BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

IOT GATEWAYS NETWORK COMMUNICATION ANALYSIS

ANALÝZA SÍŤOVÉ KOMUNIKACE IOT BRAN

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **JAN ZBOŘIL**
AUTOR PRÁCE

SUPERVISOR **Mgr. KAMIL MALINKA, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2022

Brno University of Technology
Faculty of Information Technology

Department of Intelligent Systems (DITS) Academic year 2021/2022

Bachelor's Thesis Specification

Student: **Zbořil Jan**
Programme: Information Technology
Title: **IoT Gateways Network Communication Analysis**
Category: Security

Assignment:

1. Familiarise yourself with the concept of IoT gateway and its networking capabilities and methods for data analysis of network traffic generated by IoT devices.
2. Create at least 3 instances of environment (based on gateway type) consisting of gateway and multiple IoT devices communicating through gateway, simulate network traffic of these devices. Focus on gateway outbound traffic.
3. Capture and analyze relevant data from different environment instances, compare them and present them in uniform and comprehensible format.
4. Based on results, discuss possible attack vectors such as gateway traffic fingerprinting or end-device classification.
5. Compare your results with expected results and other studies already carried in this field of research.

Recommended literature:

- Amar, Yousef & Haddadi, Hamed & Mortier, Richard & Brown, Tosh & Colley, James & Crabtree, Andy. (2018). An Analysis of Home IoT Network Traffic and Behaviour.
- P. Junges, J. François and O. Festor, "Passive Inference of User Actions through IoT Gateway Encrypted Traffic Analysis," 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), 2019, pp. 7-12.
- Cvitić, Ivan & Perakovic, Dragan & Periša, Marko & Botica, Mate. (2020). Definition of the IoT Device Classes Based on Network Traffic Flow Features. 10.1007/978-3-030-34272-2_1.

Requirements for the first semester:

- Items 1 to 3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Malinka Kamil, Mgr., Ph.D.**
Consultant: Hujňák Ondřej, Ing., UITS FIT VUT
Head of Department: Hanáček Petr, doc. Dr. Ing.
Beginning of work: November 1, 2021
Submission deadline: May 11, 2022
Approval date: November 3, 2021

Abstract

Modern IoT gateways are mainly developed by private companies behind closed doors. This results in a closed ecosystem, where only a tiny amount of information about traffic is available to the public. Therefore, to gain knowledge regarding the operation and communication of such gateways, it is necessary to examine and analyse network traffic flowing to and from such gateways.

This thesis's primary goal is to capture and process network traffic data of multiple commercially available gateways intended for home use, analyse their communication behaviour, compare the results to other studies carried out in this area, and discuss possible attacks on used gateways, based on gathered data. Communication data were obtained by deploying a controlled environment and analysed using Zeek, together with Wireshark software. Collected communication data can be further used by researchers in the areas of networking or security.

Abstrakt

Současné brány internetu věcí jsou nejčastěji vyvíjené soukromými společnostmi. Toto tvoří základ pro proprietární software, o němž výrobci zveřejňují jen málo informací. Proto je pro získání znalostí o způsobů chování těchto zařízení nutné sledovat jejich síťový provoz.

Cílem této práce je prozkoumat síťovou komunikaci několika komerčně dostupných bran pro domácí použití a na základě získaných dat porovnat jednotlivé brány, ověřit výsledky již existujících studií v tomto odvětví IT a zjistit možné bezpečnostní nedostatky těchto produktů. Síťový provoz byl odchycen v rámci uzavřeného prostředí. Získaná data prošla analýzou pomocí nástrojů Zeek a Wireshark. Získané znalosti zhodnocují stav zabezpečení IoT bran pro domácnost. Odchycená datová sada je volně publikovaná za účelem dalšího výzkumu.

Keywords

IoT, IoT gateways, network traffic analysis, attacks on IoT devices

Klíčová slova

Internet věcí, brány internetu věcí, analýza síťového provozu, útoky na zařízení internetu věcí

Reference

ZBOŘIL, Jan. *IoT Gateways Network Communication Analysis.* Brno, 2022. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Mgr. Kamil Malinka, Ph.D.

Rozšířený abstrakt

V současné době zažívá internet věcí pro domácí použití výrazný nárůst z mnoha pohledů. V domovech má chytrá zařízení více lidí, než kdykoliv v historii. Na trhu vystupuje nejvíce různých výrobců s nespočtem produktů. Tento rozvoj začal ve druhé polovině druhého desetiletí 21. století, když na trh přinesly své zboží technologičtí giganti jako Amazon, Apple či Google. Ostatní společnosti se rychle přizpůsobily situaci na trhu a pokusily se připojit do nově vznikajícího segmentu.

Tento rychlý rozvoj, kdy každá společnost hledala vlastní řešení, položil základ mnoha problémům se současným Internetem věcí (IoT). Vzniklá řešení si společnosti chránily, což vyústilo ve výrobu a prodej closed-source produktů. Utrpěla též kompatibilita mezi zařízeními. Toto výrobci pojali jako svou výhodu a další šanci, jak finančně profitovat, když kolem svých řešení vytvořili uzavřené ekosystémy, často s jednoduchým, uživatelsky přívětivým, prostředím, jež značně abstrahuje jednotlivé akce a chod zařízení.

Vznikl požadavek sjednotit komunikaci mnoha koncových zařízení a senzorů. Tento problém řeší brány internetu věcí (IoT gateways). Tato zařízení dnes již umožňují kromě samotné komunikace více zařízení také agregaci dat na společné médium, nečastuji Ethernet, a ovládání zařízení vzdáleně s využitím cloudových řešení. K těmto vlastnostem výrobci začaly postupně přidávat další, například hlasové asistenty. Hranice mezi bránou a zařízením internetu věcí se tedy tenčí. Pomocí IoT bran mnozí výrobci také sbírají uživatelská data určené pro další různá využití.

Akademická sféra může působit jako protipól a kontrolní prvek vzhledem ke komerčním zájmům výrobců. Je potřeba zkoumat chování IoT bran a hledat v nich bezpečnostní chyby s cílem upozornit na tyto bezpečnostní mezery výrobce a dožadovat se jejich nápravy. Vzhledem k uzavřené architektuře IoT bran je často jedinou možností získání dat o jejich chování analýza jejich síťové komunikace. Právě toto si klade za cíl tato bakalářská práce. Na třech vybraných, komerčně dostupných, branách pro domácí použití (Aeotec Smart Home Hub, Amazon Echo, Google Nest Mini) a jedné, vyznávající filozofii open-source (softwarová brána Home Assistant), byl proveden odchyt síťového provozu za běžného provozu.

Odchyt byl proveden ve dvou režimech – aktivním a pasivním. V rámci aktivního režimu bylo cílem získání dat obsahující záznamy o opakovaném provedení operace zapnutí a vypnutí chytré žárovky Phillips Hue. Pasivní odchyt spočíval v sestavení prostředí a sledování provozu po dobu jednoho týdnu bez vnějšího zásahu do prostředí. Cílem bylo získat data o síťovém provozu v době, kdy s branou nikdo aktivně nezachází. Sledované byly DNS dotazy, jednotlivé datové toky komunikující skrze protokoly TCP/TLS, UDP, HTTP, NTP, a jejich vlastnosti. Odchycená data jsou volně publikována za účelem dalšího výzkumu.

Nad získanými daty byla provedena analýza pomocí program Zeek (dříve známým jako Bro). Takto zpracovaná data byla podrobena ruční analýze v tabulkovém editoru se vzájemnou kontrolou údajů a získáváním dalších podrobností o datových tocích pomocí nástroje Wireshark. Výsledkem analýzy je vytvoření virtuálního otisku komunikace, podle nějž je možné bránu identifikovat (tzv. Fingerprinting), poukázat na zajímavé prvky komunikace, zhodnotit zabezpečení jednotlivých bran, uvést statistiky pro jednotlivé brány i jejich celkový souhrn a porovnat výsledky s již existujícími studiemi, například Amar a spol [2]. Ve výsledcích jsou prezentovántovány očekávané prvky komunikace, jakožto například zjištění, že komerční brány generují více provozu než open-source brány, nebo že na základě parametrů komunikace (DNS koncové body, poměr protokolů v komunikaci, vzory nalezené v komunikaci), je možné vyvinout nástroj detekující přítomnost brány v síti, Výsledky také obsahují potvrzení či vyvrácení některých tvrzení prezentovaných studiemi zmíněných v Kapitole 3.

IoT Gateways Network Communication Analysis

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Mgr. Kamil Malinka Ph.D. The supplementary information was provided by Ing. Ondřej Hujňák. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Jan Zbořil
May 11, 2022

Acknowledgements

I would like to thank both the supervisor of this thesis Mgr. Kamil Malinka Ph.D., and my consultant Ing. Ondřej Hujňák, for their advice on the form, content and other necessities, which had to be done in order to complete this thesis.

My biggest thanks comes to my family, who supported me in everything, not only writing this thesis. My gratitude belongs to my friend Tim, who provided me with his excellent moral support.

Contents

Chapter 1

Introduction

Today, the world of Internet of Things (IoT) is largely incompatible. Companies conducting business in this area of information technology are, figuratively, waging wars on each other.

IoT faces many problems, fitting into many categories. According to the article by Hua-Dong Ma [29].These categories are:

- Non-uniformity – the format of data from different devices is inconsistent.
- Inaccuracy - sensors may use various sampling methods, which are not always accurate.
- Discontinuities – network outages or other problems can disturb the continuous stream of data.
- Incomprehensiveness - not enough relevant information is used to come to a conclusion.
- Incompleteness – dynamic changes in the environment may cause the loss of data.

In the last ten or so years, there has been a great industry push to enable the *devices* to communicate with each other to establish a global network of interconnected points.

Before this initiative to standardise IoT communication, there existed, and still exist, manufacturers, developers, and facilitators of IoT devices who operate without regulation or a thought about the security of their products [37]. All of this is happening despite ISO's many studies that proved that the economic improvement, increased innovation, and simplification of the manufacturing process could be the direct consequence of applying standardization.

Without this urge for standardization, each manufacturer used a slightly different way and format of the massages when communicating with an IoT sensor.

The breaking point happened in the 2010s when IoT transformed from being an instrument for modernization of industry to being a convenience in the everyday life of many people around the globe. With the coming of product families of devices like Amazon Echo, Google Home, or Apple HomeKit, there came the need of producing user-oriented IoT gateways which would provide simple UI for managing user's home automation network, and which would be able to aggregate all the communication protocols and standards into one box. This box's purpose is to abstract anything regarding the setup of devices and communication with them.

This development brings new problems to the light of the day. Manufacturers often developed these gateways behind closed doors. This situation has evolved to the current state

when few big players exist on the market with IoT gateways for home use (Called *Smart Home IoT, SHIoT*, by Cvitić et al. [10]). These companies provide means of integration and development for their respective platform.

Therefore, there emerges a problem regarding data coming out from these closed-off gateways into the cloud. Due to no standardization, cooperation between manufacturers and proprietary solutions, there is no easy way of getting insight into the operation of devices in question.

This thesis aims to create a network traffic analysis of a few commercially available IoT gateways marketed towards home use. Chosen gateways are *Aeotec Smart Home Hub*, *Amazon Echo Dot*, *Google Nest Mini* and *Raspberry Pi with Home Assistant* software installed. This analysis suggests attack vectors, such as traffic fingerprinting or device classification, and the prevention of exploitation of these attack vectors. Comparison with already researched information in this area is made to validate the results of the aforementioned analysis.

To achieve the aforementioned target, the following steps had to be carried out:

- Select IoT gateways to experiment with.
- Create a functional physical network consisting of given gateways and other devices to simulate network traffic.
- Capture and analyse network traffic of selected IoT gateways.
- Present the newly learned information. Compare the results to existing research, like [2, 11, 26], and discuss possible vulnerabilities.

The points listed above together form the practical part of this thesis.

The Chapter 2 explains the basic concepts of IoT devices and gateways.

The Chapter 3 provides a broader context regarding IoT research as a whole. This knowledge can help to better comprehend topics related to this work.

The next Chapter, 4, introduces the ways of capturing network data and means of its analysis. In the following Chapter, 5, the selection of devices used in this thesis is described in more detail, together with particular methods used for the experiments.

The final Chapter, 6, presents the results of the conducted analysis. It discusses possible security deficiencies and their countermeasures. It demonstrates the analysis results in relation to other studies already carried out, and it points out the differences or similarities.

Chapter 2

Introduction to IoT

This Chapter introduces the primary division of devices used in IoT networks and links that interconnect them to one working network. Section 2.1 describes IoT gateways—the foundation of this thesis. Section 2.2 explains the use of IoT devices as the end devices of IoT networks.

2.1 The Purpose of an IoT Gateway

The things in the *Internet of Things* would be completely utterly unless they had a convenient way of communication between themselves and other non-IoT enabled devices connected to the Internet. It is, in fact, the goal of these *things* to be constantly connected and provide means of control to the user no matter of his physical location. IoT gateways are one of the links which make this interconnectivity possible.

The concept of an IoT Gateway can be comprehended similarly to the one of a router in the realm of classic computer networks. Likewise a router is a border point between the local network and the outer world, which can be shrunk behind one of its interfaces; the IoT Gateway forms a border between the devices and the Internet. Moreover, unlike the classical router, it brings together all the different forms of protocols like ZigBee or Z-Wave [31] used in communication between various IoT devices and a gateway and transposes them to be able to flow through the twisted pairs of the Ethernet from the gateway to the cloud.

Categorization of IoT Gateways

IoT gateways, also known as IoT hubs, can be devised into categories based on many parameters. These range from type and location of use (gateways for home or industrial use) to the mean of implementation (dedicated hardware or software gateway running on a multipurpose device).

Another purpose of an IoT gateway is to process data before forwarding them to the cloud. Deduplication or aggregation can be one of these processes whose goal is to reduce the amount of data flowing into the cloud. Improved security is often cited as a benefit of using an IoT gateway [32]. This is not an absolute truth, because security can be a significant pitfall of such devices in some cases. Security vulnerabilities can appear, especially if a gateway is being used after the end-of-life or a period of support from its manufacturer. The role of an IoT gateway in a network is manifested in Figure 2.2.

(a) Aaeon AIOT-IGWS01 industrial IoT gateway. Original image is taken from [1]

(b) Aeotec Smart Home Hub, intended for home use. Original image is taken from [31]

Figure 2.1: Visual comparison between an industrial 2.1a and home-use 2.1b IoT gateway.

IoT Gateways for Industrial Use

Industrial gateways, such as is shown in Figure 2.1a, can be a part of the backbone of entire company-wide or city-wide operation [34]. This use values open standards, flexible architectures, an enormous amount of devices connected simultaneously, security, *big data* processing in local or *edge* environments and analytical tools working, often in real-time, for monitoring and failure detection.

IoT Gateways for Home Use

On the other hand, gateways used in a home environment, like Aeotec Smart Home Hub, shown in 2.1b, are mainly made to be easily operated by the end-user. These products often have lower connectivity, are designed to be operated via a smartphone application, and are more energy efficient than their industrial counterparts. This means these gateways are often just a proxy, pre-processing data at the edge [44] and sending it to the cloud to be further processed there.

Nowadays, the line between a gateway and a device is getting blurry in the home segment of IoT business. Home monitoring centres like Amazon Echo, Google Nest or Apple HomePod serve simultaneously as a gateway and as a device in a sense that, for example, voice assistant included as a feature of the products mentioned above enables a user to get info from the cloud.

2.2 IoT devices

IoT devices form the lowest tier in the IoT network hierarchy, which is demonstrated in Figure 2.2. They play a role of end devices. Similarly to IoT gateways, their foremost purpose differs based on the environment they are deployed in and features desired by a customer. In IoT networks targeted towards home use, devices are usually used to form a *Smart Home*. This umbrella term contains the simplest devices like humidity sensor or window or door contact sensor, which entire use case is to detect real-life status of an object or an environment and send the data to a user, traditionally via an IoT gateway.

On the other hand, a more complex device, smart television, for example, can be also categorised as an IoT device. In connection with an IoT gateway, these devices can coop-

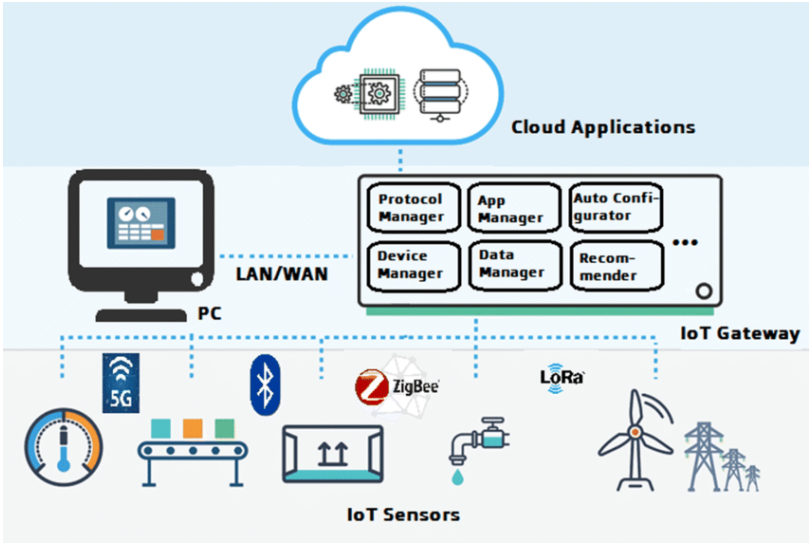


Figure 2.2: The diagram shows the basic architecture of an IoT network consisting of devices, a gateway, cloud operations, and a control station. Original image is taken from [17].

erate and provide more elaborate actions. This can be, for example, turning off a heater, and rolling down the blinds when the temperature and light sensor measures values beyond a certain threshold.

