last, I would like to thank my partner Benjamin for supporting me in all phases of the thesis. I could not have done it without you.

Helene Skjelsbæk Knudsen, 01.06.2023

Contents

Figures

Tables

Acronyms

AP Access Point. 11, 12

AQM Air Quality Monitor. 1, 3, 8, 16, 19, 20, 22, 24, 25, 27, 94, 117, 119

ARP Address Resolution Protocol. 12

BSS Basic Service Set. 12

CRC Cyclic Redundancy Check. 12

DNS Domain Name System. 15

HTTPS Hypertext Transfer Protocol Secure. 15

ICMP Internet Control Message Protocol. 15

IoT Internet of Things. 1, 3, 4, 7, 9–11, 15–20, 123, 126

IP Internet Protocol. 12

ISP Internet Service Provider. 12

MAC Media Access Control. 3, 12, 15, 24, 32

pcaps Packet Capturings. 30, 33, 44

RQ Research Questions. 2, 119–121

SD Standard Deviation. 102, 106, 110, 114

SSID Service Set Identifier. 12

TCP Transmission Control Protocol. 15

TVOC Total Volatile Organic Compounds. 8, 20, 21, 27

UDP User Datagram Protocol. 15

VOC Volatile Organic Compounds. 8, 27

VS Code Visual Studios Code. 30

Wi-Fi Wireless Fidelity. xi, 3, 4, 7, 10–13, 17–22, 24, 25, 28, 35, 119, 126

WPA Wi-Fi Protected Access. 12

Chapter 1

Introduction

This chapter introduces the master thesis while presenting the background and motivation, followed by the research objectives and research questions that will be answered throughout this research. Scope and delimitation gives a clear understanding of what is included and not in the thesis. The thesis structure outlines and gives a brief understanding of each of the following chapters in this thesis. The contributions are presented in a separate subsection.

1.1 Background and Motivation

The Internet of Things (IoT) exists of a growing number of physical and virtual devices connected to the Internet to perform smart tasks [1]. Every-day devices can be equipped with smart functionality to improve our lives, but also to improve critical societal functions such as in health care or industrial technology. The devices can range from a robot vacuum cleaner that users can control through their phone or cameras installed for elders to stream to a nurse who resides centrally. These smart devices can communicate and connect to each other and other services using the Internet and makes out an IoT system. The devices analyzes how users, machines or eco-systems behave and act accordingly. An emerging request for smart devices has resulted in a rapid growth in IoT devices worldwide [2]. The devices are becoming more user friendly, smarter with added functionality and aesthetically and more suitable to place or wear in any environment.

We spend a lot of our lives inside, breathing in the air that is available in the indoor space [3]. The air affects our health and can potentially cause chronic health problems, for example lung cancer or respiratory infections [4]. Common air pollution's, such as smoke or car exhaust, are easy to sense and avoid for people not trying to get effected by the dangerous particles they emit. It is also more wide-known that good outdoor air is beneficial for our health, not considering that the air indoor can also severely affect our health [5]. Being more aware of our indoor environment and considering the fact that Internet of Things devices are evolving rapidly, indoor Air Quality Monitor (AQM)s are increasing in popularity and functionality [6]. The air quality monitors are also developing into becoming

smaller, more affordable and appealing to include in our home environment while adopting several different sensors to report on the indoor air quality trying to become a more popular choice for users.

As users are installing these sensors inside their own homes and allow them to monitor their home environments all day, the air quality monitors will be collecting data about the environment and behaviour that affect the air quality monitor sensors. Therefore, it is interesting to look further into how easy it is to collect this data and infer what kind of user behaviour is ongoing in the environment. As harmless as a passive sensor that is just collecting data about different indoor climate rates may seem, it is important to understand the risks one takes when installing these and connecting them to the Internet. Understanding what kind of private information is possible to infer from these devices and what makes the differences can be crucial when deciding which air quality monitor on the market to buy and install in our home.

1.2 Research Objectives and Research Questions

This thesis will conduct a network attack called *passive network eavesdropping attack*, and launch it against a group of individual air quality monitors residing in a home environment. In order to decide which devices to use and how to carry out an attack, a survey of the devices will be presented. A justification for which test cases to trigger the sensors and devices is important when analyzing the results. When the passive network eavesdropping attack has been carried out and data from the different test cases are collected, the results will be analyzed to see if and how much private information can be gathered from the different devices. The results will also look into if there are significant differences between the air quality monitors. Lastly, the research will investigate how the private information inferred can be used by malicious actors in a harmful way for users having the air quality monitors installed in their home.

Based on the problem description, motivation and research objectives, the following Research Questions (RQ) have been raised and will be answered throughout this master project:

- **RQ1:** What kind of information can be gathered from air quality monitors when carrying out a passive network eavesdropping attack?

- **RQ2:** What are the differences in level of inference on different air quality monitors from different vendors?

- **RQ3:** How can the private information gathered be misused by an adversary?

1.3 Research Scope and Delimitation

This research presents three different air quality monitors, all selected from different vendors. The AQMs communicates over the same protocol, Wi-Fi, but have different functionalities, applications and sensors. A Wi-Fi sniffer together with *tshark* running on *Kali Linux* will be used to capture and store wireless traffic. To answer the research questions, a baseline traffic pattern will be compared to traffic during triggered scheduled user events. The goal is to investigate what kind of private information it is possible to infer from conducting a passive network eavesdropping attack on the different devices. The analysis and evaluation will be based on network traffic patterns and give an understanding of how this information can be misused.

This thesis will not look into decrypting traffic if the air quality monitors encrypt the communication. The focus will be on conducting a passive network privacy inference attack based on network traffic patterns and therefore only look at non-encrypted data whether it is the whole packet or only the header. The research will not look into different factors of how to do a successful attack, such as distance, materials of the building or signal strength of the sniffer. The thesis will not cover all phases of a passive network eavesdropping attack, but a prerequisite of this is that the attacker has gained a strong enough wireless access to the user's network and can read traffic sent from and to the devices in the environment. We will not look into how to identify the IoT devices as they are already known with Media Access Control (MAC) addresses in this research. But for readers information, there are other researches that have looked into identifying different IoT devices, such as in [7] and [8].

1.4 Contributions

This thesis contributes with research on different individual air quality monitors and what kind of private information that can be inferred from them. Even though there are other researches that have carried out a passive network eavesdropping attack trying to infer private information from IoT devices, a lot of these researches investigate on IoT devices that can clearly pose a threat to user's privacy if inferred, such as cameras, watches or motion sensors. In a lot of researches found online, air quality monitors are often a part of an IoT environment and only one type of air quality monitor is included in the same environment, making it harder to compare the differences between different brands, manufactures, functionality and sensors.

Research on specifically air quality monitors is popular when it comes to their functionality and sensor and how good they can read the indoor environment, but when looking into security and privacy of the devices, the research decreases significantly. Therefore, will this research contribute to not only looking into if a passive network eavesdropping attack can expose private information, but also if there are differences when multiple indoor air quality monitors are placed to

observe the same environment and are exposed to the exact same test cases.

1.5 Thesis Structure

This rest of this thesis is structured in the following chapters:

Chapter 2 - Background
The background describes important information applicable for the research and results. General information about air quality monitors, how they work, how the sensors work and what factors to consider when selecting which air quality monitor is presented. The concepts of private information inference, passive network eavesdropping and Wi-Fi are explained as these are relevant for the tests conducted.

Chapter 3 - Related Work
The chapter presents a selection of previous research done by others on the problem topic. Security and privacy issues of specifically air quality monitors are highly weighted in this chapter. The chapter also presents research done on misusing of private information on other IoT devices, including smart environments with several different IoT devices.

Chapter 4 - Method
The method describes how this research has been carried out. A survey for choosing specific air quality monitors are presented with a description of the devices. To ensure reproducibility, the environment setup is shown with all the components used. The test cases are presented by giving justification on why they were chosen, how they are designed and details on how they are carried out. A description on how the data captured are analyzed is also included in the method.

Chapter 5 - Evaluation and Analysis Results
The chapter presents the results from the research to answers the research questions. A general procedure applicable for analyzing all tests is presented. Then a subsection with standard baseline traffic from the devices is presented. The results from the tests are presented in separate subsections.

Chapter 6 - Discussion
This chapter answers the research questions defined in this thesis. It also discusses the work and results for the tests carried out. The challenges and mistakes and how this thesis could have been done differently will be brought to light, but also strengths and decision that were made to be able to discover the findings in the evaluation and analysis results.

Chapter 7 - Conclusions and Future Work
This chapter concludes the research by answering the research question in short

term and summing up this master thesis, while focusing on the contributions made. Included in this chapter is also future work suggesting how to further look into the research topic.

Chapter 2

Background

This chapter gives an overview of the necessary background information required for this research. Air quality monitors are explained in general with its use, capabilities and sensors. Passive network eavesdropping and private information inference are introduced as these will be the vulnerability exploited in this research. Lastly, an overview of the communication protocol IEEE 802.11, Wi-Fi, is presented as this is where the passive network eavesdropping attack will be launched.

2.1 Air Quality Monitors

Air quality monitors are used as sensors to collect sensor data from different sources in the air [9]. The IoT devices can store data in different ways. Cloud storing is the most preferred storage solution for IoT devices, but also local servers, internal storage and memory cards can be used to store data from air quality monitors [10]. Displaying data to users of air quality monitors is important to give the user either information about the indoor air quality, or recommendations on how to improve the air quality [10]. The most developed and recent method of doing so, is through an application on the users phone, but also solutions where users can login to a website exists. Many air quality monitors also have a screen on the device that shows certain values from the sensors [10].

As indoor air quality monitors can be equipped with different features, a brief explanation of some of the sensors on an air quality monitor is necessary to understand why and how private information can be gathered from these sensors:

- **Carbon Dioxide (CO_2)** is a chemical formula that is mainly made from human or animal combustion [11]. This implicates that the more humans or animals that are in the same indoor environment, the higher occurrence of CO_2 will be collected by sensors and transmitted to the air quality monitors. However, plants, sunlight and ventilation can bind CO_2 and reduce the amount of CO_2 particles in an indoor environment.
- **Noise** is an interesting feature of air quality monitors as it is considered a health problem [12]. Exposure to loud noises at a small amount of time or

long-term noise can harm peoples health and result in hearing difficulties. As we use a lot of time in our home, this sensor is applicable here.

- **Volatile Organic Compounds (VOC)** is a collective term for any combination of carbon, with the exception of some compounds [13]. These harmful compounds can be found in gases from building materials, cigarettes, cleaning articles, painting or cooking to mention some. Total Volatile Organic Compounds (TVOC) is a term for defining the total amount of VOCs. The amount of VOC in an indoor environment can be reduced by ensuring fresh air or using kitchen fans [14]. HCHO, also known as formaldehyde, is classified as a VOC [15]. eCO_2 is often used as an AQM measure, with means estimated concentration of CO_2 from TVOC.
- **Humidity** is calculated from the ratio between water vapor in the air compared to the maximum amount of water vapor possible in the air [14]. Even though humans can endure high variations in humidity, low humidity can result in health problems such as irritated eyes, dry skin or dry mocus membranes. High humidity can begin mold processes that can damage indoor furniture or walls and lead to health problems like asthma or allergy. User behaviour that can affect the humidity includes taking a shower, drying clothes or cooking.
- **Temperature** is a measure unit for how hot or cold an environment is and is measured using a thermometer. It is the most commonly used unit for how comfortable humans feel. A too high temperature can result in lack of energy and sleepiness and too low temperature can result in reduced muscle function or heighten symptoms of rheumatism [16].

Several factors play a key part when selecting which air quality monitor is most suitable for the users needs. Considering the different functionality an air quality monitor can have, it is also important to look into transmission range, power consumption and maintenance [10]. However, the main goal of an air quality monitor is to monitor the air and therefore the sensors on the devices should be a top priority when selecting a device. Air quality monitors can specialize in one measuring unit in the air or have the functionality to measure several air quality factors. They will be incorporated in users homes and therefore the appearance will also be a factor when choosing the right device [17].

There are several companies that manufacture and sell air quality monitors. Some companies sell air quality monitors whose main goal is to monitor the indoor air quality, such as Airthings [18], Netatmo [19] and Mill [20]. Other companies integrates the air quality monitor with other devices that have a main goal of changing humidity or CO_2 levels in the indoor environment based on the sensor values to better the indoor air conditions. Examples of these are Phillips [21] and Dyson [22]. These companies have different air quality monitors that users can choose from based on their needs.

Considering security in air quality monitors, the standard and functionality varies [23]. Some vendors of air quality monitors use secure communication channels and authentication to access data, while others do not have authentication

enabled to collect data from their devices. However, as technology evolves to support more efficient and greater computing power for the air quality monitors, it becomes easier to develop and implement stronger security. One the other side, air quality monitors are beginning to become more popular as they are increasing in number of sensors and functionality, which is often prioritized over security and privacy issues [6].

2.2 Private Information Inference

Considering the amount of information an air quality monitor can collect about an individual or a smart home, privacy leakage is a vulnerability [24]. Privacy is referred to as every person's right to have control over their own data, hereby knowing how their data is used or distributed to other parties [25]. When malicious attackers try to gather sensitive information about an individual user or group of users, it is so called a privacy attack [26]. The attacker aims to target the confidentiality of the user, while gathering information such as location, preferences, personal behaviour or similar private information. A challenge with an IoT system compared to other computer systems is that many of the IoT devices are always on, connected in a user's home and sense user behaviour in a passive and non-intrusive way. This makes it even more difficult to understand the scope of all the private information gathered by an IoT system [25].

IoT systems in smart homes are often made up by different brands and communication protocols and does not follow one standard like IoT systems in an industrial environment can do [27]. There is no central management of security patching or having a strong baseline of security that protects the private data stored on a sensor. A big threat to IoT privacy is identification and profiling, where an attacker can link a user to their behaviour in the environment [25]. An adversary can then study the data from a user's environment and identify what that user is doing. An example can be that by reading an increase in data from an IoT device communicating to a server for IoT coffee appliances, an attacker can know that the user is making coffee at a specific time of day. By collecting this data over a longer period of time, the attacker will know the routines of a user. Even though a morning coffee can be something a user posts openly on social medias, not having control of the data and who can access and process it is a violation of privacy. This could also be launched against several IoT devices in an environment at once and knowing user specific routines.

A challenge when it comes to privacy for IoT systems is weighting functionality against privacy and security [28]. There are security measures that can be implemented for the devices to mitigate private information inference, such as authentication and authorization, data anonymization, and cryptology. However, not only attackers wanting to know about user's private information poses a threat to users' IoT data, but also interested parties wanting to know user behaviour to make more suitable solutions and targeted advertisement to make more money. Hence, users need to be aware of where their data is shared to ensure that private

information is not shared without them knowing. On the other side, users are requesting better functionality for the IoT devices, wanting them to behave more customized and optimized to their needs which requires more data to be shared. Although security vs privacy is a known issue in the computer world, IoT devices are resource restrained and preferably as ascetically integrated in user's home as possible. Therefore, it is a challenge applying security measures to ensure privacy for these devices [28].

2.3 Passive Network Eavesdropping

Eavesdropping network communication can either be passive or active. When an attacker conducts an active eavesdropping attack on a target, the data in the communication link is both inferred and modified [29]. However, in a passive network eavesdropping attack, the eavesdropper does not modify any data on the link, but simply gathers data transmitted without changing the communication. The overall goal of eavesdropping is often to access private information that is sent on the communication channel. This can be anything from credentials to secret messages, or as in this research, private information about users and their habits [29]. Since the communication is not affected by a passive network eavesdropping attack, the attacker does not need to be as careful in disguising itself as in an active eavesdropping attack.

There are security measures to reduce the chance of being eavesdropped, both actively and passively. Since eavesdroppers aim to listen to the communication, encrypting the traffic is an effective way to make it harder to conduct an eavesdropping attack. Also, coverting channels to hide the identity of the communicating parties can be used to prevent the eavesdroppers from finding the right communication channel to eavesdrop [29].

To perform a passive network eavesdropping attack on wireless communication channels, it is not necessary to join the targeted network, but the attacker needs to be within the signal range of the network [30]. In order to carry out a passive network eavesdropping attack, the right hardware and software must be configured [31]. A wireless network sniffer can collect wireless traffic for the corresponding communication protocol within its signal range. This device needs to be configured to capture packets that are not meant for the device it is connected to, normally a computer or server, but every packet within its signal strength. This is often called promiscuous or monitoring mode.

It exists a number of network sniffers for every wireless communication protocol, such as Wi-Fi, Bluetooth, ZigBee or Z-wave [31]. Some computers even have a built-in wireless network card that can be used to monitor traffic, not directed to the computers address. Once the sniffer is configured to capture packets, a monitoring tool is necessary to collect, store and analyze the packets. Different software monitoring tools can be used for this purpose, and it is often recommended to use specific ones based on the operating system of the system that carries out the attack. A common choice for both Linux and Windows is Wireshark [32], which is

a free monitoring tool. With both the hardware sniffer and software monitoring tool configured correctly, a network eavesdropping attack can be carried out.

2.4 Communication Protocols

Air quality monitors send their sensor data on the network to store and analyze it and communicates with a hub or a users phone to display the sensor values. As they exist with different functionality and specifications, their communication protocols differs, like other groups of IoT devices [10]. Wi-Fi is the most preferred protocol with Bluetooth and ZigBee following [33]. As this thesis is limited to air quality monitor devices using Wi-Fi to communicate, the next subsection will elaborate on this protocol.

2.4.1 IEEE 802.11 - Wi-Fi

Wi-Fi [34] is one of the most used technologies for communicating and it is defined, developed and standardized by WiFi Alliance [34]. Wi-Fi is based on the standard IEEE 802.11 Wireless LAN set by IEEE Standard for Information Technology [35]. The transmission range for Wi-Fi is up to 100 meters and it uses 5-60GHz in the frequency band [17].

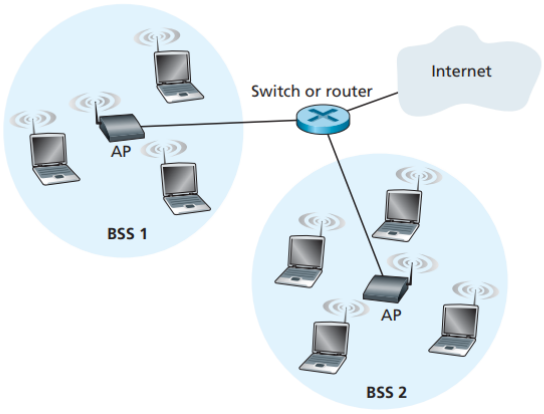


Figure 2.1: A Wi-Fi network [36]

Figure 2.1 illustrates how a Wi-Fi network works. In a Wi-Fi local area network each communicating node, such as computers, IoT devices or mobile phones, connects to an Access Point (AP), which connects the devices to the Internet [36]. The AP works as a base station for the devices and provides the basic components that

makes up the fundamental for Wi-Fi, called Basic Service Set (BSS). The AP connects to the Internet commonly using a switch or router, which can be integrated in the same physical device. When connecting to a Wi-Fi network, a Service Set Identifier (SSID), often set as a default name by the Internet Service Provider (ISP) or a customized name made by the network administrator, is visible for devices trying to connect to the network. The SSID is used to distinguish multiple APs to choose the right one to connect to. Each AP will periodically send out broadcast messages in a beacon frame containing its MAC address and SSID. When the device has chosen the right AP to connect to based on the beacon frame, an association begins between the AP and the device to create a wireless link. This link will be used by the AP to send packets from the Internet directed to the device and from the device to Internet [36].

A network packet sent from a device communicating in a Wi-Fi network is called a data frame [36]. The data frame is illustrated in Figure 2.2. The frame control field contains several different fields for encryption, acknowledgment and association to mention some [36]. The duration field holds the value of how long to reserve a channel to transmit the data frame and acknowledgement. Sequence numbers are used by a receiver to be able to make up the order the packets are sent if they arrive in the wrong order over the medium. As shown in Figure 2.2, there are 4 different MAC address fields with values such as source and destination MAC addresses [35]. The Cyclic Redundancy Check (CRC) field is controlled by the receiver to check for bit errors. The final field is the payload where the transmitted data relies in an Internet Protocol (IP) or Address Resolution Protocol (ARP) packet [36].

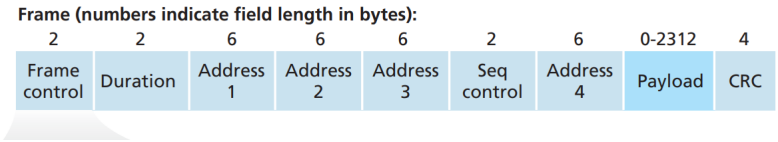


Figure 2.2: Wi-Fi data frame [36]

An attacker sniffing network traffic on a Wi-Fi network will observe different packets. Wi-Fi uses three different categories of frames: management, control and data frames [37]. Management frames are used to manage Wi-Fi traffic. When connecting to a network, association and authentication packets are sent, and when disconnecting, disassociation and de-authentication packets are transmitted. Beacon and probe frames are also sent to find the right network and manage the connection. The control frames are used to assist both management and data traffic [38]. The frames do not contain any body, but rather header information that can be used to ensure management and data frames. The last group of frames sent on a Wi-Fi network is data frames. The data frames is where the actual data that the the parties wants to transmit to each other relies.

Security mechanism on Wi-Fi called Wi-Fi Protected Access (WPA)-2 and WPA-

3 are commonly used and have stronger encryption and integrity where defense against attacks are implemented. This thesis will not investigate any further on security mechanisms of Wi-Fi, but more information about security on Wi-Fi can be found in [39].

Chapter 3

Related Work

This chapter presents relevant research within this thesis' research scope. The main emphasis is on research done on security and privacy on air quality monitors, but it is also relevant to look at security and privacy issues for IoT devices. A larger scope of research is published on IoT devices in general and they can hold many of the same functionalities and vulnerabilities.

3.1 Security and Privacy Issues of Air Quality Monitors

A research by Sivaraman et al. [40] tested several security parameters on different IoT devices which included two types of air quality monitors, Awair Air Monitor and Netatmo Weather Station. The devices confidentiality, integrity, authentication, access control and a reflection attack were tested. In the tests for confidentiality, both the devices passed. However for the integrity and authentication tests both air quality monitors failed tests for Domain Name System (DNS) security and DNS spoofing. Netatmo Weather station passed the applicable tests for access control, but the Awair Air Monitor had open ports for Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) and were vulnerable for Internet Control Message Protocol (ICMP) and UDP distributed denial of service attacks. The Awair Air Monitor were also vulnerable for ICMP Reflection attacks [40]. The results of this research emphasizes the need to investigate the same IoT devices from different vendors as they can expose private information differently even though they have the same functionality.

A low-cost air quality monitoring system, called a.com, is analyzed in [6] by Luo et al. The authors used a Hypertext Transfer Protocol Secure (HTTPS) proxy tool called mitmproxy to analyze the security, data integrity, architecture, and communication of a.com. Their results shows that this particular system use unencrypted communication and MAC addresses to identify the sensors. Two attacks were carried out, man-in-the middle and populating false data in the communication link. These vulnerabilities are a big security challenge, as it is possible for an attacker to falsify any a.com device if spoofing their MAC address [6]. To be able to do a man-in-the-middle attack, an active sniffer is used and the result shows

security vulnerabilities for an air quality monitoring system for a more comprehensive attack than passive network eavesdropping attack.

A literature review by Zagi et al. [41] collects and presents the results of several IoT privacy related articles. The comparison shows that IoT devices have several privacy issues that needs to be addressed. Even though this research is not specifically on air quality monitors, it is applicable to general IoT devices and therefore relevant to this master thesis. The research highlights that security in IoT networks is a challenge. Since the devices communicate over different protocols, it is stated in the research that one standard secure channel for all devices is not realistic. Another security issue brought up is errors in configuration. Users often buy off-the-shelf devices and install themselves, which can lead to weaknesses in the system. In addition, resource constraints in IoT devices can effect the possibility and means to incorporate security features in the devices [41].

A risk when indoor AQMs monitor users environment is that private information about users living or working habits can be leaked [17]. A research by Zhao et al. [17] reviews several aspects of security in communication protocols for indoor air quality monitors. The change in CO_2 in a users environment can reveal information about when a user is sleeping or working. Such information can be misused by a malicious attacker. The research investigated 91 papers and only 2 of them looked into the challenges and possible solutions to data privacy-related issues. The lack of research on security for AQMs shows the need to provide more specific research on air quality monitors and their security and privacy issues.

A decentralized framework for wireless communication protocols for air quality monitors is proposed by Mrissa et al. [42]. The framework uses encryption and onion routing technology to send data. The onions that transmit the packets do not access the encryption keys that encrypts the body of the onion, as the keys are only included in the last layer of the onion message. The framework can therefore protect against malicious attackers eavesdropping private information from the air quality monitors. The sensors nodes communicates with sink nodes that initiates the communication, which means that a change in the environment for the user will not result in data being transmitted immediately. Other techniques such as padding, data-link-layer encryption, timing intervals and randomized paths are also used to prevent eavesdropping [42]. However, it is not mentioned to what degree real-life air quality monitors use this framework.

3.2 Misusing of Private Information Using Data from IoT Devices

A research conducted in [43] investigates one IoT hub communicating with up to 16 different IoT devices. The results showed that it is possible to infer up to 90% of user behaviour with a passive eavesdropping attack. Even though the traffic between the devices and the hub was encrypted, they were able to infer the users action by comparing the users action with the observed encrypted application data

sizes received by the gateway.

Ziegeldorf et al. aims to classify different privacy threats and challenges for IoT devices [44]. They suggest seven different categories: identification, localization and tracking, profiling, privacy-violating interaction and presentation, life-cycle transitions, inventory attack and linkage. The threat of identification refers to collecting data about an individual that can be used to identify that person, such as an image, name or address. Location and tracking is stated as an important feature of IoT systems and has become more specific and can also track users indoor activity. Collecting location data can also track the users behaviour and compare indoor and outdoor routines to forecast a user's position. Since IoT devices integrate as a part of our everyday life, profiling for directed and personal preferences can be gathered for misuse. Privacy-violating interaction and presentation is referred to as a threat more applicable to the future since this type of technology is not too common. However, identifying that presenting private information in a real-world environment is a privacy threat is important for the development of these services. For life-cycle threats, the research highlights that IoT devices do not have established standards for total memory wipes or physical destruction, leaving possible private information stored history of private data. Inventory attacks combines information about IoT devices and their characteristics and can be used by burglars to target potential victims. Lastly, the research states linkage between interconnected systems that shares data in a way that was not disclosed to the sources when they where isolated [44].

N. Apthorpe et al. [45] proposes a three step strategy an attacker can carry out to passively network eavesdrop traffic from IoT devices to infer private information. The attacker first have to identify the different IoT devices in a smart home and recreate the smart home in its own environment. Then by doing normal user cases and looking at the variations from general traffic, it is possible to infer users private behaviour. However, the less functionality the devices have, the easier it is to infer the user behaviour as a change in traffic can correspond to the functionality being used [45].

A research testing several smart home devices by Apthorpe et al. [46] revealed that it is possible to infer user behaviour based on traffic rates. Traffic analysis from an IoT camera reveals when a user is actively looking at the camera stream and a sleep monitor were inferred to show when a user is sleeping. The research is based on traffic analysis of only traffic rate and headers. All the devices tested were encrypted [46]. The same results were found by Apthorpe et al. [47] where the authors examined four different IoT devices: a personal assistant, smart power outlet, sleep monitor and security camera. They used two different attack perspectives where one is outside of the network on a wired channel, and the other adversary is sniffing Wi-Fi traffic. The Wi-Fi sniffer is referred to a neighbour or a strong radio receiver within signal range of the network. The results showed that it was possible to infer user behaviour easily by looking at changes in traffic patterns for the different devices [47].

Another research by Acar et al. [48] shows that a passive network eavesdrop-

ping attack on several popular IoT devices can reveal user behaviour in a targeted environment. The devices ranged from light bulbs and smart plugs to motion and camera sensors, but no air quality monitors were included in the test. The devices communicated over different communication protocols, such as Wi-Fi, Bluetooth and ZigBee. Their method of analysing raw data had 4 stages: device identification, device state detection, device state classification and user activity inference. The results showed that they could infer user activity by an accuracy of 90% [48].

