



University of
Zurich^{UZH}

Design and Evaluation of a Generic IoT Network Topology Detection and Visualization Approach

*Andrina Vincenz
Zurich, Switzerland
Student ID: 12-608-576*

Supervisor: Advisor 1: **Prof. Dr. Stiller**
Advisor 2: **Katharina Müller**
Date of Submission: October 26, 2023

Zusammenfassung

Diese Arbeit beschäftigt sich mit der Analyse von Netzwerk Topologien mit Fokus auf das Internet of Things (IoT). Das Ziel dieser Arbeit ist die Entwicklung eines ganzheitlichen Analyse- und Visualisierungstool zur Darstellung der Topologie von IoT Geräten. Zunächst wird der Begriff IoT geneauer definiert und verschiedene Netzwerkprotokolle vorgestellt. Zur Validierung des Tools werden verschiedene Experimente durchgeführt. Die Experimente basieren auf Datensätzen, welche den Verkehr von 3 verschiedenen Protokollen aufzeichnen, um einen ganzheitlichen Ansatz zu gewährleisten.

Diese Arbeit zeigt, dass das Visualisierungstool fähig ist verschiedene Protokolle zu verarbeiten und die Topologie zu generieren. Das Tool identifiziert auch bestimmte Objekt Gruppen. Dazu gehören zum Beispiel Geräte, die nur Nachrichten senden, was genutzt werden kann um Geräte Typen zu bestimmen. Eine weitere Funktion vergleicht Netzwerk Topologien auf ihre Mitglieder, womit unter anderem Geräte identifiziert werden, die nicht mit einem böartigen Knoten in Kontakt kamen. Allerdings ist die Qualität der resultierenden Topologie vom Datensatz abhängig, insbesondere dem Vorhandensein von Geräte Adressen. Darüber hinaus gibt es bestimmte Sicherheitsmassnahmen von Firmen, die das Tracking von Geräten erschweren. Ein Beispiel hierfür ist die dynamische Addressierung. Hat das gleiche Gerät mehrere Adressen, wird es mehrmals in der Topologie aufgeführt. Anschliessend werden mögliche Erweiterungen für das Tool vorgeschlagen.

Abstract

This project analyses different network topologies with a focus on Internet of Things (IoT). The aim of this work is the development of a uniform analysis and visualization tool to represent the topology of IoT devices. First, the term IoT is defined and different network protocols involved in IoT systems are introduced. Further multiple experiments are conducted to validate the tool. The experiments are based on data sets that record the traffic of 3 different protocols to ensure a generalization approach.

This projects shows that the visualization tool is capable of processing different protocols and generating the topology. The tool further defines specific device groups, such as the most active nodes in the topology. Another function compares topologies with regard to their members. This functionality is used among other things to identify nodes not in contact with a malicious device. However, the quality of the resulting topology strongly depends on the data set, in particular on the presence of device addresses. In addition, there are certain security measures taken by companies that prevent the tracking of IoT devices. One example of this is dynamic addressing. If the same device has several addresses, it is listed several times in the topology. Subsequently, possible tool enhancements and extensions are presented.

Contents

Abstract	i
1 Introduction	1
1.1 Structure	2
2 Background	3
2.1 IoT Networks	3
2.1.1 TCP / UDP Standard	6
2.1.2 BLE Standard	10
2.1.3 UWB Standard	14
3 Related Work	17
4 Methodology	19
4.1 Experimental Design	19
4.1.1 Data Collection	20
4.1.2 Data Preprocessing	22
4.1.3 Algorithm	26
4.1.4 Visualization Approach	37
4.2 Experimental Setup	39
4.2.1 Experiment 1: UWB	39
4.2.2 Experiment 2: Network	43
4.2.3 Experiment 3: BLE	45

5	Results	47
5.1	Result 1: UWB Data Set	47
5.1.1	Result 1.1: Raw UWB Data Set	48
5.1.2	Result 1.2: Adapted UWB Data Set	53
5.2	Result 2: Network Data Set	56
5.2.1	Result 2.1: Normal Network Data Set	57
5.2.2	Result 2.2: Network Data Set under Attack	61
5.3	Result 3: BLE Data Set	64
5.3.1	Result 3.1: Small BLE Data Set	64
5.3.2	Result 3.2: Medium BLE Data Set	68
5.3.3	Result 3.3: Large BLE Data Set	71
6	Evaluation	73
6.1	Evaluation 1: UWB Data Set	73
6.1.1	Evaluation 1.1: Raw UWB Data Set	74
6.1.2	Evaluation 1.2: Adapted UWB Data Set	78
6.1.3	Evaluation 1: UWB Comparison	79
6.2	Evaluation 2: Network Data Set	80
6.2.1	Evaluation 2.1: Normal Network Data Set	80
6.2.2	Evaluation 2.2: Network Data Set under Attack	82
6.2.3	Evaluation 2: Network Comparison	84
6.3	Evaluation 3: BLE Data Set	85
6.3.1	Evaluation 3: Network Comparison	85
6.3.2	Evaluation 3: Run Time Algorithm	88
6.4	Limitations	89
7	Conclusions and Future Work	91
7.1	Conclusion	91
7.2	Future Work	93

<i>CONTENTS</i>	vii
List of Figures	100
List of Tables	102
A Appendix	105
A.1 Data Sets	106
A.2 Communication Graphs	106
A.3 Installation Guidelines	106
A.4 Timelines of Experiment 2.2	107

Chapter 1

Introduction

IoT or Internet of Things is network of devices collecting and storing data. IoT is on the rise while leading to a shift in communication patterns. Before IoT, traditional communication systems relied on an end-user to initiate any form of interaction [1].

IoT on the other hand is a network of different "Things" ranging from smart devices to vehicles or even houses. As a result, IoT facilitates objects to collect and transfer data, thus not relying on human actions [2]. The network of IoT devices is connected over the "Internet", enabling the transfer of data. Traditional communication standards do not meet the requirements of IoT networks alone because devices are responsible for connections and the transfer of data. This key functionality is achieved with ranging, hence multiple protocols are needed to enable the communication between smart objects [3].

This key features allows for a broad field of applications [4]. IoT networks can track different objects, ranging from stationary such as an inventory in a large factory to mobile devices like a key. With IoT systems starting to appear in every day life, the number of interconnected devices as well as the data traffic increases significantly. In summary, IoT is a network of networks, connecting a large amount of diverse devices transferring data over a broad spectrum of technologies or protocols [5]. Thus keeping an overview over all IoT devices can be increasingly challenging.

The heterogeneity of devices and networks are the biggest challenge in IoT networks. An type of topology analysis tool in networks with multiple different technologies needs to follow a uniform approach. Additional company specific implementation of IoT systems enhance the difficulty of a generalized tool. A broad spectrum of diverse protocols and device groups means a lot of different possible solution to choose from. There has been some research on trying to uniform IoT network components, but no proposed approach is used on a big scale [6; 7; 8]. As generalizing IoT systems has not caught on, another solution is to standardize the implementation. There are some solutions in form of defining key features or requirements for IoT systems, but no specific implementation details or frameworks are proposed [9; 10]. This again leads to a lot of different implementations of IoT systems, missing the mark of standardization.

While there is no solution regarding the generalization of IoT systems, visualisation approaches for network topologies exist. One research area visualizes IoT networks with the help of artificial intelligence (AI) while focusing on detecting malicious activity [11; 12; 13]. There are multiple projects presenting visualization approaches for internet networks

in a mesh topology [14; 15]. Multiple tools which generate multi-layered internet topologies have been researched. While some projects do visualize the underlying network topology, the objective is to simulate a network and its resources to detect anomalies or bottlenecks [16; 17]. Then there are tools which generate any synthetic network topology, but the nodes and connections have to be entered manually or by uploading an already generated topology [18; 19]. So far every tools is dependent on entering nodes and edges manually. Further projects actually work directly with network traffic data, meaning the topology is generated based on some packet capture tool data [20; 21]. In summary, research covers multiple aspects of IoT topologies, from simulating networks to identifying the underlying topology based on traffic data. This project takes it a step further, by not only working with network data, but by further analyzing and visualization different IoT network protocols such as Bluetooth Low Energy (BLE).

1.1 Structure

The objective of this project is to develop an uniform visualization approach for IoT network topologies. This work is structured into seven chapters. Chapter 1 starts with a introducing the topic by giving an overview of IoT networks and application areas.

Chapter 2 discusses the theory behind IoT networks. Different network protocols are researched and presented with the focus on topology relevant information. Thus different addressing schemes, topology options or device types are analysed. Further similar projects are compared and summarized in Chapter 3.

The gained knowledge from the first two chapters build the foundation of the visualisation tool. Chapter 4 gives an overview of the used methods, libraries and design choices of the produced code. At the start of the project, different network traffic data sets are collected. As this project follows an generalization approach, the data sets are processed into a uniform schema. An algorithm then performs all the necessary calculations necessary for the topology graph. An external library is used to generate the network topology. In this chapter the experiments are introduced. Each experiment consists of a different IoT network protocol and is used to validate the approach while analysing the resulting topology graph.

The results of the three experiment are shown in Chapter 5. This includes the topology with all devices. Additionally, different sets of IoT devices are identified. Each of the experiment is further evaluated in Chapter 6. The experimental results are used twofold, first to analyze the visualization approach for correctness and run time. Secondly, the actual network topology is researched and compared with other networks. This chapter finishes with the limitations of this project.

The last part is the Section 7.1 which concludes this project by summarizing the findings and presents some possible enhancements for the visualisation approach.

Chapter 2

Background

The background section gives an overview of IoT networks and the protocols researched in this project. In the first step, the basics of an IoT network are presented. In order to analyze the topology of IoT networks, diverse protocols involved are described. For each of the protocols, the frame structure, addressing, device types and topology versions are explained since these parts are important for topology detection.

2.1 IoT Networks

IoT, or the "Internet of Things", is a network consisting of multiple devices (Things) communicating with each other over different protocols (Internet). Figure 2.1 shows an example of an IoT network and illustrates the involvement of "Things" and "Internet".

The **Things** part refers to devices that are heterogeneous and consist of various types, namely end-devices, smart devices and data systems. More precisely, [23] define three requirements for a device to be a thing: A thing is identifiable, able to communicate, and able to interact. This means a device has a unique identifier in the form of an address and can use this address to communicate with other devices. Further, this address is used to interact with the device for end users, for example, to configure the device. End devices, such as smart watches with sensors, collect data or light bulbs, which can be turned on or off, execute user actions. These end-devices require low power usage, have less memory and range, are limited in connections to end points but have a faster data transfer. Then smart devices, such as a phone, are involved by controlling the end device (turn on light) and forwarding data to other data systems such as a server (e.g., heart-beat collected from a smart watch). Smart devices and data systems are able to connect a lot of endpoints over a long range, but the transfer of the data is slower and the power usage is higher.

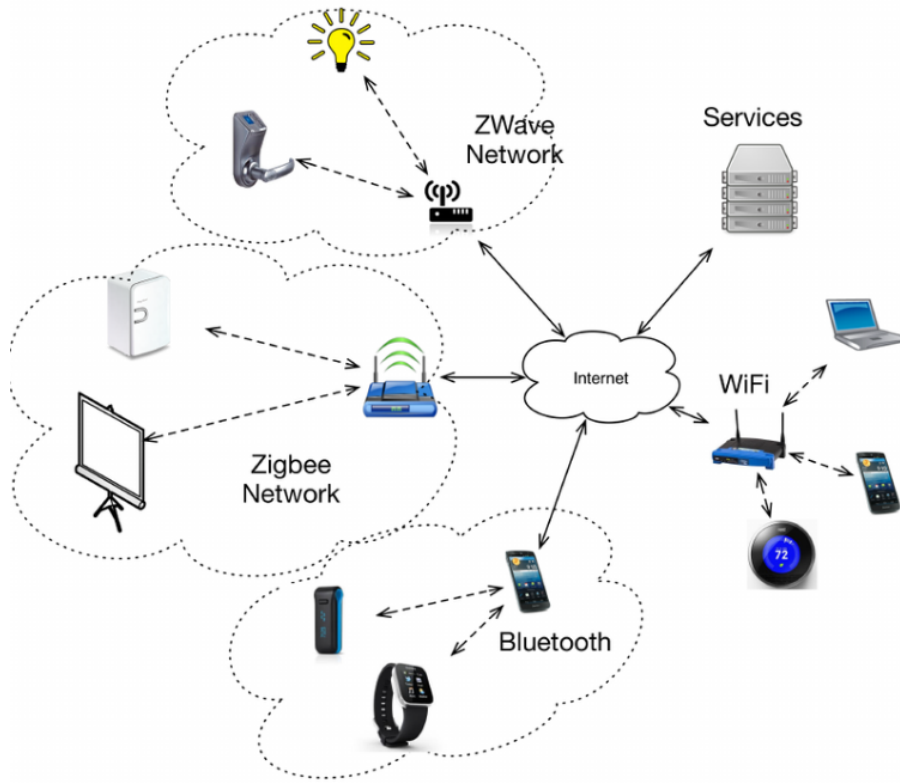


Figure 2.1: IoT Network [22]

Based on the 3 device categories, the following 2 IoT connection types can be derived, enriched with the main requirements. These connection types form the **Internet** part [24; 25]:

1. End-device to smart device: short range, fast data transfer, limited end-points, low power consumption
2. Smart device to data system: long range, slow data transfer, unlimited endpoints, high power consumption

The end-device communication with smart devices is illustrated with the different clouds at the end in Figure 2.1. While the higher level communication type is displayed in the middle with the "Internet" cloud [26].

As discussed, the **Internet** part represents the two types of communication or more clearly, the network protocols used by the devices when communicating. IoT network protocols and standards have diverse key features and requirements [27]. This can be a short-ranged protocol with limited device capabilities for communication with the end device or a protocol that can handle an unlimited amount of devices for the end-to-end connection [22].

There are Wi-Fi protocols defined in the IEEE 802.11 standard and used in WLANs and contain end-to-end communication protocols and can handle an unlimited amount of devices over long distances [28]. These protocols are used for the communication between smart devices and data systems. An example is the Transmission Control Protocol (TCP) used for connection-oriented communication or the User Datagram Protocol (UDP) acting connection-less.

Then there are Wireless personal area network (WPAN) protocols defined in the IEEE 802.15.4 standard used for data transfer over low-cost and low-power connections [29]. These protocols are used to directly interact with the end devices since the requirements are the same. These protocol stacks include Bluetooth Low Energy (BLE) and Ultra-Wide Band (UWB).

In this project, three data sets capturing three different protocols are analyzed. For the higher level protocols, or smart devices to data systems, the two protocols TCP and UDP are further explained. For communication with the end device, the two protocols BLE and UWB are selected. These two levels of communication interact with each other. This means, that some data can be collected by an end device and sent to a smart device for further processing. This communication happens over the lower level protocols BLE and UWB dependent on the device. The collected data then is sent from the smart device to a data center in order to analyse and store the collected data. This type of communication happens over a higher level protocol, such as UDP and TCP.

2.1.1 TCP / UDP Standard

The TCP and UDP protocols are the protocols for the transmission of data over the internet for end-to-end communication. Both standards are transport layer protocols and part of the internet protocol suite [30]. TCP and UDP are long-range (hundreds of meters) protocols that allow for communication between an unlimited amount of devices [31]. While both protocols enable network communication, each protocol possesses distinct key features and hence the use is dependent on the application [31; 32; 30]. The main difference is TCP enables reliable communication and hence is used for connection-oriented communication. This is achieved by opening a secure communication channel for the transfer of data. The connection establishment happens over a three-way handshake between two devices before actually sending data. This connection has to be terminated again, which happens over a four-way handshake. This direct connection leads to a few advantages, such as the possibility of sending data as a stream and the verification of a successful data transmission. On the other hand, the additional transfer of messages leads to a slower transmission and a need for more bandwidth than other protocols. UDP on the other hand is connection-less, meaning data is just sent without initially setting up a connection. As a result, the data transmission is unreliable but very fast. This allows for the transfer of time-sensitive data for real-time application. While both protocols differ in frame structure, they have the same topology options, device types, and addressing, as both protocols are part of the same suite.

Frame Structures: The first structure to have a closer look at is the header of the UDP protocol and its fields as seen in Figure 2.2. The UDP header consists of the following fields [32; 30]:

- Src. / Dest. Port: The value of this field gives the endpoints of the connection in the form of the port number of the destination and source device.
- Length: Indicates the size of the whole datagram.
- Checksum: The checksum is used to indicate if the header is damaged.



Figure 2.2: UDP Header Structure (based on [32; 30])

The basic structure of a TCP header is given in Figure 2.3 and the fields are [32]:

- Src./Dest. Port: The value of this field gives the endpoints of the connection in the form of the port number of the destination and source device.
- Seq./Ack. Number: This number is used to check the correct order of frame arrival .
- Flags: This parameter indicates the flag in TCP:
 - ACK: Successful data transfer
 - PSH: Alert of incoming data
 - SYN: Initiating connection
 - FIN: End of data transfer
- Window Size: The field value defines the sender's receive window.
- Checksum: The checksum is used to indicate if the header is damaged.

Src. / Dest. Port	Seq. / Ack. Number	Flags	Window Size	Checksum
----------------------	-----------------------	-------	-------------	----------

Figure 2.3: TCP Header Structure (based on [32; 30])

Both structures only show the header of the protocol. The header is placed in front of the user data and a segment (TCP) or datagram (UDP) is created. The TCP segment contains more fields than the UDP datagram. The reason is the reliability of the transfer, enabled through key features such as enhanced error checking. These capabilities need more information to work and hence more storage space.

Addressing: While each protocol has its own frame structure with protocol-dependent information, the data is sent over a lower level on the internet protocol suite and both frames are further encapsulated and enriched with the same headers [33]. This data encapsulation is shown in Figure 2.4. The user data in yellow on the application layer is transferred to another device. Depending on the protocol used on the transport level, a UDP or TCP header in blue is added to the front of the user data. This header contains the port address of both endpoints. The TCP/UDP header combined with the now-called UDP/TCP data are further processed by adding an IP header in green. The IP header contains additional information and flags. The key information added here is the destination and source IP address. The IP address is a logical address of a device and can have different forms, depending on the version (IPv4 vs IPv6) [33]. The IP address of a device can change, as it is configured by software components. This happens on the internet layer and the finished package is handed to the link layer as data to transfer. Further, a link header in red is generated and added to the front. The address used on this layer is the MAC address. The MAC address is the physical hardware identifier of the devices and cannot be changed. This is the final form of the frame and contains all necessary information for the data transfer.

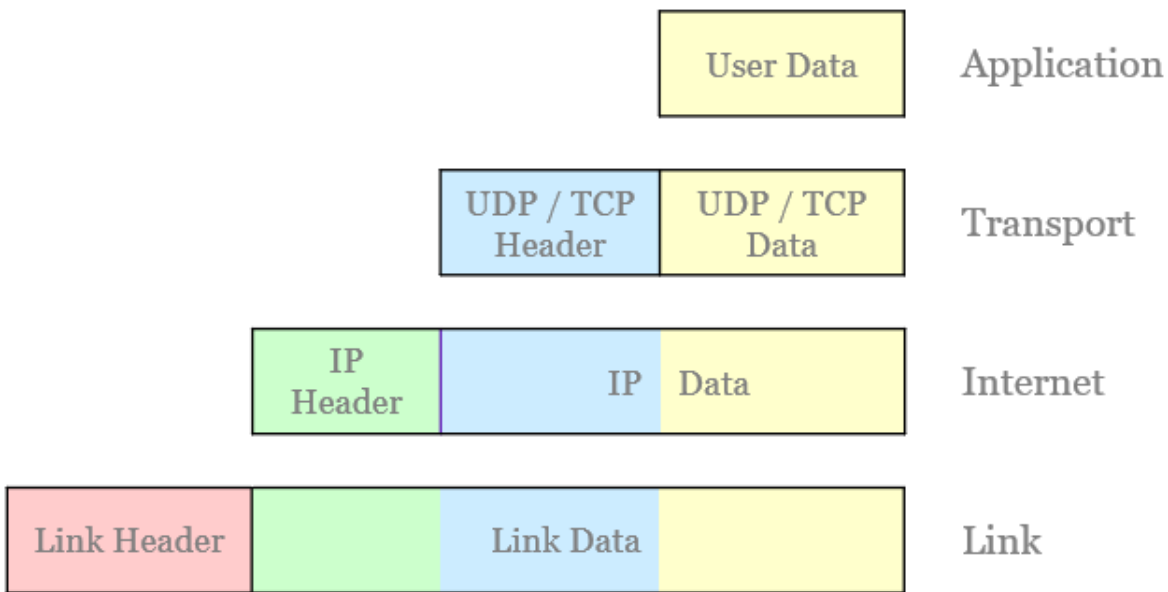


Figure 2.4: Frame Encapsulation (based on [32; 30])

Device Types: On the higher level network, the communication generally happens between two hosts, a sender (end device or client), and a receiver (data system or server) [34]. The host which initiates the communication is typically named the sender or the client [35]. When compared to the IoT network device types, the client typically takes over the role of the smart device. The receiver host sits on the other end of this message and is typically named "server". The server role translates to the data system type in an IoT network [36].

Topology Options: As soon as two or more devices start to interact, they form a network or a topology. The topology can take multiple forms and a few options are presented here with a focus on IoT networks [37; 38]:

- Point-to-Point/Multipoint Topology: This type forms when two hosts or one host to many communicate directly.
- Star Topology: If one central entity serves multiple hosts a star topology results.
- Mesh Topology: In a mesh topology all involved devices are connected to each other and, hence are fully connected.
- Hybrid Topology: When various types of different topologies are combined, a hybrid topology is formed.

The options are endless, as some network topologies are formed naturally when connecting to another device. There are also fixed topologies, which are configured at the start and do not change.

2.1.2 BLE Standard

BLE is a part of the wireless LAN (WLAN) 802.15 standard by IEEE and is used for short-range (50-60 m) data transmission. In IoT networks, BLE is used for the interaction between a smart and end device. The key characteristics of BLE are its low power consumption and unlimited amount of connections. BLE compared to classic Bluetooth, can handle a lot more data faster. However, due to the restrictive battery life of the end devices, BLE is not suitable for high data transmissions. The low power consumption is achieved by end devices sleeping until another device initiates a connection or by waking up at a certain time interval [39].

The BLE protocols allow for connection-less (broadcast) and connected communication according to [40; 39]. In a connection-less type, the devices send out messages, and devices in reach are able to listen to these devices or broadcast data for applications not needing a connection. The receiving device is passive and never actually interacts with the sender device. The sending device has no control over which device in reach can access the sent data. This BLE communication type is mainly used for applications where range plays a factor, for example, smart marketing or sensors that send their findings to a phone every day at the same time. On the contrary, the connected or uni-cast communication depends on devices interacting and hence connecting before actually sending any data. In a connected environment, both devices can send and receive messages. The connected type is more secure because the devices know the communication partner and hence more security mechanisms can be implemented, for example, anything involving a key exchange is now possible as the keys can be exchanged. Applications depending on connected communication transfer sensitive data, for example, health-related information, or the devices need to interact in order to work. For example, smart watches need to be configured and hence need to be accessible. Or more simply, as soon as a pairing process is required, a connection is established.

Frame Structures: In the BLE standard, there are two kinds of packages, data and advertising, and both types use the same packet format as seen in Figure 2.5 [41]. The structure consists of multiple fields, such as the access address of the communication channel and the Cyclic Redundancy Check (CRC) field, used for checking for errors that happened during the transmission. The Protocol Data Unit (PDU) field contains the information to send and is dependent on the package type. If data is sent after establishing a connection, the data channel PDU schema is used to transmit the data. This data channel PDU has some additional security mechanisms, for example, MIC as seen on the right side in Figure 2.5.