In industrial settings, devices are mainly performance, real-time and security oriented [28]. These devices also have to be more physically robust than they home-use counterparts in order to sustain sometimes extreme environments in industrial production. Their fault rate also has to be kept to minimum. These requirements result in devices often being certified by international organizations like IEC.

IoT devices usually do not communicate with other nodes in a network via typical network protocols like Ethernet or Wi-Fi. Instead, they take an advantage of low power wireless protocols specially designed for this use. Their purpose is to relatively frequently transfer only a small amount of data to a gateway, or a control station if the IoT gateway is omitted from the network. There are more types of these protocols [21], depending on their use. Therefore, these protocols can be classified either as *IoT data protocols* or *Network protocols for IoT*, depending on which network layer they operate, when parallels to TCP/IP model are used. Detailed information about TCP/IP is located in Chapter 4.1.

Protocols belonging to the category of data protocols are MQTT (Message Queuing Telemetry Transport), CoAP (Constrained Application Protocol), AMQP (Advanced Message Queuing Protocol), DDS (Data Distribution Service), or standard HTTP (Hypertext Transfer Protocol). Network protocols for IoT are Wi-Fi, Bluetooth, ZigBee, Z-Wave or LoRaWan. However, this thesis is concerned with data sent from an IoT gateways to the Internet. More detailed description of the device protocols is therefore not needed.

Chapter 3

Related Work

This Chapter aims to provide a wider view to the academic research concerned with IoT gateways and their properties. An overview of the research, which went into this section of IT in the last decade can be helpful in order to fully comprehend, how the topic of this thesis slots in the whole area of IoT research. Sections of this Chapter reflect on periods and main concerns of researchers in these periods.

Researchers have already carried out studies in the field of the traffic and security of IoT gateways. Amar et al. [2] built out an environment consisting of many devices and gateways, and captured the traffic for 22 days. Their work explains the learned device characteristics or other non-expected results, especially in how the devices were set up and how they operated. They talk about how devices communicated during the 22 days of capturing their traffic, and they discuss global statistics, including total bytes sent per device, per protocol, etc. This thesis methodology is based on the one in this study, in order to have files, which can be compared to each other and therefore the discoveries in Amar's study can be either confirmed or disproved. Unlike this thesis, which only research IoT gateways, their work also discusses the traffic of end devices and network infrastructure.

Ivan Cvitić et al. published a research [11], in which they were successfully able to sort IoT devices into several classes based on characteristic of their network traffic flows. The paper also includes devices communicating via a Zigbee protocol, among those using Ethernet or Wi-Fi. Based on the coefficient of variation of received and sent data (C_u index), and data transformation, used for the data to behave like a normal distribution, so statistical tests for this particular distribution could be used; they were able to link the device to one of four classes of devices sharing similar behaviour.

In his book [30], Petr Matoušek also discusses the methods of packet classifications and data filtering. He explains the algorithms (prefix based search, divide and conquer methods) and data structures (tries) used to classify data; how to implement them for packet filtration on routers and switches, and the working principle of access control lists (ACLs). Information contained in headers of packets is used to achieve this goal. He also briefly mentions the existence of DPI—deep packet inspection.

In a research [26], Pierre-Marie Junges et al. used captured traffic analysis to infer user actions, such as turning the smart light on or off. They took the position of an outsider, looking into the traffic between the LAN and the cloud. They noted, that identification of the devices, especially from the TLS handshake, can be extracted from such traffic. First, they identified the problems of deducting information from the traffic. They claim they are the following: *no individual IoT device signature*, *gateway abstraction* and *encryption*. They also made several assumptions, on which their later action is based on: *sending actions*

to the IoT devices-actions in one command are sent as one; *incidence of the actions on the packet size*, *command size stability* and *data structures similarity*. They then captured network traffic, while operating multiple IoT devices, using combinations of different actions on various devices. After measuring the size of datagrams from the start of captured TLS streams, and doing computations, they were able to distinguish the correct operation taken with 98.4% accuracy. However, authors rely heavily on assumption, that the IoT device sends traffic to the cloud after an action inside the network is taken, which is not always true, as is later revealed in this work.

Chapter 4

Overview of Network Traffic, its Capturing and Analysis

To analyse the traffic flowing through the network, one must first understand how the messages are sent and received and what their content is. To standardise this content format, two network communication models had emerged—TCP/IP and OSI.

This standardization is one of the reasons that IoT devices now can cooperate and function together, despite using different protocols to communicate. This Chapter concerns the network traffic due to the nature of this work, which deals with the traffic of IoT gateways, which flows through classic computer networks to the cloud.

4.1 The principle of network communications

Scientists in the 1960s and 1970s were more intrigued by the idea of connecting multiple computers, in order to allow direct communication between them. Their research led, at last, to the creation of two reference models. These are the *OSI reference model* and *TCP/IP reference model*. Differences between these two models are described in next sections and in Table 4.1.

OSI model

The ISO OSI[1] model was the first step towards the standardization of networking. It was presented in 1983 by Day and Zimmermann[15]. It discusses seven layers, with each of them having a signal purpose. Together, these layers forms a working hierarchy providing a functional communication.

Protocols specified in this model are now not in active use any more. However, the model itself is still regarded as theoretically valid as the functions of its layers can be projected to models and protocols used today.

This model has been accused of being flawed and unpractical by many in the computer scientific community, therefore the second model, the TCP/IP model, is now used.

TCP/IP

TCP/IP is now a go-to architecture when dealing with data transmission over a network. This suite of protocols had formed in the 1970s. Its name is derived from the names of

[1] abbreviation for International Organization for Standardization, Open Systems Interconnection

L num.	ISO OSI model	L num.	TCP/IP model	Protocols
7	Application	4	Application	DNS, HTTP, FTP, SMTP, SNMP
6	Presentation			
5	Session			
4	Transport	3	Transport	TCP, UDP
3	Network	2	Network	IPv4, IPv6, ICMP
2	Data Link	1	Network Interface	Ethernet, 802.11, Token Ring
1	Physical			

Table 4.1: Table showing differences between the ISO OSI and TCP/IP models.

two most important protocols in the entire architecture — *Transmission Control Protocol (TCP)* and *Internet Protocol (IP)*. Since then, it contains complete description of network communication from the lowest level of a wire to high level of network applications. This architecture operates with a concept of a layer, similarly to the OSI model. Each layer is responsible for one of the steps, resulting in a functioning way of sending data over a network. Layers utilise data coming to them from higher layer. They take this upper layer data and add their own generated data. This process is called encapsulation. Data coming from the upper layer to the lower one is called PDU (protocol data unit) and is referred to differently on each layer.

The *data link layer* is the lowest one in the TCP/IP hierarchy. It operates only in a scope of a local area network. This layer's crucial function is delivering frames (name for PDU on data link layer) between hosts in LAN. Frame mainly contains a MAC addresses of sender and destination, error detecting code and data of a higher layer (payload). Fields in frames on the data link layer differ, depending on a media type.

The *internet layer* provides methods of delivering data to a destination based on globally unique[2] identifiers called IP addresses. PDU of the internet layer is called a packet. It uses best-effort delivery [30]. This means that the IP layer tries to deliver data to a destination via the finest way it can find. It is not responsible for delivering each packet to the set destination. However, in case of failed delivery, the internet layer provides ways of notifying a sender of a failure. ICMP protocol is used for this task. Addressing information is the most valuable asset of this layer, which is relevant for this thesis.

The *transport layer* is the layer above the internet layer. This layer's main goal is to establish and keep a communication channel between two hosts alive. It provides *logical connection* between processes [30]. The transport layer offers two types of sending data depending on how the data is handled in case of failure during transport. Transmission Control Protocol (TCP) loads data from the application layer and sends them as a stream of packets to a network. With help of sequence numbers, it can keep track of packet order and non delivered packets. This mechanism enables TCP to operate as a reliable protocol. In case a PDU is dropped or not delivered, the TCP is able to, and required to, send the data again. User Datagram Protocol (UDP) is a non-reliable alternative to TCP. It forfeits the ability to automatically retransmit lost data or deliver packets in a right order in exchange for increased speed and reduced overhead.

Application layer is the highest in TCP/IP hierarchy. It provides many services, each usually utilising a specific protocol for communication. Most information about the content, which IoT gateways exchange with the cloud, originates in this layer. Application layer also

[2]Today, this is not strictly true, due to usage of technologies such as NAT.

provides a security protocols for communication, making an analysis harder, sometimes even impossible. Due to limited computation power of IoT devices, the cryptographic protocols often remain unused in IoT.

4.2 Delivering Data to the Point of Capture

The process of capturing data from a network consists mainly of two parts. These are *a.* how to get data flow to the point of capture, and *b.* how to capture the data itself at the point of capture. The process of diverging the data or its copy from its original path is called *Tapping*. Other names for this process are *Sniffing the wire* or *Tapping the network* [38]. *Tapping* is an analogy of a classic tap used on pipes for opening or closing a valve. There are two widely used methods of delivering network traffic to the desired destination. These methods are *Port Mirroring* and using a *Hardware TAP*.

Other methods, like Cisco's *Embedded Packet Capture* feature on selected routers of this vendor [7], can be used for capturing data flowing through a network. However, capturing packets using this method is designed only for short-term usage—as an assistance during debugging, for example. Even if captured pcap files can be exported from a router using FTP, the vendor does not recommend using Embedded Packet Capture for long-term monitoring. This method has not been selected for packet capturing in this thesis, due to aforementioned design choices made by its creators.

Port Mirroring

The idea of port mirroring is to divide traffic on the second layer of the OSI model. This divided traffic is then *a.* switched as it would be without port mirroring in place, and *b.* send out through the interface configured as the *Destination port*. A sniffing machine connected to this specified port then receives all the packets which are ingressed into the switch through all the ports configured as *Source port* or *Monitored port*. This is shown in Figure 4.1.

Port mirroring can be configured to divert the traffic flowing into the *monitored port* located on the same network switch as the port mirroring is running on or other switches in the local network. This feature is not available on all the switches and requires a special VLAN to be running, which is used to deliver data from the *monitored port* to the switch where the port mirroring is configured. Using a VLAN for port mirroring opens the potential for simultaneously capturing data from more sources (more physical ports). Then a list of source ports is called an *Administrative source*, and a list of effectively monitored ports is called an *Operational source*.

Cisco calls the process of port mirroring SPAN, which is an abbreviation of *Switched Port Analyser*.

TAP

TAP is an abbreviation of the English words *test access point* or *terminal access point* [19]. It is a physical device consisting of one or more input and output ports. These are the ports that are used to deliver the unchanged network traffic from its source to its intended destination. Nevertheless, a network tap device includes more ports. Ports in question are *Monitor* ports that output all the traffic coming to the TAP's *Network* ports. Some TAPs can capture the data in different directions in two separate channels. This is demonstrated

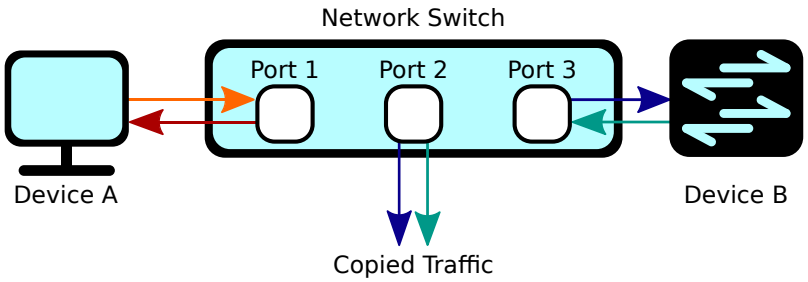


Figure 4.1: Network switch with port mirroring activated. The image shows the scenario where Port 2 is configured as a mirroring port to the Port 3. Traffic flowing through Port 3 is sent out to its original destination (Port 1) while being simultaneously sent to the mirroring port.

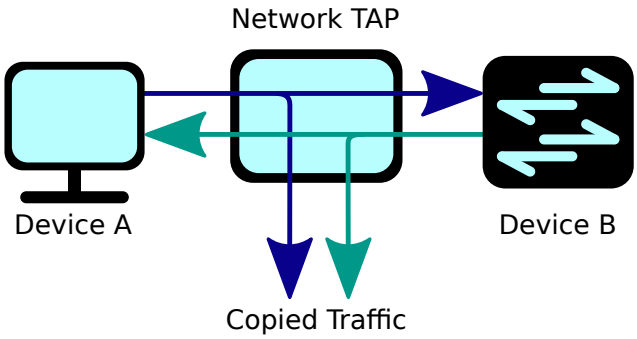


Figure 4.2: Diagram of a TAP operating between two network devices. Network TAP copies traffic out of the wire without impeding the original traffic. The Figure illustrates the TAP, which uses separate outbound ports for each direction of the original traffic.

in Figure 4.2. On the other hand, there are TAPs that use one aggregated port to achieve the same result. TAPs supporting both of the methods mentioned above exist.

TAPs are often active, powered using an electrical cord connected to a wall socket. This power delivered to TAP is used to regenerate and amplify a signal. Active TAPs can incorporate an internal battery used as a means of power to the device in the event of a power outage. Complementary to an active TAP is a passive TAP, which does not require any external power.

Installing a TAP into a physical network requires breaking an existing Ethernet line. Therefore, TAPs cannot be used without interrupting network functionality. Thus, TAPs are often being installed during network shutdowns and maintenance.

Network hub

Given the nature of the operation of basic hubs, these devices can be used to deliver traffic to the point of capture. These days, when the dominant L2 device in a network is a network switch, finding a hub is getting more challenging due to the obsolescence of such devices. Even if there is a hub available, its use poses some disadvantages. In the same manner, as for TAPs, hubs cannot be inserted into the network without a disruption in the network's functioning. Another significant disadvantage of using a hub is its core principle of flooding all incoming traffic on all its ports, excluding the in-port. The outcome of this is the

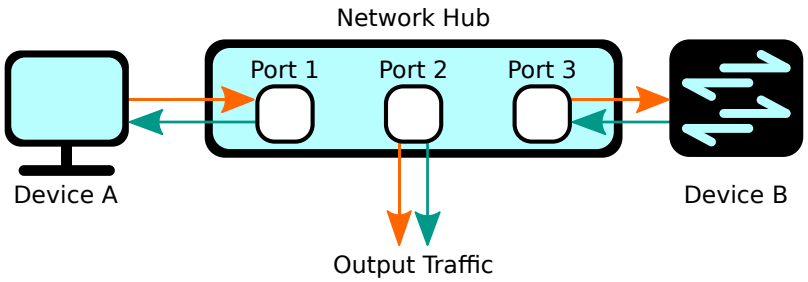


Figure 4.3: Operation of classic network hub. Communication between Device A and Device B is flooded out of the hub, through every port (2). The capturing station, connected to such port, has the ability to capture all the traffic flowing through any port on the hub.

wastefulness of network resources, like bandwidth. This is one of the main reasons hubs have been pushed out of the L2 device market. The operation of a hub can be seen in Figure 4.3.

Hubs in which only three ports are used—two for devices and one for copied traffic—function as an aggregated TAP.

4.3 Means of Capturing

Now, when the traffic, meant to be captured and analysed, is diverged or copied to the capturing device, it is time for the capturing itself. In order to be able to capture traffic flow, the device's interface of a NIC has to be configured to operate in so-called *promiscuous mode*. This mode enables the network card to listen to all the traffic coming to its interface. Had it not been switched to promiscuous mode, the NIC would discard any frames not addressed to it [38].

The NIC converts incoming data from a binary form to a readable form with promiscuous mode enabled. Data in readable format can be saved to file, analysed, or processed by specialised software. Before doing these operations, the traffic can be filtered using *Berkeley Packet Filters* (BPF) [46]. These filters can be used to select only the traffic that a user wants to analyse, either by specifying which traffic to discard or which traffic is to be kept. Table 4.2 shows some examples of such filters. These filters can be combined in order to get more desired results. The syntax for comparison and logical operators follows the standard of the C language. BPF filters also support a C language binary operators +, -, *, /, %, including bitwise operators like << or &.

Many network tools, called analysers and sniffers, can be used to capture, analyse and save the traffic in a form that a human can read. Each of them provides its user a different level of depth regarding information obtained from captured traffic, a different way of data presentation, and some of them can provide advanced functions like being able to follow network streams. The examples of such tools are *Tcpdump* [45], *Wireshark* [8], *TShark* [24], *SolarWinds Network Performance Monitor* [42] or *Kismet* [27]. Some of these tools are specialised. For example, Kismet's main domain of operation is wireless networks.

Tcpdump

Tcpdump is a CLI based network packet sniffer and analyser. This piece of software, originally written in 1988, is distributed under BSD licence [45]. Throughout the years, it

Filter Syntax	Syntax Meaning	Example
dst host *host*	True if the IPv4/v6 destination field of the packet is *host*.	dst host merlin.fit.vutbr.cz
src host *host*	True if the IPv4/v6 source field of the packet is *host*.	src host merlin.fit.vutbr.cz
ether src *ehost*	True if the Ethernet source address is *ehost*.	ether 30:9c:23:03:1f:84
dst portrange *port1-port2*	True if the packet is IPv4 TCP, IPv4 UDP, IPv6 TCP or IPv6 UDP and has a destination port value between *port1* and *port2*	dst portrange 67-68
ip proto *protocol*	True if the packet is an IPv4 packet of protocol type *protocol*. *Protocol* can be a number or one of the names icmp, icmp6, igmp, igrp, pim, ah, esp, vrrp, udp, or tcp.	ip proto udp
ip multicast	True if the packet is an IPv4 multicast packet.	

Table 4.2: BPF filters, examples taken from [46]

gained popularity, resulting in it now being preinstalled on Apple's macOS and many Linux distributions. Port of Tcpdump for Windows also exists.

Tcpdump uses the Libpcap library for capturing packets. Tcpdump is highly configurable and enables a user to get various information. Tcpdump can, for example, sniff only on a selected interface, capture a given amount of packets, list all network interfaces currently configured on a system, et cetera. All options for Tcpdump can be found on corresponding manual pages [25]. Tcpdump supports filtering captured traffic using BPF filters described in Chapter 4.3. A demonstration run of Tcpdump is shown in Figure 4.4.