A method for network traffic analysis is suggested in [49] by Papadogiannaki and Ioannidis. They describe that to be able to infer user behaviour, a two-step process is required. First, the analyzer needs to know what normal network activity is and then compare it collect data when there are changes from the normal traffic is the network. To be able to distinguish normal events from abnormal events a machine learning algorithm is proposed. Even though the research is aimed for benign actors, the same method can be used for malicious actors. It is important to understand that network sniffers can also be used by network administrators to prevent and detect attacks and protect its own network, as well as for malicious actors wanting to access private information about others.

Chapter 4

Method

This chapter describes the material and method used for testing, the environment setup and the test cases that have been conducted. First, a survey for selecting the air quality monitors to be tested in this research is introduced. Then, the method for creating the environment, carrying out the tests, and analyzing the results are presented in a method tree and explained in detail.

4.1 Air Quality Monitor Selection Survey

To select the air quality monitors to use in this thesis, the problem description and research questions have been used as a reference. In order to answer the RQs, the air quality monitors need to be manufactured from different vendors. To find the specific AQMs to use in this research, several online sources were used and compared. As this research is conducted by NTNU Gjøvik in Norway, it is also preferable that the devices are bought and available in Norwegian stores or webpages. The devices chosen should also be popular and easy accessible for any user.

It exists many solutions that integrate air quality monitors within other devices, but as this thesis aims to investigate what private information is possible to infer from an air quality monitor, the devices should only have air quality monitoring functionality. Considering these factors, the following criteria are made out to select the devices:

- The devices are manufactured from different vendors.
- The devices communicates over Wi-Fi.
- The devices are available for any user.
- The devices only monitors indoor air quality.
- The devices are available in Norwegian stores.

Tibber [50] is a Norwegian power company that specializes in combining smart technology, as with IoT devices, and live app representation of the power consumption and smart devices [50]. Tibber has over 400,000 users in Northern Europe, which makes them a natural choice for many when integrating a smart

home device to their environment [51]. On their website it is possible to buy several IoT smart devices for a home environment. They recommend one AQM which communicates over Wi-Fi, Netatmo Smart Indoor Air Quality Monitor. Netatmo [19] is a company that specializes in consumer technology. They have one indoor air quality monitor which uses four sensors to monitor the indoor air quality of an environment. The sensors are; humidity, CO_2, noise and temperature.

Elkjøp [52] and Komplett [53] are two of Norway's biggest electrical stores with a wide range of different smart home devices both in store and online. Its therefore a natural choice when users are looking for any electronic devices including smart devices. When searching for "Air Quality Monitors" on their webpages, Elkjøp shows 4 different vendors and Komplett 8 different vendors. Both vendors lists Netatmo Healthy Home coach, which is already chosen. A part from this, both Elkjøp and Komplett also recommends Mill Sense. Mill [20] is a Norwegian company that manufactures and sells products for indoor climate and heating, with the goal of developing devices that fits the indoor interior environment. They offer one air quality monitor called Mill Sense which communicates over Wi-Fi. The sensors integrated are eCO_2, humidity, temperature and TVOC.

When sorting the devices on client reviews on Komplett, the highest ranking manufacturer of air quality monitors is Nedis SmartLife [53]. Nedis [54] is an electronic company with the goal of making electronic-related solutions based on the newest technology. They have different solutions for air quality monitors, but the one that communicates over Wi-Fi is called Nedis SmartLife Air Quality Monitor. This device monitors CO_2, HCHO, humidity, temperature and VOCs.

This research will consists of air quality monitors from the selected three different vendors. The devices only works as air quality monitors and they have different sensors. The chosen devices from the survey are represented in Table 4.1.

Table 4.1: Air quality monitor devices selected for this research

Vendor	**Air Quality Monitor**	**Communication protocol**	**Sensors**
Netatmo	Smart Indoor Air Quality Monitor	WiFi	CO_2 Noise Humidity Temperature
Mill	Sense Smart Climatesensor	WiFi	eCO_2 Humidity Temperature TVOC
Nedis	SmartLife Air Quality Monitor	WiFi	CO_2 HCHO Humidity Temperature VOC

4.1.1 Netatmo Smart Indoor Air Quality Monitor

The Netatmo Smart Indoor Air Quality Monitor entails 4 different sensors: humidity, CO_2, noise and temperature. It can be integrated with several smart indoor air quality monitor devices installed in users home, using HomeKit. It communicates over Wi-Fi to its own app called *Healthy Home Coach*. Figure 4.1 shows Netatmo Smart Indoor Air Quality Monitor and its corresponding application, Healthy Home Coach.

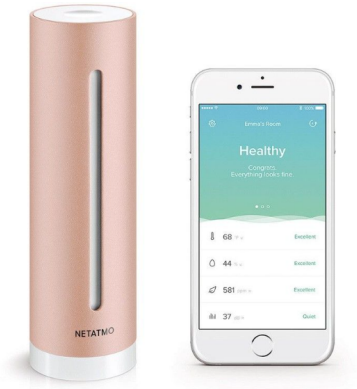


Figure 4.1: Netatmo Smart Indoor Air Quality Monitor and corresponding application Healthy Home Coach [55]

The Netatmo Smart Indoor Air Quality Monitor sends notifications to the user's application when the device sense high temperature, CO_2, humidity or noise and low temperature or humidity. These are default enabled, but can be turned off by the user. The values cannot be changed, but are stated by Netatmo in the application. The unit displays live readings to the user, together with a graphical view of values over a longer period of time. When first installed, the device needs at least 7 days to finish calibrating and read the environment values correctly.

4.1.2 Mill Sense Air Quality Monitor

Mill Sense Air Quality Monitor measures the indoor air quality with 4 different sensors: humidity, TVOC, temperature and eCO_2 [20]. The monitor communicates through Wi-Fi and connects its data to their application called *Mill Norway*. It is possible to use Mill Sense together with other Mill units, such as heaters or air purifiers, to change their values based on the sensor data from Mill Sense [20]. Figure 4.2 shows Mill Sense and its corresponding application, Mill Norway.

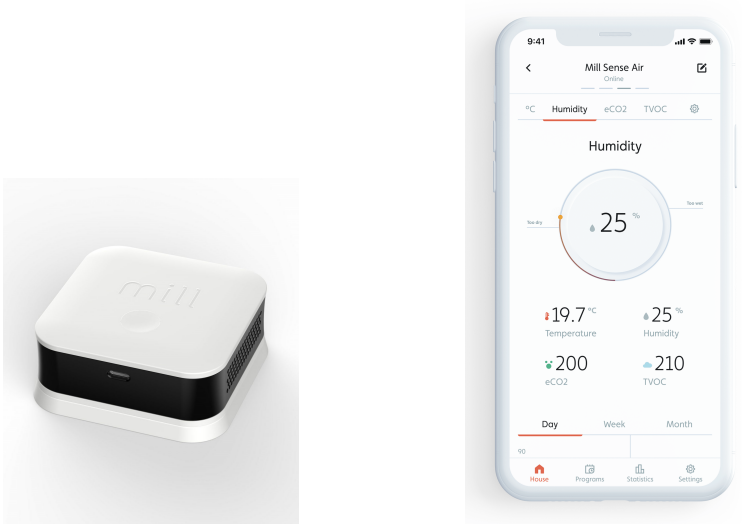


Figure 4.2: Mill Sense and corresponding application Mill Norway [56]

The AQM does not need power to stay connected and can be placed in any preferred room. The application does not provide notifications to users of threshold levels, but displays the current levels in the app together with graphs that shows variations back in time. Instead of sending notifications to a users phone, the device shows different light responses on the physical unit. These threshold values can be customized. The sensor data collected from the environment is sent to the cloud and back to the users app. It is possible to choose at what time interval the device will send sensor data, from every minute to every hour. When the unit is turned on for the first time and installed in the environment, the device needs at least 5 days to calibrate its sensors.

4.1.3 Nedis SmartLife Air Quality Monitor

Nedis SmartLife Air Quality Monitor has 3 different indoor AQM sensors: humidity, temperature and VOC [57]. The monitor communicates over Wi-Fi to the app, *Nedis SmartLife*. As Nedis sells several other smart devices, the unit can be integrated in a smart environment together with other devices. Figure 4.3 shows Nedis SmartLife Air Quality Monitor and its corresponding application, Nedis SmartLife.

Figure 4.3: Nedis SmartLife and corresponding application Nedis SmartLife [54]

Nedis SmartLife does not have notifications enabled default, but it can easily be set and customized by the user. Every sensor reading from the device can be set to notify whether levels are too high or too low. The readings on the app shows live data as well as a graphical view of previous readings. For setup and calibration in an environment, Nedis does not specify any calibration time.

4.2 Method Tree

This research is structured in six different main activities, shown in Figure 4.4:

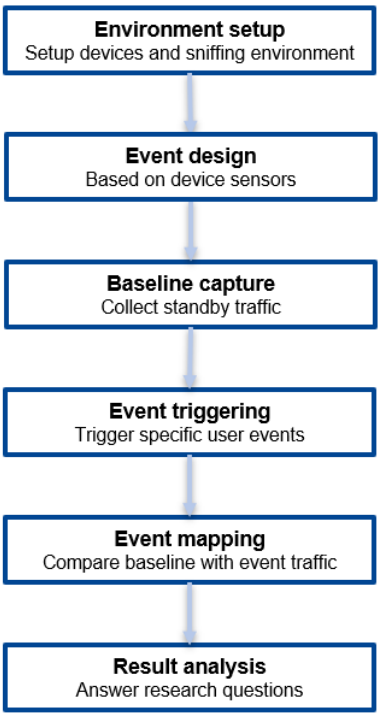


Figure 4.4: Method tree

4.2.1 Environment Setup

The first main activity is to set up the environment with both hardware and software correctly configured. First, the air quality monitors need to be functioning in the environment. Then a sniffer will be installed and tested to capture traffic sent to and from the devices. After this, the capturing and analysis platform is setup to store, process and analyze the data captured from the sniffer. On this platform, software designed to capture the necessary data needs to be installed and configured.

Air Quality Monitors
The AQMs need to be installed, connected to their app and calibrated. The devices were installed and connected to the app as explained in the user manual attached to the devices. It is desirable that the air quality monitors give notifications when a certain threshold value is met. However, for Mill Sense it is not possible to set notifications, but Nedis SmartLife and Netatmo Smart Indoor Air Quality Monitor were set with the same threshold values for notifications. For Mill Sense the interval for sending sensor data to the application were set to every minute. The MAC addresses for the air quality monitors can all be found in their corresponding apps, since identifying and discovering the devices are out of scope for this thesis.

Table 4.2 describes each AQM, their MAC address and which sensor values triggers the devices to send notifications to the users phone. For the rest of this thesis, the devices will only be referenced with their manufactures name; Netatmo, Mill and Nedis to simplify the presentations.

Table 4.2: MAC-address and notification threshold values for each AQM

Air Quality Monitor	MAC Address	Notification threshold values
Netatmo	70:EE:50:91:06:DE	$CO_2 > 1150$ ppm Noise > 62db Humidity < 30% Humidity > 60% Temperature < 17 °C Temperature > 26 °C
Mill	B8:F0:09:B3:B3:78	N/A
Nedis	2C:F4:32:29:36:DC	$CO_2 > 1000$ ppm Humidity < 30% Humidity < 50% Temperature < 15 °C Temperature > 25 °C VOC > 99.9 ppm

Sniffer
In order to collect all Wi-Fi traffic within the environment, a network sniffer was configured. The sniffer needs to be able to connect to the system were the packet

capturing will take place and be set in monitoring mode. It exists a wide variety of available Wi-Fi sniffers online, but the selected sniffer in this research is the TP-Link TL-WN722N.

Figure 4.5: The Wi-Fi sniffer selected in this research: TP-Link TL-WN722N [58]

To set the device in monitoring mode, the is sniffer first plugged into a system and then the following four commands were applied:

```
sudo ifconfig wlan0 down
sudo iwconfig wlan0 mode monitor
sudo ifconfig wlan0 up
sudo iwconfig
```

Note that wlan0 was the assigned wlan for the device when it was plugged into the environment. The specific wlan addressed to the sniffer can vary from system to system so it needs to be checked with the device plugged into the system. When the sniffer is configured in monitoring mode, the device is able to collect all Wi-Fi traffic within its own signal range. In this way, the sniffer will be able to collect the traffic sent to and from the AQMs installed in the environment. To verify that the sniffer is collecting traffic on the network, Wireshark were used with the corresponding wlan, wlan0, as capturing interface.

Platform

To collect data and store it from the devices, an operating system running on a hardware component with the possibility to connect the sniffer were setup. In this thesis, a Raspberry Pi Model 3 B+ were chosen as it can continuously be powered on to capture traffic over longer periods of time. The Raspberry Pi was installed with Kali Linux version 2022.3, installed from [59], which was beneficial as it includes all the needed software to capture the traffic and is compatible with the TP-link sniffer. As the analysis will be conducted on another computer, WinSCP were used to export the files from the Raspberry Pi over the network, downloaded from [60]. Figure 4.6 shows a logical overview of the setup of the environment.

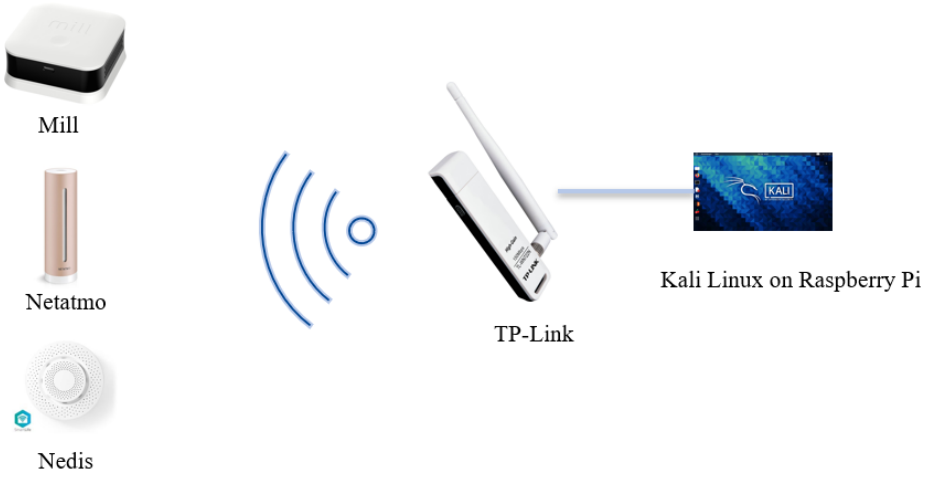


Figure 4.6: Environment setup with all devices [56], [55], [57], [58], [59]

Tshark

In order to store and process the traffic that the sniffer collects, a monitoring software was used. Tshark [61] is an open-source network packet analyzer and were used to store packets. Tshark can be run from the command line interface and has the possibility of using capture filters based on a number of parameters, such as addresses, duration or size. In order to store the packets captured, tshark was run with the option of writing its results to an output file rather than directly in the command-line, as it does default. The packets collected in the output file by tshark are possible to open and analyze in Wireshark.

To store packets from all the three air quality monitors, tshark was run from three different terminals with corresponding filter to the devices. The options of the command are explained in Table 4.3.

Table 4.3: Tshark command options used to capture traffic

Filter option	Usage
tshark	Initialize tshark
-i	Interface to be used
-f	Capturing filter
-w	Output file to store captured packets

```
Netatmo:
    tshark -i wlan0 -f "ether.host == 70:EE:50:91:06:DE" -w
    "\\Documents\Netatmo"
Mill:
    tshark -i wlan0 -f "ether.host == B8:F0:09:B3:B3:78" -w
    "\\Documents\Mill"
Nedis:
```

```
tshark -i wlan0 -f "ether.host == 2C:F4:32:29:36:DC" -w
"\\Documents\Nedis"
```

4.2.2 Event Design

In order to infer user behaviour from the devices, several test cases were designed. As the AQMs are installed and monitors the home environment at all times, the test cases are not limited in time or place in the house as they can be moved.

Looking at the different sensors and how they can be affected is important for understanding which user behaviour can possibly change the traffic pattern. The sensors from all the devices combined are; eCO_2, CO_2, humidity, temperature, TVOC, VOC, HCHO and noise. Some of the sensors will be affected by the same actions, so they will be categorized together. The following significantly different categorized sensors will be used: CO_2, humidity, temperature, VOC and noise. Examples of user behaviour in a home that affects the different sensors are:

- **CO_2** : People or animals present, cooking, windows open
- **Humidity:** Windows open, showering
- **Temperature:** People or animals present, windows open, fireplace lit
- **VOC:** Cooking, burning candles
- **Noise:** Many people present, playing music or TV

Several of the proposed user behaviours will affect not only one, but several sensors. However, it is more beneficial to focus on routines that users do every day to be able to see a pattern of a household instead of looking at one specific action that users may not do regularly. It is also important to notice that in order to infer private information from network traffic, the test cases needs to change the sensor values by some degree. Therefore, repetitive behaviour that has the possibility to change the threshold values are chosen initially.

The three different routinely test cases chosen are **cooking**, **showering** and **windows open at night**. Cooking is the most routinely behaviour in a home where dinner is normally made every day. The next test case is showering as this is a routine behaviour and will affect the sensors, particularly humidity and temperature in the bathroom. Showering can be routinely done at very different times, but evening showers are used in this case. Windows in a home can be open at several times, however, having a window open every night while sleeping can be common and should drastically change the indoor temperature and possibly humidity. Especially since the testing will be carried out during winter time in Norway. This is also a test that will last longer than the other two tests and can possibly give another aspect to the research.

Another test case that can be interesting to see if changes the traffic patterns is whether a user is home or not. Therefore, a weekend test will be carried out. The sniffer will gather traffic from several weekends when the user is present in the home environment and compare it to when the user is away for the weekend and look at the differences.

The four test cases chosen are three routine test cases and one test case not based on routines. It is important to understand that the privacy of the different test cases are different. Knowing whether a user is cooking or showering are defined as the two test cases with the least dangerous private information exposed, while window open at night can rather indicate that a user is sleeping and the home environment is not watched. The private information exposed from the weekend test is the most dangerous if revealed since it will indicate if a user is gone and therefore not in control of the home environment.

4.2.3 Baseline Capture

To be able to distinguish the events in the Wi-Fi traffic, it is necessary to capture and analyze traffic from the devices when they are not affected by any events. This is necessary to understand what normal traffic from the devices is, like presented in [49]. During the baseline capture, the devices were installed and calibrated in the environment. All notifications that will be enabled during the event triggering were enabled. The devices were reachable through their app and communicate in their own pattern. The baseline capture was on-going for several days to ensure enough data and traffic was collected. During this capture, the devices were placed in the inner hallway of the apartment. It would have been ideally to have a baseline from each room of where the test cases will be located, but due to time constraints, they were placed in a room binding all the other rooms together. See Figure 4.7 for an overview of the environment.

4.2.4 Event Triggering

In this activity, traffic from the devices were captured while the events were triggered. The routine events are each triggered 10 times and traffic from at least 1 hour before and after the event were captured to see the changes in traffic both before and after the events. Weekend testing were conducted in the course of 14 weekends, resulting in collected traffic from 7 weekends at home and 7 weekends gone.

Table 4.4 gives an overview of the time of day when each test case was carried out and in which room the devices were placed for the tests.

Table 4.4: Overview of test cases with timings and location

Test Case	**Time of day**	**Room**
Cooking	After work: 4pm to 5pm	Kitchen
Window open at night	At night: 11pm to 7am	Bedroom
Showering	Afternoon: 8pm to 9pm	Bathroom
Weekend tests	Friday: 4pm to Sunday: 11pm	Hallway

As there is only one of each device and the tests are in different rooms, the devices were moved to the corresponding room before each test. The devices were placed in the environment for at least 1 hour before each test starts to ensure that

the value changes between the room will be as equalized as possible. The exact times for when cooking, open window and showering took place were logged so it is possible to look for changes at that time.

Figure 4.7 shows an overview of the apartment layout, hereby called **the environment**, where the test cases and baseline were captured from. The different rooms are marked and shows where the devices were placed during the different tests.

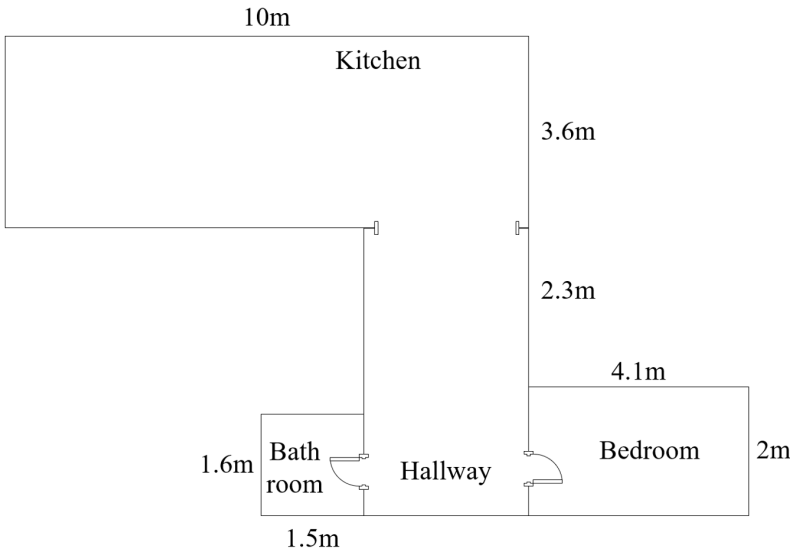


Figure 4.7: Overview of the environment for the test cases and baseline capturing. Cooking in the kitchen, showering in the bathroom, window open at night in the bedroom and baseline and weekend tests in the hallway.

4.2.5 Event Mapping

When the events had been carried out, a mapping was made to see if information could be inferred. Each event is presented both graphically and numerically with different calculations to compare. Each event were extracted from the corresponding tshark capture saved in the pcap output file. A separate pcap file for each event was made, with the timing of the specific event to see differences.

4.2.6 Result Analysis

The last activity is to analyze the results from the tests performed. Each event has initially been evaluated and analyzed with the same general procedure. Figure 4.8 shows an overview of the steps each event has been through. The results from each event are presented and explained in each subsection and in some cases, the general procedure has given insights to further analysis which is covered in the Evaluation and Analysis Results.

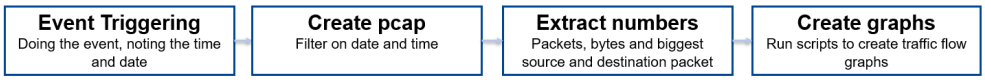


Figure 4.8: General procedure used to evaluate and analyze the test cases

To be able to run analysis on the data gathered from the different Packet Capturings (pcaps) and present it in an understandable way, two different programs were used: Wireshark and python scripts in Visual Studios Code (VS Code) with pyshark, a tshark library. Wireshark was used to extract numbers and calculations from the pcaps and VS Code was used to generate graphs from event pcaps.

When opening a pcap in Wireshark, several statistics are possible to extract. By choosing the option of "Capture File Properties", total number of packets and bytes in the capture file are shown and were extracted. The numbers from each pcap were noted down and used as event attributes to compare both the events to each other and to the baseline. Another value used to analyze the events were packet lengths. The biggest packet from each capturing were noted down to use for comparing and looking for patterns and changes during an event.

In VS Code, python was used as the programming language as it includes the library pyshark which can be used towards the tshark capture files. Four scripts with a number of different if statements were used to generate different graphs depending on system arguments passed to the script. The scripts were used to generate two graphs: one with number of bytes or packets per execution of event and one for the corresponding baseline event. The code is displayed in its whole in Appendices A, B, C and D, and in pseudo code beneath in Algorithm 1. For each of the if statements, the same code block is included and are only shown once in Algorithm 1. The graphs are all generated with bytes or packets per 2 seconds, where the y-axis is defined in the number of packets or bytes and the x-axis defined as time. The scripts use parts from the scripts explained in [62].

To create a file for each of the events, a time filter was applied to the pcaps and the remaining packets exported to a separate pcap creating one pcap file per captured event. These can be opened in Wireshark or make a graph out of to analyze each specific event. The time filter used the following format:

- **Format:** frame.time >= "Month Date, Year "Time"" && frame.time <= "Month Date, Year "Time""//
- **Example:** frame.time >= "Jan 08, 2023 "19:30:00"" && frame.time <= "Jan 08, 2023 "20:40:00""

Table 4.5 displays the different system arguments passed to the scripts to create the graphs and which values the arguments expect.

Table 4.5: Overview of system arguments for scripts

	Description	**Values**
Argument 1	Name of device	Netatmo Mill Nedis
Argument 2	Which packets to include	Inbound packets and bytes Outbound packets and bytes Inbound and outbound packets and bytes
Argument 3	Type of event	Cooking Shower Window Weekend Baseline
Argument 4	Maximum value for y-axis, in bytes or packets	<Numeric value>

Algorithm 1 Script for generating graphs

```
1: for Each date of event do                                   ▷ Graph_function start
2:     Extract the packets from the right pcap
3:     for Each packet in pcap do
4:         Extract packet length in byte and time or add packet count to time
5:         Display graph
6:     end for
7: end for                                                     ▷ Graph_function end
8:
9: if Argument 2 is Outbound then                  ▷ Set display filter to outbound traffic
10:    if Argument 1 is Netatmo then
11:        Set display filter to "wlan.sa == Netatmo MAC address"
12:    else if Argument 1 is Mill then
13:        Set display filter to "wlan.sa == Mill MAC address"
14:    else if Argument 1 is Nedis then
15:        Set display filter to "wlan.sa == Nedis MAC address"
16:    end if
17:
18: else if Argument 2 is Inbound then             ▷ Set display filter to inbound traffic
19:    if Argument 1 is Netatmo then
20:        Set display filter to "wlan.da == Netatmo MAC address"
21:    else if Argument 1 is Mill then
22:        Set display filter to "wlan.da == Mill MAC address"
23:    else if Argument 1 is Nedis then
24:        Set display filter to "wlan.da == Nedis MAC address"
25:    end if
26: end if
27:
28: if Argument 3 is Shower then                               ▷ Event if-cases start
29:    Set the dates from when the events occurred
30:    Graph_function
31: else if Argument 3 is Cooking then
32:    Set the dates from when the events occurred
33:    Graph_function
34: else if Argument 3 is Window then
35:    Set the dates from when the events occurred
36:    Graph_function
37: else if Argument 3 is Weekend then
38:    Set the dates from when the events occurred
39:    Graph_function
40: else if Argument 3 is Baseline then
41:    Set the dates from when the events occurred
42:    Graph_function
43: end if                                                     ▷ Event if-cases end
```

To be able to look even further into differences in the traffic flow during an event and standard traffic, the baseline capturing were also extracted with the same time of day and duration as the corresponding events. To have the same start and finish time for the events and the baseline, timings for all executions of the events were used. The earliest starting time and the latest finish time for that event were used on every event, but also the baseline which was captured on different dates than the events. This results in both event and corresponding baseline pcaps possible to compare as they all have the same time interval.

When these steps were finished for each execution of the events, graphs and calculations for the events were available to analyze. Average and standard deviation values were calculated for both the events and corresponding baseline pcaps. To be able to conclude that there is a difference in traffic flow from when an event is ongoing to standard traffic, different measurements were used. The measurements are both calculations and graphical views and then compared to each other, using the same principles as proposed in [46]. For calculations, the average value in packets and bytes for the baseline needed to be outside of the standard deviation for the event to conclude that there is possible to differentiate the events from other standard traffic.

The calculations for the routine tests also needed to be seen in context with the the traffic flows, as the calculations are not only made on traffic at the specific event time, but also includes traffic from a time period before and after the event. Since we do not expect the sensor values to change immediately when an event is triggered or finished, the traffic before is used as a reference to compare daily baseline traffic with the standby reference. The traffic after an event can still be affected by the sensor values since stabilizing the indoor air may take time. To be able to compare event and baseline calculations, the time before an event and the baseline needs to follow the same pattern. If the average value of the baseline were outside of the standard deviation of the event, and the traffic flow before an event were equal to the baseline traffic, then it was possible to conclude that the device exposed private information about the test case.

The graphical measurement to analyze if there was a difference in traffic flow during the events, was if it was possible to see a constant difference in the traffic before and during an event. The baseline graphs were also used here to compare even more standard traffic, than the time period before the event and look further into if it was possible to see spikes or traffic flow changes during the events. The last numerical value analyzed is the biggest packet sent or received and can be used to see if there are bigger packets in the capture files while an event is ongoing, and if there are, look into if they are associated with the specific event time. This was included because we expect bigger packets to be sent or received by the device during an event to update sensor values or send notifications, which was also possible to find in [43].