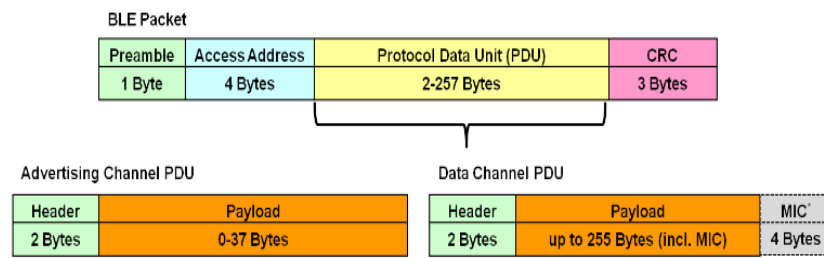


Figure 2.5: BLE Frame Structure [41]

Advertising channel PDUs are either used for broadcasting data or to discover and connect with other devices. In the case of broadcasting data, the PDU contains the broadcasting address, the sender address, and the data. If the package is used for discovery or connection establishment, the PDU includes the sender and receiver MAC address, and some additional information such as the addressing used, which will be explained in the next paragraph. If both MAC addresses are present, the devices already know each other (have connected and want to reconnect). The presence of solely the sender's MAC address indicates devices wanting to connect or searching for new devices. There are three sub types of advertising channel PDUs and their use:

- Advertising PDUs: broadcast data
- Scanning PDUs: scan for devices to connect
- Initiating PDUs: connect to a device

After two devices connect, they communicate directly and stay connected for an unlimited time frame [42].

Addressing: BLE networks have multiple options when it comes to addressing devices [43]. Figure 2.6 gives an overview of all options. Naturally, every device has a MAC address which is unique and in most cases a hardware address, meaning it cannot be changed. This is the public address of a BLE device. Public MAC addresses have one critical weakness, namely their security [44]. Once the MAC address of a device is obtained, the traffic can be tracked and captured. A solution is to use random addresses. Random addresses change periodically and hence can not be tracked. As the name suggests, random addresses are generated at random. A static random address can change after each power cycle or can even be allocated for a lifetime. It is a logical address and can be changed by a user. This means for bidirectional communication, that after every power cycle, the device has to reconnect with its previous communication partner by advertising. If a device does not want to share its address, a private random address is used. A private address can be resolvable or non-resolvable. Resolvable addresses are readable by devices with a key. This means that only devices knowing the key can identify the device, for all other devices the address is untraceable. This key has to be shared to be able to exchange messages. Non-resolvable private addresses are rarely used, as they change and are not readable by other devices. This means devices with this type of address can only send messages, as no other device knows their address.

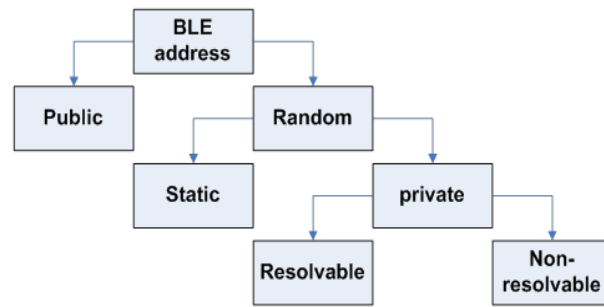


Figure 2.6: BLE Device Addresses [43]

Device Roles: The devices in BLE networks can take on different roles based on events [45; 46; 47]. The role is dependent on the communication type (uni- vs. broadcast) and changes when a certain type of message is sent. The main roles are the following:

- Broadcaster / Observer
- Master / Slave

Before any communication happens over BLE, all devices are unconnected. That means that one device, in most cases an end device, sends messages to anyone in reach. This is the broadcaster and only sends out advertising PDUs. Devices that are in reach and receive said advertisements are called observers, which are primarily smart devices. Observers are passive devices that never send messages. At the start of every BLE connection, the involved devices take on the role of observer and broadcaster. If the devices stay in a connection-less relationship, they never change roles.

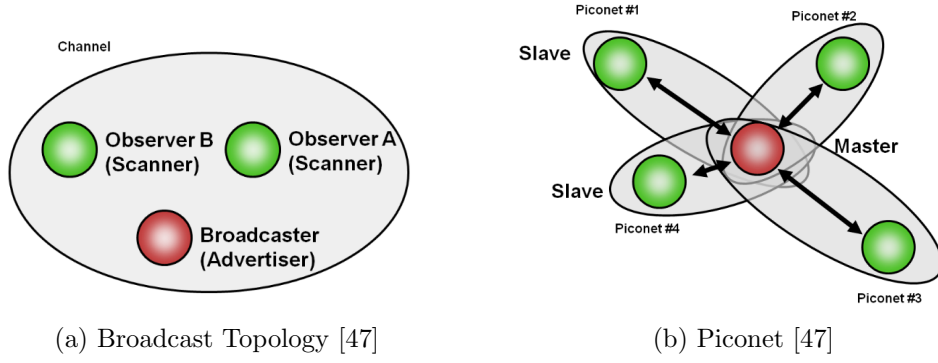


Figure 2.7: BLE Topology Options [47]

On the other hand are applications that require a pairing process in order to exchange messages more safely. To be able to do this, the broadcaster announces its presence and intent to connect. On the other side is an observer, reading all advertisements in reach and eventually initiating the connection. The broadcaster then has to allow the connection request. As soon as a broadcaster and an observer connect, they enter uni cast communication type. At this point, the observer becomes the master and the broadcaster becomes a slave and they start transmitting user data. In a connected network, masters and slaves can send and receive packages.

Topology Options: A network communicating over BLE forms one of two topologies, dependent on the communication type [47]. Broadcaster and observer communicate in a connection-less and uni-directed (one device only sends/receives) way. Figure 2.7 (a) shows an example topology and illustrates how one observer can listen to as many broadcasters as there are in reach which is two in this case. In this type of topology, the end device takes on the role of the observer and periodically reports any data that has been collected, for example, the temperature. The broadcaster, normally a smart device, collects the sent data and processes it. The other topology formed is called piconet. A piconet consists of BLE devices in a connected state, thus master and slave devices. Figure 2.7 (b) shows an example of a piconet. In the middle is the master device, in most cases a smart device such as a phone, which is connected to multiple slaves or end devices. Since the communication is bidirectional, both can send and receive messages. The master is responsible for the resources in a piconet and manages the slaves. A slave can only be part of one piconet and a master can serve an unlimited amount of slaves. An example is tracking devices. Before being able to use the tracking device, it is connected to a phone. The tracking device sends its position regularly or upon request to the phone. The phone collects the location and visualizes it to the user.

2.1.3 UWB Standard

UWB is a communication standard used to transfer data over short distances between end devices and smart devices. UWB is part of the 802.14.4 standard for low-rate wireless personal area networks (LR-WPAN) [48]. Key characteristics include, compared to other wireless communication technologies, the transfer of data by repeatedly sending pulses in a very short time frame over a large range of bandwidth. The short pulses and high bandwidth enable a low-power transmission while sending a large amount of data. On the other hand, this combination leads to a rather short range of transmission of around 10 metres . These features allow the communication over UWB to work without interference with other wireless technologies [49]. The main use of UWB is for location tracking systems, as UWB frames include an additional time stamp, enabling different distance calculation methods or time-of-flight calculations. Compared to BLE, UWB has a shorter range but sends more data faster. UWB either communicates directly between two devices or communicates with each device in range depending on the application. Both communication types require a pairing between devices beforehand. This means for the communication in range, that non-paired devices can detect one another if close but do not necessarily exchange data. Devices choose which frames they accept, based on the initial pairing. As an example, when connecting a phone to a location-tracking device, the pairing happens. After that, the phone can track the location when in range over UWB and show the position. On the other hand, the phone can also detect the UWB signal of other tracking devices within range, but the location is not shown. The pairing mechanism combined with the short time period of pulses (almost impossible to interfere with communication in a timely matter), leads to a secure protocol.

Frame Structures: UWB uses the following four frame types to transfer data and manage the communication between devices [50; 48]:

- Beacon frame: connects devices and facilitates communication
- Data frame: transfers all data
- Acknowledgment frame: confirms successful transmission of a frame
- Command frame: manages communication

The first type is a beacon frame which is vital for the operation of a network. Beacon frames are used by UWB to scan for devices in range and connect them. They further enable the low-power use of devices by managing the resources and waking up devices when needed. Beacon frames are sent by master nodes. Data frames are used to transfer any kind of data produced or collected, which in a UWB network are most likely accurate time stamps to calculate the distance and hence the location. In order to guarantee correct communication, acknowledgment frames are used to indicate a successful frame transmission. Last but not least, command frames are used to manage and monitor network resources.

Figure 2.8 shows the basic structure of an UWB frame on MAC and PHY levels. The MAC level is responsible for the successful transfer of data, while the PHY level takes care of detecting other devices and initiating communication. The MAC level format consists of three fields. The MAC header (MHR) is used to identify the frame. Hence, the MHR includes a frame control field for devices to identify the type and whether or not to process the frame. Further fields include the source and destination address if present and whether or not the frame is sent to another device within the same network. The MAC payload consists of the data to be sent. The MAC footer (MFR) uses a frame-checking sequence mechanism to detect transmission errors. The MAC layer frame is further processed and enriched with PHY layer information, hence turning into the PHY payload (PSDU). The synchronization header (SHR) added in this step is responsible for detecting signals in range by synchronizing a sending and receiving device. UWB comes with different possible implementations, such as diverse data rates. The configurations are stored in fields in the PHY header (PHR).

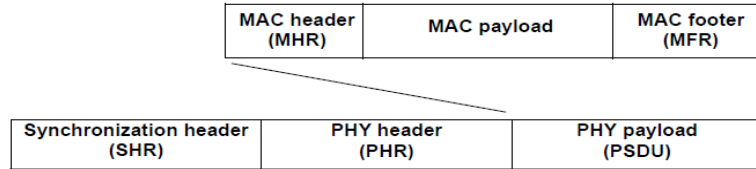


Figure 2.8: UWB Frame Structure [50]

Device Types and Addressing: In UWB networks two device types are distinguished, namely full-function devices (FFD) and reduced-function devices (RFD). These two device types can enter one or more of the following different modes: Personal area network (PAN) coordinator, coordinator, transmission device, and reception device. A PAN coordinator manages a UWB network by initiating the network, scanning for new devices, and initiating and terminating communication between devices. Coordinators take on the role of distributing messages over the network or routing. Coordinators are main-powered and hence have more functionality and are typically smart devices. Transmission and reception devices have limited functionality, as they are battery-powered and, thus are only used to communicate with a single communication partner. These are mostly end-devices in an IoT network such as sensors reporting their findings to a central entity. While an FFD can enter any, all, or a combination of the described modes, an RFD only has the choice of transmission or reception device. FFDs communicate with both device types, FFDs and RFDs. Since RFDs only require a few resources and have low memory capacity, RFDs only communicate with exactly one FFD, which is responsible for the resource distribution.

Devices in UWB networks are uniquely identified by an extended address. An extended address is composed of a short and a PAN address. Each participating device is equipped with a short address by a PAN coordinator. This short address is unique within the formed network and enables communication within this PAN. On top of that, each UWB network has its own unique PAN ID. The PAN ID facilitates communication between multiple BLE networks. Both addresses together are the extended address and enable devices from different PANs to interact.

Topology Options: In UWB there are similarly to BLE two topology options formed: star and peer-to-peer topology [50]. Figure 2.9 shows both options. Star topologies have one node which is responsible for the resource distribution in a network, thus using exactly one master node, the PAN coordinator or an FFD. Devices in a star topology are mostly associated with a specific application. The outer nodes pose as the start or end point of a data transfer, hence are end-devices or RFDs. RFD in general can only be part of star topology, as their only communication partner is a FFD. Example star topologies include home automation, PC peripherals and personal healthcare.

Peer-to-peer topologies on the other hand communicate as soon as devices are in range. Each device is an FFD as each node can initiate communication. This topology is more complex and can be used for more enhanced applications, such as wireless sensor networks or asset and inventory tracking.

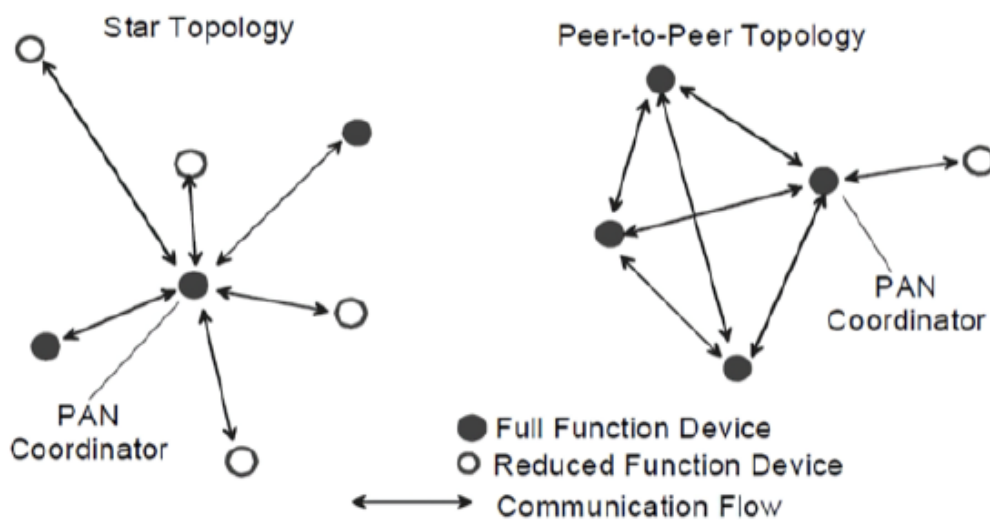


Figure 2.9: UWB Topology Options [50]

Chapter 3

Related Work

In this chapter related work to this project is presented. This includes all papers developing a visualization approach for IoT systems. The research shows, that for network data or the higher level communication, network visualization tools exist. For the other two standards BLE and UWB, theoretical work on topology detection has been researched, such as proposals to optimize BLE topologies [51].

The related work on network data can be grouped into two categories. One category contains visualization approaches based on manually entering the network topology by a user [18; 16; 17; 19]. These projects not necessary have a focus on IoT networks, rather on mesh networks. Since IoT is a network of network, which also counts for a mesh topology, these projects are also taken into account. These tools are able to visualize IoT networks, but are thought to be used for simulations of networks and not to represent real-world topologies. Hence, this tools are not further discussed. The second group contains approaches generating a network topology based traffic data sets. There are other projects, which work with AI to generate topologies and classify device groups, but since this project does not work with AI, the projects are not related. The following tools all generate a topology based on some network traffic and allow for some further analysis. GRANEF is toolkit using traffic data to visualize the underlying topology [52]. The project stores communication data and displays multiple analysis tools for the user. This project differs as it presents an additional abstraction layer above the traffic data, meaning even an application process or a file can be a node.

NetGraph developed by [53] is another graph-based topology detection tool based on log files. The data is presented in form of associations and focuses on the analysis of connection like HTTP logs. The tool has a set of predefined relations and states, which are allocated to the nodes and edges.

There are two tools which work with Pcap files. One tool similar to this approach is NetCapVis [20]. NetCapVis is a web-based analytical system with the functionality allowing a user to upload Pcap files as well as setting initial filters for data reduction before uploading. The tool then generates a topology and has the option for the user to interact and run predefined queries. PCAPFunnel also generates a topology based on Pcap files and allows for further package analysis [54]. Both projects only test their approach with network data, hence it is unclear if they work for other traffic types.

Chapter 4

Methodology

The chapter methodology is structured in two parts. In a first step in Section 4.1, the experimental design is presented. First the data processing and collecting methods are explained in Subsection 4.1.1 and Subsection 4.1.2. The algorithm calculating all necessary visualization measurements is explained in Subsection 4.1.3. Subsection 4.1.4 concludes with the presentation of the visualization approach and the library selection. In a second part in Section 4.2 the experimental setup is shown, consisting of 3 different experiments.

4.1 Experimental Design

This section gives an overview of the design of the experiment. In order to analyze an IoT network and its topology, the first necessity is to collect network traffic data as explained in Subsection 4.1.1. Generally, network traffic data is collected by a sniffing device and visualized in a packet analyzer tool. The collected data is then transformed and enriched into a uniform data schema as presented in Subsection 4.1.2. Based on this data, diverse analyses are made with an algorithm as seen in Subsection 4.1.3. This algorithm consists of different methods, calculating node and network level measurements. Finally, Subsection 4.1.4 further evaluates which visualization approach is used to visualize the collected information with the support of an external library.

4.1.1 Data Collection

The foundation of this project is network traffic data of IoT networks. The objective is to obtain data sets of UWB, BLE, and network traffic. This data consists of recordings of messages sent in a network using the mentioned protocols. There are two methods to collect the data sets in this project. The first method is to capture traffic data by using a sniffing device. The sniffing device is connected to a PC with a USB cable. A script running on the PC writes the incoming packages into a packet analyzer tool. The second option is to use network traffic data sets from an external source. These data sets have also been captured by a sniffing device. For this project, the BLE data set is sniffed while the other two data sets are prerecorded from another project.

One data set consists of many rows, each row representing a message sent between two devices in the network. The columns as seen in Figure 4.1 display different parameters of the captured data. The parameters colored in red are derived from the captured packages, while the blue information is added by the script. The most important ones are destination and source address as they represent the basis for topology analysis. Additional information added by the script includes timing or protocol information. The biggest challenge in this step is the completeness and consistency of the data. This mainly refers to the source and destination address not being empty, since this information is derived from the captured packages and therefore cannot be added manually.

Number	Time	Source	Destination	Protocol	Length	Info
--------	------	--------	-------------	----------	--------	------

Figure 4.1: Captured Parameters

Sniffing Setup

The sniffing setup of this project has the objective of capturing BLE traffic in a network. For this, two hardware components are needed. First, a sniffing device, which is able to capture BLE traffic. Then, a laptop with Linux that can present and store the captured packages. Further, some applications and libraries need to be installed. For this project, the selected sniffing device is the Ubertooth One from Michael Scott Gadgets [55]. Ubertooth One is an open-source tool with the capability of sniffing BLE traffic. This tool consists of multiple hardware parts including an antenna. The installation guide of the project gives an overview of the software components that are required for the sniffing setup [56]. The first library is called libbtbb and is used for decoding the captured BLE packages on the laptop. Further, the project provides a GitHub repository consisting of code for the Ubertooth One gadget, which allows the hardware to sniff the packages. This includes code for the configuration of the tool as well as the firmware. Additionally, Wireshark needs to be installed, as this is the package capture tool the project uses. After the successful installation of the libraries and Wireshark, the Ubertooth One is connected to the computer by plugging it into a USB port.

Before actually sniffing, a temporary pipe is created in Wireshark. In order for the sniffing tool to work, the command "ubertooth-btle" is entered in the terminal to address the Ubertooth One directly. This command includes the gadget, which traffic to sniff, and where to send the captured traffic (temporary path). After the command is entered, the tool automatically starts capturing traffic and displays the information in Wireshark. For this project, the mode for sniffing BLE is chosen, as in IoT networks a lot of end devices communicate over the low energy mode as being low powered is one of their key features.

Data Collection Challenges

As mentioned in this parts introduction, there are two main challenges in the data collection step. The data completeness refers to the source address and destination address values of the collected data not being empty. As an example, devices from different vendors using UWB for localization may have a company-specific implementation of the protocol. This means without knowing company-specific implementation details, a packet analyzer tool such as Wireshark is not able to process the packages leading to empty values in the destination or source address in the data set. The solution to this problem is further explained in Subsection 4.1.2.

The consistency of the data on the other hand means that the three captured traffic data types may record different values dependent on configurations of Wireshark. As an example when capturing WLAN traffic, Wireshark has the option of capturing "any" device in an IP-based network, meaning any communication over a WLAN standard protocol is captured. This is a capture configuration named Linux cooked-mode capture (SLL) in Wireshark specifying the diverse protocol for the sniffer to capture [57]. For this to work, Wireshark adds its own link-layer header and fills in the values. This means, that not all information can be used because the recorded packages are treated the same for every WLAN substandard. It is not always clear, which information is added from Wireshark and which information is retrieved from the captured packages. It can happen that for one protocol the MAC addresses are recorded and for another the IP addresses. For this project, this does not pose a problem. That is because, in an IoT network, the connection to the final device happens over UWB or BLE. These IoT devices are coupled to another device that is present in the overlaying network. The overlaying network works with IP addresses, so an analysis of the overlaying network level is still feasible.

4.1.2 Data Preprocessing

At this stage, the collected data sets of different network traffic types are processed into the same data schema. The data schema is derived from the needed information for the algorithm as well as the visualization approach. The main challenge in this step lies in the identification of a uniform file format and necessary parameters.

File Format Identification

Originally, the file format Pcap was chosen for its usability with packet analyzer tools and libraries such as Wireshark and tshark. Tshark is a Python library used to access the individual fields of a Pcap file to read the value. For this project, this approach does not work because of the generalization requirement of the three traffic types. Each one of these have their own naming rules. Wireshark derives the naming for the value access of the fields according to the protocol. More specifically, the "Source" column is the same in Wireshark if in csv format for all three traffic types, but when actually accessing the "Source" value of the Pcap, the field is named differently according to the protocol. As an example Figure 4.2 shows the retrieval of a source address for Pcap files for the traffic type TCP via *eth.src*. UWB on the other hand uses *wpan.src64* as seen in Figure 4.3. In order for this approach to be uniform a csv instead of a Pcap file format is used. This ensures a generic approach because the visualization approach can work with traffic types not part of this project as long as a csv file from a packet analyzer tool exists with columns specified in the following part.

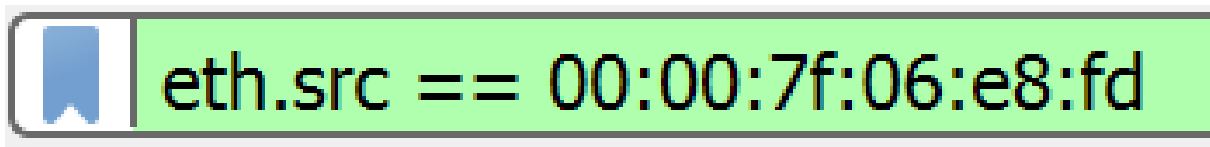


Figure 4.2: Network Source Address

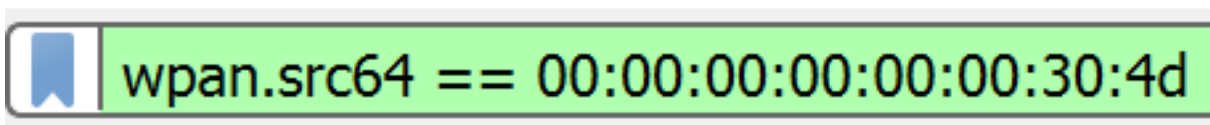


Figure 4.3: UWB Source Address

Parameter Identification

In this step, the required parameters are identified. The visualization approach needs two types of information: edge-specific and node-specific information. Edge-specific information contains data for each pair of devices that communicate. This includes the total number of packages but also per frame type. The node-specific data carries the information per node, such as the total number of packages sent by this node or per frame type, but also time-specific information such as pps.

In this project, all data sets were captured and visualized with Wireshark. Wireshark offers the possibility to add additional package information in the configuration. Since this project approach is uniform, the basic example configuration as seen in Figure 4.4 are the given parameters. The first row contains the "Number" or the number of the package in order of arrival. For this project, this number has no value and is therefore excluded. The "Time" row is needed for multiple calculations, such as the node pps or the activity span of a node. The time can have different configurations, for example, the absolute time of the day or relative time since the start of recording. For this project, the configuration for the time is the period since the start of the recording. It can be configured in milliseconds. As mentioned in Subsection 4.1.1, the "Source" and "Destination" columns show the sender node address and the receiver node address. These two parameters are the foundation for the visualization approach and must be present. The address of a node is the unique identifier of this device and is used to research the sending and receiving behavior of one node. If an address is missing, the two devices 'UNKNOWN_DESTINATION' and 'UNKNOWN_SOURCE' are introduced as addresses and filled in. This is important because, with this solution, the topology can be visualized without losing traffic information. The address field can contain diverse addresses dependent on the protocol. This approach can work with all types of addresses, as long as they are recorded. The "Protocol" section can be used for information purposes on the node level in the visualization approach but is not needed for any further calculations. Since this field is also added by Wireshark, it is not clear where the information is derived and is also dependent on the user's configuration in Wireshark. Therefore, this field is not used as it is not uniform. As an example, in one data set the protocols "IPv4" and "UDP" appear as values for the protocol field. But UDP is sent through IPv4 as UDP specifies the port number and IPv4 the MAC address. Wireshark reads the highest level of protocol it can dissect and fills in the value. The "Length" information of the packet length is not needed in this project. At last, the "Info" field contains all the information Wireshark was able to read from a package that is not already presented and accessible without decryption. The only needed information from this field is the frame type of the message if present. This information is extracted. In the example in Figure 4.4 the frame type of the package is data and this information is extracted. For visualization purposes this field is not allowed to be empty, if no frame type is present, the value "UNKNOWN" is inserted. This information is used for edge-specific information. The required parameters for this project, therefore, are: "Source", "Destination", "Time" and "Info" of a csv file.