Wireshark

Wireshark is, similarly to Tcpdump, a network sniffer and packet analyser. With its graphical user interface and the ability to dissect traffic in order to show information to a user in a friendly and comprehensible way, it became one of the most known and used software in the network monitoring community. The original version written by Gerald Combs in 1998 [8] is released under a GPL licence.

Advanced features of Wireshark are protocol dissection, live sniffing, offline analysis, ability to follow network streams (TCP, RTP, and more), ability to filter shown traffic and decryption for security protocols such as Kerberos, SSL/TLS, SNMPv3 or WPA/WPA2 under certain circumstances.

Wireshark's dissection is one of the core features of this software. It is able to show detailed information about traffic details based on the used protocol. As of 22nd December 2021, Wireshark supports 2673 protocols[3] with the ability to import user-defined dissectors for showing custom protocols or displaying existing protocols differently. User-defined dissectors can be written in C or Lua programming languages. An example of a custom dissector written in Lua can be seen in Figure 4.5 and at [49].

[3]Official GitHub repository of Wireshark dissectors can be found at [48]

```
janz  honzalinux  ~  sudo tcpdump -Avnc 1
dropped privs to tcpdump
tcpdump: listening on enp33s0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
15:07:32.982483 IP (tos 0x0, ttl 64, id 50049, offset 0, flags [DF], proto TCP (6), length
1500)
    192.168.0.50.81 > 192.168.0.18.52292: Flags [.], cksum 0x3667 (correct), seq 3311074246
:3311075694, ack 3880696991, win 1944, options [nop,nop,TS val 94189109 ecr 3941566055], le
ngth 1448
E.....@.@......2.....Q.D.Z ... N......6g.....
..65 ... g....\I.}.......
... NG(H8.EC..B.D·— ..* ... z.J.
...
...  << Output omitted >>
...
Ri..#3m.eX..  .Qn.|.f.5 ... &P9..EW`.tf..$g......2%....`..qY....P....a..G..5$....l.v....Z....\
 ... ua}.....).f.+..!U#)y ... kx.....^ ... Us..\..q.O..$.....w..q..6]4.w....gn..VLfw.i.2y;Jg; ... p
9.......9....\ ... i.........<.c._....)E.8.q.rh ... ScQ ... #S.[vIlmZq|$..DTwR..\J.c......].M....
M[Ks..f.s..L1
1 packet captured
122 packets received by filter
0 packets dropped by kernel
```

Figure 4.4: Screenshot of a packet captured by Tcpdump. This capture had been configured with the following flags: A—print packet content in ASCII, v—verbose, n—do not translate IP addresses to hostnames, c 1—capture precisely one packet. Considering that Tcpdump is using a direct link to capture off the system's NIC, it has to be run with elevated privileges.

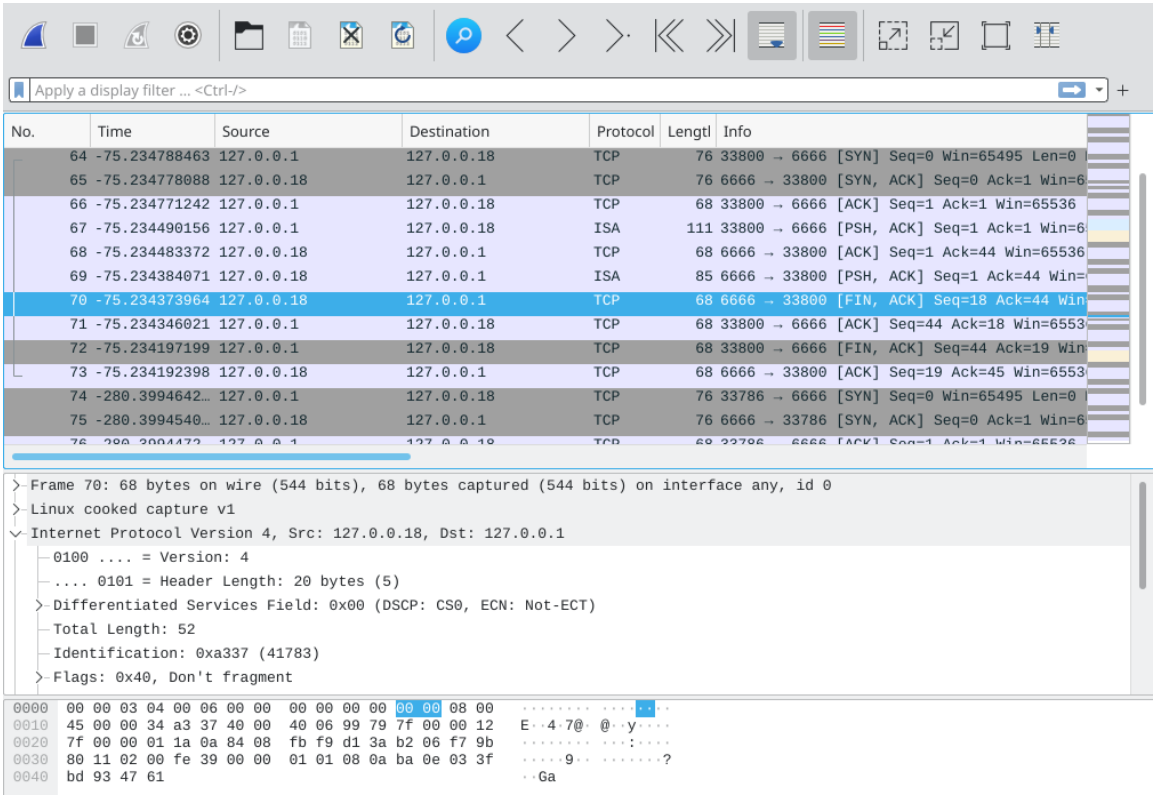


Figure 4.5: Screenshot of Wireshark application with a pcap file opened. The section in the middle shows a list of captured packets with essential information like source and destination addresses shown. The section below the list displays dissected data of a selected packet. The bottom field presents frame data in raw format—encoded as hexadecimal numbers (left) and ASCII (right).

Tshark [8], an abbreviation of Terminal-based Wireshark, is a command-line interface tool based on Wireshark supporting the same file format, filters and other options as Wireshark. The running of the application without any options results in output as if Tcpdump had been used. In Linux operating systems, Tshark comes bundled in Wireshark packages. Given the CLI operation of Tshark, tools like PyShark or Termshark [24] have emerged, enabling more throughout analysis of traffic on a command line.

4.4 Means of Traffic Analysis

Analysing captured data is a significant step in gaining knowledge of a researched topic. Many tools exist for this exact purpose. These tools can be divided into offline analysers and online analysers. Most of these tools are not only analysers but analysing traffic is only one of their functions. The other features are a GUI, used to present data in aggregated and logically divided sections, network flow monitoring in real-time, providing various graphing tools, creating general statistics, providing tools for stream following or filtering, et cetera.

Wireshark for Traffic Analysis

Sniffing network traffic is not the only use of Wireshark. Its ability to dissect many protocol formats and follow a network stream makes it a powerful tool for analysing traffic. Wireshark provides the following statics:

- General properties of a captured file.
- Used protocol hierarchy with the amount of traffic using given protocol.
- Follow distinguished conversations and streams.
- Show traffic endpoints.
- Plot I/O utilization graph.
- Display aggregated information about widely used protocols such as DNS or HTTP.
- Follow IP telephony conversations.

Zeek

Zeek [47], formerly known as Bro, is network monitoring software that can operate as a live traffic analyser or analyse existing pcap files. Zeek is able to dissolve a pcap file and produce logically differentiated logs, each containing statistics for a given category. These logs also contain transcripts of L4 protocols used. This includes HTTP sessions or DNS requests and replies. It also provides means of detection of non-standard communication patterns like DoS or attempts to brute-force SSH. Zeek is able to export this data into JSON format, which can then be imported into SIEM software for visualization. Zeek comes with a scripting language that can be utilised to get desired info from traffic, which is not extracted from captured data by default.

Other visualisers and monitors

Data formatted by Zeek can be uploaded into visualization systems like Kibana [16] to get data presented to a user in a friendly and comprehensive way. Kibana shows graphs of protocols usage, IP address map, data amount statistics, et cetera. More systems similar to Kibana exist. These are, for example, Prometheus [33], Datadog [14] or Grafana [20].

Chapter 5

Setup, Implementation, Capturing and Data Processing

A set of gateways, IoT devices, and other networking devices had to be thought of and established into a working network to carry out the experiments and analyse the traffic. This Chapter describes the process from selecting devices to gathering outputs of an analysis of IoT gateway traffic.

5.1 Selection of IoT gateways

When selecting gateways to be used for this thesis, the following points played a role:

- Market availability for general, non-technical customers.
- Well known products, even for general public.
- Closed-source architecture[1].
- Compatibility with the same devices in order for the results to be comparable.

These properties of selected gateways were selected in order for the scope of the research results to cover the largest possible amount of end users. The closed-sourced architecture was chosen to show that the traffic analysis can bring useful information even for the devices that have their implementation and internal functionality concealed. Based on these point, following gateways were selected, their summary is in Table 5.1:

- Aeotec Smart Home Hub
- Amazon Echo Dot, 4th gen
- Google Nest Mini, 2nd gen
- Raspberry Pi with Home Assistant software

[1]Except Home Assistant enabled Raspberry Pi, for comparing proprietary devices with open source counterparts.

Gateway Name	Manufacturer	Connection to router	Control App	MSRP[2](January 2022)
Aeotec Smart Home Hub	Aeotec/ Samsung	Ethernet	Samsung Smart Things	$135
Amazon Echo Dot, 4th gen	Amazon/ Foxconn	Wi-Fi	Amazon Alexa	$49.99
Google Nest Mini, 2nd gen	Google Nest	Wi-Fi	Google Home	$49.00
Raspberry Pi 4 Home Assistant	Element14/ Sony	Ethernet	Home Assistant	starting at $35 (Raspberry Pi 4)

Table 5.1: Table showing IoT gateways selected for data analysis. Data from [31, 39, 36, 35].

5.2 Selection of IoT devices

Devices selected to simulate an environment of the local network behind an IoT gateways have been chosen mainly regarding their compatibility with all the gateways stated above.

Phillips Hue Smart Light Bulb with Hue Bridge

Phillips Hue is an ecosystem of different lighting solutions made by this Dutch company. Hue line of products first appeared on the consumer market in 2012. From this year to the present, Phillips has revised Hue line-up, and they now provide complete solutions for the home automated lighting system.

Hue line offers two modes of operation [40]: *a.* Bluetooth connection, and *b.* connection through a central hub. The first option is limited to a maximum of 10 devices and is limited compared to the second option. Connecting lights to a central Hue Bridge device is recommended. When connected to the bridge, all the light can be controlled centrally through one point of contact, and control can be even delegated to another device, such as Google Nest or Aeotec Smart Hub.

Although the Hue Bridge itself is an IoT gateway, its operation is limited only to other Hue products. Therefore, this thesis considers it a device and not the border point between smart home LAN and the cloud.

Signify Holding, the parent company of Phillips has been awarded as the best IoT security company [41] and is the first company conducting business in smart lighting to be awarded security certifications.

5.3 Turris MOX Router

Researched IoT gateways were connected to the Turris MOX modular router to gain Internet connectivity. This router is set up to function as a switch, default gateway, but mainly as a point of capture.

[2]MSRP - Manufacturer's Suggested Retail Price

Figure 5.1: Basic MOX Pocket Wi-Fi package consisting of mainboard module with processing module, memory and Wi-Fi NIC. Original image is taken from [13]

Turris is an open source project by CZ.NIC [12], with the primary aspiration being security. Turris MOX is created with modularity in mind. Basic configuration, serving as a router can be expanded with various modules to add storage capacity, switched RJ-45 ports, wireless radio, or support for PoE. An example of built configuration of a MOX router is shown in Figure 5.1. The Turris router runs a Turris OS, an OpenWrt Linux distribution. Turris OS provides Sentinel security software, dynamic firewall, honeypot-as-a-service or a simple VPN client. Because it is Linux based, popular CLI based programs like Tcpdump can be run on it, which is indeed used for capturing data for the purpose of this thesis.

5.4 Environment Deployment and Capturing Methodology

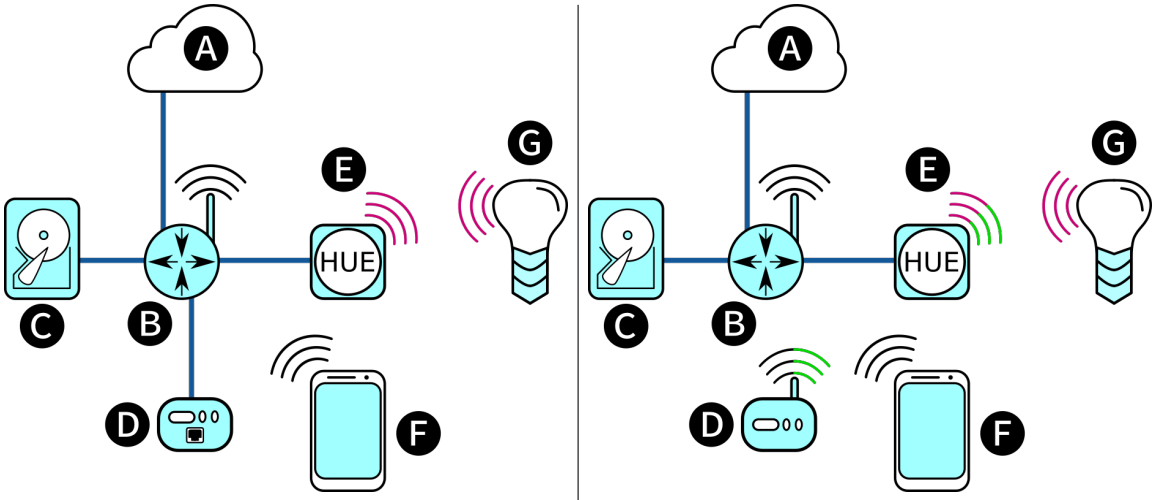
All gateways were connected to the Turris router either by Wi-Fi (Amazon Echo, Google Nest Mini) or Ethernet (Aeotec Smart Home Hub, Raspberry Pi). An External SSD manufactured by Samsung was connected to the Turris router, utilising its USB port. This SSD was formatted to the *ext4* file system and permanently mounted through *fstab*.

Turris Router was set up to work as the LAN's default gateway, with 192.168.1.1 as its IP address. A DHCP reservation was used for all the IoT gateways in order to streamline the analysis process.

Devices were set up using their respectable Android application. Following applications were used: SmartThings[3] for Aeotec control, Amazon Alexa[4] for Amazon Echo, Google Home[5] for Google Nest Mini and Home Assistant[6] for Raspberry Pi. Phillips light bulb was first connected to the Phillips Hue Bridge using the Philips Hue application[7]. When connected to the bridge, it was then connected to the smartphone via the respectable application listed above. During the setup, sending telemetry and other optional data was always enabled, even in the case of the open-sourced Home Assistant.

During the setup process, there emerged problems with setup of the Amazon Echo and Google Nest Mini devices. Amazon Echo treats the connections to the Phillips Hue Bridge in two ways. First, the Hue Bridge can communicate directly with the controlling device,

[3] SmartThings application in Google Play
[4] Amazon Alexa application in Google Play
[5] Google Home application in Google Play
[6] Home Assistant application in Google Play
[7] Philips Hue application in Google Play



(a) Environment diagram with an Ethernet wired IoT gateway. Capturing is done on the Ethernet port of B, leading into D.

(b) Environment diagram with a Wi-Fi wireless IoT gateway. Capturing is done on the WLAN interface.

Figure 5.2: The Figures show how the devices, which the LAN was composed of, were connected. A—the cloud; B—Turris router; C—external SSD for captured files storage; D—researched IoT gateway; E—Phillips Hue Bridge; F—smartphone with control application; G—Phillips Hue light bulb. Wireless communication between the devices is colour coded.

and it eliminates the need for an Echo Gateway entirely. Communication with Amazon servers is then transmitted from the controlling smartphone. Second, the Google Nest Mini setup failed with an undisclosed error in the Google Home application. The gateway was functional, except the voice commands were not working. Non-operating connection to the Internet was the reason, according to the error message, spoken by the Google Nest. Device settings in the smartphone application were not working either. These problems were solved by factory reset of the device and repeating the setup process.

The schema of the built environment is in Figure 5.2.

Capturing Methodology

Data collection was performed for each of the gateways separately. No other device except the IoT gateway itself, Turris MOX router, Phillips Hue light bulb and Phillips Hue Bridge was connected to the LAN during the data capturing. This was meant to reduce any unwanted traffic in the network.

Traffic was captured in two different operation modes for each gateway, producing two data sets for each gateway. These data sets were *a. Passive capture*, and *b. Active capture*.

Passive capture data was collected during a span of one week, with a maximal deviation of less than $\pm$ 1 hour. During this time, no outer influence in the LAN had been introduced. All traffic captured is therefore passive, hence the Passive capture name. All outbound traffic captured in this mode is solemnly generated by IoT gateway and other network devices without any interference by the user.

Active data capture was also performed on each of the gateways. Active data capture consisted of sets of the same number of actions performed on the Phillips Hue light bulb.