However, for the weekend test, baseline traffic were not used to compare. In this test, the weekends at home and the weekends gone are compared to each other to look at differences in the same way as explained for the routine beha-

vioural tests. If the calculations showed that the average values for packets and bytes during a weekend gone was outside of the standard deviation for a weekend at home, we can conclude that it is possible to infer this private information on the device. For this test, the graphical view of the traffic flow were used to strengthen the result if they gave the same result as the calculations.

Chapter 5

Evaluation and Analysis Results

The evaluation and analysis result chapter presents all gathered and analyzed data from the four different tests carried out in this thesis. First, the baseline for each device is shown both separately and to compare to each other. For each event; cooking, showering, window open and weekend, several graphs and calculations are presented side by side to analyze. This is divided into categories with the three different devices. Then the actual events are compared to the baseline for each device to look further into differentiate the events from standard traffic. The results are commented and evaluated in the same sections that presents the results.

5.1 Baseline

The capturing of baseline traffic was conducted over the course of 10 days in the hallway of the environment. During the baseline, the devices were not directly affected by the specific events, such as cooking, showering or window open in the same room as the devices resides. The baseline traffic will be used to look at standard traffic from the devices and to compare this to the events in both graphs and calculations in Sections 5.2-5.4. The Wi-Fi traffic from the capture files were encrypted and therefore it is not possible to extract any values from the payload of the packets. This applies to all the devices. Since decryption of traffic is out of scope for this thesis, the results will only analyze patterns and no payload information. The filter used for all the baseline files is:

- frame.time >= "Mar 06, 2023 "00:00:00"" && frame.time <= "Mar 15, 2023 "24:00:00""

5.1.1 Netatmo Baseline

Figures 5.2 and 5.1 show the graphs for Netatmo from the baseline capturing from 6th of March 2023 to 15th of March 2023. For the baseline graphs, it is possible to see that packets are sent continually at a rate of around 250 bytes per

2 seconds and 2 packets per 2 seconds. In addition to the continuously stream of packets, the traffic flow also has spikes that looks random in both time and size. As these graphs shows the total packets and bytes sent and received, it can also be beneficial to look at what the graphs would look like if filtered on packets and bytes sent and packets and bytes received separately.

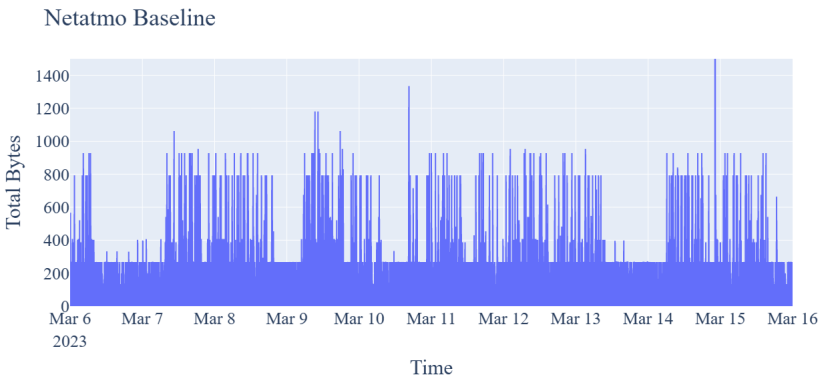


Figure 5.1: Netatmo baseline capture with total number of bytes as the y-axis

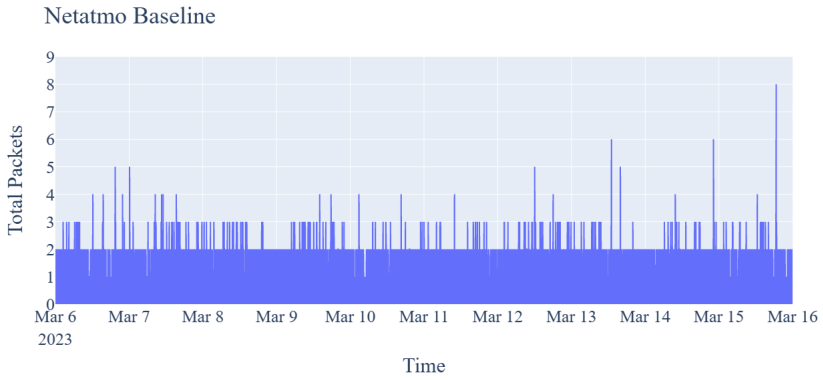
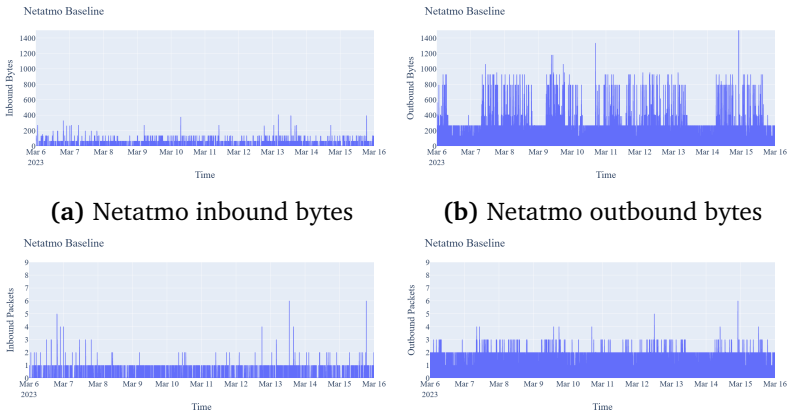


Figure 5.2: Netatmo baseline capture with total number of packets as the y-axis



(a) Netatmo inbound bytes **(b)** Netatmo outbound bytes

(c) Netatmo inbound packets **(d)** Netatmo outbound packets

Figure 5.3: Netatmo baseline inbound and outbound bytes

For the inbound and outbound bytes and packets for Netatmo it is clear to see from Figure 5.3 that the device sends a lot more than it receives. Calculations made on the baseline traffic are presented in Table 5.1.

Table 5.1: Calculations for Netatmo baseline capture

		Numbers
Total	Packets	110,735
	Bytes	14,959,396
	Average bytes/second	17
	Average packet size	135 bytes
Inbound	Packets	1,042
	Bytes	83,446
	Average bytes/second	0
	Average packet size	80 bytes
	Biggest packet	377 bytes
Outbound	Packets	109,693
	Bytes	14,875,950
	Average bytes/second	17
	Average packet size	136 bytes
	Biggest packet	1,150 bytes

Comparing the inbound and outbound graphs in Figure 5.3 with the total graphs, in Figure 5.1 and Figure 5.2, shows that the outbound graphs stands for the majority of the packets and are very similar to the total graphs. The same is numerically shown in Table 5.1, where over 99% of the total packets are outbound traffic. Therefore, it will be better to only display the events with graphs for total traffic to evaluate and analyze for the rest of the thesis.

5.1.2 Mill Baseline

Figures 5.5 and 5.4 show the graphs for Mill from the baseline capturing from 6th of March 2023 to 15th of March 2023. The baseline traffic for Mill shows that the traffic varies a lot. As this device does not send live updates, but every minute, more spikes are included as it does not always send packets. It is also possible to see the spikes more clearly if the time range is smaller, this will be visible further when looking at the event and baseline comparison graphs for each event.

Graphs for inbound and outbound traffic have also been made for Mill to see the differences for the packets sent. Figure 5.6 displays the different graphs for each of the traffic directions. Numerical calculations for the baseline traffic are presented in Table 5.2.

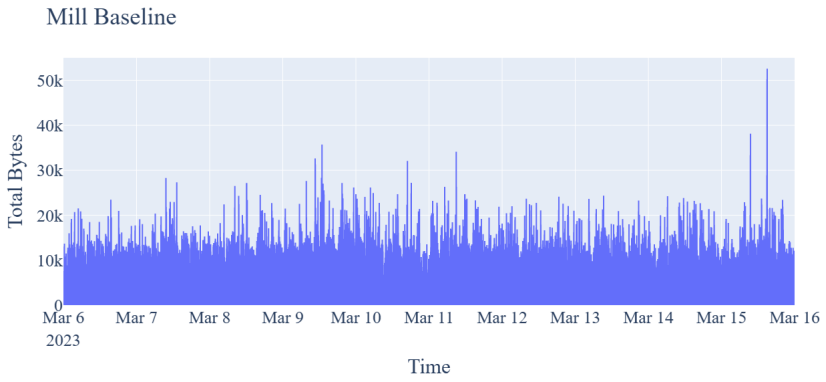


Figure 5.4: Mill baseline capture with total number of bytes as the y-axis

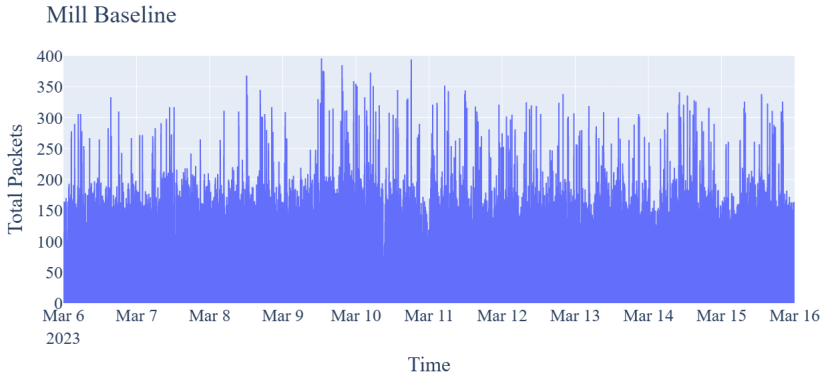
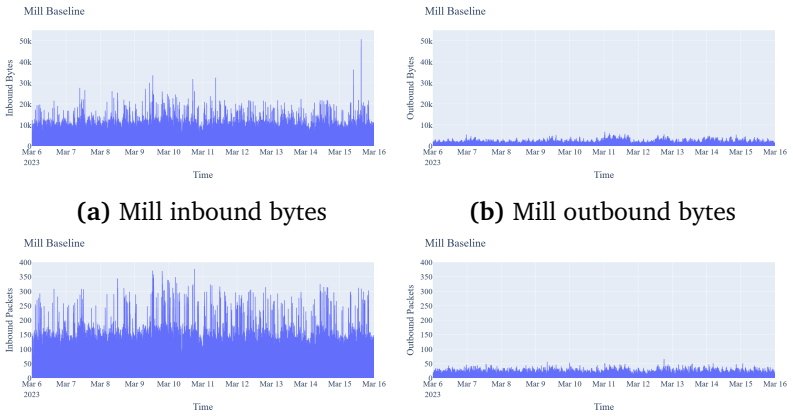


Figure 5.5: Mill baseline capture with total number of packets as the y-axis



(a) Mill inbound bytes **(b)** Mill outbound bytes

(c) Mill inbound packets **(d)** Mill outbound packets

Figure 5.6: Mill baseline inbound and outbound bytes

Table 5.2: Calculations for Mill baseline capture

		Numbers
Total	Packets	1,236,753
	Bytes	129,253,290
	Average bytes/second	149
	Average packet size	105 bytes
Inbound	Packets	942,112
	Bytes	95,458,773
	Average bytes/second	110
	Average packet size	101 bytes
	Biggest packet	1593 bytes
Outbound	Packets	294,640
	Bytes	33,794,517
	Average bytes/second	39
	Average packet size	115 bytes
	Biggest packet	456 bytes

As Figure 5.6 shows, the device receives a lot more packets and bytes than it sends. As the inbound graphs do not differ much from the total graphs, it will be best to proceed with the analysis in a total traffic aspect where both inbound and outbound traffic are included. This is also reflected in Table 5.2, which shows that 76% of packets and 74% of bytes are inbound traffic.

5.1.3 Nedis Baseline

Figures 5.8 and 5.7 show the graphs for Nedis from the baseline capturing from 6th of March 2023 to 15th of March 2023. The traffic sent and received by Nedis

is characterized by varying a lot with high spikes especially for packets. The larger spikes, which are more visible in Figure 5.8, showing the packets, occurs almost everyday. Since traffic is encrypted, it is not possible to see what these spikes are, but for further analysis it is important to understand that normal traffic for the device, can be large spikes occurring around the same time each night around 3am.

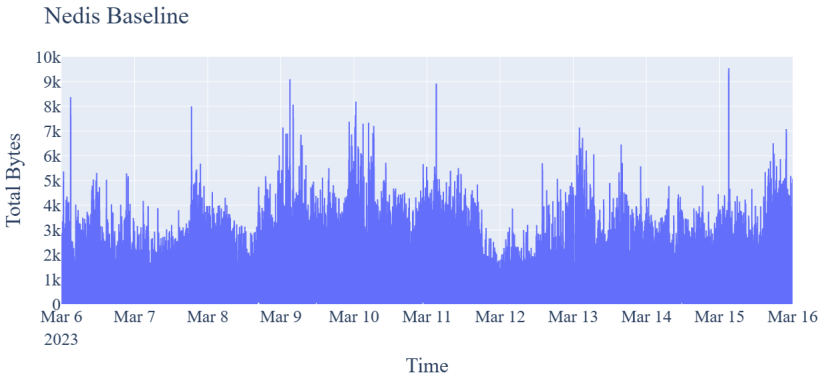


Figure 5.7: Nedis baseline capture with total number of bytes as the y-axis

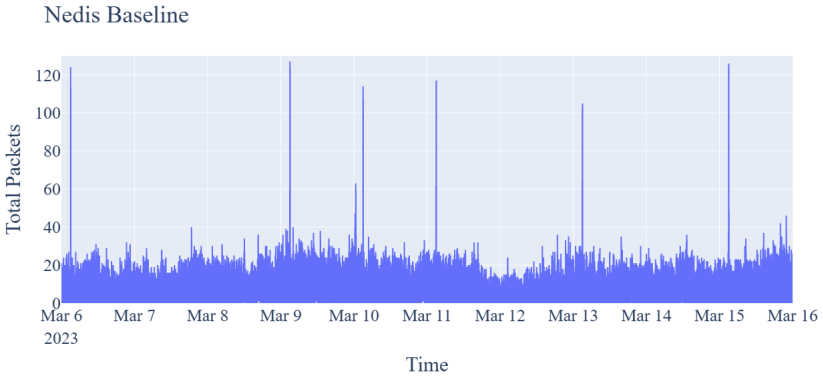
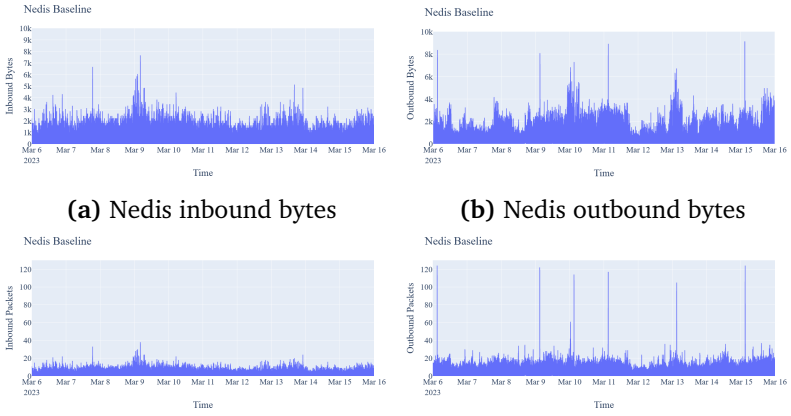


Figure 5.8: Nedis baseline capture with total number of packets as the y-axis



(a) Nedis inbound bytes **(b)** Nedis outbound bytes

(c) Nedis inbound packets **(d)** Nedis outbound packets

Figure 5.9: Nedis baseline inbound and outbound bytes

Table 5.3: Calculations for Nedis baseline capture

		Numbers
Total	Packets	2,428,701
	Bytes	295,022,494
	Average bytes/second	341
	Average packet size	121 bytes
Inbound	Packets	451,495
	Bytes	88,595,049
	Average bytes/second	102
	Average packet size	196 bytes
	Biggest packet	522 bytes
Outbound	Packets	1,977,206
	Bytes	206,427,445
	Average bytes/second	238
	Average packet size	104 bytes
	Biggest packet	485 bytes

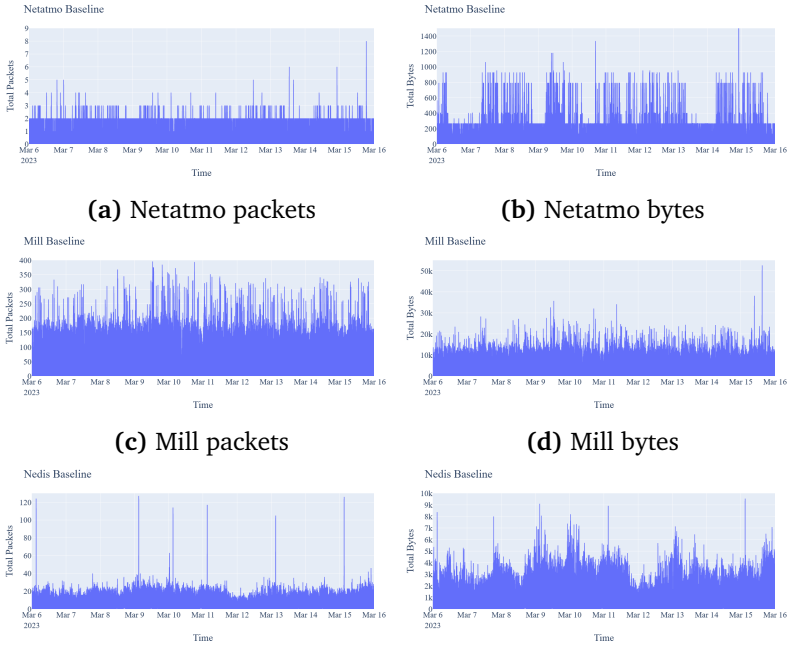
Figure 5.9 shows differences in inbound and outbound traffic for Nedis. Figure 5.9 shows that the spikes are packets which the device receives. Even though the graphs for inbound and outbound traffic from Nedis can look similar, the calculations presented in Table 5.3 shows that 81% of the packets and 70% of the bytes in the baseline are traffic sent from the device. Therefore, looking at the graphs in total will give the most to analyze. Another aspect of looking at the graphs in total, compared to inbound and outbound separately is that since the traffic is encrypted, it is not possible to know what makes the possible changes.

5.1.4 Baseline Summary and Comparison Between the Devices

This subsection summarizes and compares the baseline traffic for all the three devices. Table 5.4 shows the differences in packets and bytes sent and received to each devices during the baseline capture. As the table shows, the three devices send different amounts of packets and bytes. Nedis sends the most while Netatmo sends the least amount in standby. For inbound and outbound traffic, Netatmo and Nedis sends more packets than it receives, while Mill receives more packets than it sends. The same is shown in Figure 5.10 for the total number of bytes and packets, in Figure 5.11 for inbound traffic and in Figure 5.12 for outbound traffic.

Table 5.4: Baseline capture summary for all the devices

		Netatmo	Mill	Nedis
Total	Packets	110,735	1,236,753	2,428,701
	Bytes	14,959,396	129,253,290	295,022,494
Inbound	Packets	1,042	942,112	451,495
	Bytes	83,446	95,458,773	88,595,049
Outbound	Packets	109,693	294,640	1,977,206
	Bytes	14,875,950	33,794,517	206,427,445

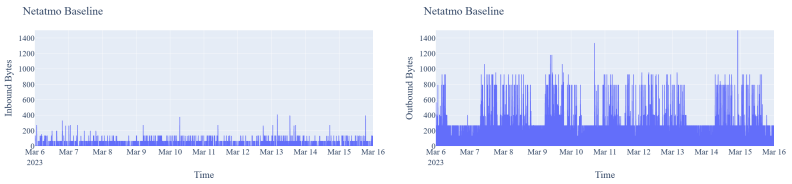


(a) Netatmo packets **(b)** Netatmo bytes

(c) Mill packets **(d)** Mill bytes

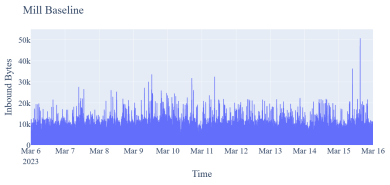
(e) Nedis packets **(f)** Nedis bytes

Figure 5.10: Traffic comparison between the baseline graphs with total number of packets and bytes for all devices

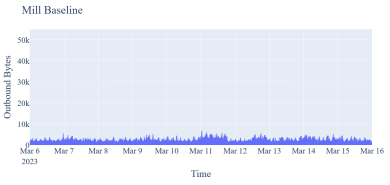


(a) Netatmo inbound bytes

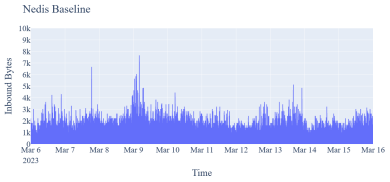
(b) Netatmo outbound bytes



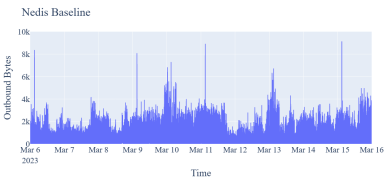
(c) Mill inbound bytes



(d) Mill outbound bytes

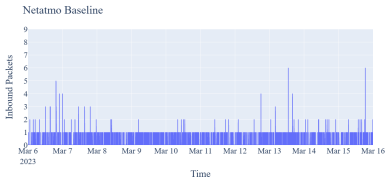


(e) Nedis inbound bytes

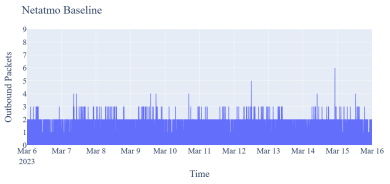


(f) Nedis outbound bytes

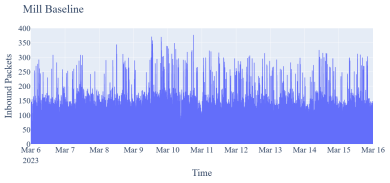
Figure 5.11: Inbound and outbound baseline bytes comparison for all devices



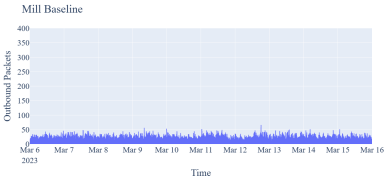
(a) Netatmo inbound packets



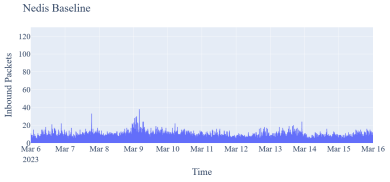
(b) Netatmo outbound packets



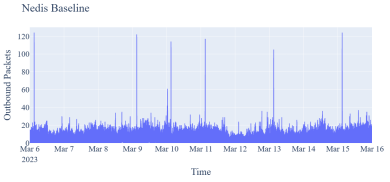
(c) Mill inbound packets



(d) Mill outbound packets



(e) Nedis inbound Packets



(f) Nedis outbound packets

Figure 5.12: Inbound and outbound baseline packets comparison for all devices

5.2 Test Case 1: Cooking

This chapter presents the results and analysis conducted on Test Case 1: Cooking. The first subsection will present general information applicable to all the devices, and the following subsections will present the result and analysis for each of the devices separately.

5.2.1 General

The cooking events are 10 in total and presented in Table 5.5. Every device have the same time and dates for this event.

Table 5.5: Date and time for Test Case 1: Cooking

	08.01	09.01	11.01	16.01	18.01	19.01	25.01	30.01	31.01	01.02
Started cooking	15:58	15:59	16:05	16:02	16:04	16:01	16:02	16:01	16:01	16:02
Finished cooking	16:22	16:21	16:37	16:25	16:25	16:18	16:13	16:19	16:21	16:22

To be able to look even further into if it is a similar traffic pattern to each event that can be used to identify it, the same start and finish time have been used for every graph. To have the same amount of time on each event, the earliest start time and the latest finish time are used as filtering values for each of the pcaps for the cooking event. 30 minutes before and after these times were used as the start and finish time for the capture files, while the actual event time given by Table 5.5 is marked red on the graphs. This gives the following values to use for further analysis:

- Earliest cooking start: 15:58
- Latest cooking finished: 16:37
- Packet capture files start: 15:28
- Packet capture files end: 17:07

These timings give the following filter added to create the pcaps for each event:

- frame.time >= "Month Date, Year 15:28:00" && frame.time <= "Month Date, Year 17:07:00"

5.2.2 Netatmo

Table 5.6 presents calculations from all the cooking events and Table 5.7 presents the calculations from all the corresponding baseline pcaps. Table 5.8 compares the average and standard deviation values from the events to the baseline. Figure 5.13 presents a graphical overview of the packets and bytes from Table 5.6 including average values.

Table 5.6: Calculations on cooking events for Netatmo

Dates	Packets	Bytes	Biggest packet
08.jan	901	123,630	407 bytes
09.jan	703	97,019	407 bytes
11.jan	847	114,473	407 bytes
16.jan	1,082	146,001	407 bytes
18.jan	948	129,907	407 bytes
19.jan	828	111,629	407 bytes
25.jan	430	58,926	407 bytes
30.jan	815	110,838	407 bytes
31.jan	864	115,302	136 bytes
01.feb	805	109,050	407 bytes

Table 5.7: Calculations on comparing baseline files for the cooking event for Netatmo

Baseline	Packets	Bytes	Biggest packet
06.mar	584	77,780	134 bytes
07.mar	972	132,406	407 bytes
08.mar	697	94,730	407 bytes
09.mar	868	117,020	407 bytes
10.mar	825	111,863	407 bytes
11.mar	745	101,534	407 bytes
12.mar	626	84,987	407 bytes
13.mar	764	101,772	136 bytes
14.mar	750	102,396	407 bytes
15.mar	812	108,703	407 bytes

Table 5.8: Traffic comparison between the cooking event and baseline for Netatmo

	Type	Packets	Bytes	Biggest packet
Average	Cooking	822	111,678	380 bytes
	Baseline	764	103,319	353 bytes
Standard deviation	Cooking	170	22,802	86 bytes
	Baseline	114	15,650	115 bytes

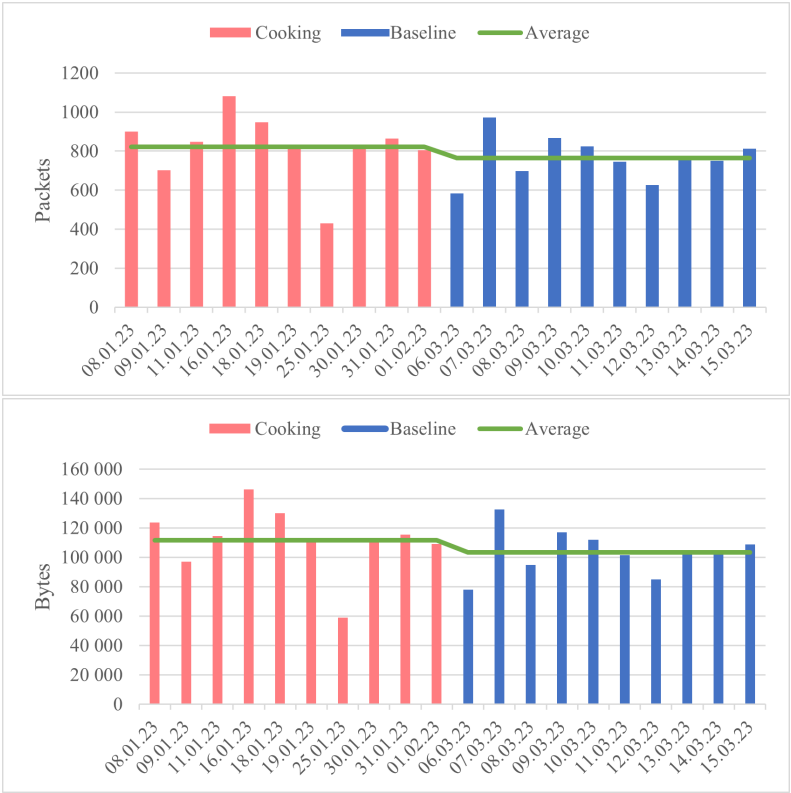


Figure 5.13: Graphical presentation of event and baseline cooking calculations with packets and bytes, including average value extracted from Table 5.8 for Netatmo

The graphs in Figures 5.14, 5.15, 5.16 and 5.17 display both bytes and packets for the cooking events in comparison with the baseline captures. The event graphs are placed on the left side of the figure and are framed in red, while the baseline graphs are placed on the right side of the figure and are framed in blue. The area marked red on the event graphs is when the event was ongoing, and not included in the baseline graphs as no event was ongoing and is only used for comparison. The x- and y-axis for all the graphs in the same figure have the same minimum and maximum values.