Number	Time	Source	Destination	Protocol	Length	Info
28	26.815140s	00:00:00:00:00:00:32:41	00:00:00:00:00:00:58:4d	IEEE 802.15.4	52	Data,

Figure 4.4: Example Captured Package

Schema Identification

As the file format is uniform and the necessary parameters are identified, the input data sets are transformed and enriched with information into six dictionaries and four sets. This part gives an overview of the sets and dictionaries which are instantiated. More detailed explanations regarding the implementation is given in Subsection 4.1.3.

Code 4.1: Set() Initialization

```
nodes();
unique_nodes_sent = set()
unique_nodes_received = set()
frame_types()
```

The three sets in Code 4.1 carry all information regarding the uniqueness of nodes, one containing all unique node addresses of the network that have either sent and/or received a package, one set for all nodes that have sent a message as well as one set for all nodes receiving at least one message. The data structure *set()* is chosen for its efficient membership calculations and uniqueness of elements. These sets are used for two purposes. Firstly, the set *nodes* includes all devices to visualize. The two sets *unique_nodes_sent* and *unique_nodes_received* contain the unique nodes which send or receive. This information is used for analysis purposes to identify nodes that only send or receive packages. The fourth set *frame_types* contains all frame types occurring in the network.

Code 4.2: Dictionary Initialization

```
edge_info_dic = {}
edge_src_dest_dict = {}
node_src_dict = {}
node_dest_dict = {}
network_info_dic = {}
time_list_received = {}
time_list_sent = {}
```

The dictionaries in Code 4.2 carry more information and can be split into four types. The dictionary is used because it has the data structure needed for the visualization approach.

To start, there are edge-specific information and visualization dictionaries, *edge_info_dic* and *edge_src_dest_dict*, composed of all the necessary information for each communication pair. One data item has the structure as seen in Code 4.3, line 1. The "device1" is the sending device and "device2" is the receiving node. The *edge_info_dic* is composed of the total number of messages between one communication pair and is therefore used for visualization. For this purpose, an additional "Frame_type" called "Total" is introduced, while the *edge_src_dest_dict* counts the number of messages of a communication pair per frame type and is used for the labeling of the edges.

Code 4.3: Example: Edge specific dictionaries

```
{('device1', 'device2', 'Frame_type'): count of occurrences}
edge_info_dic[(device1, device2, 'Total')] = 3
edge_src_dest_dict[(device1, 'device2', 'Data')] = 1
edge_src_dest_dict[(device1, device2, 'UNKNOWN')] = 2
```

As an example, there are two devices communicating. In total they exchange three messages, resulting in an entry in the *edge_info_dic* as seen on line 2 in Code 4.3. The frame types of this exchanging pair consist of 1 "Data" and 2 "UNKNOWN" types. This information is collected in the *edge_src_dest_dict* as seen on lines 3-4 in Code 4.3. Next, there is a need for dictionaries that collect node-specific information consisting of data regarding the number of messages sent or received by one node. This includes information for the total number of messages sent and received by one node as well as per frame type. For this purpose, there are two dictionaries initialized, the *node_src_dict* and *node_dest_dict*. These dictionaries have the same schema as the edge dictionaries as seen on line 1 in Code 4.3. What differs, is the device type. Instead of another node_id as the communicating device partner, these dictionaries count the total occurrences and per frame type of one node with all its communicating partners, e.g., the sending behavior. The *node_src_dict* collects all information of a node being the sending device and as a communication partner the node "ALL" is introduced as seen on line 1 in Code 4.4. On the other hand, the *node_dest_dict* collects the same information but if the device is on the receiving end of the message as seen on line 2 in Code 4.4. An additional network-level

Code 4.4: Example: Node specific dictionaries

```
{('device_is_sending', 'ALL', 'Frame_type'): count of occurrences}
{('ALL', 'device_is_receiving', 'Frame_type'): count of occurrences}
```

dictionary contains all data related to the whole network. The dictionary *network_info_dic* counts the total number of messages in a network as well as per frame type. Therefore, a new device type named "Network" is introduced. Additionally, this dictionary contains some timing information, such as the pps or the time of recording.

Lastly, there are timing dictionaries recording the sending and receiving behavior of each node as well as the pps. This information is calculated for the sending and receiving behavior of a node. As a result, there are two dictionaries, the *time_list_received* where the time activity of the node as the receiver is recorded, and vice versa for the *time_list_sent*. These dictionaries include the first time a node has sent or received a message as well as the last time. Derived by this information is the pps per node.

4.1.3 Algorithm

This section presents different measurements and visualization options to analyze the network typologies. First, different network-level measurements are introduced as well as node-specific information which is added to every node. These two parameter sets are used for the labeling of the network topology. The algorithm also presents specific device IDs and the option for selecting special nodes to visualize. Finally, the algorithm allows for the comparison of two networks by some parameters, for example, the number of devices. When comparing two networks, the network and node-level parameters indicate a change in the network. In order to analyze the algorithm, some performance measurements are also present. Additionally, an option to generate an activity diagram of the nodes is present.

Network Level

In this section, all calculations and data refer to all nodes and all messages for one defined network. The network does not have to include all nodes of a network, rather the information is calculated and recorded for a set of chosen and predefined nodes. On the network level, this project researches the following parameters.

Total messages: This is the total number of messages in the selected network. It is basically the number of rows containing the nodes selected as sending devices. This information is retrieved from the *network_info_dic* by calling *network_info_dic[("Network", "Frame_type")]* and inserting "ALL" as the frame type. **Used for:** The total count is used as visualization information as well as for the calculation of the pps on the network level.

Messages per frame type: This is the total number of messages in the selected network per frame type. The sum of all messages per frame type is the same as the total messages. This information is retrieved from the *network_info_dic* by calling *network_info_dic[("Network", "Frame_type")]* and iterating over all present frame types. **Used for:** The messages per frame type count is used as visualization information. When comparing two traffic data sets of the same network, this parameter can also indicate some change. For example, if the total message count goes up by the same amount as the total frame type count for "UNKNOWN", while all other counts per frame type stay the same, it can indicate that a device is acting differently than intended.

Time capture: This time information refers to the time interval of the recording in seconds. The interval starts when the first message is sent and concludes when the last message is sent from the network. If the network includes all nodes, the starting time is at 0 seconds and the time of the last message is the time of capture. When only choosing a subset of nodes to visualize, the timing interval is calculated by a simple subtraction of the recorded time of the first message of a chosen node from the time the last message was sent in the sub-network. The first and last time of the network is recorded during preprocessing in Subsection 4.1.2. The code for the recording is seen in Code 4.5 and shows that the "Time" value of the first row is inserted into "first_time" since the total counter for the network is increased by 1 after processing the first row. The parameter "last_time" is the

highest recorded time. Then after all rows have been transformed the "first_time" is stored as "First Activity" and the delta value as "Time of Capture" in the *network_info_dic* as seen in Code 4.5, lines 3-4. **Used for:** The time of capture is used as visualization information and further time activity analysis of the network. For example, if the chosen network only includes the "UNKNOWN_SOURCE" and "UNKNOWN_DESTINATION", and the time of capture is equal to the time of capture of the whole network, it can indicate that a master node is sending out information to all nodes. On the other hand, if the node is only active during a certain time during the day, it can indicate some foul play in the network.

Code 4.5: Time of capture

```

if network_info_dic [("Network", "Total")] == 0:
    first_time = time

if time > last_time:
    last_time = time

network_info_dic [("Network", "First_Activity")] = first_time
network_info_dic [("Network", "Time_of_Capture")] =
last_time - first_time

```

Packages per second: The method *calculate_pps* takes as an input the timing parameters "first_occurrence" and "last_occurrence" which are the times of the first and last message sent in a network. The "total_msg" is the "Total messages" count. In Code 4.6, the code is presented. First, the delta is calculated, because there is no division by 0. The delta can not be smaller than 0, but if it is 0, it indicates only one message is sent (first and last occurrence are the same) and a pps of 0,0 is returned. The same goes for nodes which are active for a short time frame of less than 1s, where the total message count is returned. For these low active nodes, another option is to approximate the number of messages for a whole second ($1 / \text{delta} * \text{total_msg}$). But since this yields in very high and not accurate numbers, the total message count is returned. This pps is always lower than the approximation. Yet a node which sent a lot of messages in a short time, will still yield in a high pps. As an example, a node sending 3 messages in 0,1s would approximate to a pps of 30. Otherwise, the actual node pps is returned. Further, due to some rounding later in the code, a node has a pps of 0,0 if the pps is very small. But since there is no actual pps of 0,0 (if a node did not send, the sending behavior is empty and not displayed), this is kept as an indicator for very low pps. On the bottom of the code, the pps of the network is calculated with the timing information recorded in Subsection 4.1.2. **Used for:** The pps is used for visualization purposes on the network level. Further, a change in pps when comparing two networks can indicate higher network traffic or an attack, since the absolute change of messages is higher, which would lead to a bigger pps when recording for the same time interval. If for example, a network is under a DDoS attack, the pps increases as more messages are sent and therefore recorded.

Code 4.6: Calculation and use of pps

```

def calculate_pps(first_occurrence , last_occurrence , total_msg):
    delta = (last_occurrence-first_occurrence)
    if delta <= 0:
        return 0, ("Only_one_Message_selected")
    if delta <= 1:
        return total_msg , ("Just_shortliy_active_(less_than_a_second))
    return total_msg/_delta

network_info_dic[("Network","pps")]_ =
calculate_pps(first_time , last_time ,_total_msg)

```

Number of Devices: This is the number of nodes in the network. It includes all devices, no matter if they only send or receive or both. It is derived from the length of the "nodes" set from Subsection 4.1.2. **Used for:** The purpose of the number of devices is on one side information for the visualization. On the other hand, it can be used when comparing networks to identify a change in participating nodes.

Number of Individual Networks: This calculates the number of individual networks in one big network. This is the only information which can only be calculated after the network is created. **Used for:** This information is used for visualizing the network-related data. Additionally, a list of lists of nodes which make up the sub-network is retrieved. These sub-networks can be individually shown and are particularly interesting with big networks including a lot of nodes and individual networks. The number of devices and the number of individual networks are used to calculate the average number of devices per individual network.

Node Level

The node-level information contains all data regarding one node and its communication behaviour. The behaviour is split into two parts, when the node acts as the sending device versus when being on the receiving end of the messages. On the node level, there are similar parameters as for the network level. This data is gathered in the preprocessing in Subsection 4.1.2 and is mainly used for the labelling of the node. The retrieved information counts either some specific type of occurrences in the traffic data, e.g. total messages sent, or tracks some timing behaviour, e.g. pps.

Total messages sent / per frame type: This counts the total number of messages where this node acts as the sender either in total or per frame type. This information is retrieved with the node ID from the *node_src_dict* alongside the individual frame type or "ALL" if it is the total count. **Total messages received / per frame type:** This is the exact same behaviour as before, the only difference is the dictionary *node_dest_dict* storing the receiving behaviour. **Total messages sent and received:** This is the addition of the two retrieved values only for the total count. The individual frame types totalling over sent or receive behaviour give no further insight into the topology and are not calculated. **Used for:** This information is used for labelling the nodes. When comparing multiple networks, a significant change in one or both of the counts of messages can indicate higher network traffic at this node or some sort of attack.

Packages per second: Is calculated exactly the same as on the network level in Code 4.6. The only difference is that here the node has two pps: one for sending and another for receiving. Therefore for both calculations, three values are needed. The total messages per behaviour is a value already present. To get the values for "first_occurrence" and "last_occurrence", the two dictionaries *time_list_sent* and *time_list_received* are consulted with the respective node ID. **Used for:** The timing information is used for the labelling of the nodes. Additional node activity behaviour is analyzed by the timing behaviour. For example, a change in total messages could also indicate a crashing node. The pps is used for the same purpose as on the network level.

Special Nodes

This project also has the option to only visualize or highlight certain nodes or sub-networks. This can become handy in the case of a lot of nodes. The visualization approach has no maximum node criteria, it can handle an unlimited amount of nodes. This means that the readability and informative value of the topology suffer, the more nodes are present in a network. Therefore, it is possible to select individual nodes, such as the "UNKNOWN_SOURCE" device. The following option or sub networks can be retrieved and visualized:

- "UNKNOWN_SOURCE" and "UNKNOWN_DESTINATION"
- Nodes with the n highest or lowest pps
- Nodes that only send messages (With "UNKNOWN_SOURCE")
- Nodes that only receive messages (With "UNKNOWN_DESTINATION")
- Individual networks
- Custom nodes

While the "UNKNOWN" node devices are selected manually as they are already known, the other sets of nodes are returned by the algorithm. The custom nodes can be known in advance (malicious nodes) or after inspecting the topology (by scrolling through the topology). The lists with device IDs belonging to either one of the subsets can be used twofold.

Code 4.7: Coloring of Special Node

```
devices_to_highlight = []
boolean_which_devices_to_highlight =
[True, True, True, False, True, True, True]
custom_nodes_to_highlight = ['example_device']
colors =
['blue' , 'yellow' , 'yellow' , 'yellow' , 'yellow' , 'violet' , 'orange']
```

Firstly, the user has the option to highlight none, certain, or all special device sets as seen in Code 4.7. This feature enables the location of certain node groups or specific devices in a big topology by highlighting them, rather than having a user search through the interactive topology. There are seven device group options to color. These device lists are identified and added in order to the list in line 1: custom nodes, highest sending pps, lowest sending pps, highest receiving pps, lowest receiving pps, only nodes that send, and only nodes that receive. To select the nodes to color, the user can just change the boolean variable from *True* to *False*. In the example, the device set with index 4 (highest receiving pps) is not colored. Beforehand, the user has the option to add any device ID to *custom_nodes_to_highlight*, as indicated by the value *example_device* in line 3.

Lastly, a color can be selected, the base configuration is 'darkblue' for the custom nodes, yellow for all nodes with extreme pps, violet for sending nodes, and orange for receiving nodes. All groups can be selected as *True*, even if a device list is empty. In the actual coloring step, an additional node color 'darkgreen' is assigned to nodes fitting into multiple device groups.

The group set is defined by selecting in a boolean *boolean_which_devices_to_highlight* (True if node set is to be colored) the correct position. A color can be assigned to each special node and the list is passed to the visualization approach, where the node is colored. A user further has the option to visualize a subset of nodes by assigning the list with the node IDs as "devices_to_select" which acts as input to the visualization method. If no nodes are specified as input, "ALL" is assigned resulting in the complete network topology. The network node computations refer to the selected subset of nodes. The code then visualizes the selected nodes and their respective communication partner. The code in Code 4.8 shows an example assignment of all nodes on line 1. Line 2 is chosen if a user only wants to inspect the "UNKNOWN_SOURCE" device while on line 3 the option of selecting the "UNKNOWN" nodes is presented.

Code 4.8: Special Node Assignment

```
devices_to_select = [ 'ALL' ]
devices_to_select = [ 'UNKNOWN_SOURCE' ]
devices_to_select = [ 'UNKNOWN_DESTINATION' , 'UNKNOWN_SOURCE' ]
```

The node with the highest or lowest pps is found by extracting the node ID with the maximum or minimum pps from the timing lists. Since there are two lists, *time_list_sent* storing the sending behavior and *time_list_received* capturing the receiving behavior of the nodes, four different values are retrieved. The code for extracting the node ID with the highest sending pps is shown in Code 4.9. The ID is found by first getting the highest pps. This project allows for selecting a multiple number N of extremas by setting the input value "nr_extremas" to the desired value (default is 1). The pps per behavior are extracted and stored in an array. In this case, the n largest pps are extracted and stored on line 3. Line 4 shows the extracting of the information to display by checking for every device ID if the device pps is in the list of n highest pps. This approach uses the lambda function because this implementation allows for multiple node IDs to be selected if multiple nodes have the same pps.

Code 4.9: Retrieval Node Ids with highest pps

```
time_list_sent_as_array = np.array(list(time_list_sent.values()))
list_pps_sent = time_list_sent_as_array[:,2]
highest_send_n_pps = heapq.nlargest(nr_extremas, list_pps_sent)
device_ids_pps_highest_send =
[f"{device}_with_pps_{round(key[2],4)}"
for device, key in time_list_sent.items()
if key[2] in highest_send_n_pps]
```

This list can be assigned to "devices_to_select" in order to visualize the identified nodes. Extracting the n largest or smallest pps can be useful when comparing multiple networks. A relatively high pps, for example, can indicate a malicious node performing a man-in-the-middle attack, as it is assumed that said node pps belongs to extrema. For presentation purposes, the pps is rounded to two digits in the topology. For the actual comparison, the pps up to the fourth digit is presented, as there can be multiple devices matching a certain pps up to the second digit but not necessarily have the same value.

To identify the devices in the network that only send or receive messages, the two sets *unique_nodes_sent* and *unique_nodes_received*, recording the communication behavior of the nodes, are used. This can be done by using the set difference calculation. To extract the node IDs of only sending devices, the set difference of *unique_nodes_sent* to *unique_nodes_received* is used. For identifying the receiving devices, the opposite set difference is used. The code for set membership is listed in Code 4.10.

Code 4.10: Retrieval Node IDs which only send or receive

```
node_ids_only_send = unique_nodes_sent.difference(unique_nodes_received)
node_ids_only_receive =
unique_nodes_received.difference(unique_nodes_sent)
```

In networks with a high number of devices, individual sub-networks may form. In order to portray the sub-networks, the devices forming them need to be known. This is achieved by the code in Code 4.11 which returns the list of nodes. Further returned is the number of individual networks. This is done by using the NetworkX library, which has a method to return both. This information is used to display in the network node. If the number of individual networks is one, all nodes are connected and there are no individual sub-networks.

Code 4.11: Retrieval Node Ids of sub Networks

```
def calculate_number_individual_networks(communication_data):
    G = nx.DiGraph()

    for key, value in communication_data.items():
        device1, device2, frame_type = key
        G.add_edge(device1, device2, weight=value)
    connected_components = list(nx.weakly_connected_components(G))

    return len(connected_components), connected_components
```

Time Activity Nodes

An additional timeline recording the timing behavior of the devices in a network is generated. This timeline is used to analyze when a node is active during the time of recording of the data set. Compared to the time calculation in the topology, this timeline does not round any numbers or changes to another format. On the y-axis, the nodes involved are listed. The x-axis stores the timing information, starting at 0.0 seconds and ending with the time of capture in seconds. The first entry in blue in the plot is the network ID and the time span of the bar is the time of capture or the run time of the experiment. The node IDs are then all listed separately and can have up to two bars. The first bar in red represents the receiving activity and the green bar shows the sending behaviour. For example, a red bar of a device starts when the first message is received and ends when the last package is sent to this node. When hovering over a bar, the time of the first event and the activity span of the node is given. An example timeline is given in Figure 4.5. The network was online for 600 seconds. Device1 was sending and receiving messages throughout the whole run time. Device2 was also sending and receiving messages, but compared to Device1, not the whole time. Device3 is an example of a device only sending and being active in the first half of the experiment since there is only half a red bar. Device4 only receives messages during the second half of the experiment as the green bar is only present after 300 seconds have passed. This timeline is used for the evaluation of the topologies and for large networks, only selected nodes are included.

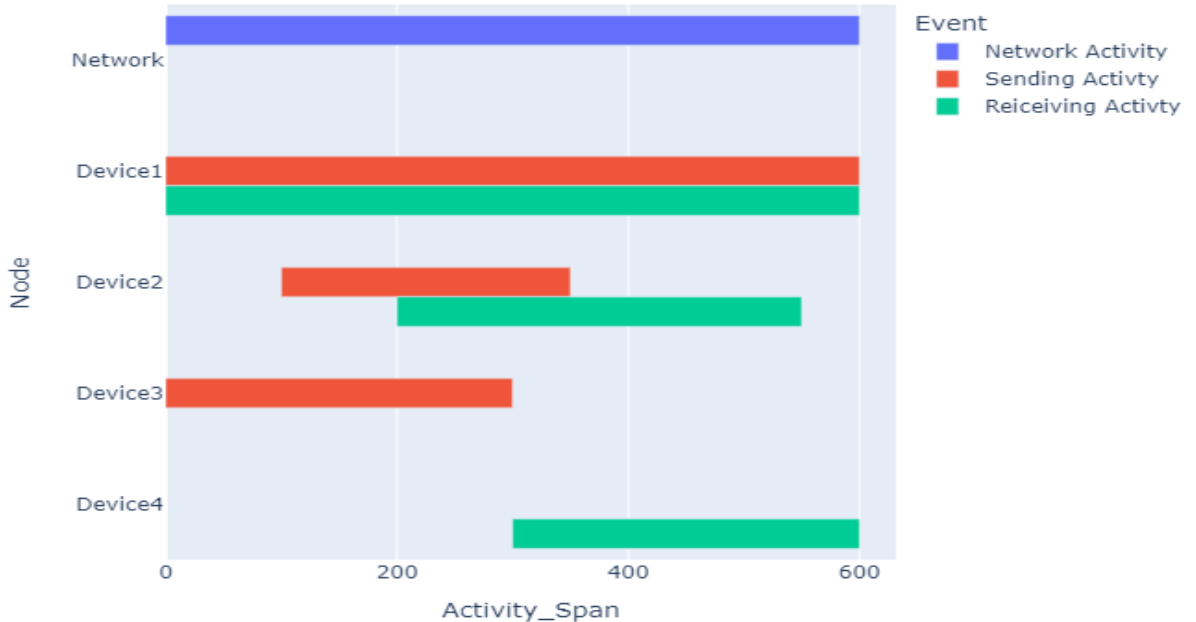


Figure 4.5: Example Activity Timeline

Network Comparison

The network comparison part allows for the processing of multiple networks together and identifies the shared devices.

For processing multiple networks, this approach has two options. First, multiple paths to different csv files can be stored and visualized together. Further, a list containing multiple sub-networks in the form of "devices_to_select" can be defined. For example `list_devices_to_select = ["ALL", "UNKNOWN_SOURCE"]`, calculates in one step the measurements for the whole network while in a second step, only the ["UNKNOWN_SOURCE"] node and its communication partners are looked at. As a result, four combinations are possible. The basic option is to select one data set and one set of devices to display, e.g., all nodes of one network. Then for one network by choosing multiple "devices_to_select", a network and its sub-networks can be compared. This can be useful to identify nodes that do not communicate directly with certain devices. For example, to find out which devices have not communicated with a malicious node.

The third option is to compare two network files with only one set of "devices_to_select". This configuration can be used for comparing a network data set recording an attack versus normal behavior. The last option is to compare multiple data sets with each other as well as the sub-networks of these main networks. This approach can be used for example to display two networks, at different times, in combination with first showing the whole network, while in a second step only visualizing the networks of the "UNKNOWN" nodes. For each of the networks, a graph and the calculations are displayed.

Further, the total number of unique devices in both networks combined is displayed. This information can be used to check whether or not two networks are formed by the same devices if the count of unique device IDs of both networks is the same as the individual networks. Another addition is the identification of nodes that only appear in one network or in every network. If for example some network members use a dynamic naming schema and change address every minute, and the network is sniffed for an hour, the topology will end in a lot of additional nodes. By sniffing the network traffic on two different occasions, the devices not appearing in both data sets, can be identified as having a dynamic naming schema and devices appearing in both sets have a static address (only if the number of participating devices stays the same). If only two topologies are compared, the addition of these two is equal to the unique devices, since the devices are either in one network or in both. On the other hand, if no devices are in both networks, the number of unique devices combined is the addition of the device count of each network.

The code for calculating network comparison measurements is shown in Code 4.12. The function calculates three different set memberships to find out the number of devices and their IDs belonging to the respective set. Starting with the union of all nodes over all network files returning the unique node IDs. The second is an intersection to check for nodes appearing in every network. The last set contains all node IDs that only appear in one network. This is achieved by finding the set difference for each specific network with the other networks and storing the nodes only appearing in the specific network. The set of node IDs is then displayed with the number of set members derived. The method displays this information in the terminal.