Gateway	Capturing mode	Connection	Raw Size	Filtered Size	Raw Packets	Filtered Packets
Aeotec	A	Eth	58.8 KiB	51.0 KiB	302	268
Aeotec	P	Eth	139.6 MiB	93.7 MiB	701 448	525 089
Echo	A	Wi-Fi	305.8 KiB	134.4 KiB	595	319
Echo	P	Wi-Fi	1.1 GiB	221.6 MiB	1 454 195	525 888
Google	A	Wi-Fi	49.6 KiB	24.1 KiB	383	172
Google	P	Wi-Fi	2.7 GiB	117.0 MiB	3 407 511	507 148
RPHA	A	Eth	19.3 KiB	29.6 KiB	289	194
RPHA	P	Eth	141.3 MiB	83.5 MiB	1 014 335	719 611

Table 5.2: Table showing the differences between the sizes of gathered pcap files of each IoT gateway, before and after filtration. Gateway's names are shortened: Aeotec Smart Home Hub = Aeotec, Amazon Echo = Echo, Google Nest Mini = Google, Raspberry Pi with Home Assistant = RPHA. Capturing mode: A = active, P = passive; Connection: Eth = Ethernet

This means repeatedly switching on and off the lights using a smartphone application meant to control the given gateway.

Data capturing was done on Turris MOX router using Tcpdump CLI program, more in 4.3. Tcpdump was saving live data to Samsung SDD connected via USB to Turris router. The router was set up to connect to the rest of the location network using Ethernet. The Turris router was serving as a DHCP server for the testing LAN. Captured data, stored in pcap files, was retrieved using a remote SSH connection to the router after the capturing process had finished.

5.5 Analysis methodology and process

Captured and sorted pcap files were manually filtered to contain only the information desired for the analysis. The main criterium for the filtering was the MAC address of a given IoT gateway. The amount of captured data, which is not helpful for the analysis, depends on the type of media carrying the data (Wi-Fi or Ethernet). Gateways communication via Ethernet suffered a lower traffic overhead than devices carrying the information using Wi-Fi. Table 5.2 shows the difference between sizes of captured, raw pcap files, and filtered pcap files. The entire dataset, composed of raw pcap files without any filtering done, is available at https://nextcloud.fit.vutbr.cz/s/TTtEqT8wJLwDAs4.

Pcaps with filtered traffic were subjugated to the Zeek software. Zeek logs were extracted from each file. These logs contained information about traffic streams, DNS queries and answers, records of transferred files, TLS connections details, statistics. Zeek logs content were copied to spreadsheet editor for easier manipulation and filtration of the data. Each pcap file had its own spreadsheet with corresponding Zeek logs.

Data, copied to the spreadsheet editor, were manually inspected, and essential information was extracted from them. This includes mainly resources regarding DNS records and queries, time intervals between the queries, endpoints and intervals of NTP streams, ports, target's hostnames, features marking and identifying the traffic stream of UDP streams, conversation destinations, intervals, number of packets in a burst for ICMP communications, intervals, endpoints, stream identifiers, features, patterns corresponding to the actions

taken during the active capturing for TCP/TLS conversations, intervals, destinations, contents of sent documents for HTTP streams. Zeek logs were regularly being cross-checked with the content of the pcap files, viewed in Wireshark. An interesting traffic behaviour, observable patterns, intriguing nuances of the traffic, cryptographic information and other significant properties of communication were searched for (both manually and with use of simple scripts) in the spreadsheet containing filtered Zeek logs. The results of this analysis were stored in a standalone spreadsheet. Graphs, histograms or traffic flow graphs were created where applicable.

Chapter 6

Experimental Results

This Chapter shows results of carried out experiments, ranging from the amount of captured data of each gateway, during passive or active operation, via comparison of different gateway's traffic, finally leading to a discussion on possible vulnerabilities. Information extracted from captured data is written in this Chapter sequentially, based firstly on *a.* gateway used, and then secondly *b.* capture mode. Summarised results for all gateways follow.

6.1 Aeotec Smart Home Hub

This section covers the experimental results for the Aeotec Smart Home Hub, which is operated by Samsung's SmartThings ecosystem.

Passive Capture Results

Aeotec Smart Home Hub ranks second out of four regarding the amount of filtered data transmitted. However, unfiltered pcap size is the lowest of all gateways, but this number is not representative of valuable information. Out of 89886558 transported bytes, 68.7 % were TCP, while 55.1 % were TLS. TLS contributed 28.6 % of all in terms of packets. UDP makes only 9.8 % of all transmitted bytes and DNS 6.2 % of bytes. Aeotec sent 39429 DNS requests, but they were asking only for three queries. These were: api.smartthings.com, fw-update2.smartthings.com and dc-eu01-euwest1.connect.smartthings.com. These queries were sent from the 192.168.1.101 DHCP static address, given to the Aeotec gateway, and the fd86:36f3:b032:0:b5ef:f6ef:d663:e589, fd86:36f3:b032:0:2a6d:97ff:fecd:4adc and fd86:36f3:b032:0:11c6:f88f:bdaa:8339 IPv6 addresses used by the gateway. Only queries sent from the aforementioned IPv6 addresses were for api.smartthings.com. All queries were sent to the Turris router, with the IP address of 192.168.1.1, serving as the LAN's default gateway. Queries and answers are listed in Table 6.1. Intervals between the requests with the same query vary from 01:05 [mm:ss] to 15:01.

This gateway held communication streams to the api.smartthings.com endpoint, TCP port 443. Both were originating from Aeotec's IPv4 address. The first type of connection happened 14446 times during the capture period. Endpoint addresses are located in Dublin, Ireland. Consulting WHOIS database, the service is provided by Amazon AWS. All the connections to this endpoint used TLS 1.2, with Diffie-Hellman as the key exchange method. The secp256r1 curve was used. TLS payload data form just 131 bytes outbound and 199 bytes inbound for the one instance of the communication. TLS handshake bytes are not included. These conversations were repeated each minute during the whole capture

Query	A	AAAA	Answers
api.smartthings.com[1]	14446	14447	18.202.137.67 3.248.90.154 34.249.105.246 34.250.163.199 34.251.239.96 34.253.133.191 52.17.91.165 52.18.35.218 52.19.161.236 52.208.106.25 52.208.23.209 54.76.18.183
api.smartthings.com[2]	4254	4255	same as above
fw-update2.smart-things.com	1013	1013	34.240.106.143 54.194.56.171 176.34.194.138
dc-eu01-euwest1.connect.smartthings.com	1	0	54.171.188.111 54.217.91.49 99.80.191.26

Table 6.1: Table showing the DNS requests by the Aeotec Smart Home Hub. Column A lists the number of requests for A records. Column AAAA shows the number of queries for AAAA records. 1. Queries sent from 192.168.1.101; 2. Queries sent from IPv6 addresses.

week. Based on its characteristics, it can be assumed that the purpose of these streams is "Check if the cloud service is available" or "Send a small amount of telemetry" type of conversation. A standard communication flow graph is shown in Figure 6.1.

Streams towards the fw-update2.smartthings.com endpoint ran 1013 times. They utilised HTTP over TLS. The endpoint is located in Dublin, Ireland, the same as the previous streams, and is served by Amazon AWS. Intervals between iterations vary from 5:01 to 15:01. Communication used SHA1WithRSAEncryption certificate. The certificate for establishing TLS stays the same in all iterations. Server Hello used a self-signed certificate, utilising the TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 cipher suite. Based on the FQDN of the endpoint, it can be assumed that the purpose of these streams was to check for software updates or security patches.

The last TCP stream held during the time of capturing had dc-eu01-euwest1 .connect.smartthings.com as its endpoint. This stream, which was already running when the capturing had begun, ran until 6.5 days into the capturing. A TCP packet with a reset flag set halted it. Immediately after this, a new stream was established, which lasted past the capture period's end. The destination IP address belongs to Amazon AWS, and it is located in Dublin, IR. According to the I/O graph in Figure 6.2, this connection is predominantly idle (thanks to the TCP keep-alive packets), or it is sending small repeating messages, 40 and 42 TLS payload bytes in size. Spikes in the graph are all originating from the server-side.

Because Aeotec does not utilise NTP for time synchronisation, the previous streams could also carry the time synchronisation data.

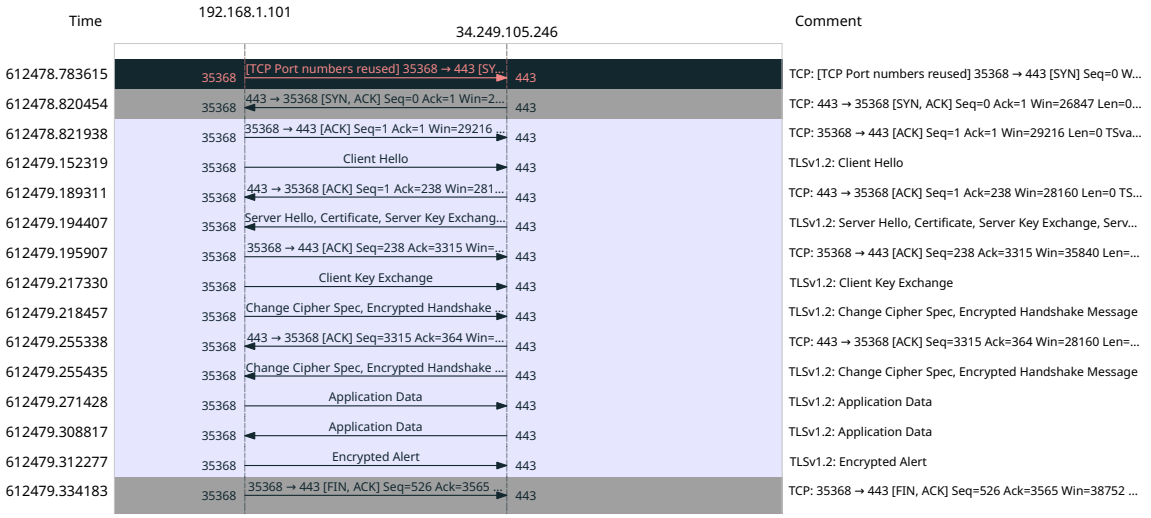


Figure 6.1: Traffic flow graph representing the typical TCP stream of the Aeotec gateway towards the api.smartthings.com endpoint.

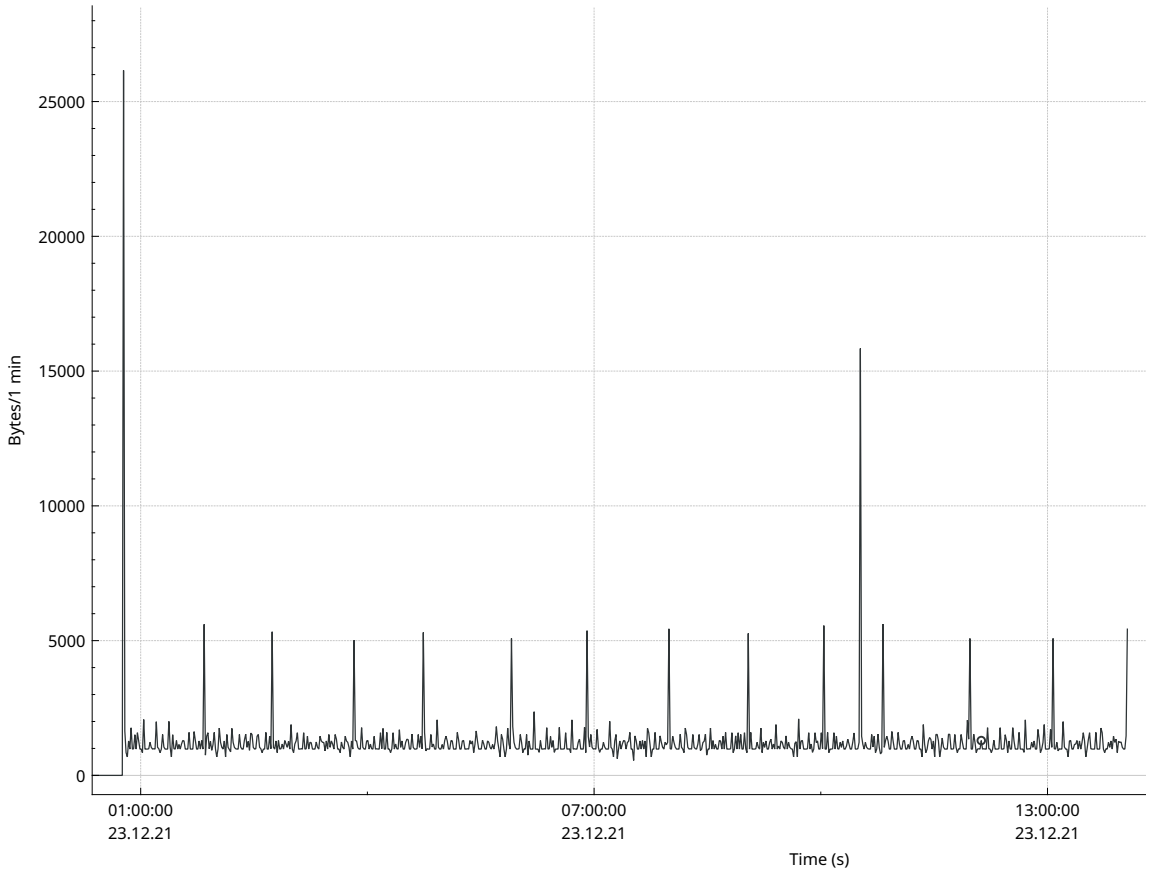


Figure 6.2: Figure shows the input/output graph of the stream from the Aeotec gateway towards the dc-eu01-euwest1.connect.smartthings.com endpoint. X-axis step is 1 minute.

Active Capture Results

During the 49 seconds of active capturing, while the Phillips Hue light bulb was turned on and off 10 times, 47869 bytes in 268 packets were transmitted. 57.4 % of these bytes are TLS, which corresponds to 45.9 % of packets. 12 UDP DNS packets were sent and received by the gateway.

All the DNS queries were for api.smartthings.com. One of the answers for the query was 3.248.90.154, which was subsequently used for TLS connection. This TCP/TLS connection used TLS 1.2. This conversation happened twice. The first stream begun 26 seconds into the capture; the second one lasted past the end of capturing, therefore it is incomplete in the pcap file. Conversation contains TCP handshake, TLS establishment and 1 outbound packet of 126 TLS content bytes, 194 bytes of inbound TLS payload data and 1 encryption alert message. Stream is correctly finished using the sequence of TCP packets with FIN, ACK flags and TCP response with the ACK flag. Destination IP is located in Dublin, Ireland, and it points to Amazon AWS services—Amazon EC2, which is, quote [5]: "Secure and resizeable compute capacity for virtually any workload." TLS certificate is issued by GeoTrust, for api.smartthings.com. Cipher and curve used is the same as in the case of passive capturing.

The second TCP/TLS stream, with high probability, carries data which are generated when operating the Phillips light bulb. At the time of capture start, the connection had already been established. Same as above, destination endpoint belongs to Amazon AWS in Dublin, Ireland. The packet's pattern is notable. Here, sizes of frames are main determinants. The found pattern consists of 42 TLS data bytes (113 frame B) outbound, 425 TLS data bytes (496 frame B) outbound and 42 TLS data bytes inbound (113 frame B), followed by 247 TLS data bytes (318 frame B) inbound. There are 57 inbound (37 times with the frame length of 113 B) and 57 outbound (20 times with the frame length of 113 B) packets. 113 B long frames seems like initiation frames. Difference between turning the bulb on and off cannot be determined by packet length. During one operation, communication consist of either *a.* 6 packets for operation (113 B outbound, 490 B outbound, 113 B inbound, 318 B inbound, 113 B outbound, 490 B outbound), while not all frames were captured or *b.* 4 packets per operation (out 113, out 490, in 113, in 318) and other packets are not carrying the action data itself, but can function as a flow control, integrity check, et cetera. Due to use of encryption, the aforementioned relation between frames and action is speculative.

6.2 Amazon Echo

The following section contains information about the results for the Amazon Echo gateway, which was controlled using the Alexa application for Android. Results presented below are extracted from data collected, when the Echo device was connected to the Internet. The pcap file, captured when the gateway was not connected to the Internet, was not considered for the analysis.

Passive Capture Results

During the span of the week, when the capturing was carried out, the Amazon Echo device queried for 23 different DNS records. This is more than any other tested gateway (Google Nest ranked second, with 21 unique queries). All the DNS queries were asking for A records only. Most DNS requests (2014) queried for d3p8zr0ffa9t17.cloudfront.net do-

Query	Amount	Min. Interval	Max. Interval
api.amazon.com	173	00:58:20	00:58:21
api.amazonalexa.com	1942	00:00:01	00:58:21
avs-alexa-14-na.amazon.com	117	00:51:24	02:00:04
todo-ta-g7g.amazon.com	2	00:00:00	00:00:00
softwareupdates.amazon.com	8	00:15:33	25:35:01
acsechocaptiveportal.com	32	00:00:02	11:03:09
arcus-uswest.amazon.com	6	24:00:00	24:00:01
d1s31zyz7dcc2d.cloudfront.net	1	-	-
dxz5jxhrrzigf.cloudfront.net	1	-	-
d3p8zr0ffa9t17.cloudfront.net	2014	00:01:08	00:05:01
dcape-na.amazon.com	1	-	-
det-ta-g7g.amazon.com	2	01:07:44	01:07:44
device-artifacts-v2.s3.amazonaws.com	15	00:00:06	03:14:33
device-messaging-na.amazon.com	2	68:47:03	68:47:03
device-metrics-us-2.amazon.com	411	00:00:00	01:44:51
dss-na.amazon.com	7	00:00:00	41:01:46
ffs-provisioner-config.amazon-dss.com	5	41:00:00	41:01:46
fireoscaptiveportal.com	31	05:15:04	06:00:01
ingestion.us-east-1.prod.arteries.alexa.a2z.com	2	164:08:08	164:08:08
mlis.amazon.com	2	00:03:23	00:03:23
msh.amazon.com	28	00:02:21	12:57:10
ntp-g7g.amazon.com	31	00:02:21	06:00:00
prod.amcs-tachyon.com	2	00:00:00	00:00:00

Table 6.2: Table showing the differences and patters in DNS queries by Amazon Echo gateway. Intervals bring a new insight, how an Echo device uses DNS, alongside the number of queries. Intervals can also function as one of the metrics for the device fingerprinting.

main name. Cloudfront is Amazon's content delivery network service [3]. Forty total unique answers belonged to the following networks: 13.27.171.0/24, 13.32.0.0/16, 18.66.0.0/16, 52.85.114.0/24, 54.192.147.0/24, 65.9.94.0/24, 99.86.247.0/24. The second-highest amount of requests queried for api.amazonalexa.com happened 1942 times. The tp.b16066390-frontier.amazonalexa.com and d1gsg05rq1vjdw.cloudfront.net domain names were returned in the answer, among other IPv4 addresses. All destinations and amount of queries is in Table 6.2. IP addresses found in answers for queries in Table 6.2 are hosted by Amazon AWS.