Figure 5.14: Graphs of traffic flows from the cooking events measured in packets with event graphs framed in red and baseline graphs framed in blue for Netatmo. Event times are marked in red on the event graphs.

Figure 5.15: Continuing from Figure 5.14

Figure 5.16: Remaining graphs from Figure 5.17

Figure 5.17: Graphs of traffic flows from the cooking events measured in bytes with event graphs framed in red and baseline graphs framed in blue for Netatmo. Event times are marked in red on the event graphs.

Table 5.6 shows that the calculations varies a lot for each event. Number of packets varies from 403 to 1,082 and bytes from 58,926 to 146,001. The same is found for the baseline calculations, which also varies with 584 packets as the smallest to 972 as the highest amount of packets and for bytes from 77,780 to 132,406 bytes. The biggest packet from both the events and the baseline is mainly 407 bytes, with a few exceptions for both. The average values for the biggest packet are similar to each other for the events and baseline. Therefore it is not possible to distinguish an event based on the biggest packet.

Considering the values in Table 5.8, the average value of the baseline is within the standard deviation of the cooking events for both packets and bytes. This results in that the calculations cannot be used to identify the cooking event for Netatmo.

The same result is further confirmed in the graphs in Figures 5.14 and 5.15 for packets and 5.16 and 5.17 for bytes. For both cooking event graphs, in red, and baseline graphs, in blue, spikes that differentiate from the continuous traffic are visible for both packets and bytes. However, these are not connected to the red area when the event is ongoing and is also present in the baseline graphs, where no event is triggered. The event graphs does not show a traffic flow change in the red marked area. This results in that it is not possible to distinguish that cooking is ongoing in the environment by looking at the graphs of traffic flow. The results from cooking event on Netatmo is that there is no changes or differences in calculations or traffic flows which can be used to identify the event.

5.2.3 Mill

The results from the cooking events for Mill are presented with both numerical values in tables and graphs and in figures where traffic patterns are analyzed. Table 5.9 presents the calculations from the event traffic with packets, bytes and biggest packet sent and received during the packet capturing. The same calculations have been made on the baseline traffic, in Table 5.10. Table 5.11 compares the average and standard deviation values for the event and baseline traffic. The calculations from these three tables are graphically presented in Figure 5.18 where packets and bytes from the event and baseline traffic are included with the average value for each of them.

Table 5.9: Calculations on cooking events for Mill

Dates	Packets	Bytes	Biggest packet
08.jan	8,875	1,097,786	456 bytes
09.jan	7,948	1,057,936	456 bytes
11.jan	10,222	1,310,644	456 bytes
16.jan	10,014	1,358,615	1,353 bytes
18.jan	9,185	1,137,586	456 bytes
19.jan	8,306	1,062,743	1,593 bytes
25.jan	8,246	1,033,976	1,583 bytes
30.jan	11,826	1,357,595	1,343 bytes
31.jan	10,464	1,205,006	456 bytes
01.feb	10,124	1,219,206	456 bytes

Table 5.10: Calculations on comparing baseline files for the cooking event for the device Mill

Baseline	Packets	Bytes	Biggest packet
06.mar	8,808	835,070	426 bytes
07.mar	9,310	940,428	1,353 bytes
08.mar	8,930	904,598	1,343 bytes
09.mar	10,675	1,046,076	456 bytes
10.mar	6,989	774,986	1,573 bytes
11.mar	9,983	1,107,006	1,273 bytes
12.mar	10,740	1,033,467	456 bytes
13.mar	8,134	826,038	426 bytes
14.mar	9,090	969,576	1,573 bytes
15.mar	6,539	740,761	429 bytes

Table 5.11: Traffic comparison between the cooking event and baseline for Mill

	Type	Packets	Bytes	Biggest packet
Average	Cooking	9,521	1,184,109	861 bytes
	Baseline	8,920	917,801	931 bytes
Standard deviation	Cooking	1,221	125,182	529 bytes
	Baseline	1,404	122,928	527 bytes

Figure 5.18: Graphical presentation of event and baseline cooking calculations with packets and bytes, including average value extracted from Table 5.11 for Mill

The traffic pattern for the events and baseline are presented in Figure 5.19 and 5.20 for packets and in Figure 5.21 and 5.22 for bytes. In these figures, all graphs have the same minimum and maximum values on the y- and x-axis to be comparable. The graphs created from the different events are placed on the left side of the figure and framed in red, while the baseline is placed on the right side of the figure and framed in blue. The event times for when the event was ongoing are marked in red on the event graphs.

Figure 5.19: Graphs of traffic flows from the cooking events measured in packets with event graphs framed in red and baseline graphs framed in blue for Mill. Event times are marked in red on the event graphs.

Figure 5.20: Continuing from Figure 5.19

Figure 5.21: Remaining graphs from Figure 5.22

Figure 5.22: Graphs of traffic flows from the cooking events measured in bytes with event graphs framed in red and baseline graphs framed in blue for Mill. Event times are marked in red on the event graphs.

Comparing Table 5.9 and 5.10 shows that the number of packets sent to and from the device is around the same value and not a significant difference between the events and baseline. The average values for packets in Table 5.11 also shows that the values are similar to each other. For bytes, the calculations shows that when an event is ongoing, the overall values are higher than for standard baseline traffic. The comparison in Table 5.11 shows that the average value for events are 266,308 bytes higher. The same is presented in Figure 5.18 where there is a bigger difference in bytes than packets for events and baseline traffic. For the biggest packet sent, the packet sizes varies from around 450 bytes to 1,550 bytes for events and baseline traffic over the same time, shown in Table 5.9 and 5.10. This results in no difference in the bigger packets sent and received during an event.

Table 5.11 shows that the average value of the baseline packets are within the range of the standard deviation of the cooking events. However, for bytes, the average value is lower than the standard deviation. This is also visible in Figure 5.18 where the difference in bytes is much clearer than for packets. When looking at Figure 5.21 and 5.22, it is a difference between the traffic before the event and the baseline traffic. The graphs marked in blue shows that when the baseline capturing was ongoing, the number of bytes sent are much lower than for the days when the events were triggered. Since this is also applicable for before the event and not just during and after an event, we can conclude that the difference in calculations for bytes are not linked directly to the event triggered, but must be affected by other changes in the environment.

The graphs in Figures 5.19 and 5.20 for packets and Figures 5.21 and 5.22 for bytes does not show a significant change in traffic pattern from when an event is ongoing, in the graphs framed in red, and the standard traffic pattern from the baseline, in the graphs framed in blue. The variations the calculations gave in bytes from the events and baseline is not very visible in the graphs as they both vary a lot. However, knowing that more bytes are sent during the events, it is possible to see that more of the event graphs have higher spikes than the baseline, but this is not applicable for all events. Even though there are some differences visible, these are not applicable for all executions of cooking for Mill. Therefore, it is not possible to show that there is a specific change in traffic patterns for cooking and not possible to identify when cooking is ongoing in the environment.

5.2.4 Nedis

In Test Case 1: Cooking for Nedis, tables with numerical calculations are presented together with graphs of the traffic flow. In Table 5.12 the total amount of packets and bytes, including the biggest packet during the event are presented for each of the 10 cooking events. Table 5.13 presents the same values, but in regards of the baseline pcaps. Table 5.14 compares the average values and standard deviation from the events and baseline for packets, bytes and biggest packet. Figure 5.23 presents the calculations from Tables 5.12, 5.13 and 5.14 with packets and bytes together with the average value for both the event and baseline traffic to easier compare it.

Table 5.12: Calculations on cooking events for Nedis

Dates	Packets	Bytes	Biggest packet
08.jan	24,494	3,318,519	485 bytes
09.jan	26,787	3,870,361	424 bytes
11.jan	24,381	3,550,947	424 bytes
16.jan	26,035	3,895,071	485 bytes
18.jan	25,398	3,233,845	424 bytes
19.jan	20,857	3,242,594	424 bytes
25.jan	24,233	3,095,693	424 bytes
30.jan	24,867	3,257,812	424 bytes
31.jan	23,882	3,179,117	424 bytes
01.feb	25,397	3,477,014	424 bytes

Table 5.13: Calculations on comparing baseline files for the cooking event for Nedis

Baseline	Packets	Bytes	Biggest packet
06.mar	13,014	1,413,457	421 bytes
07.mar	18,634	2,001,978	421 bytes
08.mar	18,643	2,019,872	421 bytes
09.mar	21,885	2,606,363	424 bytes
10.mar	17,956	2,326,898	424 bytes
11.mar	18,380	2,126,816	485 bytes
12.mar	10,728	1,312,915	421 bytes
13.mar	15,815	1,968,872	424 bytes
14.mar	15,926	1,878,449	341 bytes
15.mar	19,260	2,524,793	458 bytes

Table 5.14: Traffic comparison between the cooking event and baseline for Nedis

	Type	Packets	Bytes	Biggest packet
Average	Cooking	24,633	3,412,097	436 bytes
	Baseline	17,024	2,018,041	424 bytes
Standard deviation	Cooking	1,595	281,705	26 bytes
	Baseline	3,248	420,982	36 bytes

Figure 5.23: Graphical presentation of event and baseline cooking calculations with packets and bytes, including average value extracted from Table 5.14 for Nedis

The graphical presentation of the traffic patterns during events are presented in Figure 5.24 and 5.25 for packets and in Figure 5.26 and 5.27 for bytes. In these figures, the corresponding graphs from the baseline traffic is also included to look for traffic changes during the events. In these figures, the graphs for events are placed on the left side of the figure and framed in red, while the baseline graphs are placed on the right side and framed in blue. The event graphs are marked with red when the event was ongoing. The minimum and maximum values on the x- and y-axis are equal for all graphs within the same figure.

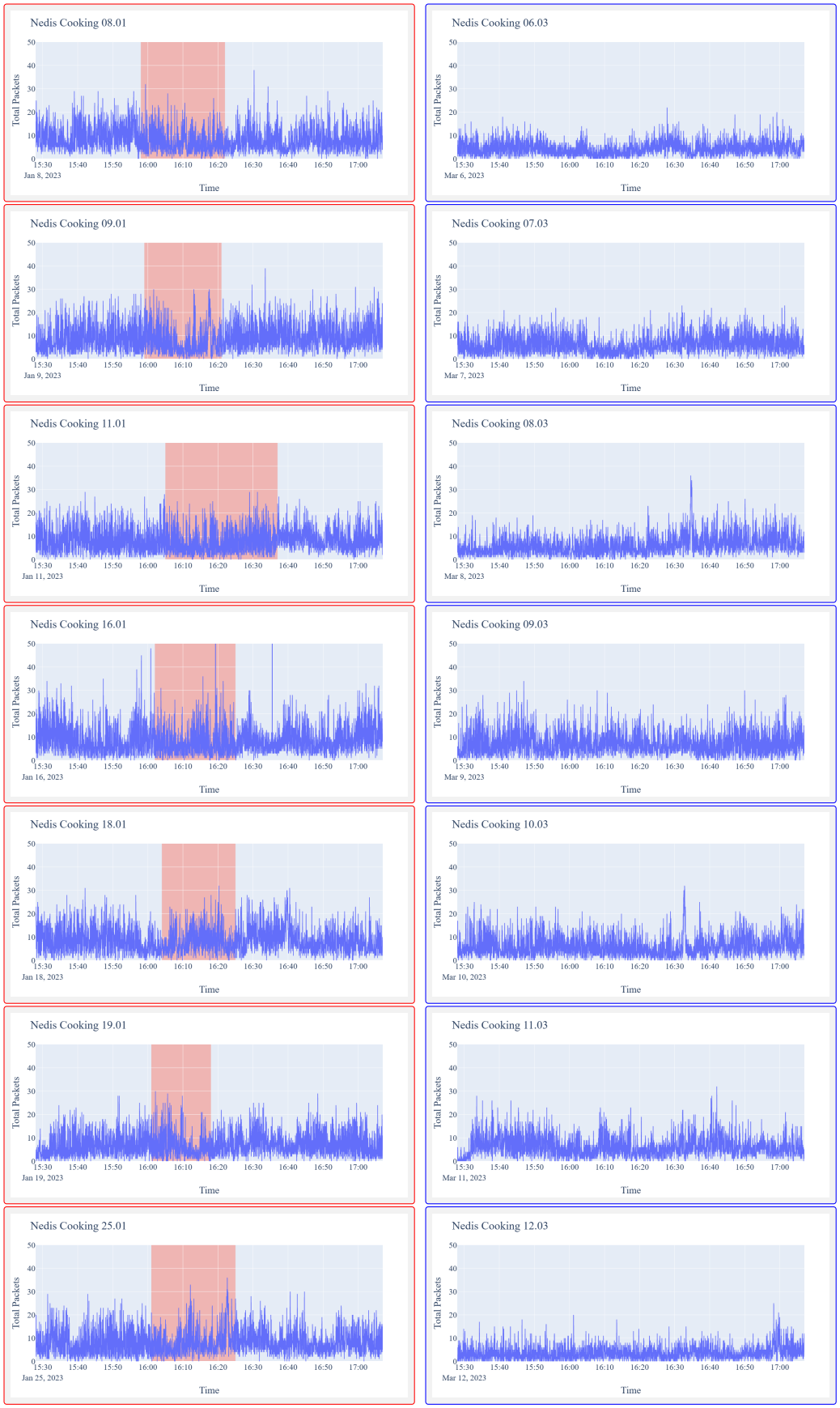


Figure 5.24: Graphs of traffic flows from the cooking events measured in packets with event graphs framed in red and baseline graphs framed in blue for Nedis. Event times are marked in red on the event graphs.

Figure 5.25: Continuing from Figure 5.24

Figure 5.26: Remaining graphs from Figure 5.27

Figure 5.27: Graphs of traffic flows from the cooking events measured in bytes with event graphs framed in red and baseline graphs framed in blue for Nedis. Event times are marked in red on the event graphs.

Comparing Table 5.12 and 5.13 shows that overall during the events, more packets and bytes are sent than during standard baseline traffic. While for the biggest packet during the cooking-time, the values are around 400 bytes and do not differ from cooking in Table 5.12 to baseline in 5.13. The same is shown in Table 5.14 where the average for both packets and bytes are higher during events. This is also visible in Figure 5.23 where all the blue marked bars are lower than the red marked graphs for events with one exception in packets.

Table 5.14 shows that the average value for the baseline is lower than the standard deviation for the cooking events for both packets and bytes. However, the traffic flows in Figure 5.24, 5.25, 5.26 and 5.27 shows that the traffic pattern before the event is not comparable to the baseline traffic as it shows a higher number of packets and bytes. This means that the difference in calculations are not necessary linked to specific event triggering, but rather other changes in the environment from the days the events and the baseline were captured.

Figure 5.24, 5.25, 5.26 and 5.27 shows that there are no significant traffic changes during the time of the event triggering that is visible for all executions of the event. It is therefore not possible to identify cooking through looking at either the calculations or the traffic flows for Nedis.

5.3 Test Case 2: Showering

This chapter presents the results and analysis for Test Case 2: Showering. The first subsection describes the general evaluation which is applicable for all the devices and the next three subsections presents the analysis and results for each of the devices separately.

5.3.1 General

The showering event has been conducted 10 times and Table 5.15 presents the 10 different dates and exact times for when the event was ongoing.

Table 5.15: Date and time for Test Case 2: Showering

	08.01	09.01	11.01	16.01	18.01	19.01	25.01	30.01	31.01	01.02
Started showering	19:59	20:14	20:01	20:12	20:02	20:00	20:03	20:00	20:01	20:00
Finished showering	20:14	20:34	20:17	20:31	20:19	20:16	20:19	20:18	20:17	20:16

To be able to compare the events to each other and the standard baseline traffic, all the pcaps created for this event will have the same start and finish time. Graphically, each event are marked at what time the actual event was ongoing to separate from before and after the event. When deciding which start and end time for these events, the earliest start and latest finish are used. These times are then added 30 minutes before and after to cover at least 30 minutes before and after each event. This gives the following values to use for further analysis:

- Earliest showering start: 19:59
- Latest showering finished: 20:34
- Packet capture files start: 19:29
- Packet capture files end: 21:04

These times results in the following filters added to create the pcaps for each event:

- frame.time >= "Month Date, Year 19:29:00" && frame.time <= "Month Date, Year 21:04:00"

5.3.2 Netatmo

For Netatmo, the 10 events for showering are presented both graphically in figures and numerically in tables. Table 5.16 shows packets, bytes and the biggest packet in each of the packet captures for the events. Table 5.17 presents the same calculations, but for the corresponding baseline pcaps. In Table 5.18, the average and standard deviation values from Table 5.16 and 5.17 are compared to each other. In context of these three tables, two graphical presentations of total bytes and packets including the average values for events and baseline are presented to compare events to the baseline traffic in Figure 5.28.

Table 5.16: Calculations on showering events for Netatmo

Dates	Packets	Bytes	Biggest packet
08.jan	870	118,119	407 bytes
09.jan	816	112,821	407 bytes
11.jan	1,075	145,838	407 bytes
16.jan	667	90,370	407 bytes
18.jan	802	109,285	407 bytes
19.jan	636	85,860	407 bytes
25.jan	897	120,580	134 bytes
30.jan	862	116,726	134 bytes
31.jan	704	94,140	136 bytes
01.feb	694	94,053	407 bytes

Table 5.17: Calculations on comparing baseline files for the showering event for Netatmo

Baseline	Packets	Bytes	Biggest packet
06.mar	614	81,600	136 bytes
07.mar	710	94,869	407 bytes
08.mar	780	94,892	160 bytes
09.mar	884	118,116	134 bytes
10.mar	594	79,258	136 bytes
11.mar	615	82,664	407 bytes
12.mar	857	116,703	407 bytes
13.mar	514	68,470	136 bytes
14.mar	808	108,120	407 bytes
15.mar	731	97,682	134 bytes

Table 5.18: Traffic comparison between the showering event and baseline for Netatmo

	Type	Packets	Bytes	Biggest packet
Average	Showering	802	108,779	325 bytes
	Baseline	711	94,237	246 bytes
Standard deviation	Showering	133	18,181	132 bytes
	Baseline	123	16,541	138 bytes

Figure 5.28: Graphical presentation of event and baseline shower calculations with packets and bytes, including average value extracted from Table 5.18 for Netatmo

Figure 5.29 and 5.30 gives the graphical presentation of the traffic flow during the showering times with the total number of packets sent and received on the y-axis. Figure 5.31 and 5.32 shows the same, but measured in bytes on the y-axis. The figures presenting the traffic flow all have the same minimum and maximum values on the y- and x-axis and event graphs are placed on the left side of the figure with a red frame, while baseline graphs are placed on the right side of the figure with a blue frame. The timings for when the event was ongoing are marked with a red area on the event graphs to look for changes in traffic pattern at that specific time.

Figure 5.29: Graphs of traffic flows from the showering events measured in packets with event graphs framed in red and baseline graphs framed in blue for Netatmo. Event times are marked in red on the event graphs.

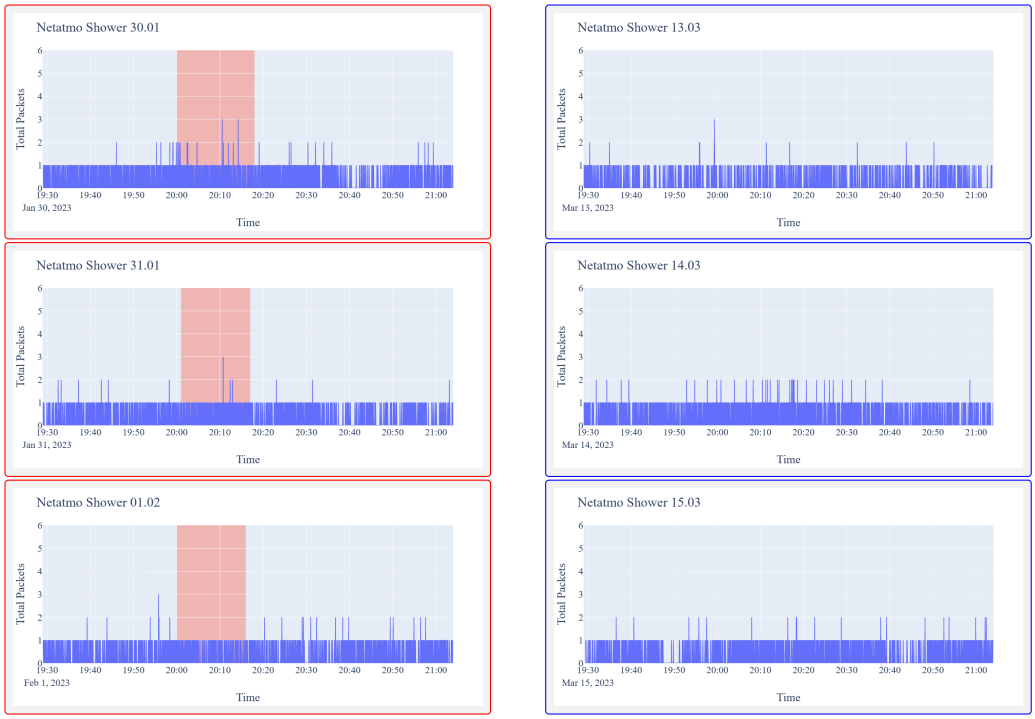


Figure 5.30: Continuing from Figure 5.29

Figure 5.31: Remaining graphs from Figure 5.32

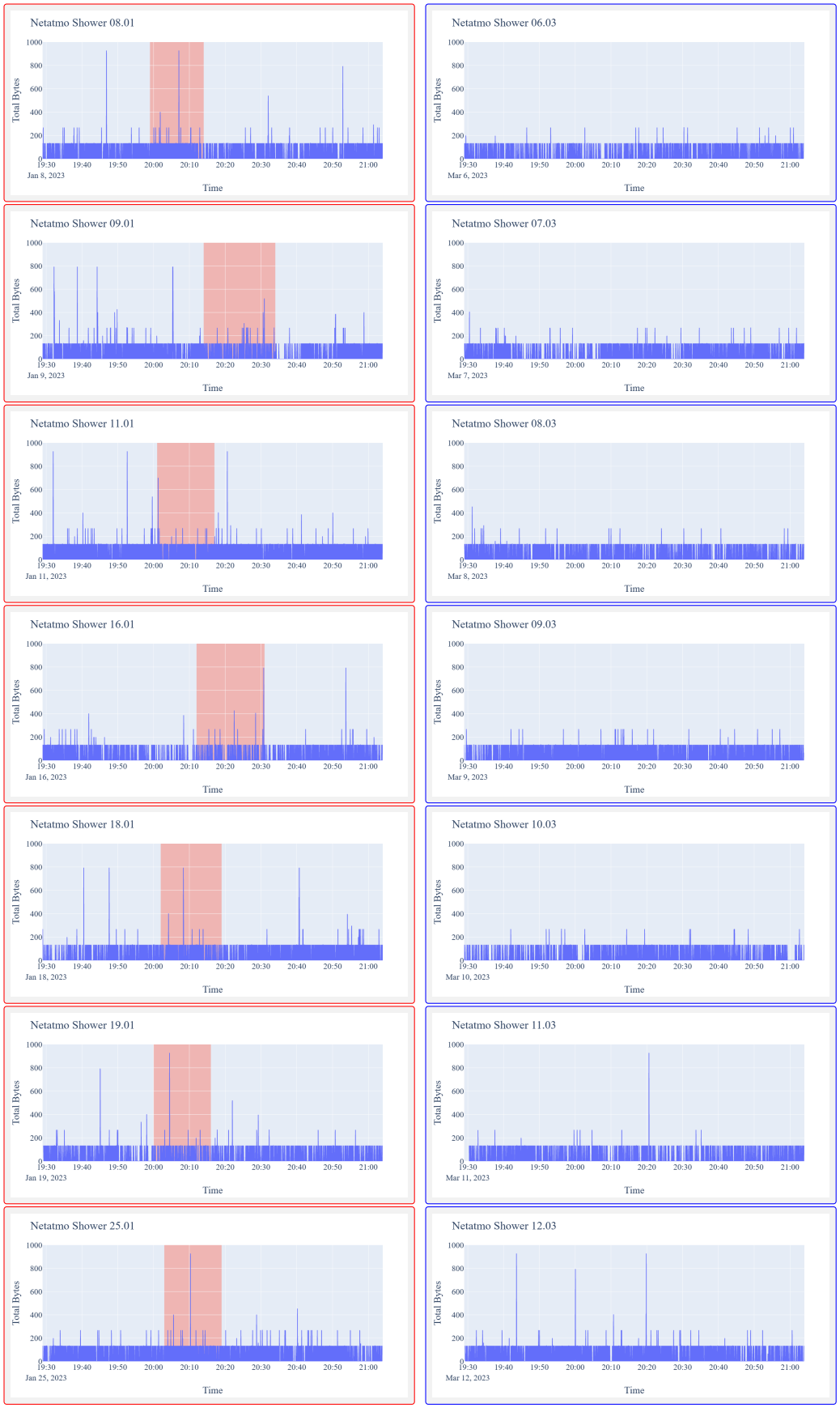


Figure 5.32: Graphs of traffic flows from the showering events measured in bytes with event graphs framed in red and baseline graphs framed in blue for Netatmo. Event times are marked in red on the event graphs.

The number of packets and bytes are relatively similar for both events and baseline traffic which is shown in Figure 5.28. For events, packets varies from 636 to 1,075 and for the baseline it varies from 514 to 884. For bytes the events varies from 85,860 to 145,838 and for the baseline it varies from 68,470 to 118,116. This results in an average value a bit lower for the baseline traffic, but not enough to see a significant change in amount of packets or bytes during the event. Even though the average baseline traffic is less, several of the baseline days exceeds both packets and bytes for some of the event traffic captures. For biggest packet sent and received, both event and baseline traffic varies between around 150 bytes to 407 bytes.

Table 5.18 shows that the average value for the baseline is within the range of standard deviation for both packets and bytes. It is therefore not possible to use these values to identify showering on Netatmo.

The graphs of the traffic flow show no distinct difference for packets in Figure 5.29 and 5.30. For bytes in Figure 5.31 and 5.32 several of the event graphs, marked in red, have higher spikes than the baseline graphs marked in blue. However, two things challenges the difference: the first is that a few of the baseline graphs also have spikes and a few of the event graphs are missing the spikes. The other one is that the spikes also occurs before the event has started and may not be associated to an event triggering. Therefore, it is not possible to see specific traffic changes when showering is ongoing for Netatmo.

5.3.3 Mill

The results from the showering events for Mill are presented in this subsection. Table 5.19 presents the total amount of packets and bytes sent and received and the biggest packet from each packet capture for the events. The same numbers are presented in Table 5.20 for the corresponding baseline captures. Table 5.21 compares the average and standard deviation values for events and baseline. The total number of packets and bytes are also presented graphically in Figure 5.33, including the average values. Events and baseline are included in the same figure to easier compare the values.

Table 5.19: Calculations on showering events for Mill

Dates	Packets	Bytes	Biggest packet
08.jan	8,400	1,112,718	1353 bytes
09.jan	7,775	970,700	456 bytes
11.jan	9,819	1,166,768	456 bytes
16.jan	8,800	1,107,872	1,421 bytes
18.jan	9,339	1,162,327	834 bytes
19.jan	8,938	1,110,851	1,353 bytes
25.jan	9,690	1,136,156	1,593 bytes
30.jan	7,838	909,992	456 bytes
31.jan	7,930	995,619	456 bytes
01.feb	7,787	905,808	456 bytes

Table 5.20: Calculations on comparing baseline files for the showering event for Mill

Baseline	Packets	Bytes	Biggest packet
06.mar	6,308	623,993	456 bytes
07.mar	5,968	675,359	426 bytes
08.mar	8,920	899,908	426 bytes
09.mar	9,241	922,721	456 bytes
10.mar	6,789	670,474	456 bytes
11.mar	6,750	686,700	1,337 bytes
12.mar	10,151	1,006,385	456 bytes
13.mar	7,032	794,714	456 bytes
14.mar	7,290	839,659	456 bytes
15.mar	7,927	809,616	426 bytes

Table 5.21: Traffic comparison between the showering event and baseline for Mill

	Type	Packets	Bytes	Biggest packet
Average	Showering	8,632	1,057,862	883 bytes
	Baseline	7,638	792,953	535 bytes
Standard deviation	Showering	801	102,055	489 bytes
	Baseline	1,381	126,913	282 bytes

Figure 5.33: Graphical presentation of event and baseline shower calculations with packets and bytes, including average value extracted from Table 5.21 for Mill

Figure 5.34 and 5.35 show the traffic flow during the events, placed on the left side of the figure and framed in red, and the baseline placed on the right side of the figure and framed in blue, measured in number of packets. Figure 5.36 and 5.37 display the same, but measured in number of bytes. The timings for when the event was ongoing are marked red on the event graphs. All graphs within the same figure have the same minimum and maximum values for the x- and y-axis.