Code 4.12: Network Comparison

```

same_nodes = set.intersection(*unique_nodes_list)
all_nodes = set.union(*unique_nodes_list)

index_counter = 0
different_nodes_nw = []
for specific_nw in unique_nodes_list:
    index_list = []
    index_list.extend(range(len(unique_nodes_list)))
    index_list.remove(index_counter)
    all_other_nodes_list = list(itemgetter(*index_list)(unique_nodes_list))
    all_other_nodes = set.union(*all_other_nodes_list)
    different_nodes = specific_nw.difference(all_other_nodes)
    different_nodes_nw.extend(different_nodes)
    index_counter += 1

```

Method	Loop	Time Complexity
<i>analyze_csv()</i>	for row in reader:	$O(n_m)$
<i>calculate_node_pps()</i>	for device in nodes:	$O(n_d)$
<i>visualize_topology()</i>	for device in nodes: for ft in frame_types:	$O(n_d * n_f)$

Table 4.1: Time Complexity Analysis

Algo Level

In this section, the methods containing loops are subject to a time complexity analysis. The time complexity of a method calculates the time of execution based on the input size. A loop that goes over every element of the input with size n has the complexity of $O(n)$. There is additional time complexity of methods that do not enter a loop. This additional component is a constant and is not part of the complexity calculations. This analysis is based on running the algorithm with one input data set and one set of devices to select. If there are multiple data sets or devices to select, the time complexity grows by the factor of the chosen input configurations or per topology analysis. In total, there are six methods. Out of the six methods, three methods enter a loop at some point and each of the methods is performed in a given order. The methods are named in Table 4.1 and the for loop is written in a second column. Two out of the three methods enter exactly one for loop. Therefore, the time complexity is equal to the input size of the looped elements. These two methods are *analyze_csv()* and *calculate_node_pps()*. For the *analyze_csv()*, the loop iterates over all rows in the given input csv file. Each row represents one message sent in the topology and the count is given by n_m . The loop of the second method *calculate_node_pps()* goes over all unique devices in the network. The complexity is derived from the iterating input hence the parameter n_d is introduced for the time complexity of all unique devices.

The only method iterating over two loops is the *visualize_topology()* method. This method iterates once over all unique devices (n_d) in the network and once over the occurring frame types. The number of frame types is associated with n_f . The maximum number of possible frame types is given by the standard. The dominant term of this method in most cases is the number of unique devices as they exceed the number of frame types.

4.1.4 Visualization Approach

In the visualization approach, all information collected in the dictionaries from Code 4.2 is presented. In the visualization approach, the information is either represented as a line or as a node. Since this project handles data sets with an unknown number of nodes and connections, the selected library needs to be powerful enough to handle an unlimited amount of nodes. In the first step, the library selection is explained followed by design choices for the selected library.

Library Selection

For the choice of the visualization library, there are multiple requirements. First, the tool needs to be powerful enough to visualize a high number of nodes in a user-friendly way. This includes a clear overview and structure of the topology as well as some interactivity such as scrolling through the nodes. Another factor is a labeling option to display node and edge level information as well as coloring of nodes which does not lead to over cluttering. Further, this tool should be easy to use and hence a python library is preferred. The following options were researched:

- NetworkX ([58]):
- Superset [59]
- matplotlib [60]
- Folium [61]
- pyvis [62]

The first tool not considered is Superset as it runs externally and needs a connection to another database. For this tool, the installation alone consists of multiple steps and is therefore not considered. The resulting four tools are all Python libraries. In order to choose a library, the same topology was created with each tool. NetworkX and matplotlib are not powerful enough and have a lot less functionality than the other two. This means, that when trying to visualize a topology with either tool, the resulting network does not give a good overview and over clutters easily. There is also no user interaction possible. The resulting two libraries are pyvis and Folium. Folium is a powerful tool, but for the best result, an underlying picture is needed, referencing the position of the objects. As this project does not work based on this kind of information, Folium is not an option. Pyvis is selected as this library fulfills all necessary requirements.

Design choices

There are two options on how to visualize the collected information: as a node or as an edge. Every device that occurs in the network is represented as a node in blue. Each node is labeled with the data from Subsection 4.1.3.

The node label for the non-network nodes contains the following information for sending and receiving behavior:

- Node ID
- Total messages sent / received
- pps sent / received
- First message sent / received
- Last message sent / received
- Message per frame type sent / received
- Total messages sent and received

The network node label contains the following information for the selected nodes:

- Total messages in network
- Total message per frame type in network
- Time of capture
- pps network
- Number of devices
- Number of individual networks

The listed parameters are all extracted and displayed, except for the timing information. While the calculations and timing span are based on seconds, the displayed time is changed into the format hh:mm:ss for reader friendliness.

An additional node color of black is introduced for the "UNKNOWN_SOURCE" and "UNKNOWN_DESTINATION" node IDs. A special network node in the color pink is added to the topology, labeled with the information from Subsection 4.1.3. Master nodes are further colored green, and identified if a message of frame type *Beacon* was sent by a node. If the "UNKNOWN_SOURCE" device is the sender of a beacon message, the color black is assigned, since the "UNKNOWN_SOURCE" can also send packages of other frame types. These node colors have a lower priority than the colors of the selected subset to color in 4.1.3, for example, if a "UNKNOWN" node has the highest pps, the color of the node is the color selected for the subset and not black. The edges contain the information specific to one communication pair from the example in Code 4.3. The edges are directed and show the flow of the messages. The label of one edge is the number of packages between a communication pair in total as well as per frame type. Here, all frame types are listed and valued with 0 if the node does not send a message of a specific type.

4.2 Experimental Setup

In this section, the three experiments are presented to research the behavior of IoT network traffic and its topology. The foundation of each experiment is a data set. In this project, the following three traffic types are considered: Network, UWB and BLE. In this step, the different data sets and collection methods are presented. Each experiment for one data set consists of the multiple calculations and visualization approaches explained in the previous sections. This includes the network level information and node/edge level information in the topology as well as the identification of some special nodes. These special nodes (except "UNKNOWN") are retrieved by the algorithm and can further be visualized. Not all special nodes are present for every experiment. The custom node can be any device and some experiments make use of this option. Additionally, a timeline with all devices or sub-networks is generated. The last part is the comparison of multiple data sets, which will be done for each experiment since there are at least two data sets of the same traffic type per experiment.

4.2.1 Experiment 1: UWB

The first experiment is made with traffic data of type UWB. The foundation of this experiment is a data set from [63]. The paper researches different localization techniques in UWB networks. To research this question a network consisting of a few devices is set up to mimic a use case. These devices communicate by sending packages over UWB. The project captures the traffic data of the devices in order to analyze the localization approach. By manually creating a network in a closed environment by setting up a known number of devices, the network topology is known. Therefore, this UWB data set is used to validate all calculations performed in the algorithm as well as the correct visualization of the topology. For this experiment, all node- and network-level measurements are done and added to the results. The special nodes analysis is done in order to validate the code returning said nodes. The method for the comparison of two networks will also be validated with this experiment.

In the paper, two localization approaches are researched, an active and a passive version, resulting in two different data sets. In this project, the two localization approaches are not relevant and will not be explained in detail. Nevertheless, both experiments use the same amount of devices and the same topology. The difference lies in the number and type of messages exchanged in this network. For this project, only the topology analysis is interesting, therefore, both data sets are potential candidates for an experiment. For the choice of data set both data sets were loaded into Wireshark. As the objective of this experiment is to validate the topology of the visualization approach by comparing it with the known topology, one requirement is that all devices appear in the captured traffic. This prerequisite is only given for one data set of the two approaches, namely the passive one. As a result, the captured UWB traffic from the experiment of the passive localization approach is chosen as the traffic data for this experiment.

Devices and Setup

In the project, the setup for the experiments includes one master anchor, three slave anchors, and one tag to localize. The master anchor acts as the master node in this setup and is responsible for the synchronization of the clock of the other anchors in order to localize the tag. All anchors are fixed in position, only the tag is able to change position as it is the device to localize. In Figure 4.6, the basic structure and topology of this experiment are shown. The green circle is the master anchor, the blue circles are the three following anchors "A0", "A1", and "A2" and the red square represents the tag. In a passive localization system, the tag calculates its position by itself. In theory, the

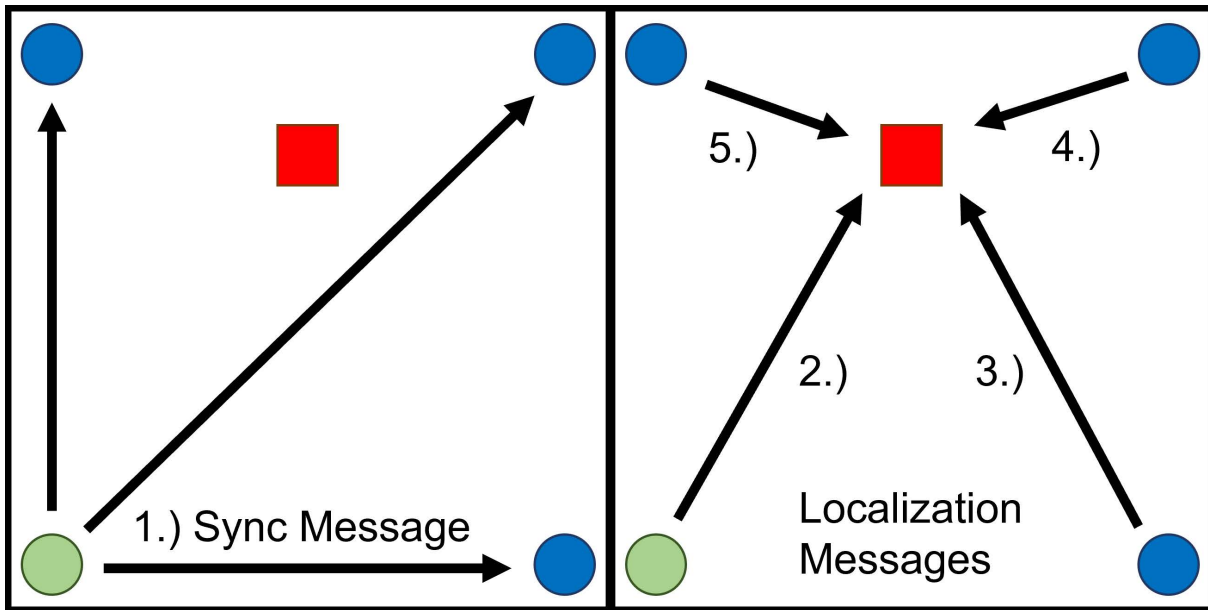


Figure 4.6: Topology in the Passive Localization System [63]

localization setup is started by all devices initializing themselves. The master anchor then sends out synchronization messages to all following anchors. In order to get the position of the tag, all anchors send out a localization message to the tag. The tag is then able to calculate its position with the help of the arrival times. This process is repeated as long as the experiment is active.

UWB Data Sets

In this experiment, the UWB data sett is used. The captured packages from this experiment are displayed in Figure 4.7. The experiment was run for 614.587 seconds and captured a total of 4'693 packages. The packages highlighted in blue are all sent by the

No.	Time	Source	Destination	Protocol	Length	Info
228	09:43:27.667485	0x304d	0x5841	IEEE 802.15.4	12	Data, Dst: 0x5841, Src: 0x304d
229	09:43:27.667485	0x3041	0x5854	IEEE 802.15.4	12	Data, Dst: 0x5854, Src: 0x3041
230	09:43:27.668480	0x3141	0x5854	IEEE 802.15.4	12	Data, Dst: 0x5854, Src: 0x3141
231	09:43:27.670806	0x3241	0x5854	IEEE 802.15.4	12	Data, Dst: 0x5854, Src: 0x3241
232	09:43:28.157427	0x304d	0x5841	IEEE 802.15.4	12	Data, Dst: 0x5841, Src: 0x304d
233	09:43:28.172992	0x3041	0x5854	IEEE 802.15.4	12	Data, Dst: 0x5854, Src: 0x3041
234	09:43:28.172992	0x3141	0x5854	IEEE 802.15.4	12	Data, Dst: 0x5854, Src: 0x3141
235	09:43:28.173985	0x3241	0x5854	IEEE 802.15.4	12	Data, Dst: 0x5854, Src: 0x3241

Figure 4.7: Experiment 1: Captured UWB Packages [63]

master anchor with the node ID "0x304d". The three other node IDs of the anchors are the "Source" field of the remaining colored packages. Anchor "A0" is associated with the address "0x3041" in yellow, "A1" refers to the orange colored line with the value "0x3141", and the red packages belong to anchor "A2" with the address "0x3241". Since this is a passive system, the tag acts as the receiver of these messages. As a result, the "Destination" field consists of the address of the tag. There are two types of addressing schemas. The anchors all have a static address, meaning all devices communicating with a certain anchor use the same address. The dynamic addressing schema occurs when packages are sent to the tag. This means that when the master anchor communicates with the tag, the "Destination" field differs from when the slave anchors communicate with the tag. This approach is configured in the setup of the experiment for security concerns. In this data set, there are two addresses for the tag recorded, namely "0x5854" when talking to the master anchor and "0x5841" for the other three. Except for six packages, all recorded rows contain both a source and destination address. The "Info" field contains additional information, for example, the frame type. In this data set, there are two frame types present, six packages of frame type "Beacon", and the rest are recorded data packages. The recorded data packages are also packages containing both source and destination addresses. This leads to six packages having no source and destination address and being of frame type beacon as pictured in Figure 4.8. In order for this data set to fit, the time configuration is changed to the start of recording. This is achieved by loading the Pcap file into Wireshark and changing the time setting. Since the input file format is csv, the packages are exported as csv files and passed to the preprocessing step.

Number	Time	Source	Destination	Protocol	Length	Info
31	0.018998s			IEEE 802.15.4	23	Beacon
32	0.020002s			IEEE 802.15.4	27	Beacon
1469	3m 8.080...			IEEE 802.15.4	23	Beacon
1470	3m 8.095...			IEEE 802.15.4	27	Beacon
2410	5m 12.653...			IEEE 802.15.4	23	Beacon
2411	5m 12.653...			IEEE 802.15.4	27	Beacon

Figure 4.8: Experiment 1: Rows with empty "Source" and "Destination" field

When looking at this data set, two inconsistencies stand out. First, the data set contains packages with no "SOURCE" and "DESTINATION" addresses. For the analysis of traffic data sets, these values are filled in as explained in Subsection 4.1.2. This means two additional nodes in the topology. The dynamic addressing of the tag poses the second inconsistency, as there are two addresses present for the same device again leading to one more device. The visualization approach can still portray this information but with two nodes. For any other experiment, this is no problem since dynamic addressing is a design choice of the referenced project and the "UNKNOWN" nodes add information to the topology. However, since this experiment's objective is to validate the design approach, some changes are made to the existing data set. The goal is to overcome the two inconsistencies. By doing so, the visualization approach is assumed to have the same amount of devices as the experiment deploys (no dynamic addressing of the tag and no "UNKNOWN" nodes). The first is to get rid of the two addresses for the same tag. This is achieved by changing the two destination addresses "0x5841" and "0x5854" of the tag to the value "tag". The second change happens to the six rows in Figure 4.8. It is assumed that these messages are sent by the master anchor since the frame type is *beacon* and in a UWB network, only master nodes send out beacon packages. Therefore, the empty "SOURCE" field is manually changed to the master anchors ID "0x304d". The destinations of said beacon packages are assumed to be the following anchor devices "A0", "A1" and "A2", because Figure 4.6 indicates the sending of a synchronization signal solely to all the anchors by broadcasting the message. This means that one row in Figure 4.8 is exchanged with three rows, all containing the master anchor's ID as "Source" and each of the slave anchors' IDs as "Destination". This means 12 ($6 \times 3 - 6$) additional rows. The value of the remaining fields stays the same. This is achieved by manually changing all rows containing empty values in the csv file. As an example in Figure 4.9, the row colored in red is the 31st captured package in the traffic data and is the csv file format of row 31 in Figure 4.8. The row colored in red is deleted and exchanged with the three rows colored in green. These rows now contain the master anchor as the source address and each of the slave anchors as the destination address. This is done for all six rows in Figure 4.8.

```
31,"0.018998s","","","IEEE 802.15.4","23","Beacon","",""
31,"0.018998s","0x304d","0x3041","IEEE 802.15.4","23","Beacon","",""
31,"0.018998s","0x304d","0x3141","IEEE 802.15.4","23","Beacon","",""
31,"0.018998s","0x304d","0x3241","IEEE 802.15.4","23","Beacon","",""
```

Figure 4.9: Experiment 1: Adapted Data Set Fields

In summary, the experiment with UWB traffic data consists of two data sets. One data set represents the captured traffic with dynamic addressing and "UNKNOWN" nodes. The other data set is derived from the original data by filling in empty values and updating the tag ID. Since the researched network in this case is manually implemented, the network topology is known. Therefore, both data sets are used to validate the code, while the adapted data set is further used to check the correctness of the topology.

4.2.2 Experiment 2: Network

The second experiment is based on network traffic data sets. The data sets were collected by the School of Engineering and Information Technology (SEIT) and stored in a cloud (<https://cloudstor.aarnet.edu.au/plus/s/ds5zW91vdgjEj9i?path=%2>). The SEIT designed and implemented a large IoT network in a lab. With this network, the lab tested diverse attacking scenarios and recorded the traffic. The SEIT named the collected data sets "ToN_IoT" which stands for "Telemetry data sets of IoT and IoT sensors, Operating systems data sets of Windows 7 and 10 as well as Ubuntu 14 and 18 TLS and Network traffic data sets" because the setup included many different data sources. The cloud path stores multiple folders. There is no information regarding the topology, e.g. the number of devices in the network, there is just some information concerning the individual data sets and the components of the network. It is also not indicated, which protocol is used exactly hence the experiment is called "Network". The data sets were captured with Wireshark, it is thus assumed that the network contains any communication sent over the WLAN standard and that it can be used to test the project's approach for the overlaying network. The folder contains raw and preprocessed data sets. For this project, the raw data sets are chosen, because the preprocessed folder contains sets that have been altered. The preprocessed data sets have been enriched with additional information (since the components of the lab were known, the captured addresses were changed) and some information has been deleted (the timing information is not present). This is assumed, as there is no explanation present regarding the preprocessing steps. Since this project follows a uniform approach and the preprocessed data sets do not represent the real-world captured traffic, the raw data sets are used to analyze network traffic with diverse protocols. Within the raw data set, the project has captured traffic on different levels. There are IoT, Windows, Linux and Network data sets. The IoT folder contains information on the end device, for example, the temperature of an IoT device. This folder contains no network-relevant information and, is thus not selected for this project. The Linux and Windows folders contain data storing values regarding the operation system, for example, usage of memory, and are hence not applicable to this project.

These raw network data come twofold. As the project researches the behavior of networks under attack, two data sets are chosen to analyze and visualize. One network that is communicating normally and one network that is under attack. For the second network, there are a total of nine hacking attacks staged and the first requirement to the data set for the experiment is that "Source" and "Destination" field value is not null. Since a lot of addressing happens in the preprocessing step, not all data sets under attack are feasible. The attacks contain for example a network under DDoS attacks, but since no "Destination" address has been captured, these kinds of data sets are not selected. Another nice feature is the identification of the malicious nodes in the network under attack. The folder gives a short overview of the attacks and an MAC address for malicious nodes. This MAC address is not captured in all data sets. For some data sets only the MAC addresses were stored by Wireshark. One data set of an attack, namely the Man-in-the-Middle (MITM) attack, fulfills both requirements. In a network under an MITM attack, nodes pose as a member of the network and intercept and forward messages. In the description of the attack, the two IP addresses 192.168.1.31,34 are listed as the malicious nodes. Since Wireshark only records MAC addresses, the data sets were loaded into Wireshark, and

the packages were checked for an IP address. This was done by going over packages and checking for a source or destination IP address matching the malicious nodes as seen in Figure 4.10. The device with the sender IP address 192.168.1.31 belongs to the captured MAC address 00:0c:29:5b:73:5b. To further emphasize the different naming in Wireshark, the IP address in this file is retrieved by calling `arp.src.proto_ipv4 == 'IP address'`. By filling in the IP address ending in .34 the MAC address 00:0c:29:b8:b1:74 was identified for the second malicious node. Here, four files were an option and the data set used for the experiment was selected randomly, namely MITM_normal1.pcap.

For the selection of the data set of a network not under attack, thirteen different Pcap files were an option. Here, multiple files were tested and all data sets fulfilled the requirement of having source and destination addresses. Therefore, one data set was chosen randomly, which was normal_13.pcap. Since both network data sets were stored as Pcap files the first step was to load them into Wireshark and change the timing configuration to milliseconds since the start of recording and downloading the files as csv.

```
. . . . .  
Sender MAC address: 00:0c:29:5b:73:5b  
Sender IP address: 192.168.1.31 (192.168.1.31)  
Target MAC address: a4:91:b1:1e:57:90  
Target IP address: 192.168.1.1 (192.168.1.1)
```

Figure 4.10: Experiment 2.2: Identification IP Address

In summary, two network data sets in raw form are identified, one under attack, and one normal. Both data sets were recorded in the same lab environment. It is not clear, which protocols were involved because the SLL capture mode is deployed in this experiment. However, since MAC and IP addresses are present, it is assumed that only protocols performing end-to-end communication are captured. All calculations and visualization options are performed in this experiment. The normal data set recorded a large amount of devices and messages. It was hence used as a measurement to confirm the approach's ability to handle a large number of devices. Since there are multiple protocols involved, the data set is also used to validate the uniform approach. The network under attack is used to have a reference point as to what such a network looks like and if the approach can identify malicious nodes with the measurements present. Further, these data sets allow for a comparison between a network under attack and one not being attacked. The network under attack is also used to identify nodes that do not communicate with the malicious nodes. For this, the experiment extracts a sub-network only containing the malicious nodes and their direct communication partners. By comparing the device IDs of the sub-network with the whole network, the nodes are identified.

4.2.3 Experiment 3: BLE

In experiment 3 the objective is to visualize BLE network traffic by analyzing real-world data sets collected by a sniffing setup as explained in Subsection 4.1.1. The data sets are collected by running the same experiment for three different time periods. The time periods are small (around 1h), medium (around 6h), and big (around 24h). The environment of the experiment is always the same. The setup is located in Wiedikon, Zurich and the start of each experiment is around 12.00 p.m.. Further, each experiment is conducted on a weekday. The resulting data sets are the base for further analyzing BLE network traffic. As these data sets are the only ones based on real-world traffic, it is possible to compare the resulting network topologies and involved devices over different time periods.

Chapter 5

Results

In this section, the results of the three experiments are discussed. For each experiment, multiple topologies and sets of special nodes are generated. A table at the beginning of each experimental result indicates the presented network topologies and sets. In the tables, an "x" marks the presented results, while all results filled with "-" yielded in nothing, for example if no source address is empty in a data set, the "UNKNOWN_SOURCE" field is "-". The number inside the brackets for the highest and lowest pps gives the number of extremes selected. Not every generated graph is presented, as there are too many and not every graph has an informative value.

5.1 Result 1: UWB Data Set

This part presents the results of the experiments with the UWB data sets. The Table 5.1 gives an overview of the conducted experiments and calculations of the UWB data sets. For both experiments, the topology and network level information is presented. Additionally, as the results to the first UWB experiment are used to verify all calculations, a random communication pair is chosen to validate the node and edge level information. All the special nodes sets were generated and are listed for both experiments. The adapted UWB set contains no "UNKNOWN" nodes while the raw set contains both. The number of extremas for the most and least active nodes is 1 for both experiments, as only a few devices are recorded. To verify the correct retrieval of a sub network, the topology involving each of the "UNKNOWN" nodes and their direct communication partner are presented for experiment 1. No nodes were highlighted, as the topology generated are clear and informative enough due to not a lot of recorded devices. The network activity of the nodes will be shown as well.