Every 5 minutes, the Amazon Echo sends bursts of 10 ICMP Echo messages towards the LAN's default gateway. Echo gateway uses NTP service for its time synchronisation; timestamps of NTP streams correspond with timestamps and intervals in the DNS queries. NTP conversation occurred 31 times, the same as was the number of DNS queries. NTP servers are: 52.45.237.36, 34.236.6.206, 52.203.151.208, 34.239.12.200, 18.205.68.150, 3.234.38.31, 34.226.24.249 and 3.214.58.173.

Amazon Echo initiates conversations with the Phillips Hue Bridge. It is an HTTP transfer. Typically, nine times in a row, the HTTP exchange is sent. Then once, another HTTP conversation is sent. This last HTTP response from Hue Bridge contains information about itself. These ten messages repeatedly appear 35 times. Spacing between the 35 streams is reasonably consistent.

The Echo takes part in HTTP communication towards 11 endpoints, which IPv4 addresses were obtained in the DNS answers for the d3p8zr0ffa9t17.cloudfront.net and acsechocaptiveportal.com queries. These streams happened 1649 times. They always consist just of an HTTP connection test.

Last HTTP based conversation reaches the fireoscaptiveportal.com endpoint. Streams repeated 30 times. Intervals between the streams copy the DNS intervals. All streams are the same, with the packets with 267 B of HTTP payload and 260 B inbound of HTTP payload. Interestingly, the response is delivered in two packets, with 204 and 400 return codes.

The Echo gateway took part in a total of 17 distinct TCP/TLS streams. The first was directed to the 65.9.90.59 IP address, which was obtained by querying the DNS for d1s31zyz7dcc2d.cloudfront.net. It lasted 42 seconds. The stream carried 569808 B inbound while transmitting 105251266 B outbound from the Echo towards the endpoint. By these parameters, it can be assumed, with confidence, that the gateway pulled a software update using this stream.

Stream, which repeated mostly—3047 times, communicated with api.amazonalexa.com. Originating payload size ranges from 1066 to 1693 bytes. Response payload is much larger—ranging from 5980 to 21141 bytes. Most connections are towards the 65.9.91.171 and 13.32.135.16 addresses (2876 out of 3047). These streams are short, with an average length of 0.37 seconds. Address changes seem logical, with clusters of conversations with the same address. This supports the theory of load balancing. All conversations are encrypted using TLS 1.3 or TLS 1.2. When using TLS 1.3, client hello frames are padded in the TLS layer to be 583 bytes long. TLS 1.2 packets do not have padding. Amazon issued all certificates. Either the TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256, or TLS_AES_128_GCM_SHA256 ciphers were used.

Connection to api.amazon.com ran 166 times. Intervals between these streams range from 00:00:00 to 02:55:02. These streams had origination packets with a TLS payload of 1354 bytes. Endpoint IP addresses do not follow sequentially in time, as in the previous stream. However, this behaviour can be caused by a relatively low amount of occurring streams. The average duration of the stream is 0.51 seconds.

Streams towards avs-alexa-14-na.amazon.com ran 130 times. There were significant differences in intervals between conversations (8 s – 2:00:00 h). Payload ranges from 931 B to 50146 B inbound and from 1990 B to 25231 B outbound. All streams have the following TLS handshake: Client Hello = 268 B, Server Hello = 1514 B, Certificate, Server key exchange = 690 B, Client Key Exchange = 180 B and Change Cipher Spec = 105 B (entire frame lengths). Stream durations ranged from 8 to 7204 seconds.

Conversation between the Amazon Echo and softwareupdates.amazon.com happened 8 times. Intervals between streams range from 00:15:33 to 1 day and 01:35:01, which corresponds with DNS queries. These streams do not presumably download software updates, as the endpoint's name may suggest. Based on the amount of data transmitted, these streams could have been checking for available updates. Certificates were issued by Digicert. The cipher used is TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256.

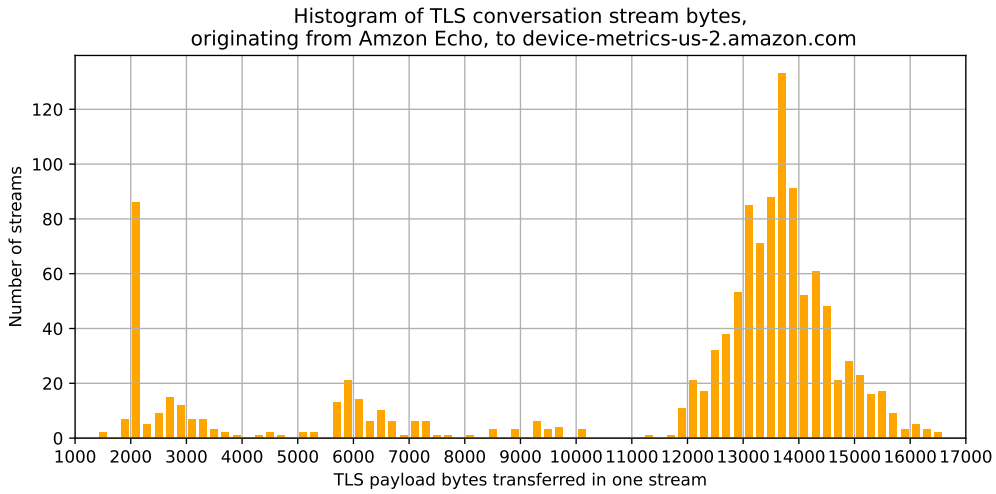


Figure 6.3: The histogram shows the number of streams towards the device-metrics-us-2.amazon.com, which transferred the same amount of TLS data. Three distinct groups based on the packet length are clearly seen. Most streams carried around 2000 B, 6000 B and 14000 B of TLS payload.

Stream towards arcus-uswest.amazon.com was established six times. These streams stood out thanks to intervals between them, which are one day exactly. Stream happened every day at 11:31:50 CET.

Communication with device-messaging-na.amazon.com happened twice. Interval was 2 days and 20:47:03. Most data is originating from Amazon Echo (88 %). The endpoint's name suggests that it is part of the Amazon Device Messaging solution that lets users send messages to Amazon devices running their app [6]. However, most of the data sent via these streams originate at the Echo gateway, negating the theory of a server sending messages to end devices.

The stream with the second-highest repetitions was directed towards the device-metrics-us-2.amazon.com endpoint, and it ran 1204 times. TLS payload without overhead ranges from 1523 B to 16580 B. Response TLS payload with no overhead always was 5317 Bytes. Intervals between streams ranged from 00:00:00 to 04:41:59. Based on the endpoint FQDN, these streams sent reports to Amazon servers. After the TLS handshake, most of the payload originated from Echo—statistics and metrics. The only payload coming back to Echo was 327 B, which was transmitted in the last TLS packet before an encrypted alert message and TCP packet with FIN and ACK flags. The amount of data carried is distorted by server hello messages and certificates, which Zeek counts into payload data. Streams can be sorted into three groups based on the amount of TLS data sent through them, as is shown in Figure 6.3.

Endpoints of all streams, including those not paragraphs above, are shown in Figure 6.4.

Comparing learnt information about Echo's traffic to the research of Amar et al. [2], both differences and similarities can be found. Even when accounting for the difference in the length of capturing, the number of DNS requests, and consequently, intervals between them, do not match. DNS query for the device-metrics-us.amazon.com endpoint in Amar's study happens 119.6 times a day on average, while in this research, the number is 58.7 times a day for the device-metrics-us-2.amazon.com. In this capturing, there was only a single query for dcape-na.amazon.com, while Amar and his team captured 1210 such requests.

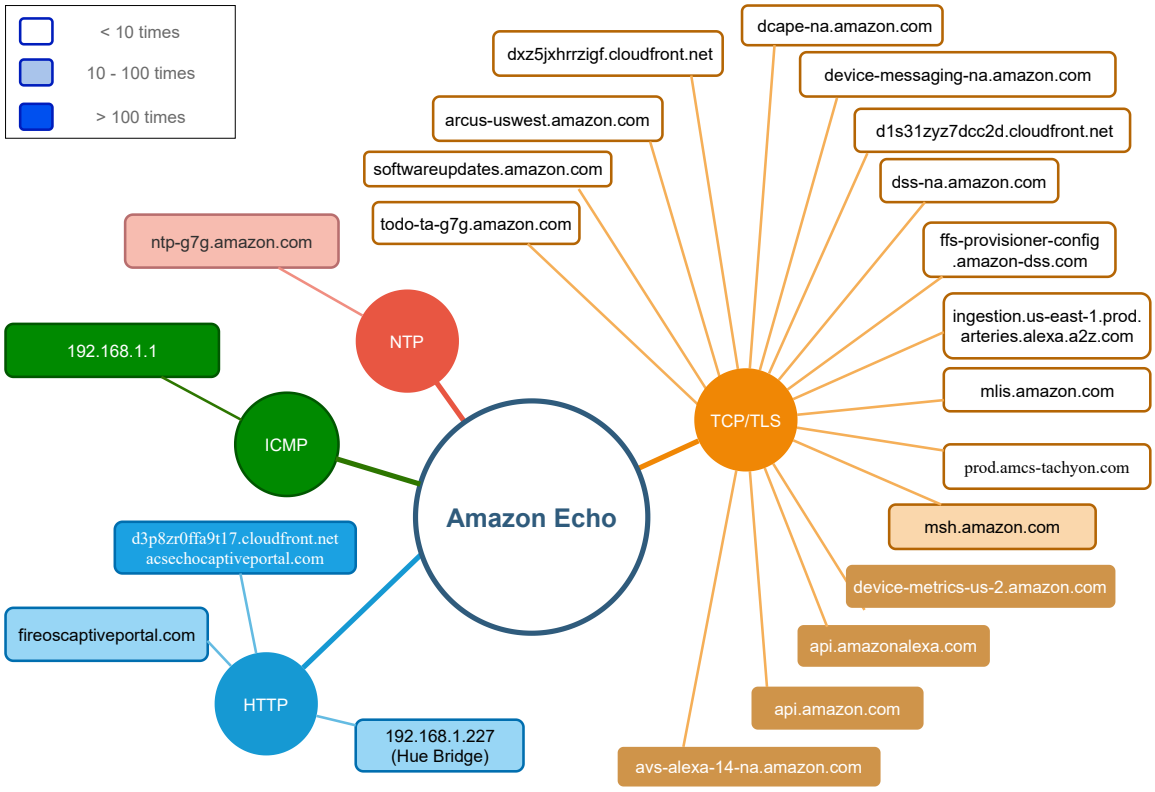


Figure 6.4: Map of endpoints to which the Amazon Echo established connection. Connection type is differentiated by colour. Colour shade marks the number of connections to the endpoint.

Unlike them, the queries for obscure sites, like www.example.org or www.example.net, never did occur. The number of ICMP connections is comparable. While the Echo device in this research sent 287.7 ICMP echoes per day, it was 308.1 echoes in Amar's study. Since the capturing time for the first and last day is not known in Amar's study, the numbers may match more closely in reality.

Active Capture Results

During the 40 seconds of active capturing, one DNS query was sent towards the Turris router. The query was asking for the A record of msh.amazon.com.

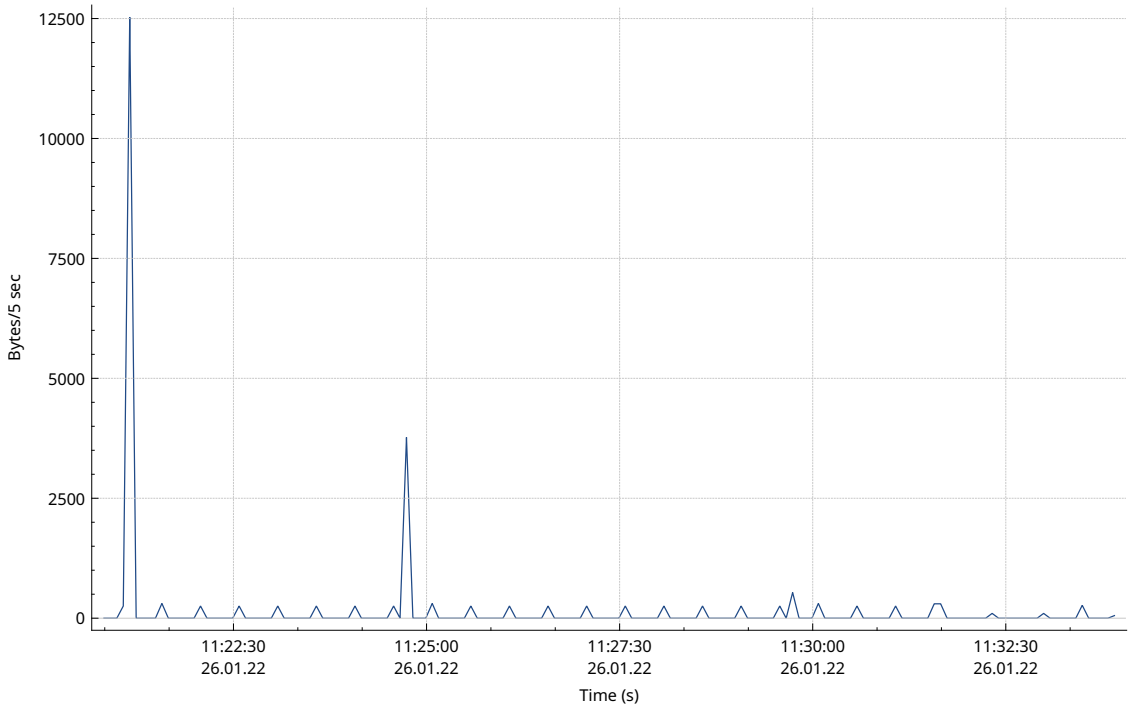
Captured streams had to be divided into two separate categories. There was a significant number of streams towards the cloud, which had not originated from the Echo gateway, but from the Android phone, which was used to send commands to the Echo. Only two distinct communication streams originated from the Echo. The first one established itself against msh.amazon.com endpoint, following the aforementioned DNS query. Communication consisted from the TCP/TLS handshake, two inbound data packet and one outbound. Then it was closed with TCP FIN, ACK. It used TLS_ECDHE_RSA_WITH_AES_128_GCM-_SHA256 and a certificate signed by Amazon. A new connection to the same host followed immediately. It carried more data than the first connection.

The second stream originating from the Echo communicated with avs-alexa-14-na.amazon.com. It was part of a much longer stream with start and end, which was not in the scope of the active pcap file. Endpoint contacted the Echo every 30 seconds with 46 TLS bytes, the Echo replied with the same amount of data. It lasted until Jan 26, 11:33:58 CET. For this whole duration, two packets, with the length of 46 TLS bytes, were being exchanged every 30 seconds. Sometimes, more than 46 TLS bytes were transferred—this is visible in Figure 6.5a. This connection was also found to be originating from the smartphone with the Alexa application used to operate the Phillips light bulb.

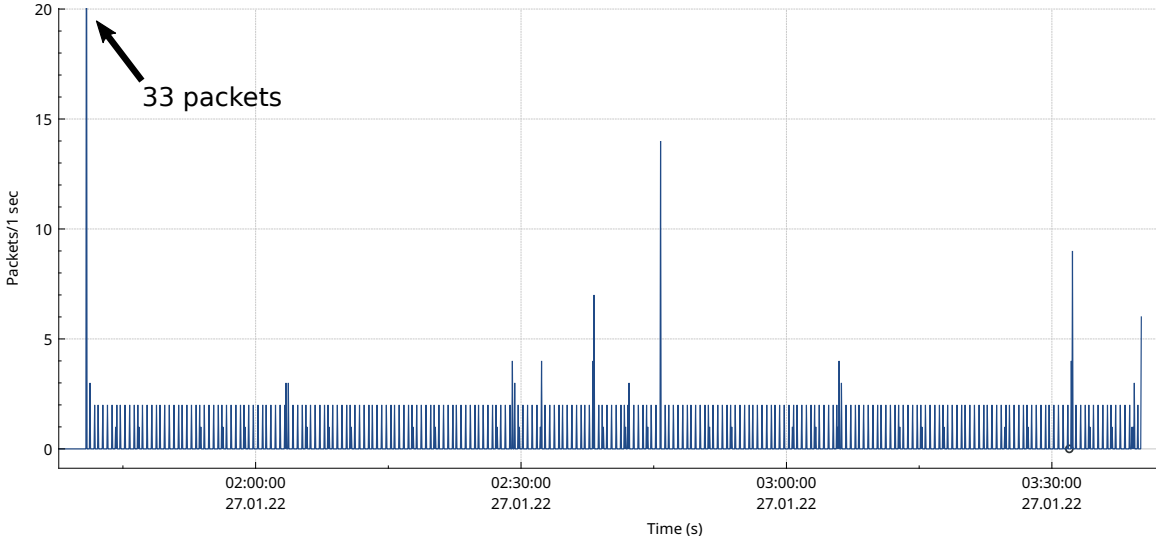
Three distinct streams were communicating between a phone with the Amazon Alexa application and cloud servers. Three of them had already been established when the active capture started. They lasted past the capture time. Only one of them can be also found in the passive capture pcap file. This stream communicated with the avs-alexa-14-na.amazon.com endpoint. A stream in the passive capture pcap was detected, however, it was a different instance, communicating with the same endpoint. It was a long stream, which mainly sent 41 TLS payload bytes both sides every 30 seconds. I/O graph of this stream is shown in Figure 6.5b.