Figure 5.34: Graphs of traffic flows from the showering events measured in packets with event graphs framed in red and baseline graphs framed in blue for Mill. Event times are marked in red on the event graphs.

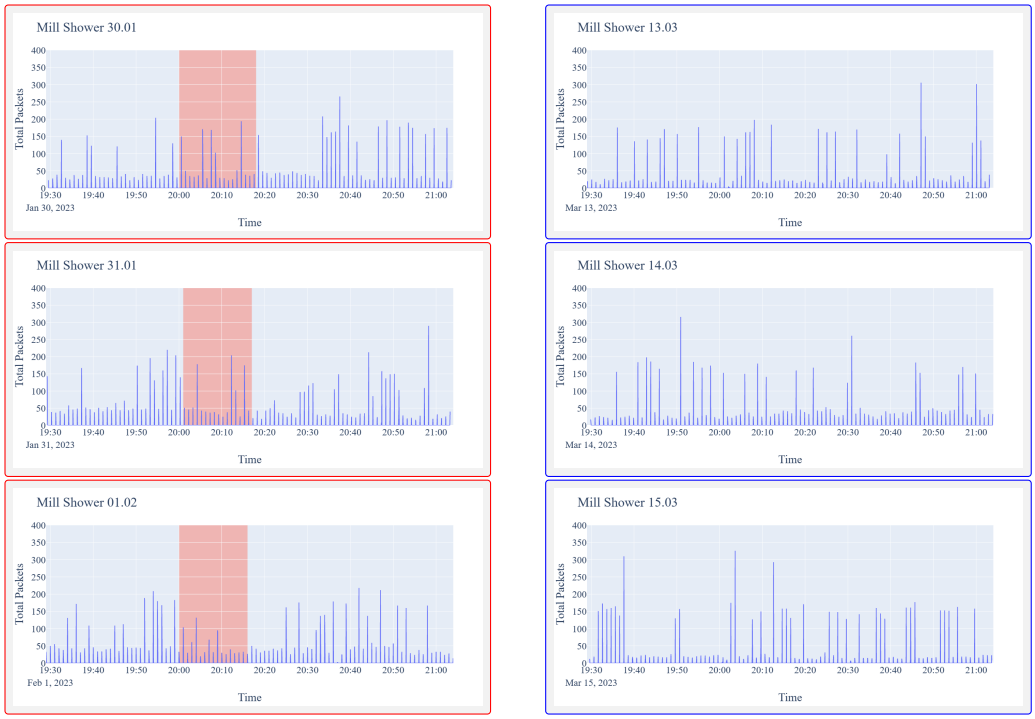


Figure 5.35: Continuing from Figure 5.34

Figure 5.36: Remaining graphs from Figure 5.37

Figure 5.37: Graphs of traffic flows from the showering events measured in bytes with event graphs framed in red and baseline graphs framed in blue for Mill. Event times are marked in red on the event graphs.

Table 5.19 shows that the number of packets varies from 7,775 to 9,819 packets during the events compared to the baseline which varies from 5,968 to 10,151 packets shown in Table 5.20. The baseline traffic has more variation than the event traffic. This also results in a higher standard deviation for the baseline shown in Table 5.21. For bytes, the same is shown, but the average value for the baseline is a lot smaller than for the events. As Table 5.20 shows, the bytes varies from 623,993 to 1,006,385 bytes, and in Table 5.19 the bytes varies from 909,992 to 1,166,768, which means that the average value for events are higher than for the baseline. Although the average value is higher, some of the baseline event days also measure similar amounts of bytes sent and received even though an event is not ongoing. This is also confirmed in Figure 5.33 where bytes for baseline are much lower than for events, but there are also events that are lower than baseline days.

The biggest packet is mainly 456 bytes both during the showering event and the corresponding baseline with some variations of bigger packets sent for both cases. This results in no significant difference during an event.

Table 5.11 shows that for both packets and bytes, the average value of the baseline is smaller than the standard deviation from the events. However, Figure 5.34, 5.35, 5.36 and 5.37 shows that that traffic patterns before the events are not comparable to the traffic patterns of the corresponding baseline graphs and therefore it is not the specific event triggering that is causing the differences in the average values for the bytes. This results in that the differences in calculations cannot be used to distinguish showering from standard traffic.

In the figures with packets, in Figure 5.34 and 5.35, and bytes, in Figure 5.36 and 5.37 no distinct differences are visible in the graphical representation of the traffic flows. There are spikes with more traffic sent before the events and during the event that are similar and does not identify the event executions. Neither from the calculations, nor the graphs for showering event for Mill it is possible to see a clear difference in traffic patterns from when the user is showering or not.

5.3.4 Nedis

This subsection presents the results and analysis for Nedis during the showering event. The total amount of packets, bytes and the biggest packet during the packet captures are presented in Table 5.22 for the events and in Table 5.23 for the corresponding baseline captures. Table 5.24 compares the average and standard deviation values calculated for the events and baseline. Figure 5.38 shows a graphical presentation of the total number of bytes and packets during the events and baseline in the same figure with average values included.

Table 5.22: Calculations on showering events for Nedis

Dates	Packets	Bytes	Biggest packet
08.jan	28,069	3,720,194	485 bytes
09.jan	23,156	3,395,004	424 bytes
11.jan	23,368	3,025,569	424 bytes
16.jan	21,026	2,598,912	485 bytes
18.jan	18,385	2,091,914	458 bytes
19.jan	25,416	3,258,851	421 bytes
25.jan	25,092	2,459,011	458 bytes
30.jan	22,415	3,302,325	485 bytes
31.jan	18,393	2,364,979	424 bytes
01.feb	20,306	2,461,409	485 bytes

Table 5.23: Calculations on comparing baseline files for the showering event for Nedis

Baseline	Packets	Bytes	Biggest packet
06.mar	12,495	1,534,872	522 bytes
07.mar	18,342	2,483,719	522 bytes
08.mar	18,318	2,116,146	424 bytes
09.mar	14,639	1,865,223	522 bytes
10.mar	9,893	1,434,033	522 bytes
11.mar	6,092	779,215	426 bytes
12.mar	10,736	1,273,551	426 bytes
13.mar	13,478	1,649,747	424 bytes
14.mar	14,239	1,668,084	522 bytes
15.mar	22,383	2,710,729	522 bytes

Table 5.24: Traffic comparison between the showering event and baseline for Nedis

	Type	Packets	Bytes	Biggest packet
Average	Showering	22,563	2,867,817	455 bytes
	Baseline	14,062	1,752,432	483 bytes
Standard deviation	Showering	3,130	540,631	29 bytes
	Baseline	4,723	571,269	50 bytes

Figure 5.38: Graphical presentation of event and baseline shower calculations with packets and bytes, including average value extracted from Table 5.24 for Nedis

In Figure 5.39 and 5.40, a graphical presentation of the traffic flow measured in number of packets is displayed. The same view is presented in Figure 5.41 and 5.42 with number of bytes as the y-axis. These figures have the event graphs placed on the left side of the figure framed in red, while the baseline graphs are placed on the right side of the figure framed in blue to compare to each other. The time of the events is marked red on all of the event graphs. The x- and y-axis have the same minimum and maximum values for all graphs within the same figure.

Figure 5.39: Graphs of traffic flows from the showering events measured in packets with event graphs framed in red and baseline graphs framed in blue for Nedis. Event times are marked in red on the event graphs.

Figure 5.40: Continuing from Figure 5.39

Figure 5.41: Remaining graphs from Figure 5.42

Figure 5.42: Graphs of traffic flows from the showering events measured in bytes with event graphs framed in red and baseline graphs framed in blue for Nedis. Event times are marked in red on the event graphs.

Comparing the values for both packets and bytes in Table 5.22 and 5.23 show that during the events, most of the values are a lot higher than for the baseline calculations. For event, the packets varies from 18,385 to 28,069 and bytes from 2,091,914 to 3,720,194 bytes. However, there are values from the baseline that matches the range of the event values, such as for 7th, 8th and 15th of March where the values for packets are over 18,000 and 2,000,000 for bytes. For the biggest packet sent, the baseline has a higher overall value compared to the event values, where baseline are mostly around 500 bytes and events are lower than 500 bytes for every day. The average values also show that the event values are higher than the baseline values for both packets and bytes. The difference in both average values and overall for events and baseline are significantly shown in Figure 5.38 where both packets and bytes are much lower for baseline than events.

Both in Table 5.24 and in Figure 5.38 a clear difference between showering and the baseline is visible. The average value for the baseline is much smaller than the standard deviation for the events for both packets and bytes. To look into if this difference is related to the event triggered, Figure 5.39, 5.40, 5.41 and 5.42 will be evaluated. In these figures it is very visible that the differences in calculations are not connected to the event triggered, but rather holds a higher number of packets and bytes before the event compared to the standard baseline traffic. Therefore, these calculations are not comparable and will not identify when showering is ongoing in the environment for Nedis.

Looking at the graphs in Figure 5.39 and 5.40 the difference in number of packets are visible. However, it does not look like the number of packets changes when the shower event starts, but are just higher during several of those days. In these figures, one can also see that some of the baseline days looks similar as the event calculations in the tables. For bytes in Figure 5.41 and 5.42, the results give the same result as for packets. The same result is found here for Nedis as for the event tests on cooking. The calculations under the event looks different than for standard traffic in the baseline, but the graphs over the traffic flows show that the differences are not related to when the event is triggered, but rather higher before and after the event. Therefore, it is not possible to see traffic pattern changes when showering is ongoing for Nedis.

5.4 Test Case 3: Window Open

This chapter presents the results and analysis on Test Case 3: Window Open. A general section applicable for all devices is presented before the results. The results from the test is presented separately in subsections for each device.

5.4.1 General

Test Case 3: Window Open has been conducted 10 times over the course of 10 different days. The days and timings for the events are presented in Table 5.25.

Table 5.25: Date and time for Test Case 3: Window Open

	08.01	09.01	11.01	16.01	18.01	19.01	25.01	30.01	31.01	01.02
Started window open	23:00	23:00	22:50	23:10	23:15	23:02	22:59	23:00	22:59	22:59
Finished window open	07:00	07:00	07:00	06:56	07:09	06:59	06:55	06:56	07:00	06:59

To be able to compare the events to standby traffic, the baseline capture has been used with the same timings as for the actual events. To easier compare the events with each other and against the baseline, all packet captures have been filtered with the same start and finish time. Due to time limitations, the baseline only includes 9 full nights and therefore only 9 corresponding baseline packet captures have been made and used for comparison in this section. To ensure that all events have at least 30 minutes before and after the event was ongoing, the earliest time for starting and the latest time for finishing the event has been used to calculate the start and finish times for the files. Then 30 minutes are added to these times to ensure that each event has at least 30 minutes before and after to see traffic changes. The timings are calculated from Table 5.25 and presented in the list beneath:

- Earliest window open start: 22:50
- Latest window open finished: 07:09
- Packet capture files start: 22:20
- Packet capture files end: 07:39

These times results in the following filters added to create the pcaps for each event:

- frame.time >= "Month Date, Year 22:20:00" && frame.time <= "Month Date, Year 07:39:00"

5.4.2 Netatmo

For the window open event for Netatmo, calculations are presented in Table 5.26 with number of packets and bytes and the biggest packet during the capture. The same calculations on the comparing baseline files to be used in this test are presented in Table 5.27. The average values and standard deviation are compared for the event and baseline in Table 5.28. The calculations are also shown graphically in Figure 5.43 with number of packets and bytes from Tables 5.26 and 5.27 combined with their average value.

Table 5.26: Calculations on window open events for Netatmo

Dates	Packets	Bytes	Biggest packet
08.jan	4,716	641,245	407 bytes
09.jan	4,026	544,344	407 bytes
11.jan	5,149	697,920	407 bytes
16.jan	4,493	608,029	407 bytes
18.jan	4,165	565,885	407 bytes
19.jan	3,877	527,113	407 bytes
25.jan	4,460	605,296	407 bytes
30.jan	3,745	509,093	407 bytes
31.jan	3,494	468,922	407 bytes
01.feb	3,858	522,632	407 bytes

Table 5.27: Calculations on comparing baseline files for the window open event for Netatmo

Baseline	Packets	Bytes	Biggest packet
06.mar	3,447	460,531	387 bytes
07.mar	4,356	591,234	407 bytes
08.mar	4,373	587,975	407 bytes
09.mar	4,513	609,590	407 bytes
10.mar	4,637	631,232	407 bytes
11.mar	4,010	545,391	407 bytes
12.mar	3,815	520,289	407 bytes
13.mar	3,571	478,194	407 bytes
14.mar	3,716	503,515	407 bytes

Table 5.28: Traffic comparison between the window open event and baseline for Netatmo

	Type	Packets	Bytes	Biggest packet
Average	Window open	4,198	569,048	407 bytes
	Baseline	4,049	547,550	405 bytes
Standard deviation	Window open	503	68,966	0 bytes
	Baseline	436	60,687	7 bytes

Figure 5.43: Graphical presentation of event and baseline window open calculations with packets and bytes, including average value extracted from Table 5.28 for Netatmo

Graphs over traffic patterns are presented in four different figures, for packets in Figure 5.44 and 5.45 and for bytes in Figure 5.46 and 5.47. The graphs within the same figure have the same minimum and maximum values on the x- and y-axis. For all the figures, the event graphs are placed on the left side framed in red and have a red area marked on the graph which shows when the event was ongoing. The corresponding baseline graphs are placed on the right side of the figure and framed in blue to compare.

Figure 5.44: Graphs of traffic flows from the window open events measured in packets with event graphs framed in red and baseline graphs framed in blue for Netatmo. Event times are marked in red on the event graphs.

Figure 5.45: Continuing from Figure 5.44

Figure 5.46: Remaining graphs from Figure 5.47

Figure 5.47: Graphs of traffic flows from the window open events measured in bytes with event graphs framed in red and baseline graphs framed in blue for Netatmo. Event times are marked in red on the event graphs.

Comparing Table 5.26 for events and Table 5.27 for baseline files, shows that the different columns are similar. The total number of packets stays around 4,000 packets for both events and baseline. For bytes, the values ranges from around 500,000 to around 700,000. This is also easily visible in Figure 5.43 where total number of packets and bytes are very similar. The biggest packet for events and the baseline is 407 bytes, with only one exception of 387 bytes for one of the baseline days. This value can therefore not be used to identify a window open in the event.

Table 5.28 and Figure 5.43 shows that the values for window open and baseline are very similar. The average value for baseline packets and bytes are inside of the standard deviation for the window open. This means that it is not possible to identify if a user has a window open using these calculations.

Looking at the graphs in Figure 5.44 and 5.45, there are no significant differences in the graphs for the events compared to the baseline on the right side. For both events and baseline graphs, the same traffic pattern is found. For bytes in Figure 5.46 and 5.47, the events varies with some nights having many and high spikes, and other nights with smaller and fewer spikes. The same pattern is found for the baseline graphs marked in blue. This shows that it is not possible to distinguish whether the window is open or not by looking at the traffic patterns in a numerical or graphical way from the device Netatmo.

5.4.3 Mill

Table 5.29 presents the number of packets and bytes and the biggest packet during the capturing of the window open event for Mill. The same calculations are included for the comparing baseline files in Table 5.30. To compare average and standard deviation values from the event, Table 5.31 is shown. The calculations are also presented graphically in Figure 5.48 where number of packets and bytes for both the event dates and comparing baseline dates are included together with their average value extracted from Table 5.31.

Table 5.29: Calculations on window open events for Mill

Dates	Packets	Bytes	Biggest packet
08.jan	49,204	6,159,905	1,353 bytes
09.jan	49,515	6,055,179	1,545 bytes
11.jan	45,680	4,998,933	1,515 bytes
16.jan	45,559	4,881,294	1,353 bytes
18.jan	56,021	7,122,722	1,593 bytes
19.jan	40,090	4,736,715	1,573 bytes
25.jan	52,728	6,166,428	1,421 bytes
30.jan	44,015	5,089,165	1,589 bytes
31.jan	40,616	4,501,888	1,353 bytes
01.feb	37,963	4,030,897	456 bytes

Table 5.30: Calculations on comparing baseline files for the window open event for Mill

Baseline	Packets	Bytes	Biggest packet
06.mar	42,541	4,463,880	456 bytes
07.mar	44,018	4,634,112	800 bytes
08.mar	51,730	5,307,249	1,583 bytes
09.mar	56,022	5,568,551	1,343 bytes
10.mar	57,262	6,237,897	1,593 bytes
11.mar	49,145	4,959,292	1,593 bytes
12.mar	44,181	4,523,140	1,583 bytes
13.mar	49,628	5,238,854	456 bytes
14.mar	42,318	4,449,260	1,589 bytes

Table 5.31: Traffic comparison between the window open event and baseline for Mill

	Type	Packets	Bytes	Biggest packet
Average	Window open	46,139	5,374,313	1,353 bytes
	Baseline	48,538	5,042,471	1,222 bytes
Standard deviation	Window open	5,782	954,686	338 bytes
	Baseline	5,678	606,684	505 bytes

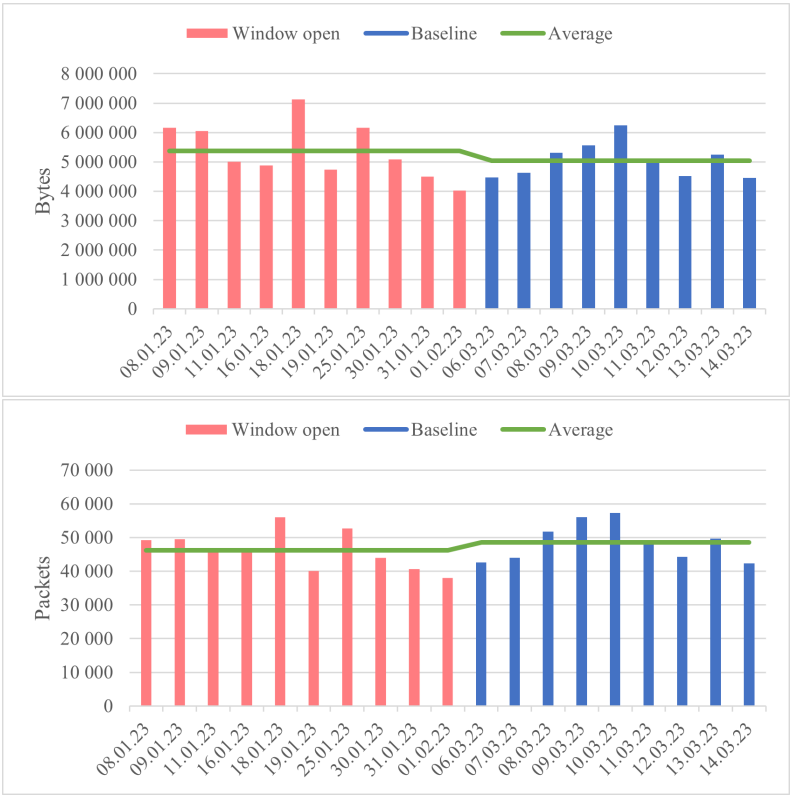


Figure 5.48: Graphical presentation of event and baseline window open calculations with packets and bytes, including average value extracted from Table 5.31 for Mill

The packet capturings from the window open event for Mill is presented with traffic flows in four different figures. Figure 5.49 and 5.50 shows the traffic patterns for packets and Figure 5.51 and 5.52 for bytes. The event graphs are marked in red and placed on the left side of the figures, with a red marked area on the graphs to show when the event was ongoing. The comparing baseline graphs are placed on the right side of the figures and framed in blue. All graphs in the same figure have the same minimum and maximum value for the x- and y-axis.

Figure 5.49: Graphs of traffic flows from the window open events measured in packets with event graphs framed in red and baseline graphs framed in blue for Mill. Event times are marked in red on the event graphs.

Figure 5.50: Continuing from Figure 5.49

Figure 5.51: Remaining graphs from Figure 5.52

Figure 5.52: Graphs of traffic flows from the window open events measured in bytes with event graphs framed in red and baseline graphs framed in blue for Mill. Event times are marked in red on the event graphs.

Comparing Table 5.29 and 5.30 shows that the number of packets sent during the events and baseline do not show a significant difference. The packets both varies from around 40,000 to 60,000 when an event is ongoing, in Table 5.29, to when an event is not triggered in Table 5.30. The same is shown in bytes as it varies from 4,000,000 to 7,000,000 for both baseline and window open. The biggest packet sent is also similar in the event and baseline capturings where most values are over 1,000 bytes, but both cases have packets that are also under that, such as 456 bytes for events and 456 and 800 bytes for the baseline.

In Table 5.29, the average value for packets and bytes for the baseline is not outside of the standard deviation for the window open event. The fact that the calculations are very similar is also shown in Figure 5.48. The result shows that these calculations cannot be used to identify if a user has a window open in the environment through the AQM Mill.

The traffic flows with packets in Figure 5.49 and 5.50 do not show a significant difference between the events on the left side, compared to the baseline packets on the right side of the figure. Both cases display graphs with variations that reaches approximately the same level of packets. It is not possible to see changes in the traffic from the red marked area under the events compared to the timings when events are not ongoing. The same result is found for bytes in Figure 5.51 and 5.52. As the results of the graphs from both the baseline and the event looks familiar, there are no traffic patterns changes which can identify that a window is open in the environment.

5.4.4 Nedis

For the window open event for Nedis, Table 5.32 presents calculations for the events, while Table 5.33 presents calculations for the comparing baseline files. Both tables includes the number of packets and bytes sent and the biggest packet captured during the event. In Table 5.34 the average and standard deviation value for the event and comparing baseline are presented. The values are also presented graphically in Figure 5.53 for number of packets and bytes including average values.

Table 5.32: Calculations on window open events for Nedis

Dates	Packets	Bytes	Biggest packet
08.jan	131,961	18,088,388	424 bytes
09.jan	121,273	15,204,877	424 bytes
11.jan	94,682	9,432,611	421 bytes
16.jan	83,344	8,875,795	424 bytes
18.jan	94,106	10,706,230	458 bytes
19.jan	97,324	12,565,538	485 bytes
25.jan	88,413	9,470,205	421 bytes
30.jan	59,874	6,939,737	424 bytes
31.jan	96,309	13,204,218	424 bytes
01.feb	101,872	12,988,077	485 bytes

Table 5.33: Calculations on comparing baseline files for the window open event for Nedis

Baseline	Packets	Bytes	Biggest packet
06.mar	77,759	8,062,010	426 bytes
07.mar	105,395	12,437,964	426 bytes
08.mar	109,084	13,759,303	424 bytes
09.mar	111,832	15,941,453	458 bytes
10.mar	115,626	15,852,737	485 bytes
11.mar	58,325	7,096,260	426 bytes
12.mar	89,493	11,740,679	485 bytes
13.mar	89,958	11,137,820	424 bytes
14.mar	91,427	11,865,346	485 bytes

Table 5.34: Traffic comparison between the window open event and baseline for Nedis

	Type	Packets	Bytes	Biggest packet
Average	Window open	96,916	11,767,368	439 bytes
	Baseline	94,322	11,988,175	449 bytes
Standard deviation	Window open	19,686	3,334,883	27 bytes
	Baseline	18,445	3,042,357	29 bytes

Figure 5.53: Graphical presentation of event and baseline window open calculations with packets and bytes, including average value extracted from Table 5.34 for Nedis

The event is also shown graphically in four different figures. Figure 5.54 and 5.55 shows traffic flows with packets and Figure 5.56 and 5.57 for bytes. All graphs within the same figure have equal minimum and maximum values for the x- and y-axis. The event graphs are placed on the left side of the figure and marked in red, while the comparing baseline graphs are placed on the right side of the figure and marked in blue. The event graphs also have a red area marked on the graphs for when the event was ongoing.

Figure 5.54: Graphs of traffic flows from the window open events measured in packets with event graphs framed in red and baseline graphs framed in blue for Nedis. Event times are marked in red on the event graphs.

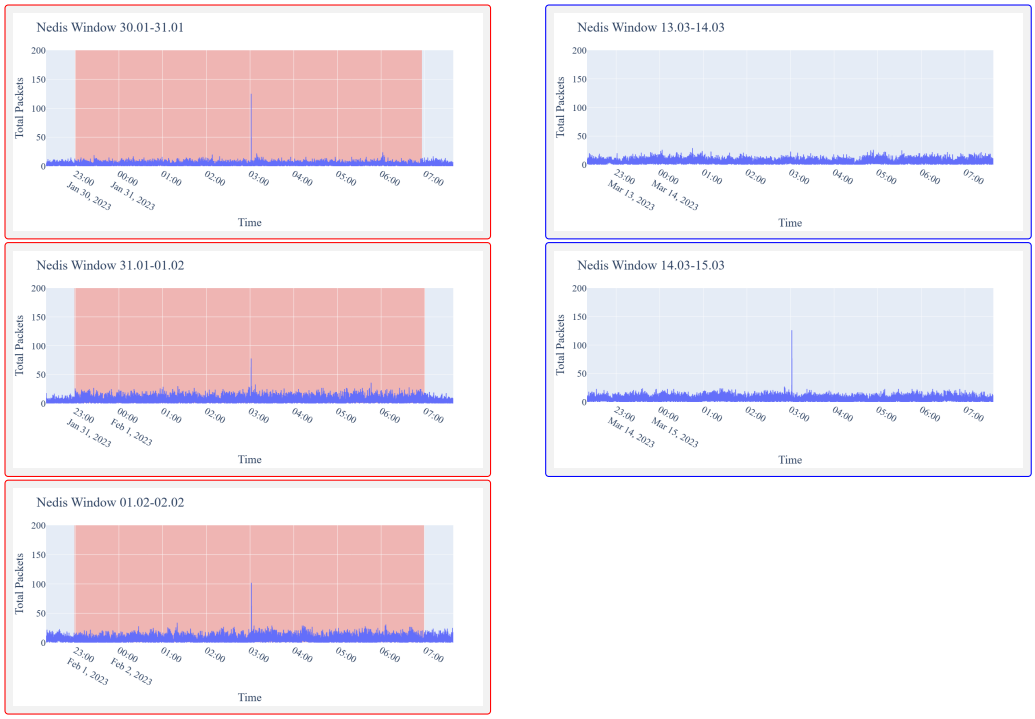


Figure 5.55: Continuing from Figure 5.54

Figure 5.56: Remaining graphs from Figure 5.57

Figure 5.57: Graphs of traffic flows from the window open events measured in packets with event graphs framed in red and baseline graphs framed in blue for Nedis. Event times are marked in red on the event graphs.

Comparing Table 5.32 for events and 5.33 for baseline calculations shows that both cases varies a lot in number of packets and bytes during the traffic capture. For both events and baseline, the number of packets varies from around 50,000 to over 100,000 packets, which is a big difference. The same applies to bytes where both varies from around 7,000,000 to over 15,000,000 bytes. There are no significant differences in the biggest packet during the capture, where all days within both cases are between 400 and 500 bytes.

Even though the number of packets and bytes varies much for each event and baseline day, they vary almost equally as the average value shown in Table 5.34 are almost the same. The average value for both packets and bytes for the baseline is not outside of the standard deviation for the window open events and cannot be used to distinguish if the window is open or not in the environment. The graphs in Figure 5.53 also demonstrates how much the packets vary, but still the average values are very close to each other.

For the packet traffic flows presented in Figure 5.54 and 5.55 the event graphs do have some more spikes overall compared to the baseline graphs. However, 4 out of 9 of the baseline days also have spikes that are similar to the event days where there are only one spike included. The bytes traffic flows presented in Figure 5.56 and 5.57 shows well the variations that were visible in the calculations presented. Both on the left side with events and on the right side of the figure with the baseline traffic there are variations easily visible. Therefore, there are no significant differences in events and baseline traffic as the variations are shown in both cases. This results in that it is not possible to identify if a window is open in the environment compared to standard traffic which is not triggered by the same event.