	UWB	
	1.1 raw	1.2 adapted
Main Topology	x	x
Network Node	x	x
Special Node Sets		
UNKNOWN_DEST	x	-
UNKNOWN_SOURCE	x	-
Max / Min pps	x (1)	x (1)
Only Send / Receive	x	x
Ind. NW	x	x
Custom Sub Topology	"UNKNOWN" nodes	
Activity Timeline	x	x
Node Highlights		

Table 5.1: Experiment 1: Analysis

5.1.1 Result 1.1: Raw UWB Data Set

In this section, the results of the unchanged UWB data set are presented and explained in detail. In a first step, as mentioned in Figure 4.7, the data set containing the dynamic addresses of the tag is loaded into Wireshark and the time configuration is changed to seconds since recording as seen in Figure 4.7. As the data set was recorded in Wireshark, the file was stored as a Pcap file in the original experiment. Therefore after changing the time configuration, the data was extracted as a csv file before loading the content into the preprocessing and visualization step.

Number	Time	Source	Destination	Protocol	Length	Info
1	0.000000s	0x304d	0x5841	IEEE 802.15.4	12	Data, Dst: 0x5841, Src: 0x304d[Malformed Packet]
2	0.001999s	0x3041	0x5854	IEEE 802.15.4	12	Data, Dst: 0x5854, Src: 0x3041[Malformed Packet]
3	0.003001s	0x3141	0x5854	IEEE 802.15.4	12	Data, Dst: 0x5854, Src: 0x3141[Malformed Packet]

Figure 5.1: UWB Time Configuration

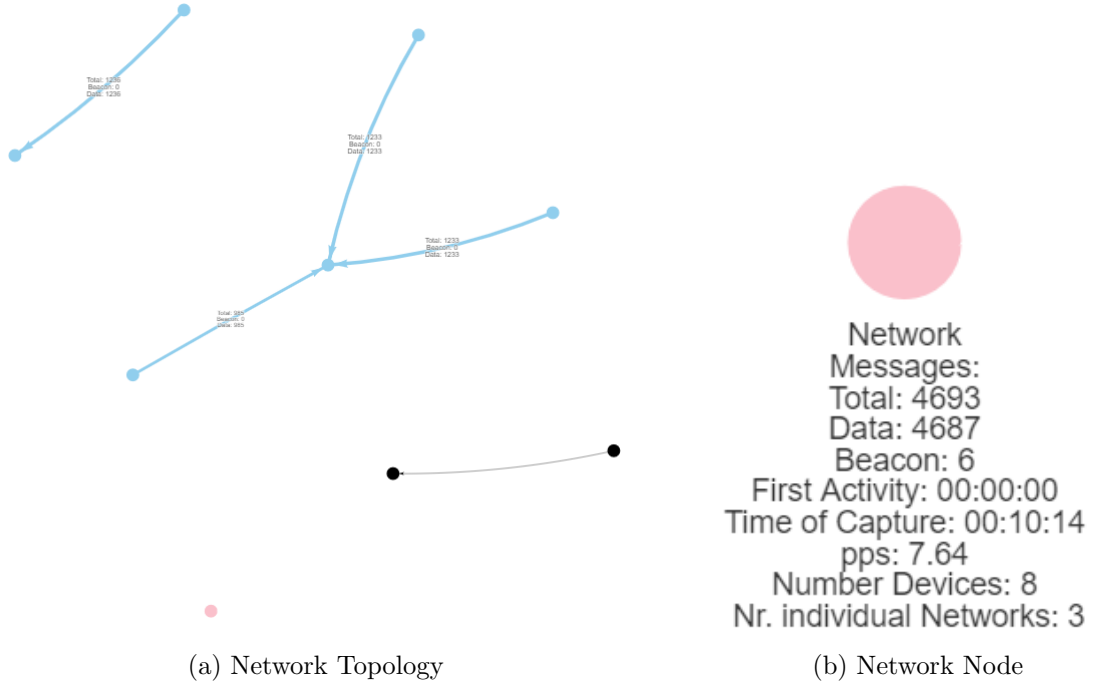


Figure 5.2: Result 1.1: Network Level

The high level topology of the UWB data set can be seen in (a) in Figure 5.2. At first, the topology consists of 9 dots. They represent the total number of devices in a network and one network node. The pink dot presents the network node, which is labeled with network information and is not part of the actual network resulting in no edge connections to other nodes. The actual network has 8 communicating devices. The nodes in blue are normal devices while the two black dots are the "UNKNOWN_SOURCE" and "UNKNOWN_DESTINATION" nodes, which are added by the preprocessing step in case of an empty "SOURCE" and "DESTINATION" field value. In this UWB data set both special nodes appeared but only communicate with each other. The blue devices communicate with each other, but still are implementing two individual networks. According to the edges, each node either sends or receives a message, as there are only edges in one direction.

Network Level information is provided at the network node as seen in Figure 5.2 (b). The information is retrieved by scrolling to the designated dot. In this experiment, the network exchanged a total of 4693 messages, consisting of 4687 data messages and 6 beacon messages. This means that in this network only two types of frames were captured. The run time of the experiment is 10 minutes and 14 seconds. This leads to a network pps of 7,64. The number of participating devices is 8 and the network consists of 3 individual networks.

Node and Edge Level information is shown at a random blue dot, one for a device which only sends messages and one device only receiving messages. For this the sub network with 2 involved devices is chosen as it includes the device with node ID "0x304d" which only sends packages and on the receiving end device "0x5841" as seen in Figure 5.3. Since this two devices only communicate with each other, the values are expected to be the same as the sending behaviour of one node is the receiving behaviour of the other. The device "0x304d" send a total of 1236 messages all being of frame type data. Since this network sends out also beacon type but this node has not the value is 0. This means a node pps for the sending behaviour of 2.01. Additional timing information is shown, in this case the node was active from the beginning of capture since the value is 00:00:00. The last message was sent after 10 minutes and 14 seconds, meaning this node was active for the same time period as the network. Total messages sent and received is the addition of the total Msg sent and total Msg received, in this case the received value is 0, therefore the total is equal to the sent value 1236. The receiving node "0x5841" has the following receiving behaviour. The node received a total of 1236 messages, all of which are of frame type data. As the device active from the start of recording and was active for 10 minutes 14 seconds, the receiving pps for this node is 2.01. The edge between the two dots show again the total number of messages between this communication pair, which is 1236 packages. This is equal to the total messages of both nodes, which is expected since they only talk with each other.

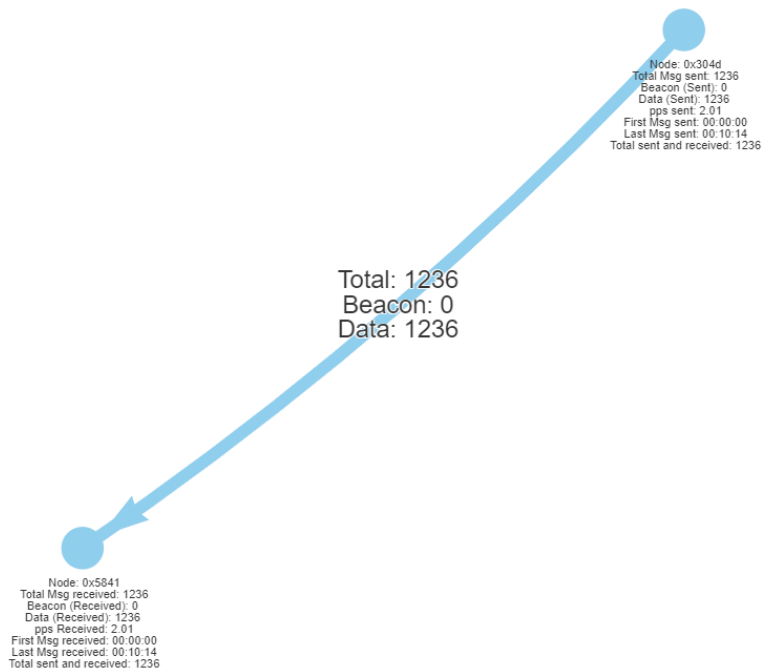


Figure 5.3: Result 1.1: Node and Edge Information

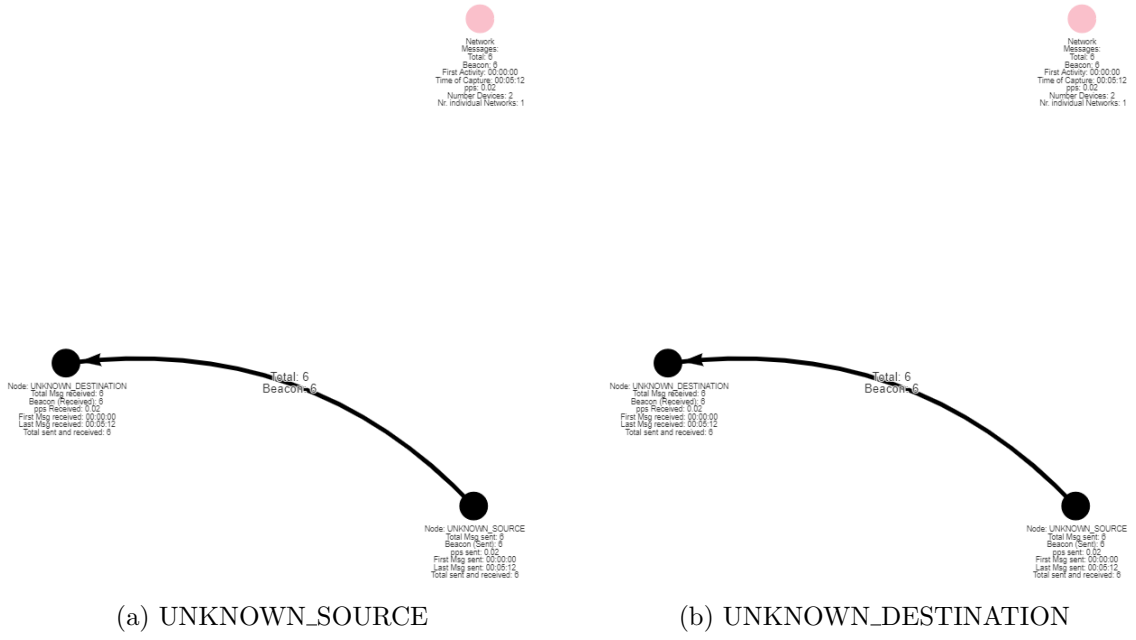


Figure 5.4: Result 1.1: "UNKNOWN" Nodes Topology Retrieval

Special Nodes can also be retrieved in form of a set. For this experiment, all nodes were retrieved and visualized. In this section, one example visualisation of the special "UNKNOWN" nodes are presented. Further only the sets containing the node ID for each special node is presented, as the network is small and the visualization part is relevant for bigger networks.

The topology in Figure 5.4 (a) is returned when the special node is defined as "UNKNOWN_SOURCE". The figure shows that the all captured packages with no source address, are sent by the "UNKNOWN_DESTINATION" node. Further all 6 messages in this network are of type beacon. The "UNKNOWN" nodes were active for from 0.02 to 312.65 seconds and have a pps of 0.02. Since they only communicate with each other, in this experiment, the retrieval of the individual nodes "UNKNOWN_SOURCE" and "UNKNOWN_DESTINATION" are expected to result in the same sub network. Therefore the "UNKNOWN_DESTINATION" node is defined as device to select and the resulting topology is shown in Figure 5.4 (b). In both figures, the network node in pink now contains only the relevant information for the chosen sub network. The two nodes were active from the start of recording until 5 minutes and 12 seconds.

The next special nodes to retrieve are the devices with the lowest or highest pps. This pps is retrieved for sending and receiving behaviour. For this data set, the return of the method with default value 1 for "nr_extremas" is given in 5.1. There is only one device per extreme pps in this network and no device has multiple extreme pps. The device with the highest sending pps of 2.01 has the node with ID '0x304d'. Highest pps when on the receiving side is the device '0x5854' with more than double the amount as the node before, namely around 5.62. Having a look at the lowest pps distribution, the value for both behaviours is the exactly the same with 0.0192 pps. The node producing this pps are when sending is the "UNKNOWN_SOURCE" node and vice versa the "UNKNOWN_DESTINATION" device.

Code 5.1: Result 1.1: Node IDs with min / max pps

```

Highest sending device(s): [ '0x304d_with_pps_2.0112' ]
Lowest sending device(s): [ 'UNKNOWN_SOURCE_with_pps_0.0192' ]
Highest receiving device(s): [ '0x5854_with_pps_5.6152' ]
Lowest receiving device(s): [ 'UNKNOWN_DESTINATION_with_pps_0.0192' ]

```

The next step is to identify the nodes which only send or only receive. This is achieved by running the code and the output is seen in 5.8. In the raw UWB data the set of nodes only sending messages consists of 5 devices, namely '0x3041', '0x3141', 'UNKNOWN_SOURCE', '0x304d', and '0x3241'. On the other hand the topology has 3 devices only receiving messages and never actually send one. These node IDs are 'UNKNOWN_DESTINATION', '0x5841' and '0x5854'. In total this network consists of 8 devices which only send or receive. This is further backed by the fact that all devices which send messages only send messages, as the number of devices per category is the same.

Code 5.2: Result 1.1: Count and Node IDs for sending / receiving

```

Number device(s) total: 8
Number device(s) sending: 5
Number device(s) receiving: 3

Number device(s) only sending: 5
Number device(s) only receiving: 3
Device(s) only sending messages: [ '0x3041', '0x3141',
'UNKNOWN_SOURCE', '0x304d', '0x3241' ]
Device(s) only receiving messages: [ 'UNKNOWN_DESTINATION',
'0x5841', '0x5854' ]

```

The last list of nodes generated contains sets of node IDs which implement a sub network and only communicate with the devices in this sub network. This means the node IDs of weakly connected graphs are returned as given in 5.3. This network consists of 3 individual sub networks with 8 devices being involved. There are two networks each made up by two devices. The third topology involves 4 devices. The first sub network consists of the two devices '0x5841' and '0x304d'. The "UNKNOWN" nodes build a separate network as well. The biggest individual network involves the devices '0x5854', '0x3041', '0x3241' and '0x3141'.

Code 5.3: Result 1.1: Individual Networks

```

Number individual networks: 3
nodes / ind_nw: 2,67
[ { '0x5841', '0x304d' }, { '0x5854', '0x3041', '0x3241', '0x3141' },
'UNKNOWN_SOURCE', 'UNKNOWN_DESTINATION' ]

```

Node Activity Timeline The Figure 5.5 shows a timeline of all nodes in the network and their activity span during the experiment. The capture time of whole data is visualized with the blue bar and is 615.58s. The 3 nodes 'UNKNOWN_DESTINATION', '0x5841' and '0x5854' only send messages in this network. While the 'UNKNOWN_DESTINATION' node is only active in the first half of recording and has an activity span of 312.65. The two other nodes were active for roughly the whole time, with device '0x5854' receiving messages for 614.58 seconds and device '0x5841' for 614.56 seconds. For the nodes sending these messages, it looks similar. The 'UNKNOWN_SOURCE' is active during the same time as the 'UNKNOWN_DESTINATION' node. The other four nodes '0x3041', '0x3141', '0x304d' and '0x3241' were all active between 613.57 and 614.58 seconds. The two 'UNKNOWN' nodes start their activity at exactly 0.02 seconds while all the other nodes are active from the start of recording at 0 seconds.

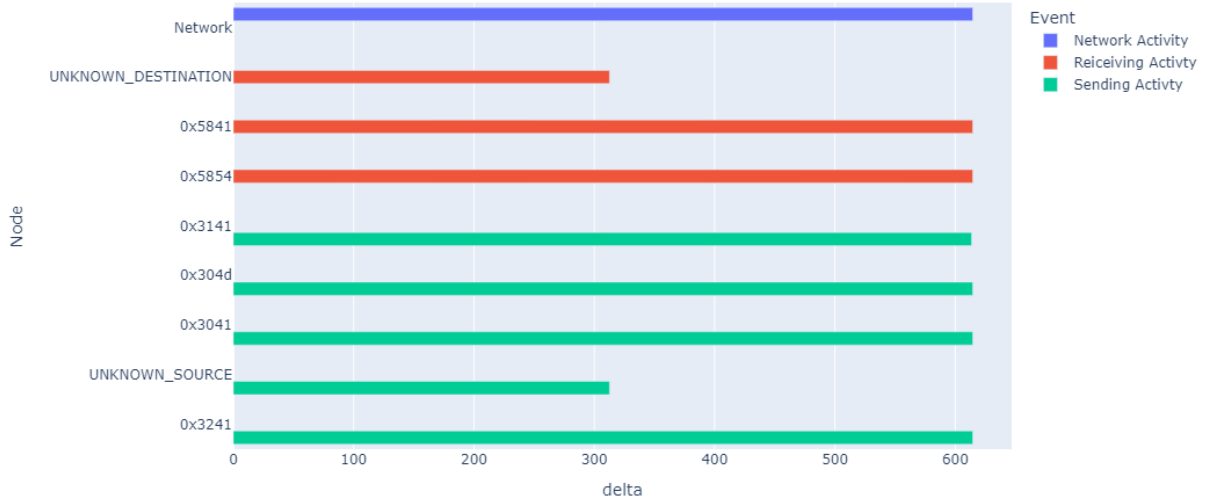


Figure 5.5: Result 1.1: Node Activity Timeline

5.1.2 Result 1.2: Adapted UWB Data Set

The network topology of the adapted UWB data set is presented in Figure 5.6 (a) and is composed of 5 dots. The green dot represents a master node, as the device sent out at least one Beacon message. The other 4 involved devices are colored in skyblue as there are no special nodes to highlight. All dots are connected, but not all devices send and receive, as not all dots have an edge to and from them.

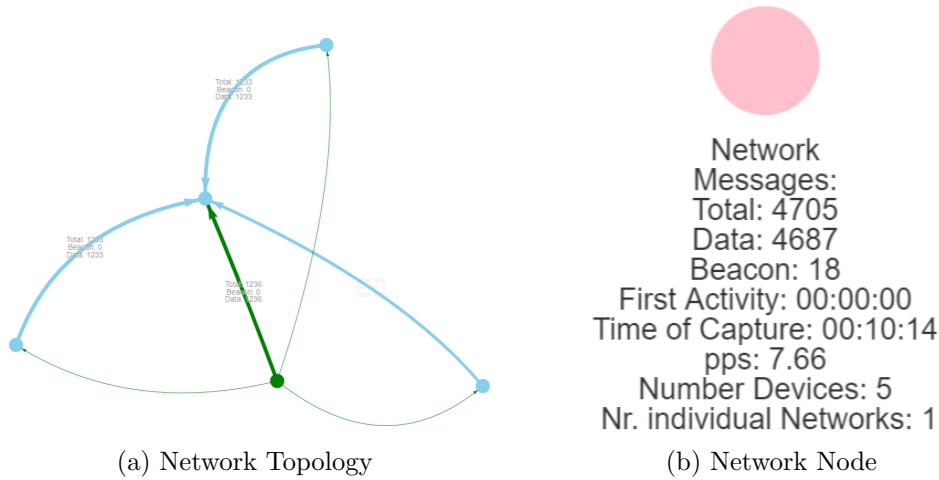


Figure 5.6: Result 1.2: Network Level

Network Level For more topology related information, the network node is consulted as seen in (b) in Figure 5.6. The traffic was captured for 10 minutes and 14 seconds. In this time 4'705 messages were sent. Out of these, 18 messages are of type Beacon, while the other 4'687 packages carried data. The messages were sent by 5 devices, which all communicate with each other, as there is only one individual network. This results in a network pps of 7,66.

Special Nodes The first node set retrieved contains all devices with a max or min pps in this topology. In this experiment, only one extrema was chosen. As listed in Code 5.4, the highest pps per behaviour and the lowest sending pps was each only matches by one device. Further each device is present in just one set, except for the device '0x3141' which has the lowest pps for both behaviours in the whole topology. The device ID set belonging to the lowest receiving pps contain three nodes.

Code 5.4: Result 2.2: Node IDs with min / max pps

```
Highest sending device(s): [ '0x304d_with_pps_2.0405' ]
Lowest sending device(s): [ '0x3141_with_pps_1.6054' ]
Highest receiving device(s): [ 'tag_with_pps_7.6264' ]
Lowest receiving device(s): [ '0x3041_with_pps_0.0192',
'0x3141_with_pps_0.0192', '0x3241_with_pps_0.0192' ]
```

In 5.5 the devices are presented, which only either send or receive. In this network, there are only two devices standing out. The first device is the 'tag' as it only receives messages but never sends and vice versa the behaviour of node with ID '0x304d'. This implies that all 4 other nodes send and receive messages, which is checked in line 2 and 3.

Code 5.5: Result 1.2: Count and Node IDs for sending / receiving

```
Number device(s) total: 5
Number device(s) sending: 4
Number device(s) receiving: 4

Number device(s) only sending: 1
Number device(s) only receiving: 1
Device(s) only sending messages: [ '0x304d' ]
Device(s) only receiving messages: [ 'tag' ]
```

As there is only one individual network, the average nodes per network is the total number of devices, e.g. 5, as seen in 5.6. The device list therefore is composed of all devices in the main network.

Code 5.6: Result 2.2: Individual Networks

```
Number individual networks: 1
nodes / ind\_nw: 5
ind nw = [ { '0x3041', '0x3241', 'tag', '0x3141', '0x304d' } ]
```

Node Activity Timeline The time activity of the involved devices is shown in 5.7. The two devices at the bottom are the two which either only send ('0x304d') or receive ('tag'). These two devices are active for the whole time during the recording of the traffic. For all devices which send or receive, a common behaviour is detected, namely all devices only receive messages during the first half of the recording. In the second half, all devices except for the tag only receive messages. This implies, that the tag is the sending device of all packages captured in the second half. This is further backed by the fact, that the tag has the highest sending pps.

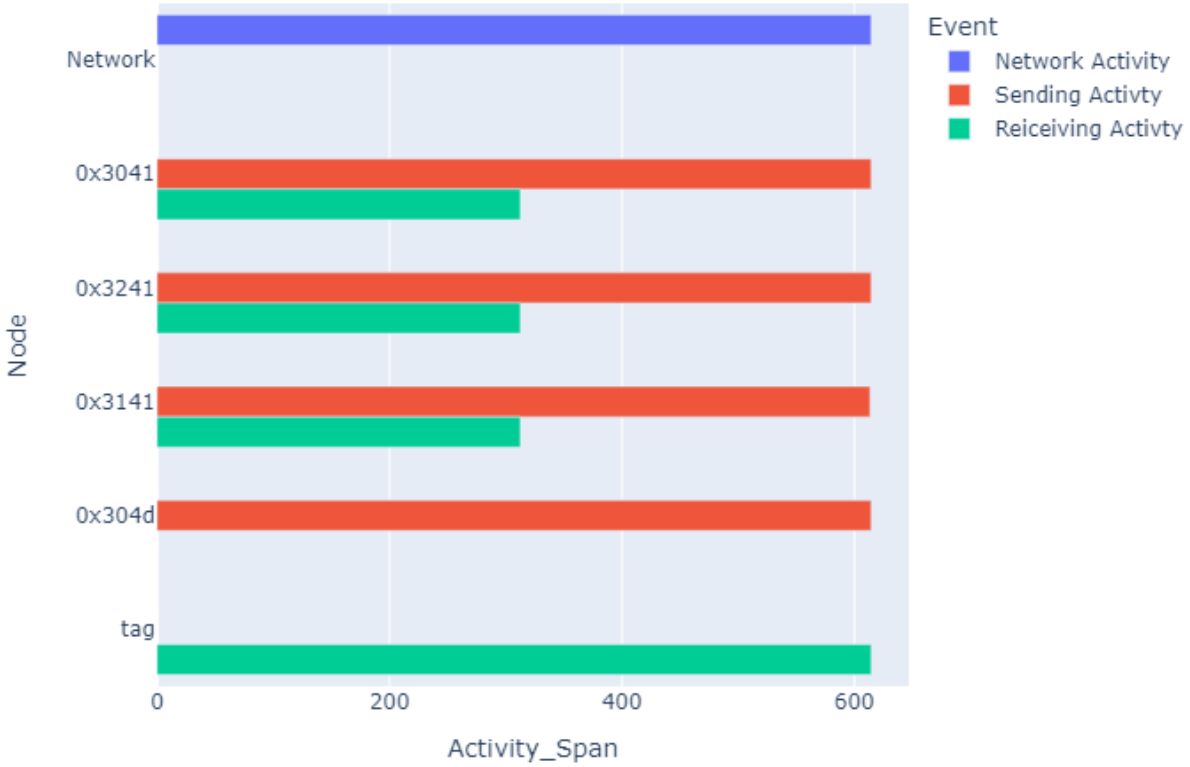


Figure 5.7: Result 1.2: Node Activity Timeline

5.2 Result 2: Network Data Set

In this section the results for the network data set are presented. To start the network topology with all nodes is shown and the network level data analyzed. Table 5.2 gives a summary of the presented results. The node and edge level will not be further discussed as the information is visualized and can be retracted by scrolling into the original communication graph. For the experiments The "UNKNOWN" nodes are further inspected if present, which is only the case for the "UNKNOWN_DESTINATION" node in the topology not under attack. Opposite to experiment 1.1, the "UNKNOWN" devices are not visualized as their own sub network, rather just as a screenshot of a node in the main network. The devices with extreme pps are extracted, in this experiment with 3 extremas. To analyse the communication behaviour the absolute number of devices per category are presented and the device IDs listed, if they have a reasonable size. There are no custom nodes in Subsection 5.2.1 as opposed to Subsection 5.2.2 where the sub network contains the malicious nodes and their respective communication partners. In both data sets special nodes are highlighted. Further a time line involving all nodes is generated for the network under attack but not presented. For the normal network, the tool is not power full enough to visualize all nodes and since no custom nodes are selected, no timeline is generated.