6.3 Google Nest Mini

This section presents results of the traffic analysis for the Google Nest Mini gateway. The initial setup of the device failed with further unspecified error. The device seemed to be working as intended, with two caveats noted. First, the device setting in the Google Home application was blank. Second, the connected Hue light bulb could not be operated via voice commands. The Hue light bulb could, however, be operated by using the Google Home application. Other voice commands (asking for current time, etc.) were working. The device setup was successful after the factory reset. The pcap, captured while the Google Nest was in this partly functioning state, was not part of the analysis.



(a) An I/O graph for the TCP/TLS stream from the Amazon Echo towards the avs-alexa-14-na.amazon.com endpoint. The initial spike consists of a TLS handshake, where a certificate makes the most of the data. Besides the occasional spikes, the conversation is mostly idle. This stream continues past the time scope of the active capture and is detected in the passive capture pcap file. It ends on 26. 01. 2022, 11:33:58 CET.



(b) Stream with the same endpoint found in the passive capture file. The number of sent packets in the span of a second is shown in this Figure instead of a number of bytes. It shows similar spikes as the connection from 6.5a, thus confirming the hypothesis that these different connections to the same endpoint are almost identical concerning the amount of sent data.

Figure 6.5: Comparison of the Amazon Echo's streams directed to the same endpoint, each captured in different capture mode.

Passive Capture Results

Google Nest sent 33887 DNS queries, divided into twenty DNS distinct queries for A or AAAA records, one PTR request and five different multicast DNS queries for discovering services and devices in the LAN. All of these queries are shown in Table 6.3. Intervals with the value 00:00:00 are caused either by sending two exact queries directly after each other or by sending queries for A and AAAA records simultaneously. As seen in Figure 6.6, the purpose of the endpoint cannot be determined solely by its IP address, nor does it work the other way. One server might host multiple services, each accessible via a different domain name.

All queries, but few exceptions, were directed to Google's DNS—8.8.8.8. These exceptions occurred when the device could not get an answer from 8.8.8.8. It then sent the request to the default gateway—the Turris router. The number of queries to 191.168.1.1 was single-digit for each distinct endpoint, even when the total number of queries for the endpoint was larger than a thousand. There were 419 requests for the PTR record of 110.37.251.142.in-addr.arpa. The answer was always prg03s13-in-f14.1e100.net, Google's server in Prague, Czechia (the 1e100 domain being the scientific notation of a googol).

The DNS query for an A record of dl.google.com was sent on the second day of capturing at 03:10:44 CET. TCP/TLS stream followed. The endpoint is Google's HTTP download server [18] serving Chrome, Earth, Android SDK, et cetera.

From the DNS communication, it can be said that Google uses the following networks for hosting their services:

- 108.177.0.0/16
- 142.250.0.0/16
- 142.251.0.0/16
- 172.217.23.0/24
- 216.58.0.0/16
- 216.239.0.0/16
- 2a00:1450:4014:800::
- 2001:4860:4806::

Google Nest used NTP for the time synchronisation. Endpoints for NTP communication were time[0-4].google.com. NTP synchronisation occurred 674 times. Endpoints are not changing sequentially based on time. The minimal interval between NTP streams is 00:01:44, while the maximal interval is 00:19:58. Maximal interval corresponded with DNS queries for time.google.com. There is one DNS query before every NTP stream.

The gateway periodically checks for connectivity to the default gateway (9653 times) and DNS server (9748 times). These ICMP echo messages are sent in batches of two requests for each endpoint, with the interval between batches never being longer than 00:03:07.

Interesting conversations happened on the UDP port 10101—serving as both source and destination port. Endpoints were multicast IP addresses 224.0.0.250 and 239.255.255.251. This stream ran 13638 times. Sometimes, two or three packets with the interval of 1 minute were aggregated to one stream. The final number of connections is, therefore, bigger than 13638. Every packet carried 36 B of UDP payload. The payload was the same for all packets—0a2033393131424534424638353045454532463931354233373735303942333831391001 in hexadecimal. The minimal interval between connections was 00:00:59, while the maximal interval was 00:06:00. Time-to-live of these packets was four. If this was a TCP connection, this connection could suggest the infection by BrainSpy and Silencer trojans that operated on the TCP port 10101 [43]—the same port as the Google Nest uses.

The Nest gateway communicated with three endpoints via plain HTTP. The first connection occurred seven times towards the http://clients1.google.com/generate_204 URI. The conversation is always the same. The second stream arose just once and lasted 242

Query	Amount	Min. Interval	Max. Interval
110.37.251.142.in-addr.arpa	419	00:00:00	03:59:53
android.clients.google.com	3	00:00:00	48:00:00
clients1.google.com	7	21:50:54	25:24:19
clients3.google.com	508	00:00:44	00:39:34
clients4.google.com	2027	00:00:01	00:10:00
clientservices.googleapis.com	3	04:29:24	10:08:23
connectivitycheck.gstatic.com	255	00:00:00	01:01:13
device-provisioning.googleapis.com	3	48:00:00	48:00:00
dl.google.com	1	-	-
fcm.googleapis.com	325	00:00:45	00:39:59
geller-pa.googleapis.com	452	00:00:00	05:59:58
google.com	46	00:16:32	04:00:01
home-devices.googleapis.com	1	-	-
lycraservice-pa.googleapis.com	34	00:00:00	00:42:47
mtalk.google.com	36	00:00:05	23:56:03
play.googleapis.com	2148	00:00:00	01:05:00
time.google.com	1350	00:00:00	00:19:58
tools.google.com	434	00:00:00	00:35:54
www.google.com	19604	00:00:00	00:02:13
www.googleapis.com	46	00:16:35	03:59:59
www.gstatic.com	9	00:00:00	07:53:23
_8e6c866d._sub._googlecast._tcp.local	476	00:00:00	21:30:44
_googlecast._tcp.local	1686	00:00:00	24:46:44
google-nest-mini-6f8186e66483e4ec33a41a91ead50751._googlecast._tcp.local	6	00:00:00	03:04:18
6f8186e6-6483-e4ec-33a4-1a91ead50751.local	90	00:00:00	00:46:44
_services._dns-sd._udp.local	88	00:00:00	00:05:01

Table 6.3: The Table shows all the DNS queries that Google Nest asked for during the one week of capturing. The query on the first line is for the PTR record. DNS requests in the middle sections are for the A or AAAA records. The last section contains multicast DNS queries used to discover services and devices on the local network.

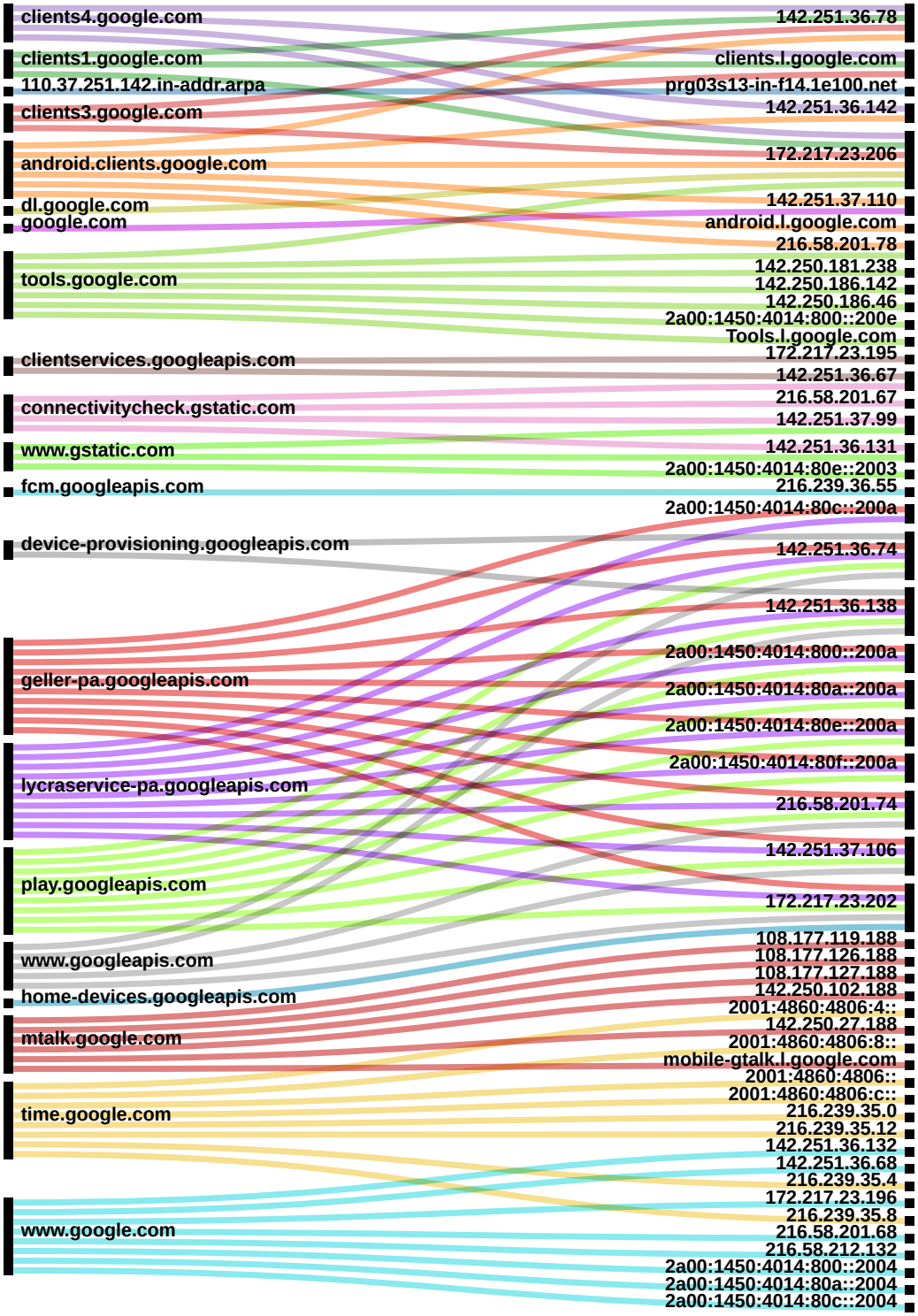


Figure 6.6: The graph demonstrates the overlaps of DNS queries (on the left) and answers for these queries (on the right). This demonstrates that the destination of the traffic cannot be determined solely by the destination IP address because several endpoints may be covered by one IP address—the server is running multiple services, each hosted under a different domain name.

Destination IP	Destination endpoint	Client hello packets
142.251.36.138	play.googleapis.com	866
	www.googleapis.com	11
142.251.36.74	play.googleapis.com	710
	www.googleapis.com	14
142.251.37.106	play.googleapis.com	757
	www.googleapis.com	14
172.217.23.202	play.googleapis.com	297
	www.googleapis.com	22
216.58.201.74	play.googleapis.com	334
	www.googleapis.com	7

Table 6.4: The Table demonstrates the problem of associating an IP address with an endpoint using the QUIC protocol. The Table contains data identified as belonging to one type of stream/conversation. The right-most column is populated by numbers of QUIC client hello packets, identifying the beginning of a single QUIC conversation.

seconds. It used HTTP POST request to http://clientservices.googleapis.com/uma/v2; media type was application/vnd.chrome.uma (4324 bytes). The last HTTP stream repeated 188 times, and it reached connectivitycheck.gstatic.com endpoint. Requests were usually generated after 20 seconds. The URI always contained the 'generate_204' string, similar to the first HTTP communication mentioned.

Google Nest was the only tested gateway that used the QUIC protocol. Six different communication groups, utilising the QUIC protocol, were observed. QUIC (Quick UDP Internet Connections) protocol was developed by Google, and it is meant to reduce overhead and latency of TCP, while keeping its benefits. It is trying to implement TCP, TLS and HTTP on UDP [22].

The first QUIC stream was directed toward the tools.google.com endpoint. Out of the 407 occurrences, 406 had the 216.58.201.78 IP address as their destination. The typical conversation consisted of client hello → rejection → client hello → connection. It was protected using TLS. The authentication algorithm used was AES-GCM, and the Curve25519 was used for key exchange. Client hello and server handshake packets were padded to 1392 B (frame length) or 1358 B (UDP payload). The next stream, which repeated 92 times, had www.google.com as its endpoint. Again, the client hello and server handshake packets were padded to 1392 B (frame length) or 1358 B (UDP payload). The same cipher suite was used. Most frames sent from Google Nest were 75 B or 1392 B long (frame length)—33 B or 1350 B of payload—471/555 frames (84.86 %). Intervals ranged from 00:01:29 to 11:10:03.

The rest of the QUIC streams had the same characteristics as the aforementioned did (padding, payload lengths, et cetera.). Sorting them based on the endpoint rendered difficult for the DNS answers contained IP addresses, which belonged to more than one endpoint. This is best demonstrated by data contained in Table 6.4. Four remaining distinguishable QUIC streams demonstrate the same behaviour.

Google Nest Mini communicated via 11 different TCP/TLS streams. In all of them, the client and server negotiated to use the TLS_AES_128_GCM_SHA256 cipher with the x25519 curve. They always used TLS 1.3. While the average stream duration, when using the QUIC protocol, was less than 1 second, communications via TLS were much

longer—2113.5 seconds on average. All TLS client hello packets were padded to have 512 B or 585 B of TLS payload.

The longest streams took 19901 seconds on average. They were directed to endpoints obtained by DNS queries for mtalk.google.com. Most of the streams were comprised of TCP keep-alive packets.

Twenty-eight streams were directed to www.gstatic.com endpoint. Most data originates from the endpoint, which supports the fact that www.gstatic.com is part of Google's CDN [23].

Communications between the Google Nest and the 216.58.201.74 IP address were showing similar properties as the QUIC based stream in Table 6.4. Out of 29 connections, two were for play.googleapis.com, a single one for www.googleapis.com, while the rest—27 connections—reached geller-pa.googleapis.com. The amount of TCP non-overhead data was similar: from 3718 to 3767 bytes outbound. Inbound data created clusters from around 6100 B to 7400 B, and from 9159 B to 9220 B. Streams to different endpoints showed noteworthy differences in stream duration—from less than a second, via 243 seconds, to a duration of 1849 seconds.

Active Capture Results

During the active capturing, which lasted 40 seconds, 4 DNS queries were detected. Two of them asked for the A record of www.google.com, while the other two queried for AAAA records of the same endpoint.

Two ICMP streams were captured. They share the same traits as the ICMP bursts described in the section about the results of the passive capturing 6.3.

There were three TCP/TLS conversations, one using UDP/QUIC and one using HTTP. All of them were also found in the pcap file of the passive capturing. The HTTP conversation is equal to the passive conversation, which repeatedly sent out the same hexadecimal strings and ran on the UDP port 10101. In the pcap file, there is no hint suggesting a possible conversation with the cloud regarding the operation of the Hue light bulbs.

6.4 Raspberry Pi with Home Assistant

Analysis results for the Home Assistant software gateway are contained in this section. Home Assistant has been installed as a standalone operating system on a Raspberry Pi computer. It differs from the other tested gateways in its form—software against hardware gateway, and the type of software philosophy—Home Assistant is an open-sourced solution. As was the case of the other gateways, anonymous telemetry data collection was enabled.

Passive Capture Results

The Home Assistant sent A or AAAA DNS queries for six endpoints. This is the second-lowest number, after Aeotec, in tested gateways. However, three of these were sent 30 times in total, which brings it the total of A or AAAA queries to a lower number than Aeotec gateway had. These queries can be seen, among others, in Table 6.5.

Querying for an IP record of cognito-idp.us-east-1.amazonaws.com was made to gain a way to authenticate itself towards Amazon's server. Quote Amazon AWS, [4]: "Amazon Cognito lets you easily add user sign-up and authentication to your mobile and web apps. Amazon Cognito also enables you to authenticate users through an external iden-

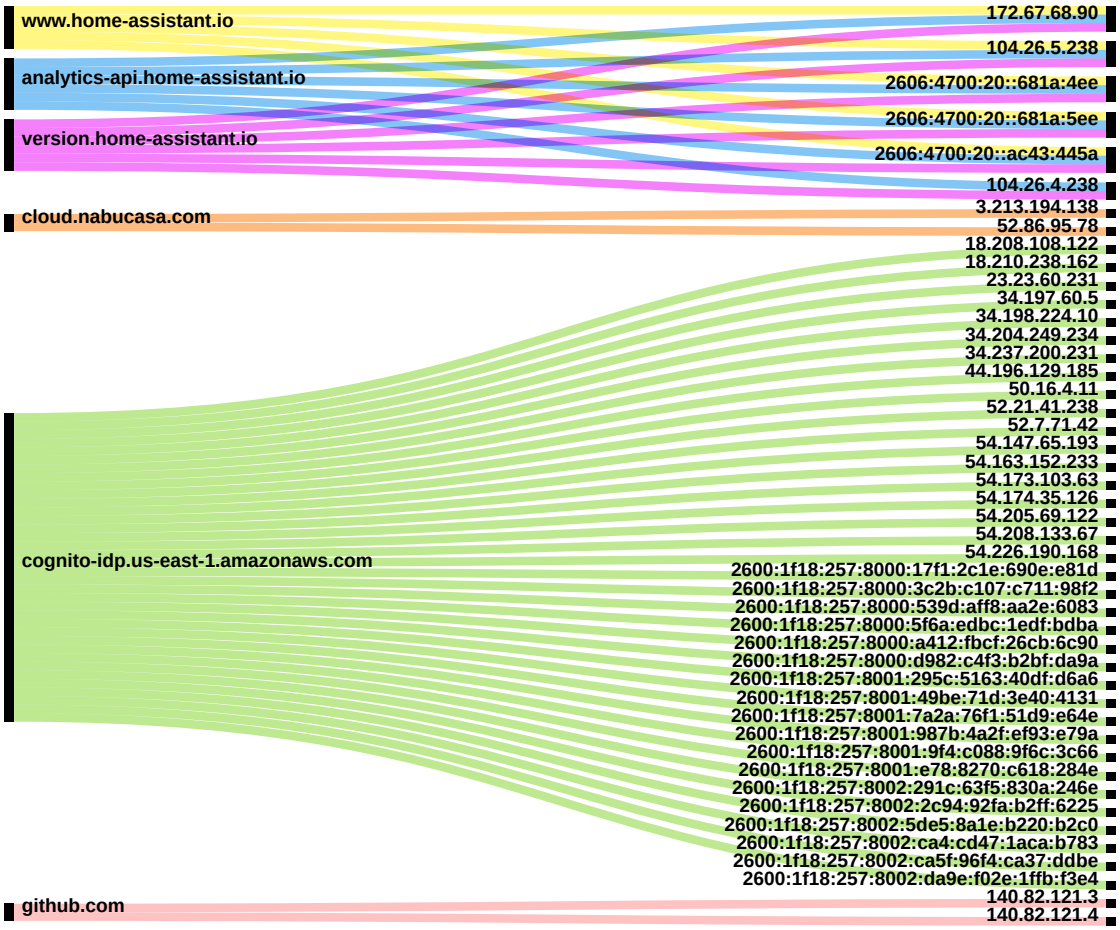


Figure 6.7: Diagram showing the running of multiple services on one endpoint—all three domain names under the *.home-assistant.io wildcard domain share the same IP address.

tity provider and provides temporary security credentials to access your app's backend resources in AWS or any service behind Amazon API Gateway." This is presumably used to get to cloud.nabucasa.com, another queried name, which provides cloud solutions for Home Assistant and is served by Amazon AWS.