5.5 Test Case 4: Weekends

This chapter presents the results and analysis for Test Case 4: Weekends. First, a section with general results which applies to all the devices are presented. Afterwards, the results for each of the devices are presented separately in their own subsection.

5.5.1 General

Test Case 4: Weekends is tested over the course of 14 different weekends. 7 weekends when the environment was occupied, meaning that the user was at home, and 7 when the environment were not occupied, meaning that user was not home during this time. The different dates are described in Table 5.35.

Table 5.35: Dates for Test Case 4: Weekends

	Dates
Weekends at home	13.01.2023-15.01.2023
	27.01.2023-29.01.2023
	03.02.2023-05.02.2023
	17.02.2023-19.02.2023
	10.03.2023-12.03.2023
	28.03.2023-30.03.2023
	31.03.2023-01.04.2023
Weekends gone	23.12.2022-25.12.2022
	30.12.2022-01.01.2023
	20.01.2023-22.01.2023
	10.02.2023-12.02.2023
	24.02.2023-26.02.2023
	03.03.2023-05.03.2023
	17.03.2023-19.03.2023

The times for weekends at home and gone were from 16:00 at Friday to 22:00 at Sunday. In a weekend were the home was occupied, it was variably occupied. Some weekends a lot of time were spent in the home and other weekends it was just occupied during evenings or daytime, but every occupied weekend were spent sleeping at night in the home. One weekend, 28.03.2023-30.03.2023, were tested from Tuesday to Thursday, but are treated as Tuesday=Friday and Thursday=Sunday.

This results in the following filter used on the pcaps to created the files:

- frame.time >= "Month Date, Year 16:00:00" && frame.time <= "Month Date, Year 22:00:00"

5.5.2 Netatmo

For the weekend test for Netatmo, calculations are listed in Table 5.36, for weekends at home, and Table 5.37, for weekends gone. The different values are total number of packets and bytes during the packet capturing and the biggest packet sent in bytes, biggest src, and the biggest packet size in bytes received, biggest dst. Table 5.38 compares the average and Standard Deviation (SD) for all weekends.

Table 5.36: Calculations for weekends at home for Netatmo

Date	Packets	Bytes	Biggest src	Biggest dst
13.01-15.01	31,640	4,285,664	407 bytes	418 bytes
27.01-29.01	25,465	3,436,632	407 bytes	154 bytes
03.02-05.02	24,887	3,379,806	444 bytes	396 bytes
17.02-19.02	23,654	3,202,102	1,130 bytes	154 bytes
10.03-12.03	24,881	3,372,230	407 bytes	136 bytes
28.03-30.03	27,555	3,743,121	407 bytes	154 bytes
31.03-02.04	28,445	3,852,003	407 bytes	418 bytes

Table 5.37: Calculations for weekends gone for Netatmo

Date	Packets	Bytes	Biggest src	Biggest dst
23.12-25.12	22,367	2,987,324	134 bytes	136 bytes
30.12-01.01	22,553	3,012,868	134 bytes	136 bytes
20.01-22.01	24,631	3,288,842	134 bytes	154 bytes
10.02-12.02	20,320	2,715,486	134 bytes	136 bytes
24.02-26.02	21,332	2,849,186	134 bytes	136 bytes
03.03-05.03	19,023	2,538,937	134 bytes	136 bytes
17.03-19.03	23,905	3,191,924	134 bytes	136 bytes

Table 5.38: Traffic comparison of weekend values from Table 5.36 and 5.37

		Packets	Bytes	Biggest src	Biggest dst
Average	Home	26,647	3,610,223	516 bytes	261 bytes
	Gone	22,019	2,940,652	134 bytes	139 bytes
SD	Home	2,756	373,892	271 bytes	140 bytes
	Gone	1,963	262,109	0 bytes	7 bytes

Graphs of packets are presented in Figure 5.58 and bytes in Figure 5.59. In these figures, the weekends at home have a green frame and placed on the left side, while the weekends gone have an orange frame and placed on the right side of the figure. All graphs in the same figure have the same maximum value on the y-axis and have the same timings on the x-axis to be easily comparable.

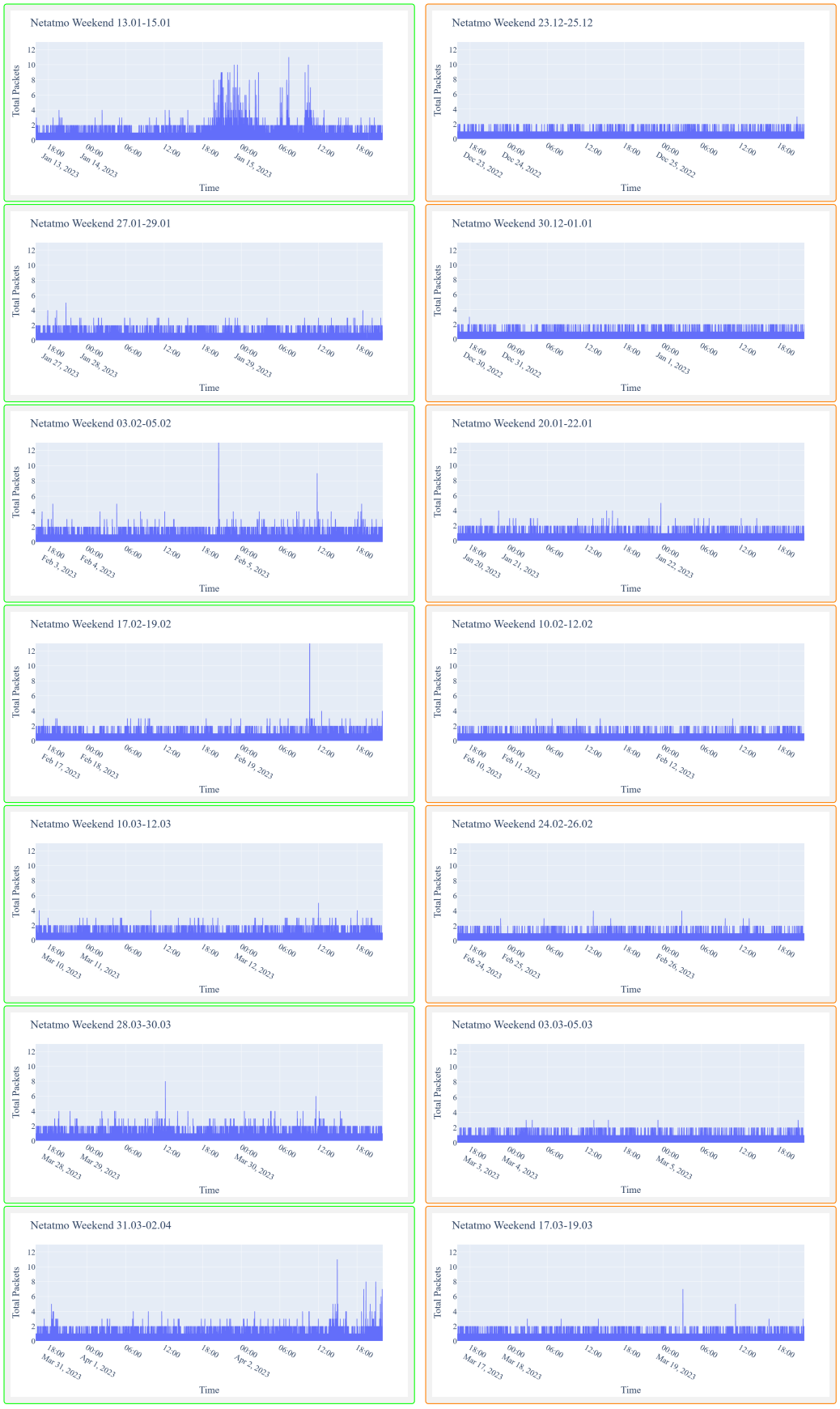


Figure 5.58: Traffic patterns for weekends at home, marked in green, and gone, marked in orange, measured in total amount of packets sent and received for Netatmo during the weekends

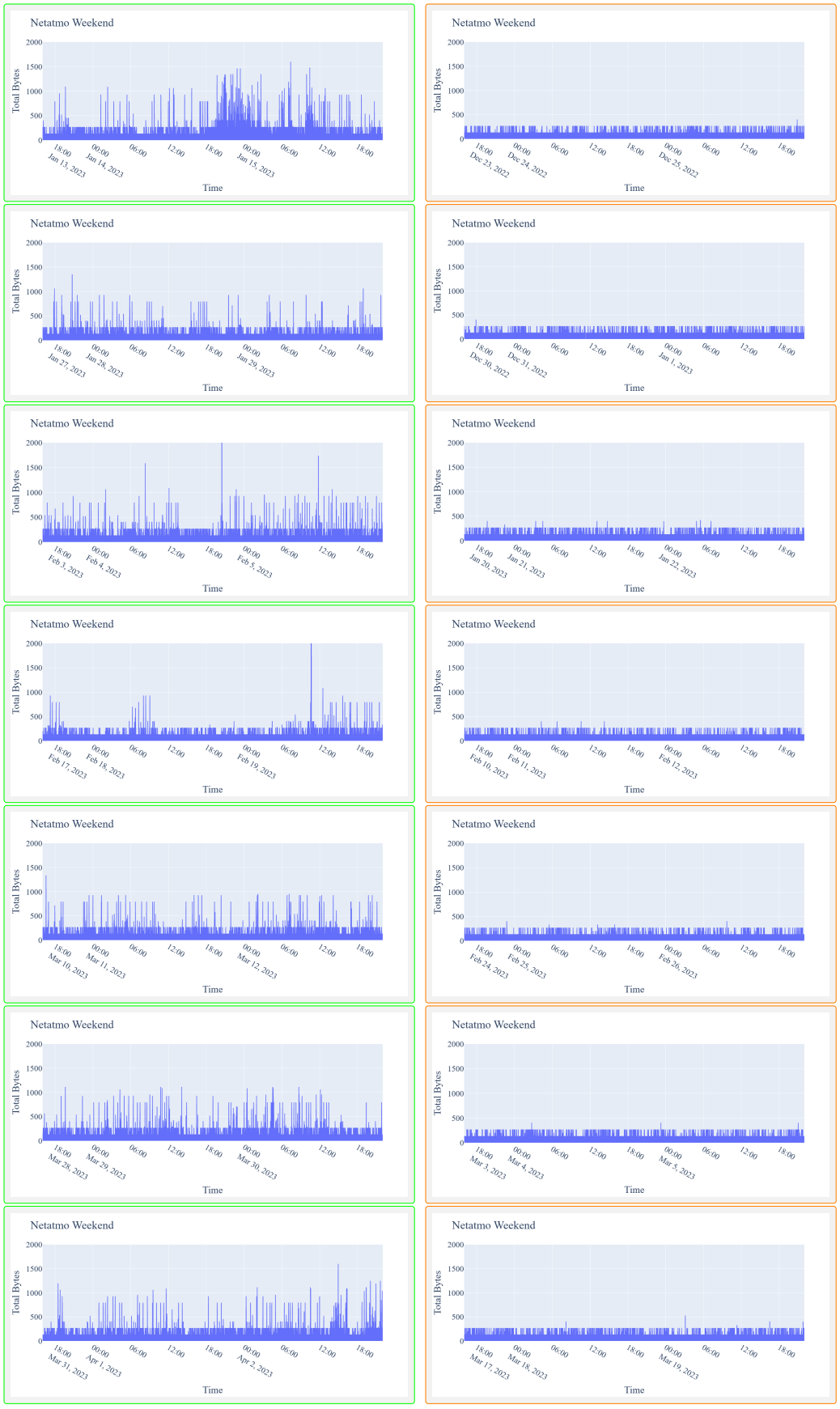


Figure 5.59: Traffic patterns for weekends at home, marked in green, and gone, marked in orange, measured in total amount of bytes sent and received for Netatmo during the weekends

The calculations for weekends show a significant difference from when the home is occupied to when its not. Comparing the total packets between Table 5.36 and 5.37 does not show a big difference with 26,647 packets at home to 22,019 packets when gone in average. However, in bytes the difference is more significant. The difference of the average value for weekends at home compared to gone is nearly 500,000 bytes, as shown in Table 5.38. It is also shown in the table that the average value for packets and bytes for the weekends gone, are smaller than the standard deviation than for the weekends at home. Therefore, we can conclude that there are differences in the number of packets and bytes sent when a user is home or not and can be used to infer private information.

Another difference is in the biggest packet sent in the *biggest src* column. For the weekends gone, in Table 5.37, the biggest packet sent is only 134 bytes, while for the weekends at home, in Table 5.36, the biggest packet sent is always much larger than 134 bytes with a variation from 407 bytes to 1,130 bytes. Since the biggest packet overall for gone weekend are 154 bytes, this will be used as a reference to look at differences to packets sent while at home.

The graphs in Figure 5.58 with packets shows small differences in home and gone activity. The home graphs have more spikes, but there are also some graphs for gone that have small spikes and look similar to the home graphs. However, for the graphs measured in bytes in Figure 5.59 it is a clear difference to when the home is occupied or not. All the graphs for weekends at home, marked in green, shows a very different traffic pattern than the graphs for the weekends gone, marked in orange. This corresponds to the findings from the calculations in Table 5.36 and 5.37.

Since it is possible to see a difference in biggest packet during weekends at home and gone, it is also interesting to see if these bigger packets only occur a few times during the weekend or if it is sending larger packets during the whole weekend. Therefore, a new filter have been applied to the pcaps to see how many packets during a weekend at home is actually bigger than the traffic sent on a weekend gone. A filter filtering on only packets larger then 154 bytes have been applied to the pcaps, and have the following format:

- frame.len $>$ 154

Table 5.39 shows the results from the filtering. The percentage of packets that are over 154 bytes in Table 5.39 is very small. A graphical view is therefore presented in Figure 5.60 to see if the amount is enough to see differences from the weekends gone which does not have any packets over 154 bytes. The graphs in Figure 5.60 shows that it is possible to see if a home is occupied by looking at the packet sizes that are sent.

Table 5.39: Amount of packets bigger than 154 bytes sent on weekends at home for Netatmo

Dates	Packets	Percentage
13.01-15.01	251	0.8%
27.01-29.01	194	0.8%
03.02-05.02	271	1.1%
17.02-19.02	171	0.7%
10.03-12.03	224	0.9%
28.03-30.03	446	1.6%
31.03-02.04	275	1%
Average	**262**	**0.99%**
SD	**90**	**0.30%**

Figure 5.60: Traffic patterns for weekends at home with filtered with packets over 154 bytes for Netatmo

The last evaluation on Netatmo weekend events is to compare the graphs for

bytes in Figure 5.59 against values from the application of the device. This is presented in Figure 5.61 and 5.62. The graphs are extracted from the app, Healthy Home Coach, and the sensor values are in CO_2. The y-axis, which are measured in ppm are not equal for each of the graphs from the app, as this is not possible to change manually in the app. The x-axis is given in time.

The Figures in 5.61 and 5.62 shows the same pattern as for the bytes from the traffic captures on the left side of the figure. When the home is not occupied, the CO_2 value does not change much which the corresponding traffic flow in the byte-graphs shows, and when the home is occupied, the CO_2 levels varies a lot and the corresponding traffic flow also varies a lot. It is therefore shown that it is possible to infer whether a user is home or not by looking at the traffic pattern both graphically and with calculations.

Figure 5.61: Traffic flows in bytes for weekends at home, framed in green, and gone, framed in orange, compared to CO2 values from the corresponding dates in the application, Healthy Home Coach

Figure 5.62: Remaining graphs from Figure 5.61

5.5.3 Mill

For weekend testing with the device, Mill, Table 5.40 and 5.41 presents the calculations for the weekends at home and gone. The values presented are total number of packets and bytes sent and received and the biggest packet sent, biggest src, and received, biggest dst. Average and standard deviation values from the tables are combined and compared in Table 5.42.

Table 5.40: Calculations for weekends at home for Mill

Date	Packets	Bytes	Biggest src	Biggest dst
13.01-15.01	285,543	33,269,620	456 bytes	1,593 bytes
27.01-29.01	240,434	29,179,764	456 bytes	1,593 bytes
03.02-05.02	223,095	28,543,539	456 bytes	1,593 bytes
17.02-19.02	237,964	25,637,948	456 bytes	1,583 bytes
10.03-12.03	295,395	31,161,679	456 bytes	1,593 bytes
28.03-30.03	342,001	39,951,155	456 bytes	421 bytes
31.03-02.04	275,456	30,155,191	456 bytes	421 bytes

Table 5.41: Calculations for weekends gone for Mill

Date	Packets	Bytes	Biggest src	Biggest dst
23.12-25.12	259,516	34,066,162	456 bytes	421 bytes
30.12-01.01	255,487	32,809,172	456 bytes	421 bytes
20.01-22.01	256,455	29,417,721	456 bytes	421 bytes
10.02-12.02	270,718	30,482,741	456 bytes	421 bytes
24.02-26.02	218,536	25,525,312	456 bytes	421 bytes
03.03-05.03	236,655	23,064,715	456 bytes	421 bytes
17.03-19.03	278,146	28,612,487	456 bytes	421 bytes

Table 5.42: Traffic comparison of weekend values from Table 5.40 and 5.41

		Packets	Bytes	Biggest src	Biggest dst
Average	Home	271,413	31,128,414	456 bytes	1,257 bytes
	Gone	253,645	29,139,759	456 bytes	421 bytes
SD	Home	41,205	4,546,021	0 bytes	571 bytes
	Gone	20,224	3,870,041	0 bytes	0 bytes

Graphs of traffic flows are presented in Figure 5.63 for packets and in Figure 5.64 for bytes. The graphs marked in green on the figures are weekends at home and are placed on the left side of the figures, while the graphs marked in orange on the figures are weekends gone and placed on the right side of the figures. The graphs in the same figure all have the same maximum values for the y-axis to be easily comparable.

Figure 5.63: Traffic patterns for weekends at home, marked in green, and gone, marked in orange, measured in total amount of packets sent and received for Mill during the weekends

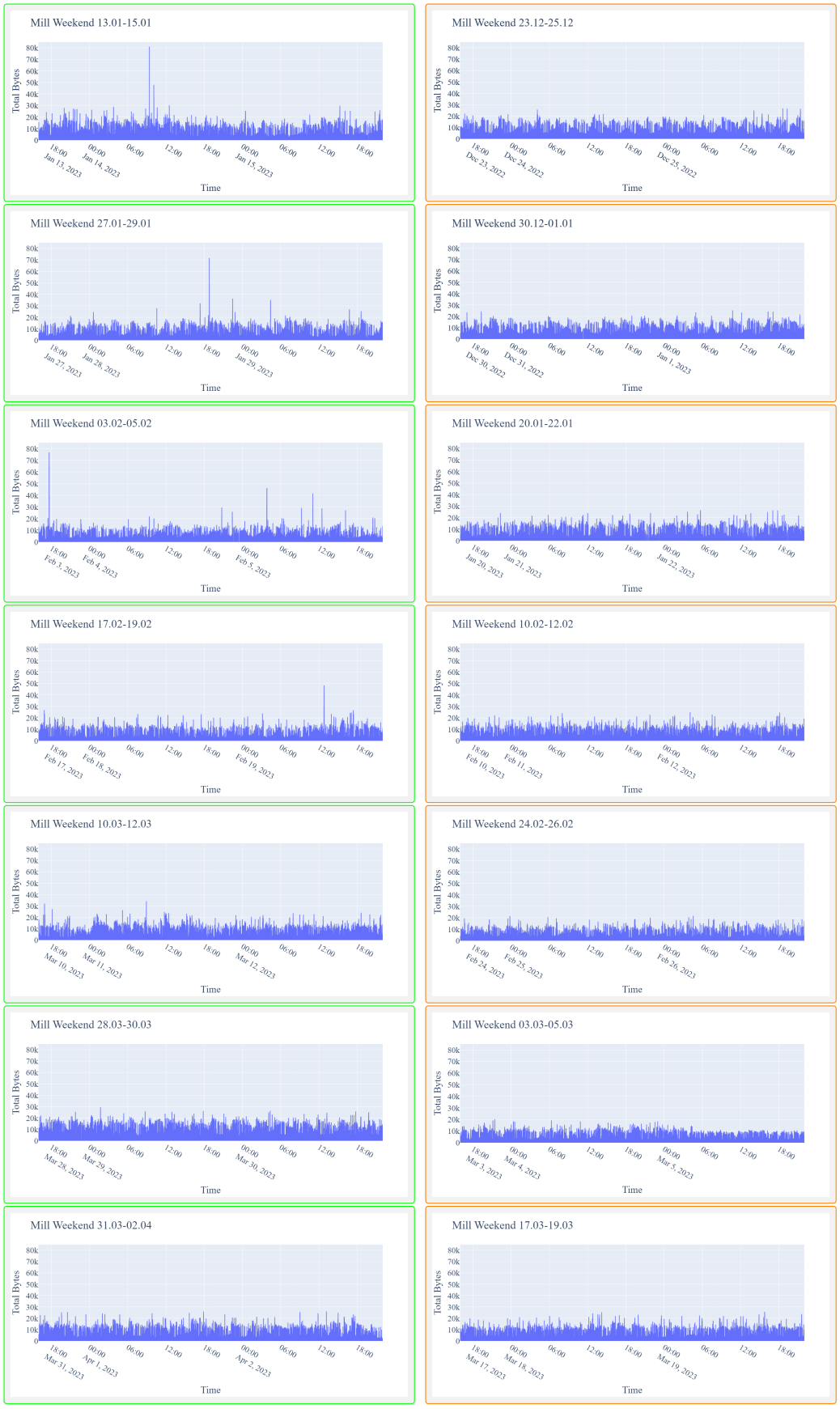


Figure 5.64: Traffic patterns for weekends at home, marked in green, and gone, marked in orange, measured in total amount of bytes sent and received for Mill during the weekends

Looking at the values in Table 5.40 and 5.41, the total packets and bytes remains the same whether or not it's a weekend at home or gone. In regards of the largest packets, all the weekends gone have the same size; 456 bytes for packets sent from the device and 421 bytes for packets sent to the device. For the first 5 weekends at home it looks like there's a pattern change that the device receives much bigger packets than during a weekend gone with a size of 1,583 and 1,593 bytes, but as the last two weekends at home shows their biggest packet is actually the same as for a weekend gone.

The values in Table 5.42 shows that the average value for packets and bytes for the weekends gone is not smaller than the standard deviation for the weekends at home. Therefore, these calculations cannot be used to infer whether a user is home or gone.

The graphs in Figure 5.63 and 5.64 shows that the traffic pattern for weekend at home or gone look the same. There are variations in the both weekend at home or gone, but not a constant pattern which distinguishes weekends at home from gone. This means that is it not possible to distinguish whether a user is home or not by looking at the traffic patterns for Mill.

5.5.4 Nedis

Table 5.43 and 5.44 shows the calculations for weekends at home and gone for Nedis. The values included are total amount of packets, total number of bytes, the biggest packet sent, biggest src, and the biggest packet received, biggest dst. Table 5.45 compares the average and SD values for all the weekends at home and weekends gone.

Table 5.43: Calculations for weekends at home for Nedis

Date	Packets	Bytes	Biggest src	Biggest dst
13.01-15.01	565,380	66,062,682	485 bytes	522 bytes
27.01-29.01	546,269	58,677,501	424 bytes	522 bytes
03.02-05.02	501,936	62,219,835	485 bytes	522 bytes
17.02-19.02	501,747	64,037,197	485 bytes	522 bytes
10.03-12.03	482,205	60,333,999	485 bytes	522 bytes
28.03-30.03	578,033	65,159,251	424 bytes	426 bytes
31.03-02.04	621,200	81,810,849	485 bytes	458 bytes

Table 5.44: Calculations for weekends gone for Nedis

Date	Packets	Bytes	Biggest src	Biggest dst
13.01-15.01	678,979	98,642,277	485 bytes	522 bytes
27.01-29.01	655,417	91,850,956	485 bytes	650 bytes
03.02-05.02	518,316	66,069,105	485 bytes	522 bytes
17.02-19.02	426,157	53,137,691	424 bytes	522 bytes
10.03-12.03	357,715	42,390,770	424 bytes	522 bytes
28.03-30.03	507,506	65,163,914	485 bytes	522 bytes
31.03-02.04	634,018	87,175,396	554 bytes	522 bytes

Table 5.45: Traffic comparison of weekend values from Table 5.44 and 5.43

		Packets	Bytes	Biggest src	Biggest dst
Average	Home	542,396	65,471,616	470 bytes	522 bytes
	Gone	539,730	72,061,444	477 bytes	540 bytes
SD	Home	49,893	7,665,988	30 bytes	40 bytes
	Gone	121,922	21,010,071	44 bytes	48 bytes

Figure 5.65 presents the graphs for weekend traffic measured in packets and Figure 5.66 presents the graphs for weekend traffic measured in bytes. Both figures have the weekends at home placed on the left side of the figure framed in green and the weekends gone placed on the right side of the figure framed in orange. All graphs in the same figure have the same values for the x- and y-axis to be comparable.

Figure 5.65: Traffic patterns for weekends at home, marked in green, and gone, marked in orange, measured in total amount of packets sent and received for Nedis during the weekends

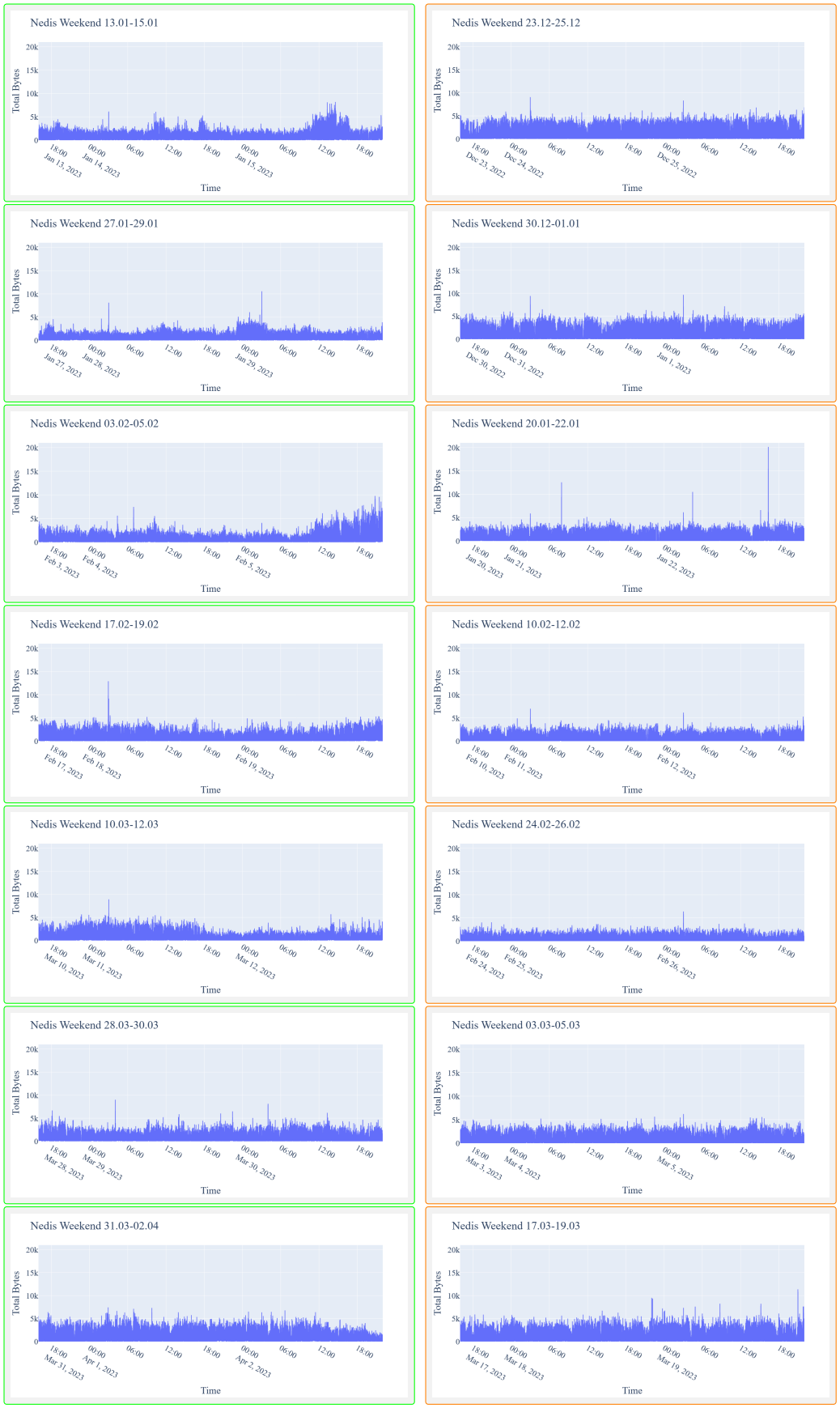


Figure 5.66: Traffic patterns for weekends at home, marked in green, and gone, marked in orange, measured in total amount of bytes sent and received for Nedis during the weekends

The results in Table 5.43 and 5.44 shows that the amount of packets does not change whether its a weekend at home or gone. Both tables show that there are sent and received around 500,000 to 600,000 packets regardless if the user is home or not. The amount of bytes varies a lot in both tables from 58,677,501 bytes to 81,810,849 bytes for the weekends at home and from 42,390,770 bytes to 98,642,277 bytes for the weekends gone. Looking at the biggest packet received and sent, shows very similar values for both weekends at home and gone. Table 5.45 also gives the same results. Neither average values for packets, bytes or biggest packets received or sent differs significantly between a weekend at home or gone. The standard deviation for weekends gone are some higher than for weekends at home which means that the values differ more from each other, but since the average is similar and the values differ both higher and lower than the average, it is still not possible to see any significant differences here.