	Network	
	2.1 normal	2.2 under attack
Main Topology	x	x
Network Node	x	x
Special Node Sets		
UNKNOWN_DEST	x	-
UNKNOWN_SOURCE	-	-
Max / Min pps	x (3)	x (3)
Only Send / Receive	x	x
Ind. NW	x	x
Custom Sub Topology		Malicious nodes
Activity Timeline		
Node Highlights	x	x

Table 5.2: Experiment 2: Analysis

5.2.1 Result 2.1: Normal Network Data Set

In this section the results for the network data set in a normal setting are presented. The left network in Figure 5.8 shows the topology of a WLAN network with diverse communication protocols. At first glance, there are a lot of individual networks, as seen with the devices only communicating with each other. In the middle there are a few nodes forming a larger network. In a second step, devices to highlight are selected to visualize some special nodes set as seen in b). In this topology, the devices in yellow have one of the highest 3 pps of all nodes, for sending and receiving. The nodes in red have one of the lowest pps in this network topology. The same color for both behaviours is configured in order to not over color the topology. The topology with highlighted nodes shows that for a network topology involving a lot of devices, this feature enables the readability of the topology, as the user can locate certain nodes. The topology in b) shows that for example a the node rather at the bottom, has a high and a low pps. By scrolling into the topology the node ID can be identified. In the colored topology, all nodes seem to be talking with each other and there is no node forming its own network. This can be then checked by only visualizing the node groups and check the network node value for the number of individual networks if equals 1. This will not be further researched, as this network topology is only used as an example for highlighting a certain device group.

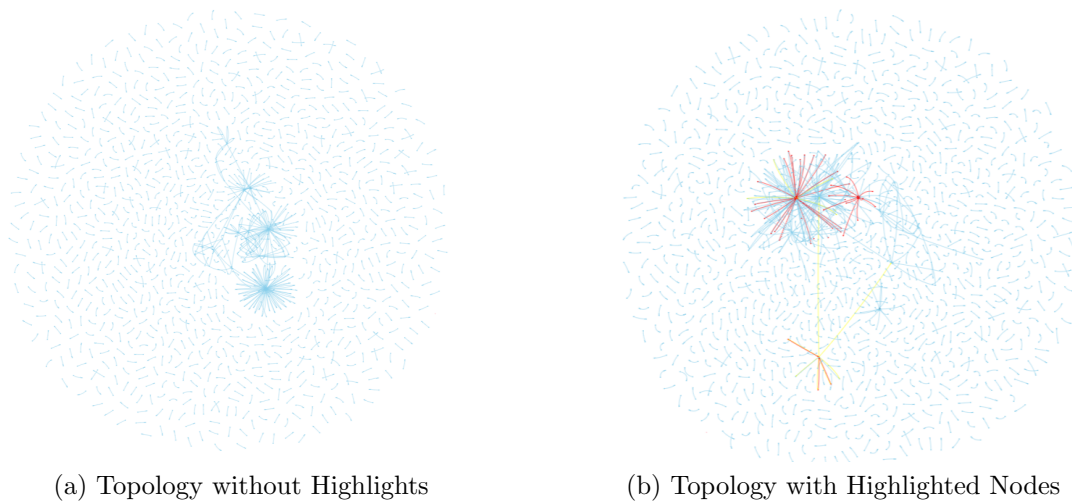


Figure 5.8: Result 2.1: Network Topology

Network Level is checked for more information, by looking at the network node in Figure 5.9 is consulted. The node shows that the communication was recorded for 25 hours and 48 seconds. Within this period, the devices sent a total of 1'918'446 messages. This lead to a network pps of 21,3. The messages were sent by a total of 2139 devices. The distribution of frame types is not uniform as 1'916'537 (99.9%) are of type "UNKNOWN". From the rest there are 16 of type "psh", 39 are data packages and the rest of 1851 have the frame type 1851 "ack". The number of individual networks are 1002, which was visually implied that there are a lot of individual device pairs.

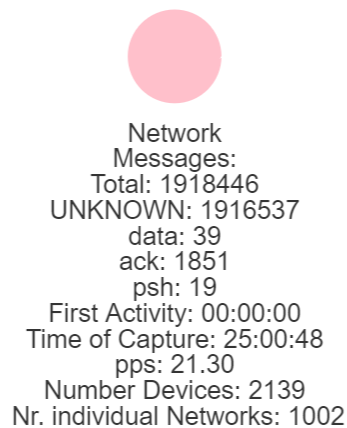


Figure 5.9: Result 2.1: Network Node

Special Nodes in this experiment are retrieved and visualized. The set of "UNKNOWN" nodes are tested for presence. In this experiment all messages recorded a source address and therefore no "UNKNOWN_SOURCE" is present in this topology. On the other hand, the "UNKNOWN_DESTINATION" node is present implying missing sender addresses in some packages. By scrolling into the network, the black dot is identified and seen in Figure 5.10. The node received a total of 311'118, which are 16.22% of all messages received in the network. The node was active for 25 hours and 48 seconds leading to a receiving pps of 3.46. Further all messages are of type "UNKNOWN".

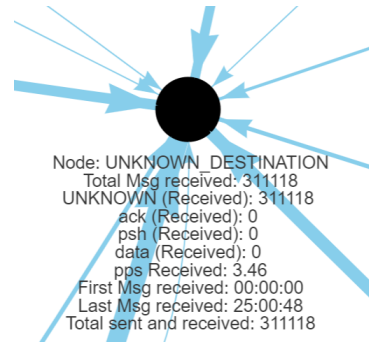


Figure 5.10: Result 2.1: "UNKNOWN_DESTINATION" node

The next nodes to analyze are part of a set with either a high or low pps. In this experiment, the number of extremes is set to 3, meaning for each of the extreme pps per behaviour, the 3 highest and lowest pps are selected and the nodes matching these returned. As the pps are rounded, some devices display the same pps but differ a small amount. In this section, both min and max pps per behaviour are presented in combination with number of devices. For the maximum pps per behaviour, also the device

Code 5.7: Result 2.1: Node IDs with min / max pps

```
Highest sending device(s): [ '2600:9000:5301:4c00::1_with_pps_200.0 ',
'2600:9000:5303:a200::1_with_pps_200.0 ',
'2600:9000:5306:6700::1_with_pps_100.0 ' ]
Highest receiving device(s): [ '2600:9000:5300:6200::1_with_pps_200.0 ',
2405:6e00:10ce:2c00:20c:29ff:fed2:b002 with pps 29.906 ',
'2a01:8840:c0::1 with pps 34.1463 ' ]
```

lists are presented as an example with multiple extremas in 5.7. The devices lists for the minimum extremas are stored and visualized but not presented, as they contain too many devices. This network contains a total of 3 nodes having one of the three highest sending pps in the network as seen on line 1 in 5.7. Two of the selected nodes have a sending pps of around 200,00 while the third highest node has half the pps with 100.00. The highest receiving pps also has one stand out node, with a pps of 200,00. The second and third highest pps are significantly lower with around 30 pps. The smallest pps recorded is 0,0 for both behaviours and all extremas. The reason for this is the rounding, which happens in the visualization stages. The calculations are performed with higher digits and therefore more concrete. For the experiment this just means that even tough all 3 printed minimum extremes are 0,0, but in reality they differ. While the minimal pps is the same

for both behaviours the number of devices matching this pps is not. While for the lowest sending behaviour, 29 devices in this network were identified the receiving minimum pps was matched by 24 devices.

For the special nodes which only send or receive, the devices lists are not presented as they are too long. In a WLAN network with a total of 2139 devices, 1085 of these device send messages. Out of these 1033 or 95.21% are devices which only send messages and do not receive any. Similarly more than half of the devices in the network receive at least one message, while 95.3% or 1054 out of those devices only receive messages and never send. This implies 52 devices which send and receive ($2139 - (1033 + 1054)$).

Code 5.8: Result 2.1: Node count for sending / receiving

```
Number device(s) total: 2139
Number device(s) sending: 1085
Number device(s) receiving: 1106

Number device(s) only sending: 1033
Number device(s) only receiving: 1054
```

The last special nodes set to analyze is the one containing all the individual sub networks. Here no device lists are presented as they are simply too long. As seen in 5.9, this topology is formed with 1002 individual networks averaging around two nodes.

Code 5.9: Result 2.1: Individual Networks

```
Number individual networks: 1002
nodes / ind_nw: 2,14
```

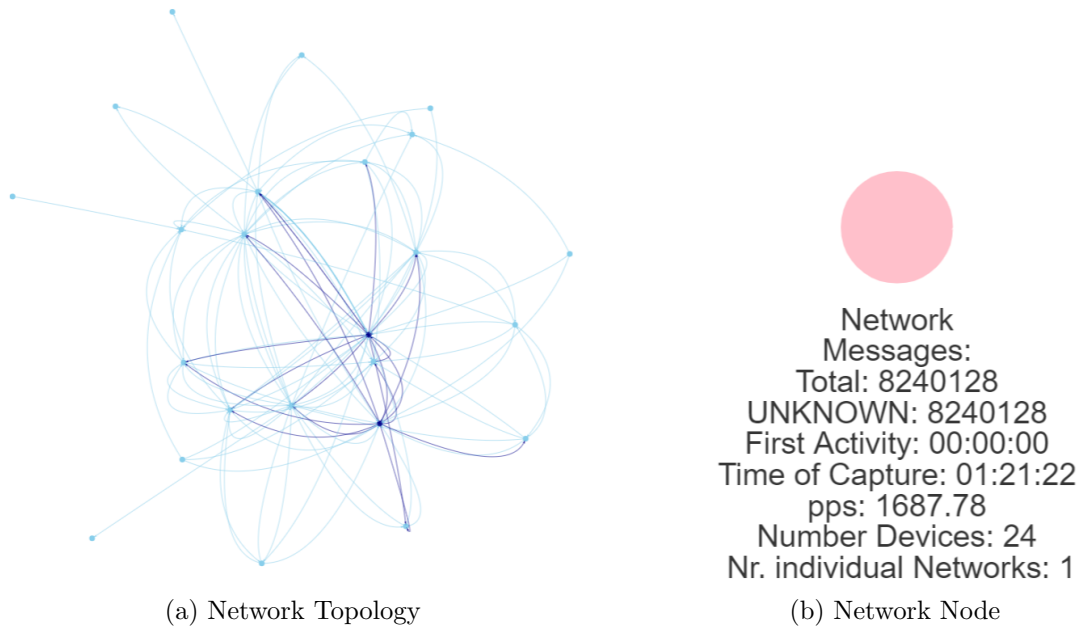


Figure 5.11: Result 2.1: Network Level

5.2.2 Result 2.2: Network Data Set under Attack

The results to the data set recording a network under attack are discussed here. First the topology involving all devices is shown and the network specific information presented. In this experiment, the malicious nodes are known. Hence in the configuration for this topology, custom nodes to color are the two malicious devices. The special nodes are extracted and visualized. For this experiment, the device lists and number of devices per special nodes set are displayed. Further for the custom node configuration, a sub topology only involving the malicious nodes is generated and presented.

Figure 5.11 (a) shows the topology of a network communicating over various WLAN protocols while being under a MITM attack. The nodes posing as malicious nodes are coloured in darkblue. The nodes not directly communicating with the malicious devices are coloured in skyblue. At first glance, it seems that more than half of the nodes are colored in darkblue and are communicating with the malicious nodes.

Network Level The network traffic data under attack was captured for 1 hour 21 minutes 22 seconds as seen in the network nodes of the topology in Figure 5.11 (b). In this time period 24 devices sent a total of 8'240'128 messages. No frame types were detected, therefore all messages are of type "UNKNOWN". This leads to a pps of 168,78. The topology also has all devices connected and hence only one individual network is present.

Special Nodes The first list of special nodes extracted are the devices having an extreme pps. In this experiment, 3 extremas are chosen because there are two malicious nodes present. In this topology, each node has an unique pps, leading to exactly three devices per category. When looking at the two lists containing the devices with minimal pps per behaviour in 5.10, the difference in pps is small. This means that there is no device standing out with a special behaviour. On the other hand for the highest sending pps, there are two devices with ID 00:0c:29:5b:73:5b and d4:3b:04:31:39:54 which have a value 20x higher then the third biggest pps recorded. Similarly for devices which receive a lot of messages, the two identified devices stand out with a higher pps than the third highest recorded value. But for this behaviour, it is only approximately 3,6 times higher.

Code 5.10: Result 2.2: Node IDs with min / max pps

```
Highest sending device(s): [ '00:0c:29:5b:73:5b_with_pps_820.5803 ',
'a4:91:b1:1e:57:90_with_pps_41.0891 ', 'd4:3b:04:31:39:54_with_pps_804.1024 ' ]
Lowest sending device(s): [ 'd4:dc:cd:b4:26:3e_with_pps_0.0123 ',
'dc:56:e7:14:5f:3b_with_pps_0.0039 ', 'ec:1f:72:f1:28:6d_with_pps_0.0034 ' ]
Highest receiving device(s): [ 'a4:91:b1:1e:57:90_with_pps_275.983 ',
'00:0c:29:5b:73:5b_with_pps_1010.2613 ', 'd4:3b:04:31:39:54_with_pps_974.2888 ' ]
Lowest receiving device(s): [ '01:00:5e:00:00:01_with_pps_0.0063 ',
'01:00:5e:00:00:02_with_pps_0.0029 ', '00:0c:29:db:1d:dc_with_pps_0.0741 ' ]
```

In this topology, there are 17 devices which send messages to 20 different devices as seen in 5.11. By implication, there should only be 4 devices which only send and 7 devices just receiving messages. This implication is true, as seen on line 5-6. Further the device lists are provided, as an option to visualize the sup topology.

Code 5.11: Result 2.2: Count and Node IDs for sending / receiving

```
Number device(s) total: 24
Number device(s) sending: 17
Number device(s) receiving: 20

Number device(s) only sending messages: 4
Number device(s) only receiving messages: 7
```

As already implied on the network node, all devices are connected in some way, leading to only one network topology involving all devices as seen in Code 5.12.

Code 5.12: Result 2.2: Individual Networks

```
Number individual networks: 1
nodes / ind. nw : 24.0
```

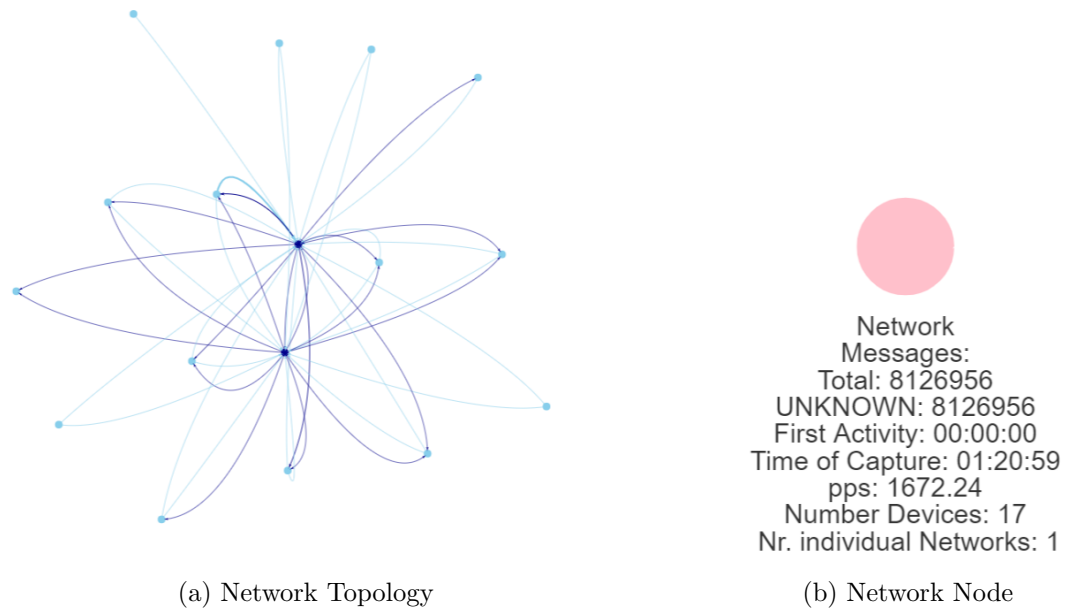


Figure 5.12: Result 2.2: Network with Malicious Nodes

Custom Nodes In a second step, the network topology only involving the malicious nodes is extracted and presented in (a) in Figure 5.12. The topology shows with the darkblue edges, the sent messages of the malicious nodes. The skyblue edges are the messages sent from the direct communication partner to the malicious nodes, or all the messages the malicious nodes receive. The darkblue dots are the malicious nodes itself.

Network Level A look at the network node reveals some more information in Figure 5.12 (b). The extracted sub topology involves 17 devices. The malicious nodes together were active for an 1 hour 20 minutes 59 seconds. In this time frame, the devices sent a total of 8'126'956 messages. This means a value of 1672,24 for the network pps. No individual networks were formed, which means that the malicious nodes communicate with each other.

	ble		
	3.1 small	3.2 medium	3.3 big
Main Topology	x	x	x
Network Node	x	x	x
Special Node Sets			
UNKNOWN_DEST			
UNKNOWN_SOURCE			
Max / Min pps	x (3)	x (3)	x (3)
Only Send / Receive	x	x	x
Ind. NW	x	x	x
Custom Sub Topology	No broadcast		
Activity Timeline		x	
Node Highlights			

Table 5.3: Experiment 3: Analysis

5.3 Result 3: BLE Data Set

In this section the data sets capturing BLE traffic are presented. Table 5.3 shows all conducted experiments and indicates which results are shown. For each experiment, the main topology and network node is displayed. Further all special node lists are extracted and presented if in size. In all three topologies, both "UNKNOWN" nodes are part of the devices and each experiment, a sub network is created but not presented. For Subsection 5.3.1, a sub network excluding the broadcast address is created. As there are too many devices in the resulting topology, there is no time line generated involving all devices, as the tool is not powerful enough. As an example on how to use the timeline tool in this case, the activity span for a individual sub network is presented for Subsection 5.3.2. No nodes sets are highlighted in the main topology.

5.3.1 Result 3.1: Small BLE Data Set

In this part, the results of the smallest BLE data set are discussed. For this experiment, all special node sets were retrieved. As seen in Table 5.3 not all results are also presented, for example all device lists if too long. The resulting topology is seen in Figure 5.13 (a) based on all devices to select and no highlighting. The topology is not visually appealing as there seem to be thousand of devices involved, centering around one node in the middle. The node in the middle has the address "ff:ff:ff:ff:ff:ff" and is used by the BLE standard to reference a broadcasting message. A broadcast message is sent to all devices in range, meaning the address only occurs as a destination address. The devices in range are the observers and listen to this address. There seems to be "UNKNOWN" nodes as there is a black arrow on the bottom left. Around the middle, there are a lot of individual networks made up by around 2 devices, which seem to communicate connection-oriented, as there are one to one.

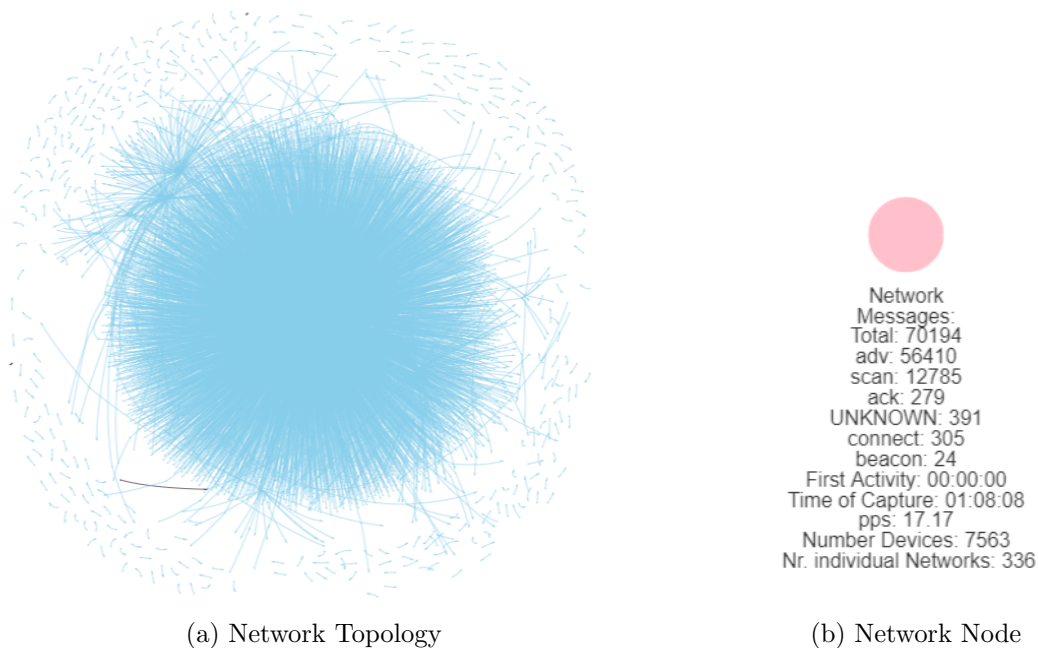


Figure 5.13: Result 3.1: Network Level

Network Level For all network level information the network node is consulted in Figure 5.13 (b). In experiment 3.1, the sniffer was run for 1 hour 8 minutes and 8 seconds. Within this time frame, there were a total of 70'194 BLE packages captured. Out of these packages, most were of frame type "adv" which are advertising packages and form over 80% (56'410). The "scan" frame is the second most appearing type with around 18% (12'785). The last 2% are of diverse types found in IoT networks. This leads to a network pps of 17,17. In total there are 7'563 devices forming 336 individual networks.

Special Nodes In this section the special node lists are presented if in size. The network topology has both 'UNKNOWN' nodes but they only communicate with each other and the sub topology is not presented. The individual sub networks will also not be presented as there are too many. Device count and IDs of nodes with one of the 3 highest or lowest pps per behaviour are seen in Code 5.13. For the highest behaviours there are 3 devices. As the network pps is 17,17 there is no device which sends more, as the highest sending pps is around 10, all other devices have a lower pps than the network. Another pattern arises for the highest receiving devices, as only one device has a higher pps, namely 'ff:ff:ff:ff:ff:ff' with a pps of 156. This is the broadcasting address and it is expected that this node receives a lot of packages, as it is a universal address and the same for every device. The lowest pps per behaviour device set contain all devices which rarely send or receive any messages and therefore match one of the lowest pps. In this experiment the all pps were below 1. In the smallest BLE data set there are a total of 5957 devices which rarely send and on the other hand 641 devices which receive a small number of packages.

Code 5.13: Result 3.1: Count and Node IDs with min / max pps

```
Highest sending device(s): [ '10:94:97:23:02:d0_with_pps_8.6099',
'61:e6:5a:0e:2f:7b_with_pps_10.3552', '64:d2:c4:95:a9:68_with_pps_9.9819' ]
Nr Lowest sending device(s): 5957
Highest receiving device(s): [ 'ff:ff:ff:ff:ff:ff_with_pps_156.0716',
'49:f2:9f:3e:62:4a_with_pps_1.3493', 'Unknown_0x50655496_with_pps_7.1751' ]
Nr Lowest receiving device(s): 641
```

The next set are devices in the topology which only send or receive frames. For this network, as listed in Code 5.14 out of a total of 7563 devices, big majority of 6839 only send messages. On the other hand, there are 647 nodes acting only as the destination of a message. Hence, there are 77 devices which send and receive $(7563 - 6839 - 647 \text{ or } (6916 + 724 - 7563) / 2)$.