DNS queries for www.home-assistant.io and analytics-api.home-assistant.io were answered with the same range of IP addresses. This is similar behaviour to the one observed in Google Nest's results, and it is further demonstrated in Figure 6.7. These requests always directly followed each other. Since these queries were sent so close to each other, they used the same UDP source port. Together with a query for version.home-assistant.io, both of these queries were answered with the same list of IP addresses. Answers for these queries differed from other ones by having the DNSSEC DO bit[1] cleared explicitly to zero.

An intriguing observation not experienced elsewhere was scanning the local area network using the PTR DNS record. The Home Assistant sent batches of 254 PTR queries for the entire network with a /24 prefix every hour. Besides this, the Home Assistant scanned for services running on the network via multicast DNS. This is the only time when the traffic from the Home Assistant was sent by an interface with an IPv6 address.

[1]DO bit indicates if the DNS resolver is able to accept DNSSEC security resource requests [9].

Query	A	AAAA	Min. Interval	Max. Interval
www.home-assistant.io	7	7	00:00:00	24:00:00
analytics-api.home-assistant.io	7	7	00:00:00	24:00:00
cloud.nabucasa.com	2	0	00:00:00	00:00:00
cognito-idp.us-east-1.amazonaws.com	180	180	00:00:00	01:00:02
github.com	214	213	00:00:00	03:00:02
version.home-assistant.io	9048	6461	00:00:00	00:10:00

Query	From IPv4	From IPv6	Min. Interval	Max. Interval
<0-255>.1.168.192.in-addr.arpa.	168	-	00:59:59	01:00:00
0abe49165e1940ce9b9f2b398f8e6814.local				
_googlecast._tcp.local _home-assistant._tcp.local _hue._tcp.local _services._dns-sd._udp.local _ssh._tcp.local philips hue – 81cf9e._hue._tcp.local	54197	24768	-	-

Table 6.5: Table showing all the DNS queries demanded by Home Assistant during the passive capturing. The upper section shows all the queries for A and AAAA DNS records—numbers in those columns represent the amount of queries. The second section contains captured queries for MDNS. Each hour, the Home Assistant completed a scan of the entire LAN—with queries starting from 1.1.168.192.in-addr.arpa, through every available address from 192.168.1.0/24 network, to the last address (192.168.1.254). Numbers in the Table for this row represent the amount of sent batches—to get the total amount, it is necessary to multiply this number by 254, which is 42672.

Home Assistant used Cloudflare NTP servers to get current time information. The endpoint, to which it connected 295 (interval between streams was always 00:34:08), was time.cloudflare.com.

There were 2889 HTTP communications towards Cloudflare hosted endpoints without registered domain names. IP addresses were obtained in the DNS answers for version.home-assistant.io. The HTTP request was always for http://version.home-assistant.io/online.txt.

Ten thousand and seventy-six SSDP streams were captured. Half of them were directed towards the LAN's broadcast IP address, while the other was for the 239.255.255.250 multicast address. These happened every two hours.

TLS stream towards version.home-assistant.io ran 1040 times. 1037 out of 1040 streams sent 822 TLS bytes outbound. All its client hello packets had 517 TLS bytes, and all server hello packets only (without changing the cipher spec and data) had 127 TLS bytes. After the TLS handshake, streams were composed of three other packets. These were: Home Assistant outbound – 201 B of TLS data; Home Assistant inbound – 982 B—1889 B of TLS data (majority of packets had 1476-1498 B of TLS payload); Home Assistant outbound – 24 B of TLS data. The cipher used was TLS_AES_256_GCM_SHA384, together with the x25519 curve. Intervals between streams were not longer than 00:10:41.

The Home Assistant connected to github.com 112 times, always with two TLS streams starting simultaneously. The interval between these streams always hovered around the 3-hour mark. It used the TLS_AES_128_GCM_SHA256 cipher, unlike the other TLS streams originating from the Home Assistant, which all used the TLS_AES_256_GCM-_SHA384 cipher suite

A stream, which spanned across the entire week, was connected to 18.193.141.36. This IP address was not found in any DNS answers. It is serviced by Amazon AWS. Predominantly, it consisted of bursts of two packets—32 bytes of TCP payload outbound with TCP PSH flag set and 32 bytes of TCP payload inbound. Sporadically, these were interrupted by a SSLv2.0 packet.

Fourteen TLS communications were directed towards the endpoint with the 172.67.68.90 IP address. Depending on the destination domain name, the stream lasted on average either 15.4 s (www.home-assistant.io) or 30.4 s (analytics-api.home-assistant.io). The domain name also influenced the number of total bytes sent and received-808 B (www.home-assistant.io) or 2113 B (analytics-api.home-assistant.io) outbound and 4906 B (www.home-assistant.io) or 3836 B (analytics-api.home-assistant.io) inbound.

Communications to cognito-idp.us-east-1.amazonaws.com arose after each successful DNS query for the domain name. These streams had the following TLS payload: either 235 B outbound, 1165 B inbound, or 2218 B outbound, 2384 B inbound.

Conversations with the cloud.nabucasa.com ran during the whole week, divided into two distinct TCP/TLS streams. The first stream started before the capturing had begun, and it ran until February 18th, 20:20:03 CET. The second, ensuing, conversation began on the same day, at 20:20:37 CET and lasted past the end of packet sniffing. The two streams manifested vastly different behaviour, as demonstrated in Figure 6.8, although they communicated with the same endpoint.

Active Capture Results

During the span of 41 seconds of active capturing, no DNS query for A or AAAA record was found. The only DNS queries captured were the multicast DNS questions for discovering services running in the LAN.

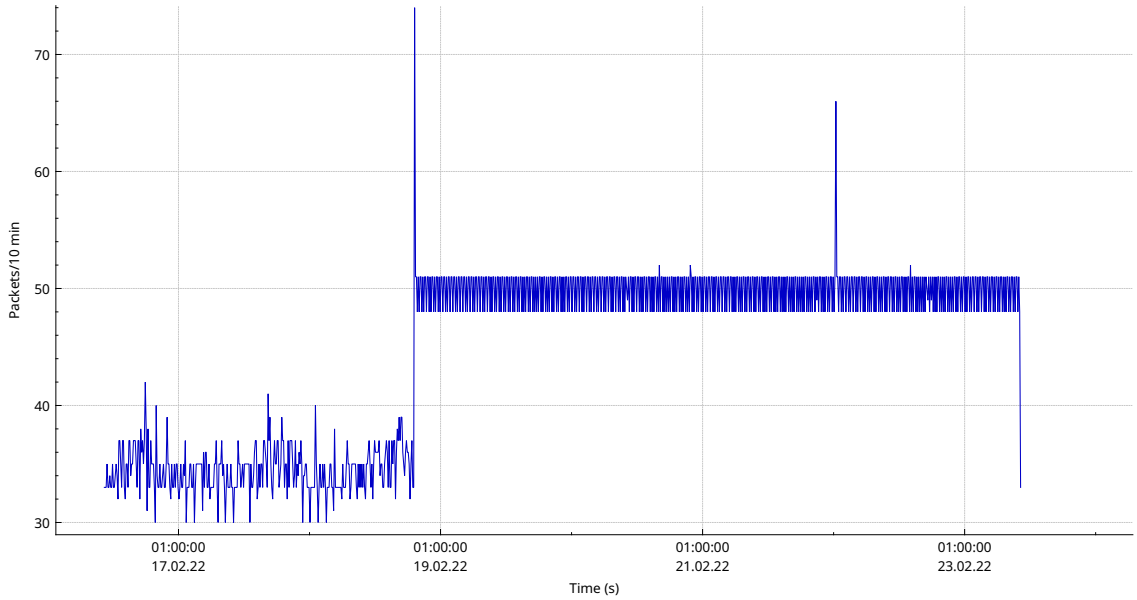


Figure 6.8: I/O graph of the two streams from Home Assistant towards the cloud.nabucasa.com endpoint shows two distinct natures of two streams, which ran shortly after each other It demonstrates, that communication with a single destination does not automatically result in the same conversation properties.

Three streams, which were also detected in the passive capturing (more in section 6.4), were found. These are communicating with the following endpoints:

- cloud.nabusaca.com
- ec2-18-193-141-36.eu-central-1.compute.amazonaws.com
- 172.65.32.248

The Home Assistant did not communicate with the cloud when operating the light bulb. However, a conversation with a controlling smartphone was observed. The start and end of this TCP/TLS stream were not captured. Twenty packets with TCP data, either 79 B or 80 B in length, were sent from the phone. The PSH flag was set. These could be the packets transferring requests for light manipulation. There was no observable difference between turning the lights on or off (from the side of a smartphone as a packet source). Fifty-seven packets with TCP ACK and TCP PSH flags were sent back to the smartphone. Responses were either 2, 3 or 4 packets long. The lengths of these packets were not unified. The lengths were contained in an interval of 13 B and 105 B of TCP payload. The first response packet usually arrived 50-60 ms after the request, with the next packet being sent from the endpoint after the next 20 ms. In the case of the fourth packet being sent, the interval after which it was sent depended on the type of action taken. For the “turn the light on” operation, the fourth packet was sent approximately 1 second after the third. In case of turning the light off, the fourth packet followed immediately after the third one.

Gateway	Packet Count	Captured Bytes	Avg. Data Rate [B/s]	Avg. Packet Rate [p/s]	Avg. Packet Size [B]
Aeotec Hub	525089	89886558	146	0.9	171
Amazon Echo	525888	223910138	370	0.9	426
Google Nest Mini	507148	114586759	189	0.8	226
Home Assistant	719611	76087337	125	1.2	106

Table 6.6: The Table shows the global statistics of captured files for each tested gateway.

6.5 Global results

In total, the capturing took place for a total of 28 days. During this time, more than 4080 MiB in 6579433 datagrams of raw data was captured, which was later filtered to 516 MiB and 2278689 packets of data used for the analysis. Most filtered data was sent and received by Amazon Echo-221.6 MiB, with Google Nest Mini coming second with 117.0 MiB of data. The Aeotec gateway transmitted 93.7 MiB of filtered data. Raspberry Pi came last, with 83.5 MiB of data sent or received. These statistics and other properties of captured files are manifested in Table 6.6.

Figures 6.9 and 6.10 show the total amount of data transmitted by transport layer protocol (6.9) and application layer protocol (6.10), both for IPv6 and IPv6. Data, of which these Figures are composed of, were obtained from the *IP.len* field of a record for a single datagram contained in a pcap file; or *IPv6.plen* field for IPv6, respectively. This decision was taken in order to make results comparable to the results in [2]. The script used for data extraction is located within the published dataset. It clearly demonstrates the differences in the amount of sent or received data and how each gateway uses different ratios of protocols for its communication. The usage of Google's QUIC protocol, based on UDP, is noticeably visible. All traffic data, including data sent via protocols, which are not clearly visible in graphs in Figures 6.9 and 6.10 due to making only a small portion of the overall traffic, are shown in Table 6.7 for IPv4 traffic and Table 6.8 for IPv6, respectively.

6.6 Learned Information and Possible Attacks Discussion

This traffic analysis showed how much information can be obtained from the traffic, even when it is encrypted using modern ciphers. **With the further use of information discovered in this thesis, it could be probably possible to identify the devices running inside the network (device fingerprinting)**, even behind the scope of the local network. Using modern pattern recognition software and other similar methods, the following categories or properties of the communication analysed in this thesis could be used as the main differentiators between each gateway: DNS queries, hostnames and addresses with which the gateways communicated, patterns in time intervals between streams towards the same endpoint, patterns found in the length of packets within streams, et cetera. The carried-out analysis already identified these traffic features. The fingerprinting of the

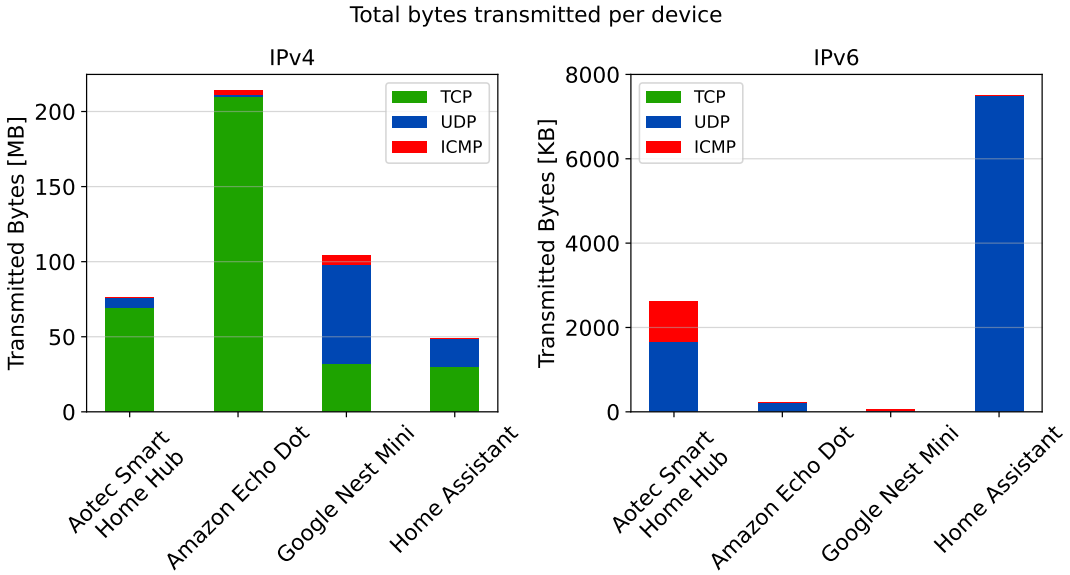


Figure 6.9: The Figure presents the total amount of transferred data to and from the gateways. It portrays the results divided into transport layer protocols and ICMP, both for IPv4 (left) and IPv6 traffic (right).

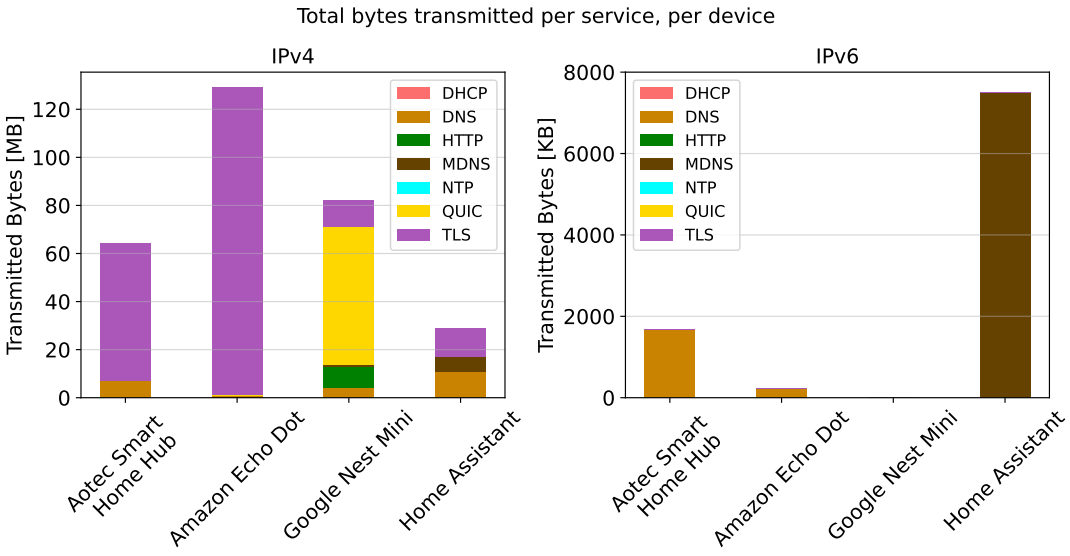


Figure 6.10: The Figure manifests the total amount of transferred data. It divides the data based on the services provided by various application layer protocols, both for IPv4 (left) and IPv6 traffic (right). The lower amount of total data transmitted is caused by script (total_data_sent.sh – available within the dataset), which used Tshark and BPF filters to obtain the amount of bytes for each service. The filter 'IP and TLS', for example, did filter out all the TCP overhead, which was sent when TLS was used, therefore the total amount of bytes is lower than in Figure 6.9.