The values in Table 5.45 shows that the average value for weekends gone are within the range of the standard deviation for weekends at home for both packets and bytes. This results that it is not possible to infer whether a user is home or gone during a weekend using these calculations from the AQM Nedis.

The graphs in Figure 5.65 and 5.66 also shows that there are no significant differences in the traffic pattern when a user is home or gone for the weekend. The spikes at around 3 am, which were also explained in the baseline capture, occurs at both home and gone. The results therefore shows that it is not possible to discovery if a user is home or not by looking at the traffic patterns from Nedis.

Chapter 6

Discussion

This chapter discusses the research questions by using the results found in the four test cases of this research. The questions will be answered sequentially. The chapter also discusses the work done in this thesis, both what was done good which lead to new knowledge and contributions, but also what could have been done differently. Challenges and mistakes which lead to decisions being made are explained here. Limitations to this research will also be included as this could have affected how strong the results are and can explain why certain decisions were made.

6.1 Answer to Research Question 1

RQ1: What kind of information can be gathered from air quality monitors when carrying out a passive network eavesdropping attack?

In this research, four different test cases have been used to test if it is possible see traffic pattern changes from when an event is triggered in the environment the device relies. By looking at calculations and graphical views over the traffic flows during the test, we were able to find results to answer the research questions in this thesis. The baseline capturings showed that all the devices communicates with a layer of security, as the packets sent on Wi-Fi are encrypted and not readable while carrying out a passive network eavesdropping attack. Therefore it is not possible to gather private information sent in the payloads to and from the devices.

For the routine behavioural events defined as cooking, showering and window open, only showering and window open gave notifications of threshold values exceeded for all executions. None of the devices showed a significant difference from when an event was triggered at a specific time compared to the traffic sent before and after the event or compared to the baseline capturings. This means that it was not possible to expose the private information of the user behaviour: cooking, showering or window open during night for any of the three devices: Netatmo Smart Indoor AQM, Mill Sense or Nedis SmartLife.

While comparing the baseline traffic to the event traffic, the devices Nedis SmartLife and Mill Sense showed that there are visible differences from the baseline capturings to right before and after the events which should expectedly have been the same. This can be a result that the traffic can be affected by smaller indoor changes such as a season or location in the environment which can change the indoor air quality. For example, during summer windows can be more open than during winter where ovens can be more used.

However, for the weekend test, one of the devices shows a clear difference from when a user is home or gone during the time period of a weekend. When Netatmo Smart Indoor Air Quality Monitor was used, it shows that it is possible to distinguish whether a user is home or gone by looking at the total number of bytes sent and received to that device during the weekend and looking at the size of the packets sent by the device which is never higher than 134 bytes when the user is gone for the weekend. This kind of information will expose private information about a user where it is possible to see whether the user is home or not for a longer period of time. These two factors proves that it is possible to infer private information from the device. For Mill Sense and Nedis SmartLife, no similar findings were discovered during the tests. These devices did not expose any private information during the weekend test.

In the weekend testing, it looked like there were differences for both Mill and Netatmo for the first 5 weekends at home, but only Netatmo had the same significantly different pattern for weekends at home and gone. For Mill, the pattern changed for the two last weekends at home where the packet size were the same as for the weekends gone. This shows the importance of including enough executions for each of the test cases. However, for over 70% of the weekends at home, a clear difference is visible when a user is at home compared to gone. Due to time constraints, no more weekends were tested, but more tests could reveal if these two weekends were an exception or if it is not possible to distinguish a weekend at home or gone for this device. For Nedis, no significant differences were possible to see during the weekend testing and therefore private information about whether a user is at home or gone during the weekend was not possible to discover during these tests.

6.2 Answer to Research Question 2

RQ2: What are the differences in level of inference on different air quality monitors from different vendors?

The baseline capturings shows that the devices communicates differently. For Netatmo and Nedis, most of the packets are outbound bytes meaning that these devices send a lot more traffic than they receive. For Mill, inbound traffic was higher than outbound traffic. Also comparing the traffic from the three devices, as shown in Figure 5.10, shows that the traffic is different from the three devices. Netatmo has a more continuous line of the packets, while Mill sends traffic peri-

odically and have more spikes and Nedis shows many spikes that are a lot higher than for the rest of the baseline period. These differences can be used to distinguish the traffic between the three air quality monitor from different vendors.

The differences in level of inference from the three different air quality monitors are only distinguishable when it comes to Test Case 4: Weekends. For the routine behavioural test cases, neither of the devices expose private information regarding when the event is ongoing or what kind of event is triggered. However, the results from the weekend test shows that Nedis and Mill is a more secure choice than Netatmo when it comes to selecting air quality monitor to install in our home. Also when it comes to identifying changes in traffic, Nedis and Mill shows that the baseline traffic can change on other factors than a triggered event and it will therefore be harder for an attacker to understand normal traffic. It is good for users to know that one should consider more factors than just the appearance or functionality of the air quality monitor when selecting what device to install in their home. Knowing that the network traffic and level of inference of an air quality monitor is not the same regardless of the manufacture, is important when selecting the right one.

6.3 Answer to Research Question 3

RQ3: How can the private information gathered be misused by an adversary?

Since the weekend test for Netatmo exposed private information, this specific test case can be used to evaluate how this information can be misused by an adversary. If an adversary has conducted a passive network eavesdropping attack against a user who has installed the Netatmo Smart Indoor Air Quality Monitor, it is possible to know if a user is gone for a longer period of time. Even though weekends were used in this test, the results will also be applicable for longer periods of time such as a week or several weeks. An adversary could use this private information to carry out malicious actions against the users home such as a burglary. Another aspect is that an attacker could learn about our habits for being home or not, such as every Easter or Christmas, the user is gone or every weekend during winter the user is not home. An attacker could also gather this information to sell to other interested parties, spreading the private information more without the user having control of it.

However, it is interesting to consider to what extent private information from the other test cases can be misused if they were identified. Both cooking and showering are events when the user is awake and doing an active action in the home environment. If an adversary were to find out the routines of a user for these two cases, it may not be able to misuse it to the extent as the results from the weekend test can. Knowing that an unwanted party knows that every day at 7am you are taking a shower, may not feel that scary and invading, but if this is combined with attacks against other devices to find out your whole daily routine, it will be a bigger invasion of privacy. Window open at night can indicate a users sleeping

patterns and therefore identifies a time period where the user is not awake in its home environment. This can feel like a more invasion of privacy as the user does not have as much control over the situation as when cooking or showering. On the other side, being able to know if a user is gone for a whole weekend is easier to misuse than just sleeping as the user is present at home.

6.4 Valuable Contributions and Challenges Faced

When starting this research, a decision to include several different air quality monitors from different manufactures were made. Looking at the results and evaluation, this was positive for the contribution as it shows that there are differences in level of private information inference from the different devices. This shows that analyzing the network traffic from one air quality monitor is not representative for all air quality monitors and is something to consider if setting up a test environment where only one air quality monitor is included.

When defining the scope and test cases, it was unclear how the devices communicated including packet sizes, amount of packets, how frequent and if traffic was encrypted or not. When starting the capturing and analyzing of data from the different devices, it became clear that the devices do not send the same amount of traffic. Netatmo Smart Indoor Air Quality Monitor sends the least amount of packets, while Mill Sense and Nedis SmartLife sends significantly more packets. For Nedis SmartLife, analyzing the amount of traffic was a challenge, both timewise as creating graphs took a long time, but also looking more into the traffic. The processing power of a standard computer were just enough to process the data and figures shown in the evaluation and analysis result chapter.

A challenge faced during the research happened for the test cases for cooking. The events were designed to change the indoor environment so much that the air quality monitors would sense values outside of the defined threshold values. However, when doing the cooking test, only a few of the executions actually triggered the notifications to be send. While for showering and window open, every execution lead to a significant change in threshold values and sent several notifications to the connected phone.

Another challenge encountered during the test, was the differences in the baseline capturings compared to the event traffic for Nedis SmartLife and Mill Sense, before the event was triggered, as we would have expected these two traffic patterns to be similar. However, the decision were made to include the baseline capturings and compare them to the event traffic because it also shows methodologically the setup that was originally designed, and it is still possible to compare traffic from when the event was ongoing to traffic before and after the event. It is unclear why there are differences in standard traffic from the devices, but differences in season as the test cases have been conducted during January and the baseline during March can be one reason. Another reason can be that the baseline was not captured at the exact same spot as were the test cases were carried out. This was because of time constraints. It was not enough time to have a

long baseline in each room and conduct the test cases during the research period even though this would have been preferred. Since the air quality monitors are IoT sensor that are always on and sense the environment all the time, differences in day to day or month to month can be significant. The indoor environment can be affected by many different factors and it is impossible to have the exact same values for the indoor air during longer period of times.

The differences in standard traffic shows that in order to have the same traffic pattern to test with, the best way would be to have the air quality monitors in a closed environment. Then the environment could only be changed by the specific event which are tested. However, this takes away the realistic parts of the test. The chance of identifying the specific events may be bigger, but if it is not applicable in a live environment, there is no use to launch such an attack against a target. Another aspect of this is that if changes in the environment is significant within the same environment then it will also be hard for an attacker to find certain signatures in their own environment to be applicable on a target environment and device.

Chapter 7

Conclusions and Future Work

This thesis has investigated three different air quality monitors to see what kind of private information can be inferred while carrying out a passive network eavesdropping attack. Three research questions have been raised and answered through the tests in this thesis. The three devices selected were Netatmo Smart Indoor Air Quality Monitor, Mill Sense and Nedis SmartLife. To test which kind of private information can be inferred from the devices, the devices were installed in a live environment and four different test cases were carried out. Three test cases which targets routine behaviour: cooking, showering and window open during night, and one test case over a longer period of time: home or gone during a weekend.

The initial baseline capturings for each device showed that the devices communicates quite differently in number of packets and bytes sent and received and packet sizes. Scripts for generating graphs of the traffic flows in a presentable way were created and used to present the results. This visual representation of the traffic flow was used together with calculations to give the results. The three routine test were compared with corresponding times for baseline capturings. For Mill Sense and Nedis SmartLife, the baseline traffic differs from the same standard traffic that is captured before an event is triggered. This shows how the network patterns of these devices can be affected by factors such as season or location. For Netatmo Smart Indoor Air Quality Monitor, the baseline traffic is comparable to the event traffic.

The routine tests were carried out over different days to have enough capturings to look for a specific traffic pattern. During these tests, only showering and window open during night triggered the sensor thresholds as expected and sent notifications to the application. For cooking, not every execution did this. When looking at the traffic patterns for these test cases and devices, there are no significant change in traffic pattern from before an event is triggered to when the event is ongoing. This applies to the three test cases cooking, showering and window open for the three devices Netatmo Smart Indoor Air Quality Monitor, Mill Sense and Nedis SmartLife.

The weekend tests were carried out over different weekends at home and gone. For the weekend test, Mill Sense and Nedis SmartLife did not expose dif-

ferences in traffic patterns from when a user was home or gone. However, for Netatmo Smart Indoor Air Quality Monitor the results showed a clear difference. When a user is home, the device sends significantly more bytes and bigger packets compared to when the user is gone. This information can be misused by an adversary to know if a user is home during a weekend or a holiday and perform malicious actions such as burglary.

Through carrying out four test cases on three different air quality monitors, the research questions have been answered. Only one of the devices exposed private information in one of the tests and shows that using this method, the majority of private information tested are not revealed. However, Netatmo reveals private information whether a users is home or not for longer periods of time by looking at the traffic patterns on Wi-Fi traffic. This research shows that different air quality monitors communicate with different network patterns on Wi-Fi. It is therefore important to understand that the choice of air quality monitor can impact how much private information it is possible to infer. This is applicable both in research cases and when choosing an air quality monitor to install in a home environment. The scripts created to generate graphs of traffic flows and methods used can be used to further test other air quality monitors also communicating on other communication protocols.

Future Work

This research is limited to investigating air quality monitors which communicates over Wi-Fi, therefore future researches could look into differences between communication protocols and include air quality monitors that communicates over protocols such as Bluetooth, ZigBee or Z-wave. Since Netatmo Smart Indoor Air Quality Monitor reveals private information during the weekend-testing, building further on these findings could be to look into timings for when the traffic pattern changes. This research only showed traffic pattern changes when the user was gone for a weekend or longer, but testing number of hours it takes before the changes occur, could increase detail in exposed information. Another aspect is testing the devices in an environment with other IoT devices to see if there are differences in test cases or if it reveals more or less private information than other IoT devices. This thesis does not look into security measurements, but as one of the tests discovers private information on the target environment, future work should look into how implement security measurements to this case.

Bibliography

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari and M. Ayyash, 'Internet of things: A survey on enabling technologies, protocols, and applications,' *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015. DOI: 10.1109/COMST.2015.2444095.

[2] A. W. Burange and H. D. Misalkar, 'Review of internet of things in development of smart cities with data management & privacy,' in *2015 International Conference on Advances in Computer Engineering and Applications*, 2015, pp. 189–195. DOI: 10.1109/ICACEA.2015.7164693.

[3] J. Jose and T. Sasipraba, 'Indoor air quality monitors using iot sensors and lpwan,' in *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, 2019, pp. 633–637. DOI: 10.1109/ICOEI.2019.8862647.

[4] J. Saini, M. Dutta and G. Marques, 'Indoor air quality monitoring systems based on internet of things: A systematic review,' *International Journal of Environmental Research and Public Health*, vol. 17, no. 14, 2020. DOI: 10.3390/ijerph17144942. [Online]. Available: https://www.mdpi.com/1660-4601/17/14/4942.

[5] J. M. Seguel, R. Merrill, D. Seguel and A. C. Campagna, 'Indoor air quality,' *American journal of lifestyle medicine*, vol. 11, no. 4, pp. 284–295, 2017.

[6] L. Luo, Y. Zhang, B. Pearson, Z. Ling, H. Yu and X. Fu, 'On the security and data integrity of low-cost sensor networks for air quality monitoring,' *Sensors*, vol. 18, no. 12, 2018, ISSN: 1424-8220. DOI: 10.3390/s18124451. [Online]. Available: https://www.mdpi.com/1424-8220/18/12/4451.

[7] A. Aksoy and M. H. Gunes, 'Automated iot device identification using network traffic,' in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–7. DOI: 10.1109/ICC.2019.8761559.

[8] J. Kotak and Y. Elovici, 'Iot device identification using deep learning,' in *13th International Conference on Computational Intelligence in Security for Information Systems (CISIS 2020)*, Á. Herrero, D. Cambra Carlosand Urda, J. Sedano, H. Quintián and E. Corchado, Eds., Cham: Springer International Publishing, 2021, pp. 76–86, ISBN: 978-3-030-57805-3.

[9] H. Mokrani, R. Lounas, M. T. Bennai, D. E. Salhi and R. Djerbi, 'Air quality monitoring using iot: A survey,' in *2019 IEEE International Conference on Smart Internet of Things (SmartIoT)*, 2019, pp. 127–134. DOI: 10.1109/SmartIoT.2019.00028.

[10] J. Saini, M. Dutta and G. Marques, *Internet of things for indoor air quality monitoring*. Springer, 2021.

[11] B. P. Haakon Haraldsen and T. Nordseth. 'Karbondioksid.' (6.040.2022), [Online]. Available: https://snl.no/karbondioksid. (accessed: 23.10.2022).

[12] U. S. D. of Labor. 'Occupational noise exposure.' (), [Online]. Available: https://www.osha.gov/noise/health-effects. (accessed: 23.10.2022).

[13] E. United States Environmental Protection Agency. 'Technical overview of volatile organic compounds.' (), [Online]. Available: https://www.epa.gov/indoor-air-quality-iaq/technical-overview-volatile-organic-compounds. (accessed: 23.10.2022).

[14] Folkehelseinsituttet, 'Anbefalte faglige normer for inneklima,' *Rapport 2015:1*, 2015, ISSN: 1503-1403. [Online]. Available: https://www.fhi.no/globalassets/dokumenterfiler/rapporter/2015/anbefalte-faglige-normer-for-inneklima-pdf.pdf, (accessed: 23.10.2022).

[15] E. United States Environmental Protection Agency. 'What is formaldehyde?' (2022), [Online]. Available: https://www.epa.gov/formaldehyde/facts-about-formaldehyde#whatisformaldehyde. (accessed: 23.10.2022).

[16] F. Folkehelseinsituttet. 'Temperature, fukt og trekk er viktig for kroppens varmebalanse.' (2012), [Online]. Available: https://www.fhi.no/ml/miljo/inneklima/artikler-inneklima-og-helseplager/temperatur-fukt-og-trekk-er-viktig-/. (accessed: 23.10.2022).

[17] L. Zhao, Y. Yang and Z. Wu, 'Review of communication technology in indoor air quality monitoring system and challenges,' *Electronics*, vol. 11, no. 18, 2022, ISSN: 2079-9292. DOI: 10.3390/electronics11182926. [Online]. Available: https://www.mdpi.com/2079-9292/11/18/2926.

[18] Airthings. 'About us.' (2022), [Online]. Available: https://www.airthings.com/about-us. (accessed:15.11.2022).

[19] Netatmo. 'Netatmo.' (), [Online]. Available: https://www.netatmo.com/en-us. (accessed:26.12.2022).

[20] Komplett. 'Om oss.' (2022), [Online]. Available: https://millnorway.no/om-oss/. (accessed:15.11.2022).

[21] Philips. 'Philips air quality solutions.' (), [Online]. Available: https://www.philips.com.au/c-m-ho/air-purifier-and-air-humidifier. (accessed:26.12.2022).

[22] Dyson. 'Air treatment, air purifier, heaters, fans, humidifiers, purifier filters.' (), [Online]. Available: https://www.dyson.com/air-treatment. (accessed:26.12.2022).

[23] L. Morawska, P. K. Thai, X. Liu, A. Asumadu-Sakyi, G. Ayoko, A. Bartonova, A. Bedini, F. Chai, B. Christensen, M. Dunbabin, J. Gao, G. S. Hagler, R. Jayaratne, P. Kumar, A. K. Lau, P. K. Louie, M. Mazaheri, Z. Ning, N. Motta, B. Mullins, M. M. Rahman, Z. Ristovski, M. Shafiei, D. Tjondronegoro, D. Westerdahl and R. Williams, 'Applications of low-cost sensing technologies for air quality monitoring and exposure assessment: How far have they gone?' *Environment International*, vol. 116, pp. 286–299, 2018, ISSN: 0160-4120. DOI: https://doi.org/10.1016/j.envint.2018.04.018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0160412018302460.

[24] K. Zhang, J. Ni, K. Yang, X. Liang, J. Ren and X. S. Shen, 'Security and privacy in smart city applications: Challenges and solutions,' *IEEE Communications Magazine*, vol. 55, no. 1, pp. 122–129, 2017. DOI: 10.1109/MCOM.2017.1600267CM.

[25] H. F. Atlam and G. B. Wills, 'Iot security, privacy, safety and ethics,' in *Digital twin technologies and smart cities*, Springer, 2020, pp. 123–149.

[26] H. Ning, H. Liu and L. T. Yang, 'Cyberentity security in the internet of things,' *Computer*, vol. 46, no. 4, pp. 46–53, 2013. DOI: 10.1109/MC.2013.74.

[27] H. Lin and N. W. Bergmann, 'Iot privacy and security challenges for smart home environments,' *Information*, vol. 7, no. 3, p. 44, 2016.

[28] M. Seliem, K. Elgazzar and K. Khalil, 'Towards privacy preserving iot environments: A survey,' *Wireless Communications and Mobile Computing*, vol. 2018, 2018.

[29] M. Sherr, 'Eavesdropping,' in *Encyclopedia of Cryptography and Security*, H. C. A. van Tilborg and S. Jajodia, Eds. Boston, MA: Springer US, 2011, pp. 378–379, ISBN: 978-1-4419-5906-5. DOI: 10.1007/978-1-4419-5906-5_109. [Online]. Available: https://doi.org/10.1007/978-1-4419-5906-5_109.

[30] M. Alyami, I. Alharbi, C. Zou, Y. Solihin and K. Ackerman, 'Wifi-based iot devices profiling attack based on eavesdropping of encrypted wifi traffic,' in *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, IEEE, 2022, pp. 385–392.

[31] R. Singh and S. Kumar, 'A comparative study of various wireless network monitoring tools,' in *2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC)*, IEEE, 2018, pp. 379–384.

[32] Wireshark. 'Wireshark.org.' (), [Online]. Available: https://www.wireshark.org/. (accessed:27.12.2022).

[33] J. Saini, M. Dutta and G. Marques, 'Indoor air quality monitoring systems based on internet of things: A systematic review,' *International journal of environmental research and public health*, vol. 17, no. 14, p. 4942, 2020.

[34] W.-F. Alliance. 'Who we are.' (), [Online]. Available: wi-fi.org/who-we-are. (accessed: 19.10.2022).

[35] 'Ieee standard for information technology—telecommunications and information exchange between systems local and metropolitan area networks—specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications,' *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, pp. 1–3534, 2016. DOI: 10.1109/IEEESTD.2016.7786995.

[36] J. F. Kurose and K. W. Ross, 'Computer networking: A top down approach,' in M. Hirsch, Ed. Boston, MA: Pearson, 2012, ISBN: 978-0-13-285620-1.

[37] P. Ebbecke. 'Protected management frames enhance wi-fi® network security.' (25 February 2020), [Online]. Available: https://www.wi-fi.org/beacon/philipp-ebbecke/protected-management-frames-enhance-wi-fi-network-security. (accessed:28.12.2022).

[38] D. Akin and J. Geier, 'Certified wireless analysis professional™ official study guide,' in G. Hancock, Ed. Emeryville, CA: Planet3 Wireless, 2004, ISBN: 0-07-225585-4.

[39] B. I. Reddy and V. Srikanth, 'Review on wireless security protocols (wep, wpa, wpa2 & wpa3),' *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, pp. 28–35, 2019.

[40] V. Sivaraman, H. H. Gharakheili, C. Fernandes, N. Clark and T. Karliychuk, 'Smart iot devices in the home: Security and privacy implications,' *IEEE Technology and Society Magazine*, vol. 37, no. 2, pp. 71–79, 2018. DOI: 10.1109/MTS.2018.2826079.

[41] L. M. Zagi and B. Aziz, 'Privacy attack on iot: A systematic literature review,' in *2020 International Conference on ICT for Smart Society (ICISS)*, 2020, pp. 1–8. DOI: 10.1109/ICISS50791.2020.9307568.

[42] M. Mrissa, A. Tošić, N. Hrovatin, S. Aslam, B. Dávid, L. Hajdu, M. Krész, A. Brodnik and B. Kavšek, 'Privacy-aware and secure decentralized air quality monitoring,' *Applied Sciences*, vol. 12, no. 4, 2022, ISSN: 2076-3417. DOI: 10.3390/app12042147. [Online]. Available: https://www.mdpi.com/2076-3417/12/4/2147.

[43] P.-M. Junges, J. François and O. Festor, 'Passive inference of user actions through iot gateway encrypted traffic analysis,' in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, IEEE, 2019, pp. 7–12.

[44] J. H. Ziegeldorf, O. G. Morchon and K. Wehrle, 'Privacy in the internet of things: Threats and challenges,' *Security and Communication Networks*, vol. 7, no. 12, pp. 2728–2742, 2014.

[45] N. Apthorpe, D. Reisman and N. Feamster, 'A smart home is no castle: Privacy vulnerabilities of encrypted iot traffic,' *arXiv preprint arXiv:1705.06805*, 2017.

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

[47] N. Apthorpe, D. Reisman and N. Feamster, 'Closing the blinds: Four strategies for protecting smart home privacy from network observers,' *arXiv preprint arXiv:1705.06809*, 2017.

[48] A. Acar, H. Fereidooni, T. Abera, A. K. Sikder, M. Miettinen, H. Aksu, M. Conti, A.-R. Sadeghi and S. Uluagac, 'Peek-a-boo: I see your smart home activities, even encrypted!' In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2020, pp. 207–218.

[49] E. Papadogiannaki and S. Ioannidis, 'A survey on encrypted network traffic analysis applications, techniques, and countermeasures,' *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–35, 2021.

[50] Tibber. 'Tibber.' (2022), [Online]. Available: `https://tibber.com/en`. (accessed:14.11.2022).

[51] H. M. Jordheim. 'Tibber nidoblet inntektene: Serverne knelte etter kundestrøm.' (24.02.2022), [Online]. Available: `https://e24.no/naeringsliv/i/ML2JjE/tibber-nidoblet-inntektene-serverne-knelte-etter-kundestroem`. (accessed:20.11.2022).

[52] Elkjøp. 'Luftkvalitetsmålere.' (2022), [Online]. Available: `https://www.elkjop.no/search/luftkvalitetsm%C3%A5ler`. (accessed:15.11.2022).

[53] Komplett. 'Luftkvalitetsmålere.' (2022), [Online]. Available: `https://www.komplett.no/search?q=luftkvalitetsm%C3%A5lere`. (accessed:15.11.2022).

[54] Nedis. 'Our story.' (2022), [Online]. Available: `https://nedis.com/en-us/business/about-nedis`. (accessed:20.11.2022).

[55] Netatmo. 'Smart indoor air quality monitor.' (), [Online]. Available: `https://www.netatmo.com/en-us/aircare/homecoach`. (accessed:29.12.2022).

[56] M. I. AS. 'Mill sense - smart indoor climate sensor.' (), [Online]. Available: `https://millnorway.com/product/mill-sense/`. (accessed:29.12.2022).

[57] Nedis. 'Smartlife air quality monitor.' (), [Online]. Available: `https://nedis.com/en-us/product/home-living/climate/thermometers/550768043/smartlife-air-quality-monitor-wi-fi-humidity-temperature-volatile-organic-compounds-voc-android-ios-white`. (accessed:29.12.2022).

[58] TP-Link. 'Tp-link tl-wn722n.' (), [Online]. Available: `https://www.tp-link.com/us/home-networking/usb-adapter/tl-wn722n/`. (accessed:29.12.2022).

[59] Kali. 'Get kali | kali linux.' (), [Online]. Available: `https://www.kali.org/get-kali/#kali-platforms`. (accessed:01.05.2023).

[60] WinSCP.net. 'Winscp free award-winning file manager.' (), [Online]. Available: `https://winscp.net/eng/index.php`. (accessed:01.05.2023).

[61] W. Org. 'Tshark(1).' (), [Online]. Available: https://www.wireshark.org/docs/man-pages/tshark.html. (accessed:01.05.2023).

[62] J. McManus. 'Visualizing network data using python: Part 3.' (14. jun 2018), [Online]. Available: https://www.automox.com/blog/visualizing-network-data-using-python-part-3. (accessed:01.02.2023).