Code 5.14: Result 3.1: Count of Node IDs for sending / receiving

```
Number device(s) total: 7563
Number device(s) sending: 6916
Number device(s) receiving: 724

Number device(s) only sending: 6839
Number device(s) only receiving : 647
```

As already discussed and referenced in Figure 5.13, there are a total of 374 individual networks, formed by roughly 22 nodes. The biggest network as seen in the main topology is centered around the broadcasting nodes. There are also a lot of point to point communication between two devices.

Code 5.15: Result 3.1: Individual Networks

```
Number individual networks: 374
nodes / ind_nw : 20.22
```

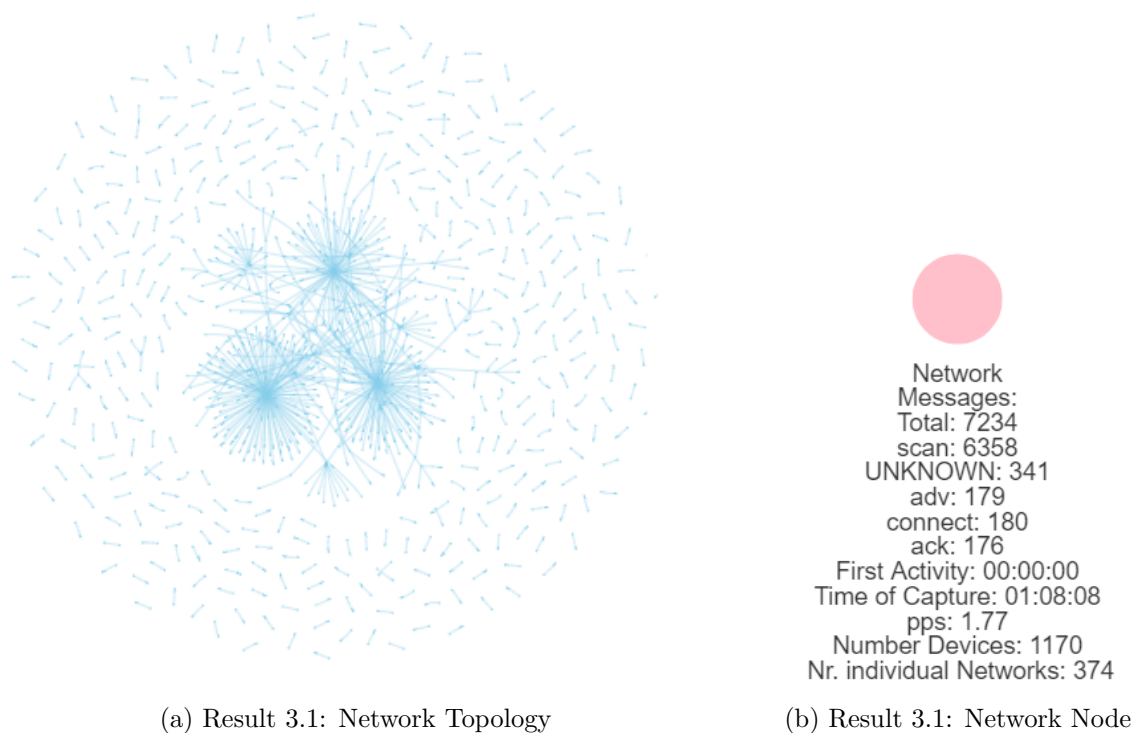


Figure 5.14: Result 3.1: Network Topology without Broadcast

Custom Nodes: For custom nodes in this experiment, all nodes which communicate more than just via broadcast are selected. This means that all messages containing "ff:ff:ff:ff:ff:ff" as destination address are dropped. Hence all devices which do not have any other destination address are excluded. This is an example as to why the broadcasting configuration in Wireshark can be important, even if the resulting topology is not visually appealing. The reason is, that all devices only communicating via broadcasting are ignored. At first, the network topology looks more user friendly in a) in Figure 5.14. But when having a closer look at the network node, the total message count is 7'234 in b). When comparing to the total of 70'194 of the main topology, the broadcast messages make around 90% of all messages. Hence, this sub network topology only presents 10% of all messages captured. The second problem is the number of devices. The new topology is formed by 1'170 devices, which means that 6'393 devices communicating over BLE are not captured. Further the pps dropped from 17,17 to 1,77 implicating less communication.

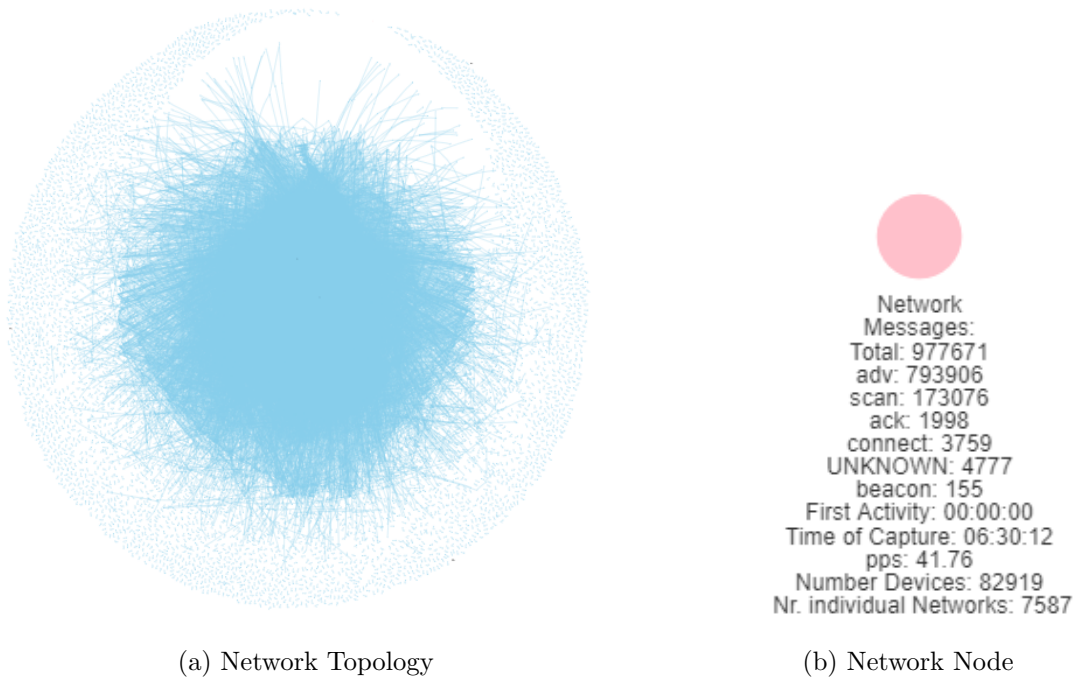


Figure 5.15: Result 3.2: Network Level

5.3.2 Result 3.2: Medium BLE Data Set

The results to the medium size BLE data set are presented in this section. For this data set all devices are selected to visualize and no nodes are highlighted. The network topology of this data set is presented in Figure 5.15 (a). The topology looks similar to the result from Subsection 5.3.1 as there is the broadcast node in the middle and the network topology is not clearly visible. There seem a lot of small individual networks pictured around the broadcasting address node. As there are two types of communication in BLE, connection-less and connection-oriented, it is assumed the pattern in the middle shows the connection-less communication or broadcasting. This means all broadcasting devices are captured and pictured. On the other hand the connection-oriented communication is pictured in form of individual sub networks, where only two devices are talking to each other.

Network Level The network level node in 5.15 (b) gives more insight, as the topology is not informative at first glance. In this experiment, the sniffer was active for 6 hours 30 minutes 12 seconds. During this period a total of 82'919 devices were recorded forming 7'587 sub networks. In this time frame, a total of 977'671 messages were captured. Hence this network has a pps of 41,76. The frame type distribution is dominated by the type "adv" with more than 80% (793'906), followed by the "scan" type with around 18% (173'076). The other types occur less than 1%.

Special Nodes For this experiment all special node set are retrieved and presented. Both "UNKNOWN_SOURCE" and "UNKNOWN_DESTINATION" nodes occur in this data set but are not presented. The first set to identify are the most or least active devices. For this the devices matching one of the 3 highest or lowest pps are selected, and the resulting IDs and counts are presented in Code 5.16. For devices which send the most, there are exactly 3, with device with ID "Unknown_0xaf9aa392" being the most active sending and receiving node. The pps is the same, so when checking the topology, this node only communicates with itself. Based on the BLE protocol, a node only interacting with itself was not expected. The address "Unknown_0xaf9aa392" is not an unique device, but an access address. The access and MAC address are part of the BLE frame and if both are present, the MAC address is used as it is unique. The reason for this behaviour are privacy concerns for the user [64]. As a solution, companies implement BLE with their own security. For connection-oriented communication, instead of using MAC addresses, only the access address of the opened channel is used for addressing both devices. Any addressing regarding this sub network happens over the unknown address, or the access channel. There is not much more information on the exact implementation, but it is assumed that if nodes communicate only with them self over a unknown address, it is a master node distributing data over an access address. For the receiving behaviour, the node which receives the second most messages is the broadcast address "ff:ff:ff:ff:ff:ff" which is expected as it is a universal address. On the other hand there are over 60'000 devices which rarely send any messages or active for a short time and 12'000 devices which receive almost no messages. In this experiment the every minimal pps was below 1.

Code 5.16: Result 3.2: Count and Node IDs with min / max pps

```
Highest sending device(s): [ '1a:d2:35:46:19:67_with_pps_22.2444',
Unknown\_0xaf9aa392 with pps 58.6656',_, '54:49:23:4c:2d:47 with pps 26.018
Nr\_Lowest\_sending\_device(s):_61001
Highest\_receiving\_device(s):_['ff:ff:ff:ff:ff:ff with pps 37.4885',
'c4:4b:77:29:c9:ec with pps 3',_, 'Unknown\_0xaf9aa392 with pps 58.6656']
Nr\_Lowest\_receiving\_device(s):_12192
```

For devices only participating in one communication behaviour, there are 69'541 nodes which only send messages and 12'817 devices only receiving frames as seen in Code 5.20. This means that in this topology only a minimum of devices send and receive, namely 561.

Code 5.17: Result 3.2: Count of Node IDs for sending / receiving

```
Number device(s) total: 82919
Number device(s) sending: 70102
Number device(s) receiving: 13378

Number device(s) only sending: 69541
Number device(s) only receiving: 12817
```

The total number of 82'919 devices are divided into 7'587 individual sub networks. Hence, there are around 11 devices per sub network as seen in Code 5.21

Code 5.18: Result 3.2: Individual Networks

```
Number individual networks: 7587
nodes / indnw : 10.93
```

Node Activity Timeline As this topology has too many involved devices, the timeline generated only shows an example of one of the 7'587 sub networks. For this, all sub networks involving more than 5 devices were extracted leaving 8 candidates. Out of the 8 one is the broadcasting network, being the only sub network with more than 10 devices. Out of the resulting 7 options, one was chosen at random and presented in Figure 5.16. The timeline shows a sub network with 10 nodes. Out of those, exactly half only send messages and the other half are the recipients of said messages as seen on the red or green colored bar. The selected nodes were only active for the first few minutes when compared to the network bar in blue.

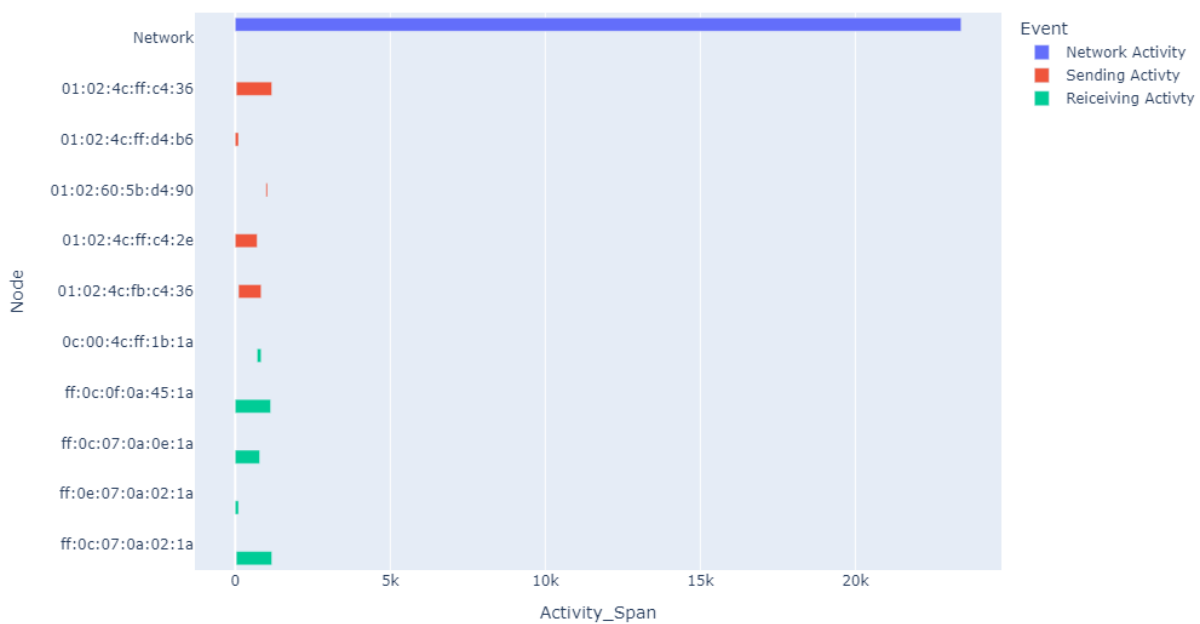


Figure 5.16: Result 3.2: Node Activity Timeline

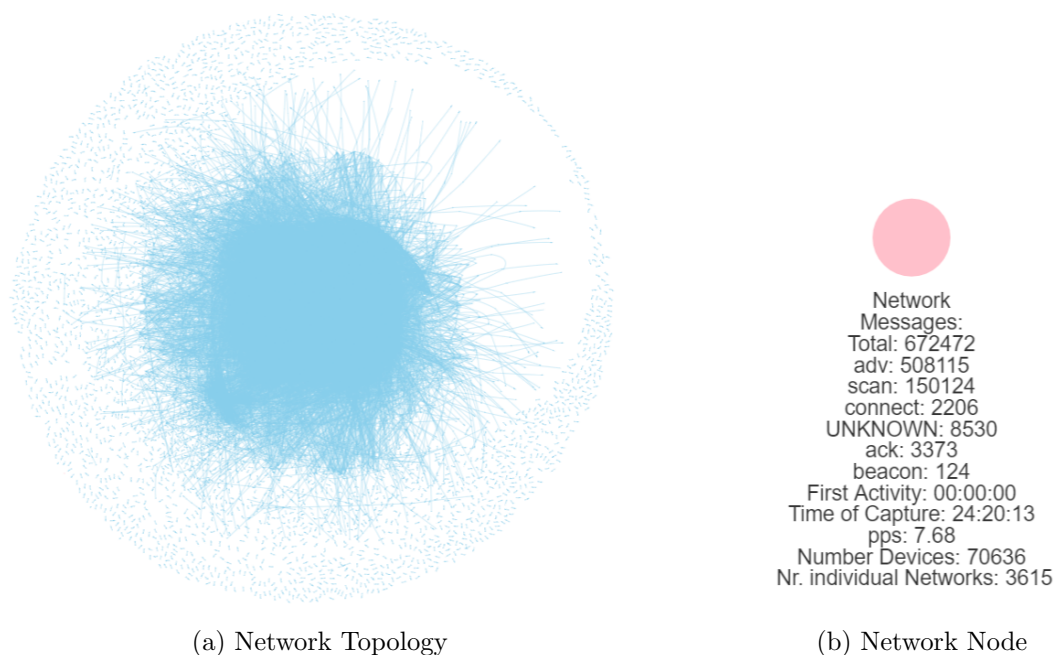


Figure 5.17: Result 3.3: Network Level

5.3.3 Result 3.3: Large BLE Data Set

The resulting topology of the large BLE data set is pictured in 5.17 (a). Similarly to the topology of the medium and small sized BLE data set, there is one very big main node, where most of the communication seems to happen. This is again the broadcasting destination address node "ff:ff:ff:ff:ff:ff". Around the big network in the middle, there seems to be a lot of devices just communicating with each other.

Network Level In this experiment, the network level information is given in Figure 5.17 (b). This experiment run for 24h 20min 13s. Within this time frame, a total of 672'472 messages were captured. The most captured frame type is "adv" or the advertising packages. The second most occurred type is "scan", which is used for scanning for new communication partners. With each less than 1% of all packages follow the resulting frame types "connect", "UNKNOWN", "ack" and "beacon". These messages were sent by a total of 70'636 devices. This leads to a network pps of 7,68. The topology consists of 3'615 individual sub topologies.

Special Nodes The first set retrieved is presented in Code 5.19 and shows the device IDs or device count of nodes matching one of the three highest or lowest pps per behaviour. When sniffing BLE traffic for over 24h, both "UNKNOWN" nodes are present, but the topology is not presented.

The three most active nodes are given on line 1. The highest sending pps of a node is 110 and belongs to device "8:b6:73:cb:b4:a4". The highest receiving pps is the same with 110 and belonging to "44:e1:50:f7:b2:3a". This implies a that these two devices only talk to each other and either send a lot of messages or are active for a short time frame. When checking the topology, these two devices only communicate with each other and are active during a small time frame just below one minute. For the highest sending devices, one access channel "Unknown_0x50657bd4" is part of this set with a pps of around 68. This node again only communicates with himself and therefore is also part as one of the highest receiving devices. There is another access channel address within the highest receiving devices. It is assumed, that the sending pps of this node is the same, namely around 27. Further his topology consists of around 56'723 devices which send almost no packages or are active for a short amount of time. On the other hand there are 5'704 node which do not receive a lot of messages.

Code 5.19: Result 3.3: Count and Node IDs with min / max pps

```
Highest sending device(s): [ 'd8:b6:73:3f:db:ec_with_pps_38.1921 ',
'd8:b6:73:cb:b4:a4_with_pps_110 ', 'Unknown\_0x50657bd4_with_pps_68.4593 ' ]
Nr Lowest sending device(s): 56723
Highest receiving device(s): [ 'Unknown\_0xaf9ab420_with_pps_27.4514 ',
'44:e1:50:f7:b2:3a_with_pps_110 ', 'Unknown\_0x50657bd4_with_pps_68.4593 ' ]
Nr Lowest receiving device(s): 5704
```

In the big network topology out of the 70'636 participating devices, a big majority of 64'204 devices never actually receive messages. In contrast there are just 5'964 nodes which never actually send a message and only receive frames. As a result, there are only 468 devices sending and receiving messages.

Code 5.20: Result 3.3: Count of Node IDs for sending / receiving

```
Number device(s) total: 70636
Number device(s) sending: 64672
Number device(s) receiving: 6432

Number device(s) only sending: 64204
Number device(s) only receiving: 5964
```

These 70'636 devices form 3'615 individual networks, leading to an average of 20 devices participating in one sub network.

Code 5.21: Result 3.3: Individual Networks

```
Number individual networks: 3615
nodes / ind_nw : 19.54
```

Chapter 6

Evaluation

In this chapter, all three experiments are further analysed and evaluated. The evaluation of the first experiment includes the validation of the visualization approach, as the expected topology is known. The second experiment researches a network topology which is under attack vs. in a normal state. The last experiment captured real-life BLE traffic for 3 different time frames, thus the evaluation compares the three data sets. The BLE data sets are also used for a run time analysis for the algorithm. At the end, the limitations of this approach are presented.

6.1 Evaluation 1: UWB Data Set

In this section the results of the UWB data set from the algorithm are compared and validated with the expected results. The expected results are either collected from Wireshark, such as network node information, or extracted from [63] such as the expected network topology. In a first step, all calculations on network, node and edge level are verified with the results of the raw UWB data set from 5.1.1. Further the special node sets are checked for correctness. In this experiment, the "UNKNOWN" nodes are retrieved and the resulting topology validated. The visualized topology from the adapted UWB data set from 5.1.2 is compared to the expected topology from 4.6. In a last step, the membership comparison of two networks is validated.

	Result	Wireshark
Messages Total	4693	4693
Data	4687	4687
Beacon	6	6
Time of Capture	00:10:14	614,57 s
pps	7,64	7,6

Table 6.1: Evaluation 1.1: Node Information Validation

6.1.1 Evaluation 1.1: Raw UWB Data Set

The raw UWB data set is used for validating the following values and topologies: network level, node and edge level, special node sets and counts as well as the correct sub network topology retrieval with the "UNKNOWN" nodes.

Network Node Validation

The first information to validate is the network node information retrieved from Figure 5.9. This information is checked for correctness with the expected value. The expected value is retrieved from Wireshark. This is achieved by loading the original Pcap file into Wireshark and use the file wide information. The information regarding the number of involved devices and individual networks is proven by visually marking the devices and individual networks. The results of the validation with Wireshark are presented in Table 6.1. The 'First Activity' is excluded, as the whole data set is part of this topology and therefore the first recorded message was at the start of the recording or at 00:00:00. The table shows, that the total message count is the same, as well as the count per frame type. The time is retrieved in seconds from Wireshark and comes to 614,57 seconds, which translated to the same format, comes to 6 minutes 15 seconds. Since the time configuration in this experiment is in milliseconds, and the time configuration of this calculation is not known, a rounding difference is assumed. The pps of the project matches the pps of Wireshark, differs in just one more digit.

For the correctness of the number devices and individual networks count, the devices and networks are colored and presented in Figure 6.1. The network node information contains 8 devices and 3 individual networks. To validate the unique device count, each dot in figure (a) is circled in red and counted by adding a number to each dot. There are 9 dots present, but the node with nr. 9 is the network node and does not count as a device. Therefore 8 unique devices are counted and returned by the algorithm. The individual networks are highlighted and circled in green as seen in figure (b). To count the networks, each individual topology is labeled with an number. Here 4 networks are counted, but again network number 4 is the network node and can be subtracted from the total count, resulting in 3 individual networks.

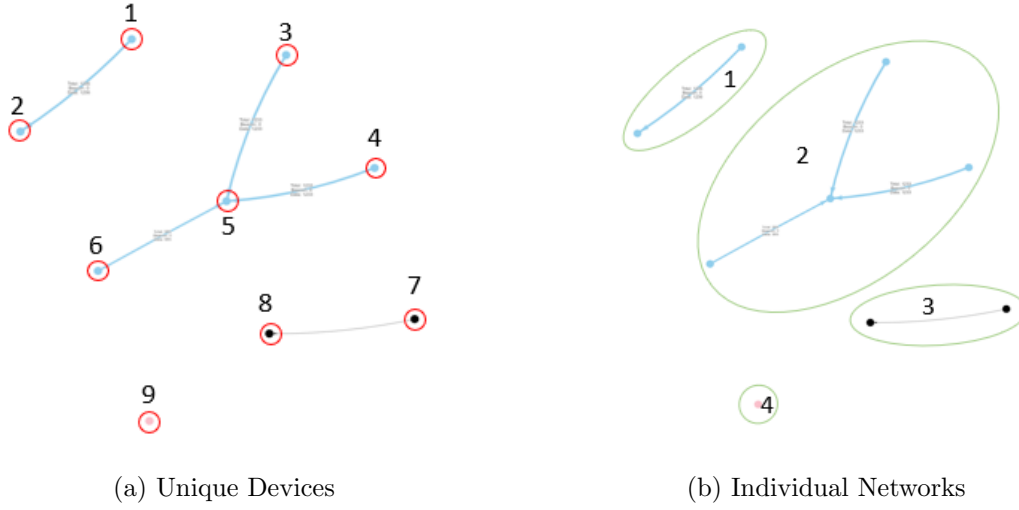


Figure 6.1: Evaluation 1.1: Networks and Device Count Validation

	0x304d	0x5841	Wireshark
Messages Total	1236	1236	1236
Data	1236	1236	1236
pps	2,01	2,01	2
First Activity	00:00:00	00:00:00	0,000s
Last Activity	00:10:14	00:10:14	614,563s

Table 6.2: Evaluation 1.1: Edge and Node Information Validation

Edge and Node Validation

In order to validate the edge and node information from Figure 5.3, the data set in Wireshark is filtered for the two node IDs '0x304d' and '0x5841'. Because they only communicate with each other and are always sender ('0x304d') or receiver ('0x5841'), the resulting data rows are the number of messages exchanged and Wireshark provides the pps and time of capture for only the selected rows. The first activity is derived from the time of the first filtered packages and the last frame carries the time value of the last activity. For the frame type count, the Pcap file is filtered for both device IDs in combination of containing the string 'Beacon' in the 'Info' field. This query resulted in zero entries, and since all other messages are of type data, all exchanged messages are data types. Again since the nodes only communicate with each other, the values are the same, once for a receiving behaviour and once for a sending behaviour. The Table 6.2 gives an overview of the values from the topology and the values extracted from Wireshark. The total messages count and type count true as they are the same. If the node pps is rounded to no digits, the node pps is 2 and the values match. The nodes communicate from the beginning, as the first package has the time stamp of 0,000s (it is package with nr. 1) which corresponds to 00:00:00. The last message was sent after 614,563 seconds, which is rounded to one more minute than the algorithm returns.