Gateway	DHCP	DNS	Plain HTTP	NTP	QUIC	TLS version	TLS data
Aeotec Hub	25536	7259768	0	0	0	1.2, 1.3	57165608
Amazon Echo	21842	1187511	748754	4712	0	1.2, 1.3	128958289
Google Nest	573344	4155497	9086641	102448	57947054	1.2, 1.3	11552892
Home Assistant	19800	11419042	0	44840	0	1.2, 1.3	12508143

Table 6.7: The Table manifests the number of bytes sent and received by each IoT gateway when communicated using the IPv4 protocol.

Gateway	DHCP	DNS	Plain HTTP	NTP	QUIC	TLS version	TLS data
Aeotec Hub	0	1671997	0	0	0	-	0
Amazon Echo	0	223027	0	0	0	-	0
Google Nest	0	0	0	0	0	-	0
Home Assistant	7470	0	0	0	0	-	0

Table 6.8: The Table demonstrates the number of bytes sent and received by each IoT gateway when communicated using the IPv6 protocol.

network connected devices is valuable primarily since the MAC address of a device—the primary identifier—is lost at the network's border.

Open DNS communication shows that the use of this protocol has its security shortcomings, especially in giving away valuable information about domains searched for and probably later used by a user. Captured DNS records can serve as the basis for the **DNS cache poisoning attack (also known as DNS spoofing)** and other man-in-the-middle attacks. The emerging use of DNS over HTTPS, DNS over TLS or DNSCrypt could mitigate this problem by obscuring the DNS data carried over the network. This does not solve the problem in its entirety because of the recursive nature of DNS servers. While the traffic towards the first DNS server might be encrypted, the given server might not use the encrypted communication when obtaining answers from other DNS servers. However, it is improbable that the potential attacker would be sniffing the traffic in a section of a network where the plain DNS is being utilised.

Most of the data was sent via TCP/TLS or QUIC protocol. Therefore, it was encrypted, which is certainly a good indication that manufacturers of tested gateways participate in the current urge to protect private data. Every device used both TLS 1.2 and 1.3. The Curve25519 elliptic curve was the single most used curve for establishing a ciphered connection, while Diffie-Hellman's algorithm was used to

Gateway	Cipher
Aeotec Hub	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
Amazon Echo	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 TLS_AES_128_GCM_SHA256 TLS_RSA_WITH_AES_128_CBC_SHA
Google Nest	TLS_AES_128_GCM_SHA256 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
Home Assistant	TLS_AES_128_GCM_SHA256 TLS_AES_256_GCM_SHA384 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

Table 6.9: The Table demonstrated different cipher suites used by each of the tested IoT gateway.

exchange keys. The ciphers used for TLS encryption by each gateway are shown in Table 6.9.

The total amount of data sent towards the cloud is particularly concerning, especially when the device should be idle. In the case of the Home Assistant software gateway, this could be remedied by disagreeing with sending anonymous telemetry data to the cloud. However, this solution is severely limited, or even not possible at all, with other gateways, because a user often cannot opt out from sending such data when using the closed-sourced gateways. The discovery of a high, even alarming, amount of data being sent to the Cloud, even when the gateway should not be exercising any signs of communication, is in line with the expectations of the author before the analysis process—it was one of the important motivators why the selected gateways were tested. **The analysis proved the author's hypothesis, that commercial, closed-sourced, gateways developed by large corporates (Amazon Echo and Google Nest) would send more diagnostic, and other data towards the Cloud.** The use of encrypted communication obscures the concrete evidence of what data are being sent towards the Cloud. The fact, that these commercial gateways connect to more endpoint is also related to the aforementioned issue, and it is caused by the existence of large infrastructure of the manufacturing companies.

Results Comparison to the Related Studies

The methodology of capturing and analysis process of this work was based on the one of Amar et al. [2]. This was done to achieve the most comparable results. From the data of Amazon Echo, it can be deduced that **traffic in this thesis and Amar's paper differ, yet they share certain similarities in other areas**. The cause for this could be either using a different version of the Amazon Echo device, different settings of the device, or simply the time difference between the two studies—during which Amazon may have changed internal processes. The time interval between the publishing of Amar's paper and this thesis is the most probable explanation, why the traffic is so different, especially given the fast-paced environment of research, development, and business in the area of IoT gateways in past years. **The conversations of Amazon Echo in this research were mainly composed of TLS** traffic (see Figure 6.10), while Amar showed that their Echo also communicated via ICMP and other protocols. The Echo in this research did not send as many DNS requests, and especially it did not send any requests for DNS resource records such as www.example.net, compared to Amar's work. Also, NTP connections happened

less often in this work than in the paper of Amar. **Between publishing Amar's work and this thesis, the general use of TLS rose. This is an expected, but positive discovery**; however, the low number of tested devices could skew the result, and it may not represent the situation in the entire IoT segment. This work differs from Amar's mainly in the type of devices tested—this study is concerned with IoT gateways only. Also, the results of this thesis emphasise more detailed characteristics of streams than Amar. **The analysis in this thesis extended the span of scenarios of the Amar's paper, when the data gathered in the active capturing mode were analysed.**

The dataset collected in this work is similar to the one featured in the paper of Ivan Cvitić et al. [11]. It could be further used to validate their research if the same process was carried out on the dataset. The work was done the same way or similarly until the point following the extraction of stream data. Different tools were used for the part of streams data extraction. The main drawback limiting the reproducibility of Cvitić's classification with the dataset collected for this work is the small pool of tested gateways.

Junges et al. [26] used traffic flow data to create a tool that can identify user taken IoT actions inside a network with 98.4% accuracy. The tool relies on several assumptions and solutions to challenges. **Their assumption that gateways send data after command execution is not always correct, as the results of this thesis manifest. The Amazon Echo and Google Nest Mini do not indicate that traffic is generated when operating a light bulb. Therefore, the tool created by Junges et al. would be unusable for these gateways**.

Chapter 7

Conclusion

In this thesis, an analysis of the network traffic of several IoT gateways intended for home use was conducted. This analysis demonstrated that valuable information about the devices running inside a local network can be obtained by observing traffic leaving and entering the network. This includes the theoretical possibility of fingerprinting of the devices inside a network based on the traffic information gathered in the analysis. Most information can be obtained from DNS queries and answers. Due to the extensive use of TLS encryption for HTTPS traffic, traffic information can be extracted from parameters such as packets lengths, patterns found in traffic, destination addresses and their DNS records, et cetera. However, the use of TLS is an excellent sign in the grand scheme of things, meaning that manufacturers are not underestimating the importance of data security and confidentiality. This thesis showed, how fast the parameters of communication (of the same device) are changing these days. It also found imperfections in the existing literature concerning the topic of IoT gateways. The thesis confirmed the author's assumption that commercial, close-sourced gateways produce, on average, more traffic than open-sourced counter-parts.

The dataset, consisting of captured traffic, is openly shared and available at https://nextcloud.fit.vutbr.cz/s/TTtEqT8wJLwDAs4. Its use for future research is another contribution of this thesis. Future research, consisting of machine learning, neural networks or other methods of finding patterns, can be trained and used on this dataset to find or confirm the emerging data patterns presented as the results of this thesis, thus creating a traffic fingerprint of the gateways.

Bibliography

[1] AAEON. *AIOT-IGWS01* [online]. 2022 [cit. 2022-01-15]. Available at: https://www.aaeon.com/ru/p/iot-gateway-systems-aiot-igws01/.

[2] Amar, Y., Haddadi, H., Mortier, R., Brown, A., Colley, J. A. et al. An Analysis of Home IoT Network Traffic and Behaviour. *ArXiv*. march 2018, abs/1803.05368.

[3] Amazon Web Services, Inc.. *Amazon CloudFront* [online]. 2022 [cit. 2021-03-05]. Available at: https://aws.amazon.com/cloudfront/.

[4] Amazon Web Services, Inc.. *Amazon Cognito* [online]. 2022 [cit. 2021-03-11]. Available at: https://aws.amazon.com/cognito/.

[5] Amazon Web Services, Inc.. *Amazon EC2* [online]. 2022 [cit. 2021-03-05]. Available at: https://aws.amazon.com/ec2/.

[6] Amazon.com, Inc.. *Overview of Amazon Device Messaging* [online]. 2022 [cit. 2021-03-06]. Available at: https://developer.amazon.com/docs/adm/overview.html.

[7] Cisco Engineers. *Embedded Packet Capture for Cisco IOS and IOS-XE Configuration Example* [online]. Cisco Systems, Inc., august 2016 [cit. 2021-10-02]. Available at: https://www.cisco.com/c/en/us/support/docs/ios-nx-os-software/ios-embedded-packet-capture/116045-productconfig-epc-00.html.

[8] Combs, G. et al. *Wireshark* [online]. 2022 [cit. 2021-12-21]. Available at: https://www.wireshark.org.

[9] Conrad, D. *Indicating Resolver Support of DNSSEC* [Internet Requests for Comments]. RFC 3225. RFC Editor, December 2001.

[10] Cvitić, I., Peraković, D., Periša, M. and Botica, M. Smart Home IoT Traffic Characteristics as a Basis for DDoS Traffic Detection. In: Lucia, K., Dragan, P. and Marko, P., ed. *3rd EAI International Conference on Management of*. EAI, December 2018. DOI: 10.4108/eai.6-11-2018.2279336. ISBN 978-1-63190-167-6.

[11] Cvitić, I., Peraković, D., Periša, M. and Botica, M. Definition of the IoT Device Classes Based on Network Traffic Flow Features. In: Springer Nature Switzerland AG. *4th EAI International Conference on Management of Manufacturing Systems*. 1st ed. January 2020, p. 1–17. DOI: 10.1007/978-3-030-34272-2_1. ISBN 978-3-030-34271-5.

[12] CZ.NIC, z. s. p. o.. *CZ.NIC* [online]. 2022 [cit. 2021-12-23]. Available at: https://www.nic.cz.

[13] CZ.NIC, Z. S. P. O.. *Turris MOX* [online]. 2022 [cit. 2021-12-23]. Available at: https://www.turris.cz/cs/mox/.

[14] Datadog, Inc.. *Datadog* [online]. 2021 [cit. 2021-12-23]. Available at: https://www.datadoghq.com.

[15] Day, J. and Zimmermann, H. The OSI reference model. *Proceedings of the IEEE*. january 1984, vol. 71, no. 12, p. 1334 – 1340. DOI: 10.1109/PROC.1983.12775.

[16] Elasticsearch B.V.. *Kibana* [online]. 2022 [cit. 2021-12-23]. Available at: https://www.elastic.co/kibana/.

[17] Felfernig, A., Polat Erdeniz, S., Uran, C., Reiterer, S., Atas, M. et al. An overview of recommender systems in the internet of things. *Journal of Intelligent Information Systems*. 1st ed. april 2019, vol. 52, no. 2. DOI: 10.1007/s10844-018-0530-7.

[18] Fitzpatrick, B. *dl.google.com: Powered by Go* [online]. Google LLC, july 2013 [cit. 202-03-07]. Available at: https://talks.golang.org/2013/oscon-dl.slide#1.

[19] Gigamon. *Network TAPs* [online]. Gigamon, 2021 [cit. 2021-12-19]. Available at: https://www.gigamon.com/products/access-traffic/network-taps.html.

[20] Grafana Labs. *Grafana* [online]. 2022 [cit. 2021-12-23]. Available at: https://grafana.com.

[21] Gregersen, C. *A Complete Guide to IoT Protocols & Standards In 2021* [online]. 2020 [cit. 2022-01-15]. Available at: https://www.nabto.com/guide-iot-protocols-standards/.

[22] Hamilton, R., Iyengar, J., Swett, I. and Wilk, A. *QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2*. Internet-Draft draft-hamilton-early-deployment-quic-00. Internet Engineering Task Force, july 2016. Work in Progress. Available at: https://datatracker.ietf.org/doc/html/draft-hamilton-early-deployment-quic-00.

[23] HemantS@TWC. *What is gstatic.com used for? All you need to know!* [online]. The Windows Club, december 2021 [cit. 2022-03-09]. Available at: https://www.thewindowsclub.com/what-is-gstatic-com-used-for-all-you-need-to-know.

[24] Jacobs, R. *TSHARK.DEV* [online]. 2019 [cit. 2021-12-21]. Available at: https://tshark.dev.

[25] Jacobson, V., Leres, C. and McCanne, S. *MAN PAGE OF TCPDUMP* [online]. The Tcpdump Group, november 2021 [cit. 2021-12-21]. Available at: https://www.tcpdump.org/manpages/tcpdump.1.html.

[26] Junges, P.-M., Francois, J. and Festor, O. Passive Inference of User Actions through IoT Gateway Encrypted Traffic Analysis. In: IFIP/IEEE. *IM 2019 - The 16th IFIP/IEEE Symposium on Integrated Network and Service Management*. Washington DC, United States: [b.n.], April 2019. ISBN 978-3-903176-15-7. Available at: https://hal.inria.fr/hal-02331783.

[27] Kismet. *Kismet* [online]. Kismet, december 2021 [cit. 2021-12-21]. Available at: https://www.kismetwireless.net.

[28] Limited, A. *Industrial Automation Solutions* [online]. 2022 [cit. 2022-01-15]. Available at: https://www.arm.com/solutions/industrial.

[29] Ma, H.-D. Internet of Things: Objectives and Scientific Challenges. *Journal of Computer Science and Technology*. 1st ed. Nov 2011, vol. 26, no. 6, p. 919–924. DOI: 10.1007/s11390-011-1189-5. ISSN 1860-4749. Available at: https://doi.org/10.1007/s11390-011-1189-5.

[30] Matoušek, P. *Síťové služby a jejich architektura*. 1st ed. Publishing house of Brno University of Technology VUTIUM, 2014. 396 p. ISBN 978-80-214-3766-1. Available at: https://www.fit.vut.cz/research/publication/10567.

[31] Nadel, B. *Aeotec Smart Home Hub review: The hub that does it all* [online]. 2021 [cit. 2022-01-15]. Available at: https://www.techhive.com/article/3639528/aeotec-smart-home-hub-review.html.

[32] Posey, B. and Lavery, T. *IoT gateway* [online]. TechTarget, april 2021 [cit. 2021-12-21]. Available at: https://internetofthingsagenda.techtarget.com/definition/IoT-gateway.

[33] Prometheus Authors. *Prometheus* [online]. 2014-2022 [cit. 2021-12-23]. Available at: https://prometheus.io.

[34] RAD. *How to Choose the right Industrial IoT Gateway* [online]. 2021 [cit. 2022-01-14]. Available at: https://www.rad.com/how-choose-right-industrial-iot-gateway.

[35] Raspberry Pi Ltd.. *Raspberry Pi* [online]. 2022 [cit. 2022-03-20]. Available at: https://www.raspberrypi.com.

[36] Rawes, E. *Google Nest Mini (2nd Gen) review: Even faster, even smarter* [online]. 2021 [cit. 2022-03-20]. Available at: https://www.digitaltrends.com/smart-home-reviews/nest-mini-review-2/.

[37] Saleem, J., Hammoudeh, M., Raza, U., Adebisi, B. and Ande, R. IoT standardisation: challenges, perspectives and solution. In: Association for Computing Machinery. *ICFNDS '18: Proceedings of the 2nd International Conference on Future Networks and Distributed Systems*. June 2018, p. 1–9. DOI: 10.1145/3231053.3231103. ISBN 978-1-4503-6428-7.

[38] Sanders, C. *Practical Packet Analysis: Using Wireshark to Solve Real-World Network Problems*. 3rd ed. USA: No Starch Press, 2017. ISBN 1593278020.

[39] Segan, S. and Greenwald, W. *Amazon Echo Dot (4th Gen) Review* [online]. 2021 [cit. 2022-01-15]. Available at: https://www.pcmag.com/reviews/amazon-echo-dot-4th-generation.

[40] Signify Holding. *How Philips Hue works* [online]. 2018-2022 [cit. 2021-12-23]. Available at: https://www.philips-hue.com/en-gb/explore-hue/how-it-works.

[41] Signify Holding. *Product Security* [online]. 2018-2022 [cit. 2021-12-23]. Available at: https://www.signify.com/global/product-security.

[42] SolarWinds Worldwide, LLC.. *Network Performance Monitor* [online]. SolarWinds Worldwide, LLC., 2022 [cit. 2022-04-12]. Available at: https://www.solarwinds.com/network-performance-monitor.

[43] Speed Guide, I. *Port 10101 Details* [online]. Speed Guide, Inc., march 2022 [cit. 202-03-07]. Available at: https://www.speedguide.net/port.php?port=10101.

[44] Thales Group. *Bridge the Gap with IoT Gateways* [online]. 2022 [cit. 2022-01-14]. Available at: https://www.thalesgroup.com/en/markets/digital-identity-and-security/iot/inspired/iot-gateway.

[45] The Tcpdump Group. *Home / TCPDUMP & LIBCAP* [online]. The Tcpdump Group, december 2021 [cit. 2021-12-21]. Available at: https://www.tcpdump.org/index.html.

[46] The Tcpdump Group. *MAN PAGE OF PCAP-FILTER* [online]. The Tcpdump Group, february 2021 [cit. 2021-12-21]. Available at: https://www.tcpdump.org/manpages/pcap-filter.7.html.

[47] The Zeek Project. *Zeek Documentation* [online]. 2022 [cit. 2021-12-23]. Available at: https://docs.zeek.org/en/master/index.html.

[48] Wireshark. *Wireshark/epan/dissectors* [online]. Wireshark, december 2021 [cit. 2021-12-22]. Available at: https://github.com/wireshark/wireshark/tree/master/epan/dissectors.

[49] Zbořil, J. *Projekt do předmětu ISA* [online]. 2021 [cit. 2021-12-22]. Available at: https://www.stud.fit.vut.cz/~xzbori20/isa.lua.