Appendices

A Script to Generate Graphs for Baseline Comparison with Packets as Reference [62]

```
#GraphsByPackets_BaselineEvents.py
from scapy.all import *
import plotly
from datetime import datetime
import pandas as pd
import pyshark
from pyshark.packet import consts
from pyshark.packet.common import Pickleable
import plotly.graph_objects as go
import sys
import numpy as np

#sys.argv[1] = Name of device
#sys.argv[2] = Choose between inbound, outbound or total packets
#sys.argv[3] = Type of event
#sys.argv[4] = Maximum value for y-axis

display=""

def graph_function():
    times=[]
    z=0
    for date in dates:
        if sys.argv[2] == "Total":
            file = r"C:\Users\Helene\Documents\IMT4905 - Erfaringsbasert master\
                Wireshark\Baseline\\"+sys.argv[1]+"\\"+sys.argv[3]+"\\"+sys.argv
                [1]+"_Baseline_"+sys.argv[3]+"_"+date+".pcapng"
            packets = pyshark.FileCapture(file)
        else:
            file = r"C:\Users\Helene\Documents\IMT4905 - Erfaringsbasert master\
                Wireshark\Baseline\\"+sys.argv[1]+"\\"+sys.argv[3]+"\\"+sys.argv
                [1]+"_Baseline_"+sys.argv[3]+"_"+date+".pcapng"
            packets = pyshark.FileCapture(file, display_filter=display)


        #Lists to hold packet info
        pktTimes=[]
        pkts=[]
        #Read each packet and append to the lists.
        for pkt in packets:
            n=1
            pktTime=(pkt.sniff_time)
            pktTimes.append(pktTime)
            pkts.append(n)

        #This converts list to series
```

```
        packets = pd.Series(pkts).astype(int)

        #Convert the timestamp list to a pd date_time
        times = pd.to_datetime(pd.Series(pktTimes).astype(str),  errors='coerce')

        #Create the dataframe
        df  = pd.DataFrame({"Packets": packets, "Times": times})

        #set the date from a range to an timestamp
        df = df.set_index('Times')

        #Create a new dataframe of 2 second sums to pass to plotly
        df2=df.resample('2S').sum()

        #Create the graph
        GraphTitle=sys.argv[1]+"\n"+sys.argv[3]+"\n"+date
        fig = go.Figure({"data":[plotly.graph_objs.Scatter(x=df2.index, y=df2['
            Packets'])],"layout":plotly.graph_objs.Layout(title=GraphTitle,
                xaxis=dict(title="Time"),
                yaxis=dict(title=sys.argv[2]+" Packets"))})

        #Set the y-axis range
        fig.update_yaxes(range=[0,sys.argv[4]])

        #Set the x-axis range
        fig.update_layout(xaxis_range=[packetstart[z],packetend[z]])

        #Set the font
        fig.update_layout(title=GraphTitle, xaxis_title="Time", yaxis_title="Total
            Packets",font=dict(family="Times New Roman", size=26))

        #Display the graphs
        fig.show()

        z=z+1

if sys.argv[2] == "Outbound":
    if sys.argv[1] == "Netatmo":
        display = "wlan.sa == 70:EE:50:91:06:DE"
    elif sys.argv[1] == "Mill":
        display = "wlan.sa == B8:F0:09:B3:B3:78"
    elif sys.argv[1] == "Nedis":
        display = "wlan.sa == 2C:F4:32:29:36:DC"

elif sys.argv[2] == "Inbound":
    if sys.argv[1] == "Netatmo":
        display = "wlan.da == 70:EE:50:91:06:DE"
    elif sys.argv[1] == "Mill":
        display = "wlan.da == B8:F0:09:B3:B3:78"
    elif sys.argv[1] == "Nedis":
```

```
        display = "wlan.da == 2C:F4:32:29:36:DC"

if sys.argv[3] == "Shower":
        #Set the shower dates
        dates = ["08.01","09.01","11.01","16.01","18.01","19.01","25.01","30.01","
            31.01","01.02"]

        #Set the x-axis range even tough packets are not sent
        packetstart=["2023-01-08 19:29","2023-01-09 19:29","2023-01-11 19:29","
            2023-01-16 19:29","2023-01-18 19:29","2023-01-19 19:29","2023-01-25
            19:29","2023-01-30 19:29","2023-01-31 19:29","2023-02-01 19:29"]
        packetend=["2023-01-08 21:04","2023-01-09 21:04","2023-01-11 21:04","
            2023-01-16 21:04","2023-01-18 21:04","2023-01-19 21:04","2023-01-25
            21:04","2023-01-30 21:04","2023-01-31 21:04","2023-02-01 21:04"]

        graph_function()

elif sys.argv[3] == "Cooking":
        #Set the cooking dates
        dates = ["08.01","09.01","11.01","16.01","18.01","19.01","25.01","30.01","
            31.01","01.02"]

        #Set the x-axis range even tough packets are not sent
        packetstart=["2023-01-08 15:28","2023-01-09 15:28","2023-01-11 15:28","
            2023-01-16 15:28","2023-01-18 15:28","2023-01-19 15:28","2023-01-26
            15:28","2023-01-30 15:28","2023-01-31 15:28","2023-02-01 15:28"]
        packetend=["2023-01-08 17:07","2023-01-09 17:07","2023-01-11 17:07","
            2023-01-16 17:07","2023-01-18 17:07","2023-01-19 17:07","2023-01-26
            17:07","2023-01-30 17:07","2023-01-31 17:07","2023-02-01 17:07"]

        graph_function()

elif sys.argv[3] == "Window":
        #Set the window dates
        dates = ["08.01-09.01","09.01-10.01","11.01-12.01","16.01-17.01","
            18.01-19.01","19.01-20.01","25.01-26.01","30.01-31.01","31.01-01.02","
            01.02-02-02"]

        #Set the x-axis range even tough packets are not sent
        packetstart=["2023-01-08 22:20","2023-01-09 22:20","2023-01-11 22:20","
            2023-01-16 22:20","2023-01-18 22:20","2023-01-19 22:20","2023-01-25
            22:20","2023-01-30 22:20","2023-01-31 22:20","2023-02-01 22:20"]
        packetend=["2023-01-09 07:39","2023-01-10 07:39","2023-01-12 07:39","
            2023-01-17 07:39","2023-01-19 07:39","2023-01-20 07:39","2023-01-26
            07:39","2023-01-31 07:39","2023-02-01 07:39","2023-02-02 07:39"]

        graph_function()
```

B Script to Generate Graphs for Baseline Comparison with Packets as Reference [62]

```
#GraphsByPackets_BaselineEvents.py
from scapy.all import *
import plotly
from datetime import datetime
import pandas as pd
import pyshark
from pyshark.packet import consts
from pyshark.packet.common import Pickleable
import plotly.graph_objects as go
import sys
import numpy as np

#sys.argv[1] = Name of device
#sys.argv[2] = Choose between inbound, outbound or total packets
#sys.argv[3] = Type of event
#sys.argv[4] = Maximum value for y-axis

display=""

def graph_function():
    times=[]
    z=0
    for date in dates:
        if sys.argv[2] == "Total":
            file = r"C:\Users\Helene\Documents\IMT4905 - Erfaringsbasert master\
                Wireshark\Baseline\\"+sys.argv[1]+"\\"+sys.argv[3]+"\\"+sys.argv
                [1]+"_Baseline_"+sys.argv[3]+"_"+date+".pcapng"
            packets = pyshark.FileCapture(file)
        else:
            file = r"C:\Users\Helene\Documents\IMT4905 - Erfaringsbasert master\
                Wireshark\Baseline\\"+sys.argv[1]+"\\"+sys.argv[3]+"\\"+sys.argv
                [1]+"_Baseline_"+sys.argv[3]+"_"+date+".pcapng"
            packets = pyshark.FileCapture(file, display_filter=display)


        #Lists to hold packet info
        pktTimes=[]
        pkts=[]
        #Read each packet and append to the lists.
        for pkt in packets:
            n=1
            pktTime=(pkt.sniff_time)
            pktTimes.append(pktTime)
            pkts.append(n)

        #This converts list to series
```

```
        packets = pd.Series(pkts).astype(int)

        #Convert the timestamp list to a pd date_time
        times = pd.to_datetime(pd.Series(pktTimes).astype(str),  errors='coerce')

        #Create the dataframe
        df  = pd.DataFrame({"Packets": packets, "Times": times})

        #set the date from a range to an timestamp
        df = df.set_index('Times')

        #Create a new dataframe of 2 second sums to pass to plotly
        df2=df.resample('2S').sum()

        #Create the graph
        GraphTitle=sys.argv[1]+"\n"+sys.argv[3]+"\n"+date
        fig = go.Figure({"data":[plotly.graph_objs.Scatter(x=df2.index, y=df2['
            Packets'])],"layout":plotly.graph_objs.Layout(title=GraphTitle,
                xaxis=dict(title="Time"),
                yaxis=dict(title=sys.argv[2]+" Packets"))})

        #Set the y-axis range
        fig.update_yaxes(range=[0,sys.argv[4]])

        #Set the x-axis range
        fig.update_layout(xaxis_range=[packetstart[z],packetend[z]])

        #Set the font
        fig.update_layout(title=GraphTitle, xaxis_title="Time", yaxis_title="Total
            Packets",font=dict(family="Times New Roman", size=26))

        #Display the graphs
        fig.show()

        z=z+1

if sys.argv[2] == "Outbound":
    if sys.argv[1] == "Netatmo":
        display = "wlan.sa == 70:EE:50:91:06:DE"
    elif sys.argv[1] == "Mill":
        display = "wlan.sa == B8:F0:09:B3:B3:78"
    elif sys.argv[1] == "Nedis":
        display = "wlan.sa == 2C:F4:32:29:36:DC"

elif sys.argv[2] == "Inbound":
    if sys.argv[1] == "Netatmo":
        display = "wlan.da == 70:EE:50:91:06:DE"
    elif sys.argv[1] == "Mill":
        display = "wlan.da == B8:F0:09:B3:B3:78"
    elif sys.argv[1] == "Nedis":
```

```
        display = "wlan.da == 2C:F4:32:29:36:DC"

if sys.argv[3] == "Shower":
        #Set the shower dates
        dates = ["08.01","09.01","11.01","16.01","18.01","19.01","25.01","30.01","
            31.01","01.02"]

        #Set the x-axis range even tough packets are not sent
        packetstart=["2023-01-08 19:29","2023-01-09 19:29","2023-01-11 19:29","
            2023-01-16 19:29","2023-01-18 19:29","2023-01-19 19:29","2023-01-25
            19:29","2023-01-30 19:29","2023-01-31 19:29","2023-02-01 19:29"]
        packetend=["2023-01-08 21:04","2023-01-09 21:04","2023-01-11 21:04","
            2023-01-16 21:04","2023-01-18 21:04","2023-01-19 21:04","2023-01-25
            21:04","2023-01-30 21:04","2023-01-31 21:04","2023-02-01 21:04"]

        graph_function()

elif sys.argv[3] == "Cooking":
        #Set the cooking dates
        dates = ["08.01","09.01","11.01","16.01","18.01","19.01","25.01","30.01","
            31.01","01.02"]

        #Set the x-axis range even tough packets are not sent
        packetstart=["2023-01-08 15:28","2023-01-09 15:28","2023-01-11 15:28","
            2023-01-16 15:28","2023-01-18 15:28","2023-01-19 15:28","2023-01-26
            15:28","2023-01-30 15:28","2023-01-31 15:28","2023-02-01 15:28"]
        packetend=["2023-01-08 17:07","2023-01-09 17:07","2023-01-11 17:07","
            2023-01-16 17:07","2023-01-18 17:07","2023-01-19 17:07","2023-01-26
            17:07","2023-01-30 17:07","2023-01-31 17:07","2023-02-01 17:07"]

        graph_function()

elif sys.argv[3] == "Window":
        #Set the window dates
        dates = ["08.01-09.01","09.01-10.01","11.01-12.01","16.01-17.01","
            18.01-19.01","19.01-20.01","25.01-26.01","30.01-31.01","31.01-01.02","
            01.02-02-02"]

        #Set the x-axis range even tough packets are not sent
        packetstart=["2023-01-08 22:20","2023-01-09 22:20","2023-01-11 22:20","
            2023-01-16 22:20","2023-01-18 22:20","2023-01-19 22:20","2023-01-25
            22:20","2023-01-30 22:20","2023-01-31 22:20","2023-02-01 22:20"]
        packetend=["2023-01-09 07:39","2023-01-10 07:39","2023-01-12 07:39","
            2023-01-17 07:39","2023-01-19 07:39","2023-01-20 07:39","2023-01-26
            07:39","2023-01-31 07:39","2023-02-01 07:39","2023-02-02 07:39"]

        graph_function()
```

C Script to Generate Graphs for Baseline Comparison with Packets as Reference [62]

```
#GraphsByPackets_BaselineEvents.py
from scapy.all import *
import plotly
from datetime import datetime
import pandas as pd
import pyshark
from pyshark.packet import consts
from pyshark.packet.common import Pickleable
import plotly.graph_objects as go
import sys
import numpy as np

#sys.argv[1] = Name of device
#sys.argv[2] = Choose between inbound, outbound or total packets
#sys.argv[3] = Type of event
#sys.argv[4] = Maximum value for y-axis

display=""

def graph_function():
    times=[]
    z=0
    for date in dates:
        if sys.argv[2] == "Total":
            file = r"C:\Users\Helene\Documents\IMT4905 - Erfaringsbasert master\
                Wireshark\Baseline\\"+sys.argv[1]+"\\"+sys.argv[3]+"\\"+sys.argv
                [1]+"_Baseline_"+sys.argv[3]+"_"+date+".pcapng"
            packets = pyshark.FileCapture(file)
        else:
            file = r"C:\Users\Helene\Documents\IMT4905 - Erfaringsbasert master\
                Wireshark\Baseline\\"+sys.argv[1]+"\\"+sys.argv[3]+"\\"+sys.argv
                [1]+"_Baseline_"+sys.argv[3]+"_"+date+".pcapng"
            packets = pyshark.FileCapture(file, display_filter=display)


        #Lists to hold packet info
        pktTimes=[]
        pkts=[]
        #Read each packet and append to the lists.
        for pkt in packets:
            n=1
            pktTime=(pkt.sniff_time)
            pktTimes.append(pktTime)
            pkts.append(n)

        #This converts list to series
```

```
        packets = pd.Series(pkts).astype(int)

        #Convert the timestamp list to a pd date_time
        times = pd.to_datetime(pd.Series(pktTimes).astype(str),  errors='coerce')

        #Create the dataframe
        df  = pd.DataFrame({"Packets": packets, "Times": times})

        #set the date from a range to an timestamp
        df = df.set_index('Times')

        #Create a new dataframe of 2 second sums to pass to plotly
        df2=df.resample('2S').sum()

        #Create the graph
        GraphTitle=sys.argv[1]+"\n"+sys.argv[3]+"\n"+date
        fig = go.Figure({"data":[plotly.graph_objs.Scatter(x=df2.index, y=df2['
            Packets'])],"layout":plotly.graph_objs.Layout(title=GraphTitle,
                xaxis=dict(title="Time"),
                yaxis=dict(title=sys.argv[2]+" Packets"))})

        #Set the y-axis range
        fig.update_yaxes(range=[0,sys.argv[4]])

        #Set the x-axis range
        fig.update_layout(xaxis_range=[packetstart[z],packetend[z]])

        #Set the font
        fig.update_layout(title=GraphTitle, xaxis_title="Time", yaxis_title="Total
            Packets",font=dict(family="Times New Roman", size=26))

        #Display the graphs
        fig.show()

        z=z+1

if sys.argv[2] == "Outbound":
    if sys.argv[1] == "Netatmo":
        display = "wlan.sa == 70:EE:50:91:06:DE"
    elif sys.argv[1] == "Mill":
        display = "wlan.sa == B8:F0:09:B3:B3:78"
    elif sys.argv[1] == "Nedis":
        display = "wlan.sa == 2C:F4:32:29:36:DC"

elif sys.argv[2] == "Inbound":
    if sys.argv[1] == "Netatmo":
        display = "wlan.da == 70:EE:50:91:06:DE"
    elif sys.argv[1] == "Mill":
        display = "wlan.da == B8:F0:09:B3:B3:78"
    elif sys.argv[1] == "Nedis":
```

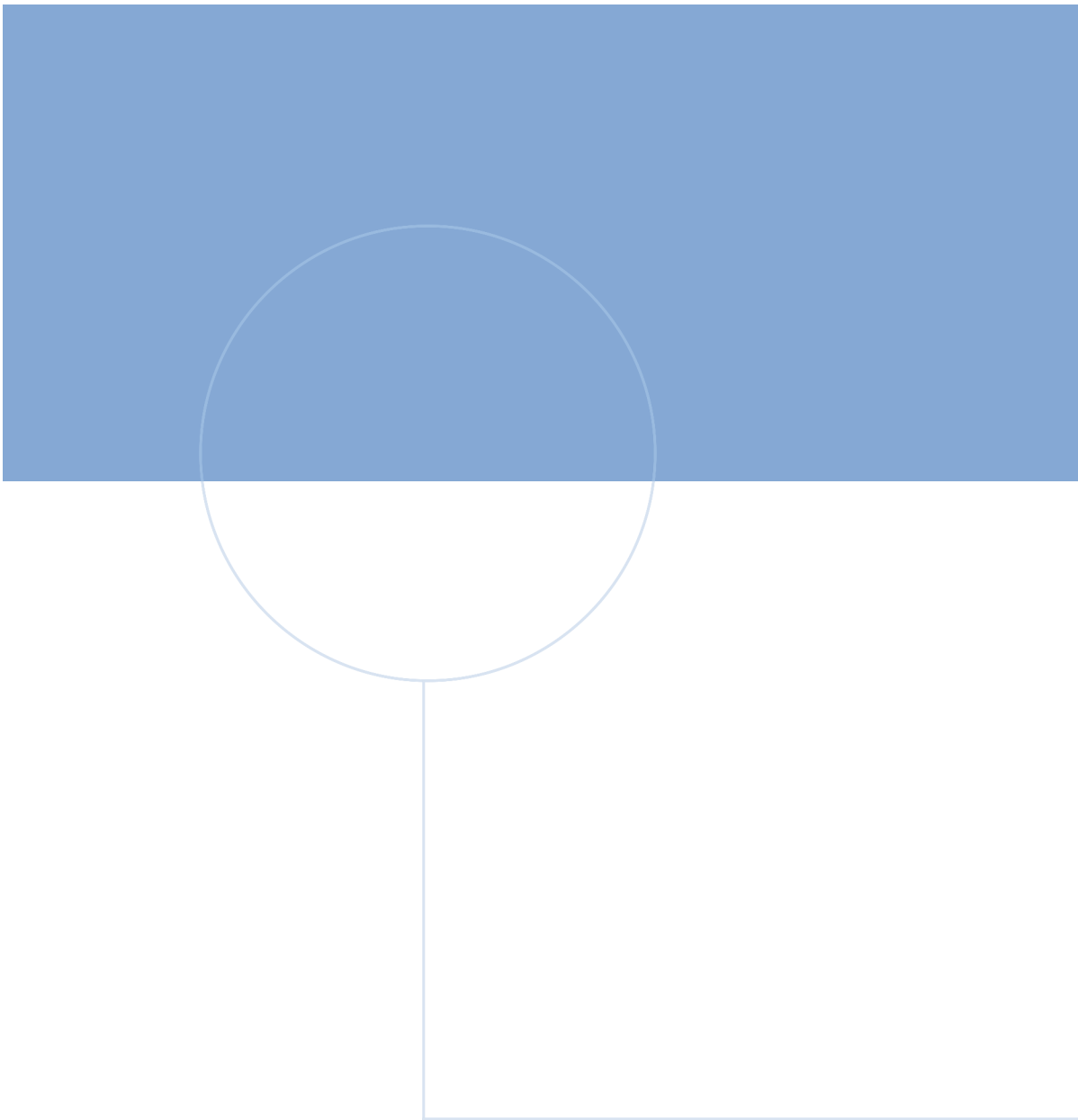
```
        display = "wlan.da == 2C:F4:32:29:36:DC"

if sys.argv[3] == "Shower":
        #Set the shower dates
        dates = ["08.01","09.01","11.01","16.01","18.01","19.01","25.01","30.01","
            31.01","01.02"]

        #Set the x-axis range even tough packets are not sent
        packetstart=["2023-01-08 19:29","2023-01-09 19:29","2023-01-11 19:29","
            2023-01-16 19:29","2023-01-18 19:29","2023-01-19 19:29","2023-01-25
            19:29","2023-01-30 19:29","2023-01-31 19:29","2023-02-01 19:29"]
        packetend=["2023-01-08 21:04","2023-01-09 21:04","2023-01-11 21:04","
            2023-01-16 21:04","2023-01-18 21:04","2023-01-19 21:04","2023-01-25
            21:04","2023-01-30 21:04","2023-01-31 21:04","2023-02-01 21:04"]

        graph_function()

elif sys.argv[3] == "Cooking":
        #Set the cooking dates
        dates = ["08.01","09.01","11.01","16.01","18.01","19.01","25.01","30.01","
            31.01","01.02"]

        #Set the x-axis range even tough packets are not sent
        packetstart=["2023-01-08 15:28","2023-01-09 15:28","2023-01-11 15:28","
            2023-01-16 15:28","2023-01-18 15:28","2023-01-19 15:28","2023-01-26
            15:28","2023-01-30 15:28","2023-01-31 15:28","2023-02-01 15:28"]
        packetend=["2023-01-08 17:07","2023-01-09 17:07","2023-01-11 17:07","
            2023-01-16 17:07","2023-01-18 17:07","2023-01-19 17:07","2023-01-26
            17:07","2023-01-30 17:07","2023-01-31 17:07","2023-02-01 17:07"]

        graph_function()

elif sys.argv[3] == "Window":
        #Set the window dates
        dates = ["08.01-09.01","09.01-10.01","11.01-12.01","16.01-17.01","
            18.01-19.01","19.01-20.01","25.01-26.01","30.01-31.01","31.01-01.02","
            01.02-02-02"]

        #Set the x-axis range even tough packets are not sent
        packetstart=["2023-01-08 22:20","2023-01-09 22:20","2023-01-11 22:20","
            2023-01-16 22:20","2023-01-18 22:20","2023-01-19 22:20","2023-01-25
            22:20","2023-01-30 22:20","2023-01-31 22:20","2023-02-01 22:20"]
        packetend=["2023-01-09 07:39","2023-01-10 07:39","2023-01-12 07:39","
            2023-01-17 07:39","2023-01-19 07:39","2023-01-20 07:39","2023-01-26
            07:39","2023-01-31 07:39","2023-02-01 07:39","2023-02-02 07:39"]

        graph_function()
```

D Script to Generate Graphs for Baseline Comparison with Packets as Reference [62]

```
#GraphsByPackets_BaselineEvents.py
from scapy.all import *
import plotly
from datetime import datetime
import pandas as pd
import pyshark
from pyshark.packet import consts
from pyshark.packet.common import Pickleable
import plotly.graph_objects as go
import sys
import numpy as np

#sys.argv[1] = Name of device
#sys.argv[2] = Choose between inbound, outbound or total packets
#sys.argv[3] = Type of event
#sys.argv[4] = Maximum value for y-axis

display=""

def graph_function():
    times=[]
    z=0
    for date in dates:
        if sys.argv[2] == "Total":
            file = r"C:\Users\Helene\Documents\IMT4905 - Erfaringsbasert master\
                Wireshark\Baseline\\"+sys.argv[1]+"\\"+sys.argv[3]+"\\"+sys.argv
                [1]+"_Baseline_"+sys.argv[3]+"_"+date+".pcapng"
            packets = pyshark.FileCapture(file)
        else:
            file = r"C:\Users\Helene\Documents\IMT4905 - Erfaringsbasert master\
                Wireshark\Baseline\\"+sys.argv[1]+"\\"+sys.argv[3]+"\\"+sys.argv
                [1]+"_Baseline_"+sys.argv[3]+"_"+date+".pcapng"
            packets = pyshark.FileCapture(file, display_filter=display)


        #Lists to hold packet info
        pktTimes=[]
        pkts=[]
        #Read each packet and append to the lists.
        for pkt in packets:
            n=1
            pktTime=(pkt.sniff_time)
            pktTimes.append(pktTime)
            pkts.append(n)

        #This converts list to series
```

```
        packets = pd.Series(pkts).astype(int)

        #Convert the timestamp list to a pd date_time
        times = pd.to_datetime(pd.Series(pktTimes).astype(str),  errors='coerce')

        #Create the dataframe
        df  = pd.DataFrame({"Packets": packets, "Times": times})

        #set the date from a range to an timestamp
        df = df.set_index('Times')

        #Create a new dataframe of 2 second sums to pass to plotly
        df2=df.resample('2S').sum()

        #Create the graph
        GraphTitle=sys.argv[1]+"\n"+sys.argv[3]+"\n"+date
        fig = go.Figure({"data":[plotly.graph_objs.Scatter(x=df2.index, y=df2['
            Packets'])],"layout":plotly.graph_objs.Layout(title=GraphTitle,
                xaxis=dict(title="Time"),
                yaxis=dict(title=sys.argv[2]+" Packets"))})

        #Set the y-axis range
        fig.update_yaxes(range=[0,sys.argv[4]])

        #Set the x-axis range
        fig.update_layout(xaxis_range=[packetstart[z],packetend[z]])

        #Set the font
        fig.update_layout(title=GraphTitle, xaxis_title="Time", yaxis_title="Total
            Packets",font=dict(family="Times New Roman", size=26))

        #Display the graphs
        fig.show()

        z=z+1

if sys.argv[2] == "Outbound":
    if sys.argv[1] == "Netatmo":
        display = "wlan.sa == 70:EE:50:91:06:DE"
    elif sys.argv[1] == "Mill":
        display = "wlan.sa == B8:F0:09:B3:B3:78"
    elif sys.argv[1] == "Nedis":
        display = "wlan.sa == 2C:F4:32:29:36:DC"

elif sys.argv[2] == "Inbound":
    if sys.argv[1] == "Netatmo":
        display = "wlan.da == 70:EE:50:91:06:DE"
    elif sys.argv[1] == "Mill":
        display = "wlan.da == B8:F0:09:B3:B3:78"
    elif sys.argv[1] == "Nedis":
```

```
        display = "wlan.da == 2C:F4:32:29:36:DC"

if sys.argv[3] == "Shower":
        #Set the shower dates
        dates = ["08.01","09.01","11.01","16.01","18.01","19.01","25.01","30.01","
            31.01","01.02"]

        #Set the x-axis range even tough packets are not sent
        packetstart=["2023-01-08 19:29","2023-01-09 19:29","2023-01-11 19:29","
            2023-01-16 19:29","2023-01-18 19:29","2023-01-19 19:29","2023-01-25
            19:29","2023-01-30 19:29","2023-01-31 19:29","2023-02-01 19:29"]
        packetend=["2023-01-08 21:04","2023-01-09 21:04","2023-01-11 21:04","
            2023-01-16 21:04","2023-01-18 21:04","2023-01-19 21:04","2023-01-25
            21:04","2023-01-30 21:04","2023-01-31 21:04","2023-02-01 21:04"]

        graph_function()

elif sys.argv[3] == "Cooking":
        #Set the cooking dates
        dates = ["08.01","09.01","11.01","16.01","18.01","19.01","25.01","30.01","
            31.01","01.02"]

        #Set the x-axis range even tough packets are not sent
        packetstart=["2023-01-08 15:28","2023-01-09 15:28","2023-01-11 15:28","
            2023-01-16 15:28","2023-01-18 15:28","2023-01-19 15:28","2023-01-26
            15:28","2023-01-30 15:28","2023-01-31 15:28","2023-02-01 15:28"]
        packetend=["2023-01-08 17:07","2023-01-09 17:07","2023-01-11 17:07","
            2023-01-16 17:07","2023-01-18 17:07","2023-01-19 17:07","2023-01-26
            17:07","2023-01-30 17:07","2023-01-31 17:07","2023-02-01 17:07"]

        graph_function()

elif sys.argv[3] == "Window":
        #Set the window dates
        dates = ["08.01-09.01","09.01-10.01","11.01-12.01","16.01-17.01","
            18.01-19.01","19.01-20.01","25.01-26.01","30.01-31.01","31.01-01.02","
            01.02-02-02"]

        #Set the x-axis range even tough packets are not sent
        packetstart=["2023-01-08 22:20","2023-01-09 22:20","2023-01-11 22:20","
            2023-01-16 22:20","2023-01-18 22:20","2023-01-19 22:20","2023-01-25
            22:20","2023-01-30 22:20","2023-01-31 22:20","2023-02-01 22:20"]
        packetend=["2023-01-09 07:39","2023-01-10 07:39","2023-01-12 07:39","
            2023-01-17 07:39","2023-01-19 07:39","2023-01-20 07:39","2023-01-26
            07:39","2023-01-31 07:39","2023-02-01 07:39","2023-02-02 07:39"]

        graph_function()
```



NTNU

Norwegian University of
Science and Technology