Device ID	pps	
	send	receive
UNKNOWN_SOURCE	0,0192	
UNKNOWN_DESTINATION		0,0192
0x3241	2,0063	
0x3142	1,6054	
0x3041	2,0063	
0x5854		5,6152
0x304d	2,0112	
0x5841		2,0112

Table 6.3: Evaluation 1.1: Special Nodes Validation

Special Nodes Set Validation

The first special node group to validate refers to the devices with a maximal or minimal pps. To identify the correct member(s) for each extrema, the pps of each node for sending and receiving is extracted. By first defining the minimal and maximal value per behaviour and rounded to 4 digits, the device IDs are identified. The Table 6.3 lists for each of the device the pps per behaviour. If a device did not send or receive a message, there is no pps, and the value in the behaviour is empty. The maximum value is colored in green per behaviour and the minimal value red. As a result, the following device IDs are identified. The highest pps for sending is 2,0112 and one device matches this pps, namely 0x304d. The minimal sending pps is 0,0192 and belongs to the UNKNOWN_SOURCE node. For each extreme receiving pps, there is exactly one device. With a pps of 0,0192 the lowest value belongs to the UNKNOWN_SOURCE node and the device 0x5854 with a value of 5,6152 received the most messages. These values and devices match the list in Code 5.1 and the correct nodes and pps were retrieved. The Table 6.3 can therefore further be used to check the correct members of the device groups which only send or only receive. The device ID can be retrieved by checking for a empty value in the table, for example if a device sends but does not receive messages, the device has a pps for sending but the receiving pps is empty. For this topology there are 3 devices, UNKNOWN_SOURCE, 0x5854 and 0x5841 which do not send any messages and are in the group of devices which only send. On the other hand the 5 devices UNKNOWN_DESTINATION, 0x3241, 0x3141, 0x3041 and 0x304d. When comparing with the output of the algorithm in Code 5.8 with the identified devices and device counts, they match. Hence, the algorithm returns the correct count and device IDs.

The last group to check are the returned individual networks from Code 5.3. As the individual network count and device count are validated, the average nodes per individual network is calculated as follows: device count / nr individual networks. For this experiment, there are 8 devices in 3 networks, hence the average number of nodes is 2,67. The individual lists should contain the respective nodes within the green boundaries from (b) in Figure 6.1. In the topology 1 are the two devices: 0x5841 and 0x304d. Network number two is formed by the following four nodes: 0x5854, 0x3041, 0x3241, 0x3141. The last network numbered 3 is composed of the 2 UNKNOWN nodes. These three sets are returned by the code and therefore the method returns the correct sub networks.

Retrieval Sub Network Validation

In order for the retrieved sub network to be correct, two requirements need to be full filled. First the correct nodes and their direct communication partners are returned. Further there needs to be a information adaption on network level, e.g. the network node carries only information regarding the selected nodes. To check the first requirement, the expected return is first defined. As seen in the overview of main network topology in Figure 5.2 (a), if the devices to select contain any of the "UNKNOWN" nodes, the sub network to retrieve is the individual network containing the two "UNKNOWN" nodes. The reason is that the two nodes only communicate with each other and therefore are the only nodes to retrieve. The expected return matches the two network topologies generated in Figure 5.4. The second node to check is the network node. The new network node generated should only contain information related to the selected sub network. In this case, the expected number of messages is equal the number of messages exchanged between the two nodes, namely 6 messages of type beacon. The first activity is equal to the earliest recorded time in the network, here it is 00:00:00. The time of capture is similar but matching the longest activity span of one of the devices, which is 00:05:12. The network pps is equal to the two nodes, exactly 0,02. The number of devices is now 2 which form 1 individual network. These numbers are equal to the network node in Figure 6.2. Hence, the correct devices are retrieved and the network node is adapted to the selected sub topology.



Figure 6.2: Evaluation 1.2: Sub Network Node

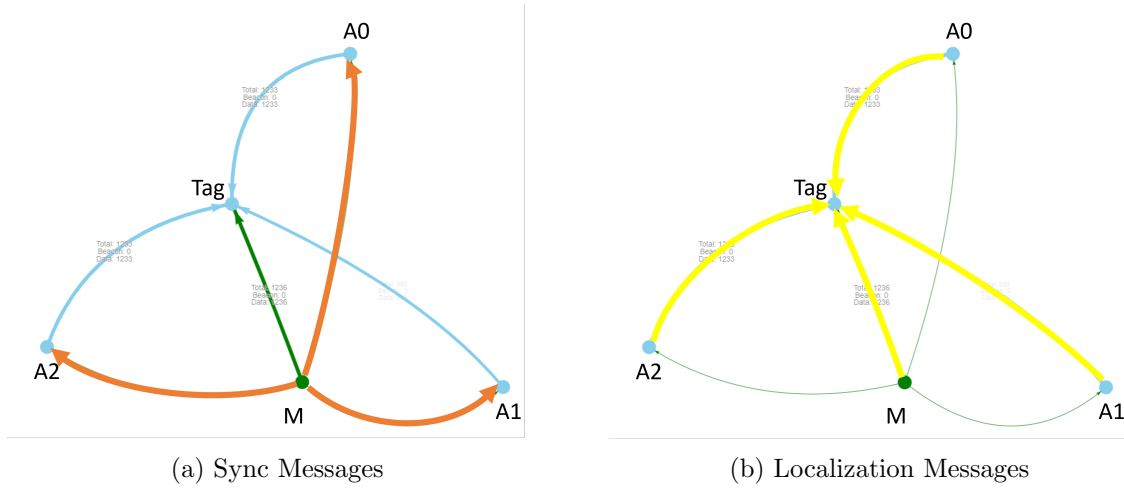


Figure 6.3: Evaluation 1.2: Visualization Approach

6.1.2 Evaluation 1.2: Adapted UWB Data Set

The adapted data set is used to verify the correctness of the visualization approach, as the expected topology is known. In order to verify the topology of the involved devices, the result of the project in Figure 6.3 is compared to the correct topology represented in Figure 4.6. First the expected topology of the sync messages on the left side in Figure 4.6 is compared to the network topology of a) in Figure 6.3. The expected topology shows a master anchor (in green) sending out sync messages to all its following anchors (in blue). As explained in the preprocessing of the adapted data set, for all 6 packages of type beacon the destination address is changed to all following anchor devices A0, A1 and A2. The source address contains the master anchor ID "0x304d". These beacon messages are highlighted in orange and are the only messages sent between the master anchor (M) and the following anchors (A0, A1, A2). As they are beacon type and the only communication between the master and the slave nodes, it is assumed that the messages carry the sync information. Therefore, the first part of the expected network topology is verified. The second part of the topology to check is the right part in Figure 4.6 and shows the localization messages sent from all anchors (in green and blue) to the tag (in red). As the system is passive, the messages are directed from the anchors to the tag and the expected frame type is data, as that is the frame type in UWB carrying localization information. This communication pattern is generated by the results in b) of Figure 6.3. The anchors (M, A0, A1, A2) communicate with the tag only via data messages colored in yellow. The tag does not send any messages and only receives. Further except for the master anchor, it is the only communication stream of the following anchors. Thus the second part of the UWB topology is correct.

6.1.3 Evaluation 1: UWB Comparison

In this part the set comparison of two or more network data sets is validated. The data sets are the raw and adapted UWB data set. To check the correctness, first the device memberships needs to be determined. The code returns all unique devices in a network, devices which appear in every topology and nodes only being part of one of the input networks. In the adapted data set the following 8 unique devices are present: '0x3241', '0x3141', '0x3041', '0x304d', '0x5841', '0x5854', 'UNKNOWN_SOURCE' and 'UNKNOWN_DESTINATION'. The adapted topology is made up by the following 5 devices: '0x3241', '0x3141', '0x3041', '0x304d' and 'tag'. Therefore the total list of unique devices contains 9 devices: '0x3241', '0x3141', '0x3041', '0x304d', 'tag', '0x5841', '0x5854', 'UNKNOWN_SOURCE' and 'UNKNOWN_DESTINATION'. The shared devices or devices in every network are the four anchors: '0x3241', '0x3141', '0x3041' and '0x304d'. The set difference of these two lead to the devices which are in one network: 'tag', '0x5841', '0x5854', 'UNKNOWN_SOURCE' and 'UNKNOWN_DESTINATION'. These derived lists are the same as in Code 6.2, which are the sets returned by the algorithm and therefore the set membership calculations are correct.

Code 6.1: Evaluation 2.2: Network Comparison

```
Combined number device(s) over all Networks: 9
Number device(s) in every Network: 4
Number device(s) in just one Network: 5
```

```
All Device(s): [ '0x3241', '0x3141', '0x3041', '0x304d', 'tag', '0x5841',
'0x5854', 'UNKNOWN_SOURCE', 'UNKNOWN_DESTINATION' ]
Device(s) in every Network: [ '0x3241', '0x3141', '0x3041', '0x304d' ]
Device(s) in just one Network: [ 'tag', '0x5841', '0x5854',
'UNKNOWN_SOURCE', 'UNKNOWN_DESTINATION' ]
```

6.2 Evaluation 2: Network Data Set

The evaluation of the second experiment starts with a closer look at the node coloring of the normal network data set. The network data set under attack concludes whether or not the malicious nodes were identified. Lastly, the two networks are compared. This includes a frame type and device behaviour distribution analysis.

6.2.1 Evaluation 2.1: Normal Network Data Set

The topology of the network data set shows a lot of individual networks communicating with each other. Considering there is no information regarding the set up of the experiment, no topology comparison is possible. Further no comparison with real world WLAN networks is feasible as the network was created in a lab environment. One limitation the SLL capture mode reveals in this experiment is the distribution of the frame types. The reason for the high amount of "UNKNOWN" packages is the SLL header Wireshark adds. In this header, for all WLAN type protocols, except for TCP, the frame type data is assumed. Since the frame type data is not derived from the package, rather from the added header, the type is not added to the "Info" field and the algorithm adds frame type "UNKNOWN". Since this network has a lot of messages sent over a thousand devices, the results are used to check the highlighting of the correct nodes.

Node Coloring

In this experiment the nodes with highest sending and receiving pps are highlighted according to the following rules. Devices which have one of the 3 highest pps (sending or receiving) are colored in yellow. There are 3 devices sending or receiving a lot, therefore maximal 6 nodes are colored in yellow. Devices which send or receive barely any messages and therefore have a low pps are colored in red. There are at most 53 devices colored in red, as there are 29 devices IDs which have a low sending pps and 24 which barely receive messages, resulting in a low receiving pps. Devices which send no messages are not in this category, as the nodes are not recorded in the device groups. Further devices which fall in either of the category, are colored in darkgreen. To check the correctness of the colors, for each color group the respective node ID are extracted and compared for correctness with the devices lists returned by the algorithm. To start the devices IDs colored in dark green are extracted and listed in Table 6.4. There are a total of 16 devices colored in dark green. The device ID is listed in the first column. The other columns indicate with an x, which max or min pps group the device belongs too. In order for the color to be darkgreen, at least two x need to be present per column, which is given. All 16 devices belong to exactly two categories, while 13 send and receive almost no messages. On the other hand, 3 devices barely receive any messages, but send out a lot.

Device ID	Sending pps		Receiving pps	
	Max	Min	Max	Min
2600:9000:5303:c700::1		x		x
2600:9000:5302:a100::1		x		x
2001:500:856e::30		x		x
2600:9000:5306:6700::1	x			x
2001:503:39c1::30		x		x
2600:1406:32::c2		x		x
2001:4860:4802:36::6a		x		x
2600:9000:5303:a200::1	x			x
2600:9000:5301:4c00::1	x			x
2a01:8840:9f::17		x		x
2001:dcd:3::2		x		x
2600:9000:5300:ce00::1		x		x
2606:2800:1::6		x		x
2600:9000:5306:4500::1		x		x
2001:4860:4802:32::6e		x		x
2001:4860:4802:32::6d		x		x

Table 6.4: Devices highlighted in 'darkgreen'

Device ID	Color	
	Expected	Actual
2405:6e00:10ce:2c00:20c:29ff:fed2:b002	yellow	yellow
2600:9000:5300:6200::1	yellow	yellow
2a01:8840:c0::1	yellow	yellow

Table 6.5: Devices highlighted in 'yellow'

The next color to check is yellow where the maximum number of nodes in this color is 6. When checking the two device lists in Code 5.7, no device ID has a maximum pps in both behaviours, so there are still 6 nodes expected to be yellow. But in Table 6.4, there are 3 devices which have a maximum sending pps, leaving 3 devices to be colored in yellow. These three devices are expected to be the nodes with the maximum receiving pps and the node IDs are extracted. In Table 6.5 these node IDs are in the first column, while the expected color is yellow and the actual color is checked and filled in in the third column. The table shows, that all 3 nodes are actually colored in yellow.

The last color to validate is red or devices with a minimal communication behaviour. First the expected number of red dots in the topology is derived. There are 53 nodes expected to be colored in red. But first the nodes which are already colored in dark green have to be subtracted. For the color red, this means 13 less nodes for the minimal sending behaviour, as well as 16 belonging to the minimal receiving behaviour. That leaves a total of 24 devices expected to be colored in red. Since the node IDs are the correct ones for the first two colors, not all node IDs are compared, rather the number of dots colored in reds is extracted, which is 24. Therefore, the highlighting of the nodes is correct and validated.

6.2.2 Evaluation 2.2: Network Data Set under Attack

In this section, the network data set under attack is evaluated by first having a look at the topography involving all devices, or mainly at the retrieved device lists sets. Since this experiment deals with a MITM attack, malicious devices are expected to have a rather high pps. The reason is that a malicious node intercepts the communication between two network nodes by disguising itself as the actual communication partner. By adding an additional node into network, disguised as a known node, the number of messages is expected to increase, resulting in a relatively high communication behaviour for said node. In a second step, the topology involving all devices is compared to the topology with only malicious nodes. In this step the devices which do not communicate with the malicious nodes, are identified by set comparison.

Malicious Nodes Identification

As stated in Subsection 5.2.2, the malicious devices have the MAC addresses 00:0c:29:5b:73:5b and 00:0c:29:b8:b1:74. When comparing with the device lists retrieved in Code 5.10, the malicious node ID 00:0c:29:5b:73:5b is present in two special node groups. The first is the set of the highest sending devices. The node has a pps of 820,58 and is on top of the list, meaning the highest pps sending of all devices in the network belongs to a malicious node. The node ID further appears in the list with the devices receiving the most messages, with a pps of 1010,26. This value is the highest pps in the whole network. The second malicious node with ID 00:0c:29:b8:b1:74 does not appear in any of the lists. By checking the topology, this node has an sending pps of 17,85 and pps when receiving of 21,9. Without any further context, the experiment detects one out of two malicious nodes, where the detected node has the maximal sending and receiving pps. By having a look in Wireshark, the reason for the rather low pps of the second malicious node can be seen in Figure 6.4. According to Wireshark, these two nodes share IP addresses in some cases. This means, that the malicious node ID detected by the algorithm also makes use of the IP address of the not detected node. How many of the messages are part of this IP sharing is not recorded in Wireshark nor listed in the original folder of the experiments. Therefore it can not be determined weather or not both malicious nodes are detectable.

[Duplicate IP address detected for 192.168.1.34 (00:0c:29:b8:b1:74) - also in use by 00:0c:29:5b:73:5b (frame 468)]

Figure 6.4: Evaluation 2.2: IP sharing of malicious Nodes

Malicious Nodes Sub-Topology

In this step, the sub topology only involving the malicious nodes is compared to the main topology by checking the network nodes and identify nodes which are in both networks or in just one. The devices appearing in both topologies communicated with at least one of the malicious nodes. Since the sub-topology is part of the main topology, the nodes which just appear in one network are the safe nodes, meaning they had no direct contact with the malicious devices. The two network nodes in Figure 5.11 and Figure 5.12 contain the information which is relevant to all selected nodes, enabling a comparison. The sub-network was active for 1 hour 20 minutes and 59 seconds compared to the 1hour 21 minutes and 22 seconds of the main network. Hence, the malicious nodes were active for almost the same time as the rest of the network. In this time, the malicious nodes network sent a total of 8'126'956 messages, leading to a network pps of 1672,24. The nodes not involved with the malicious nodes send an additional 113'172 messages, leading to a total of the main network of 8'240'128 and a pps of 1687.78. In the sub network a total of 17 devices all communicating with each other are detected. The main topology is made up of 24 devices and also no spitted networks. For further analysis, the comparison of the devices for each topology gives the following output in Code 6.2. While for all three set comparison; combined number of devices (union), number devices in every network (interception) and devices in just one network (difference), the number is given, only for the devices which appear in only one network also a list is provided. When comparing

Code 6.2: Evaluation 2.2: Network Comparison

```
Combined number device(s) over all Networks: 24
Number device(s) in every Network: 17
Number device(s) in just one Network: 7
```

```
Device(s) in just one Network: [ '01:00:5e:00:00:02 ', '01:00:5e:00:00:01 ',
'd4:dc:cd:b4:26:3e ', '01:00:5e:7f:ff:fa ', '01:00:5e:00:00:fc ',
'01:00:5e:00:00:fb ', '01:00:5e:00:00:16 ' ]
```

the set membership of both devices, the two networks combined have 24 unique devices and share 17 devices. These 17 devices are the nodes at risk, since they communicate with at least one of the nodes under attack or are one of the nodes under attack. This just highlights the fact, that one network is a complete sub network of the others since all devices are shared. The remaining 7 node IDs belong to the safe devices, or the nodes never actually communicating with malicious nodes.

6.2.3 Evaluation 2: Network Comparison

For the last part of experiment 2, the network data set in normal form and under attack are compared by first checking the difference in values from the network node followed by identifying the shared devices. For the network node information, only the pps is informative. The reason is, that even though both experiments were performed in the same lab environment, it is not the same experiment and the meta data differ naturally. The number of devices and individual networks are also a value influenced by the set up of the experiment and have no informative value when comparing two networks. As an example, the duration of the experiment is not relevant. The pps in a topology which is not under attack is 21.30 as seen in Figure 5.9. By checking the network node for the topology under attack in Figure 5.12 a pps of 1687.76 is recorded. This is an increase of around 80x the pps of a normal network.

Since there are 2 different networks, only the shared devices are presented, as the other devices are either in network 1 or network 2. For the other measurements, the numbers are listed as seen in Code 6.3. The two networks combined have a total of 2159 unique devices. Out of the total, 4 devices appear in both networks, while the other 2155 are in either one. The malicious node IDs are only present in the topology under attack. Only one shared node communicates with the malicious nodes, namely the device with ID 'd4:dc:cd:b4:26:3e'.

Code 6.3: Evaluation 2: Network Comparison

```
Combined number device(s) over all Networks: 2159
Number device(s) in every Network: 4
Number device(s) in just one Network: 2155
```

```
Device(s) in every Network: [ '00:c3:f4:0f:67:73 ',
'dc:56:e7:5b:61:41 ', 'd4:dc:cd:b4:26:3e ', 'a4:91:b1:1e:57:90 ' ]
```

6.3 Evaluation 3: BLE Data Set

The evaluation of experiment 3 skips the individual analysis for each experiments and compares the three data sets by analyzing network level information for different times of capture. This is possible as all three experiments are executed at the same place but for different time frames. These data sets are used to make a run time analysis of the algorithm with actual times.

6.3.1 Evaluation 3: Network Comparison

In this section the three BLE results are compared. First the frame type distribution over all experiments are analysed, followed by a comparison of the devices behaviour. Lastly, the network level information for each data set is presented and compared.

Frame Type Distribution The frame type distribution per experiment and frame type is given in Table 6.6. The total number gives the total count of captured messages. It is expected that experiment 1 with the shortest run time of around 1h, captured the least amount of messages. On the other hand, the medium data set which run for about 6h actually consists of the most packages. While it run for about 6x longer, the experiment captured 14x the amount of messages. When comparing to the big data set, there is an incline in captured messages, as for a run time of around 24h, an increase is expected. As the experiments started at around same time, the day differs. The medium sized data set was recorded on a sunny Friday afternoon in the city of Zurich. A possible explanation is that at this time more people were around and hence more messages captured. Another possible explanation is that a lot more random addressing is used compared to the other experiments. In this case, the device count contains the same device with different addresses. Regarding the distribution of the frame types, for each experiments the most captured type is "adv" with around 80%. This makes sense, as devices use this type for advertising the presence of a device and broadcasting data. This types of messages are most likely sent by broadcaster devices before connecting with a observer device. The second most captured type with around 20% is the type "scan". These messages are sent by a observer reacting to a advertising messages. At this point in the communication, the observer becomes a master node and the broadcaster becomes an slave node. This change in roles happens after a "connect" messages is sent by a observer. At this point the communication changes from connection-less to connection-oriented. This type of message was makes around 0,4%. Further the distribution of "ack" messages is around the same by being below 0,5%. In certain BLE networks, "beacon" types are used by the master slaves for periodically opening slots for sending data by the slaves after connection. This type makes around 0,02% of all messages. Lastly, in the first two experiments, there were around 0,5% "UNKNOWN" frame types, while for the big data set 1,27% of all messages have no type.

	Small	Medium	Big
Total messages	70'194	977'671	672'472
adv	56'410 / 80,36%	793'906 / 81,2%	508'115 / 75,56%
scan	12'785 / 18,21%	173'076 / 17,7%	150'124 / 22,32%
connect	305 / 0,43%	3759 / 0,38%	2206 / 0,33%
ack	279 / 0,4%	1998 / 0,2%	3373 / 0,5%
beacon	24 / 0,03%	155 / 0,02%	124 / 0,02%
unknown	391 / 0,56%	4777 / 0,49%	8530 / 1,27%

Table 6.6: Frame Type Distribution

	Small	Medium	Big
Devices	7'563	82'919	70'636
only sending	6'839 / 90,45%	69'541 / 83,87%	64204 / 90,89%
only receive	647 / 8,56%	12817 / 15,46%	5964 / 8,45%
send and receive	77 / 0,99%	561 / 0,67%	468 / 0,66%

Table 6.7: Device Behaviour Distribution

Device Behaviour Distribution For the device behaviour distribution, for each experiment devices which only communicate in one direction are compared. The results are given in Table 6.7. For all three experiment, the majority of devices only send messages. The small and big topology are formed by over 90% of devices which only send messages and for the medium sized data set it is over 80%. When comparing to the BLE standard, this makes sense because of the roles and communication type in the standard. It is assumed that these devices are broadcaster as this role only sends advertisement messages. The broadcasting part also leads to the assumption of a connection-less communication. This can also be compared to the previous table, where around 80% of all captured messages are of this type. On the other hand are devices which only receive messages, which make around 9% for the small and big data set and 15% for the medium experiment. It is assumed that these devices are slave or master nodes which only communicate in one direction. Further devices which send and receive make less than 1%. It is assumed that these devices communicate connection-oriented and have formed some kind of master slave architecture where both devices send and receive. The evaluation of the BLE experiment uncovered one challenge. Namely the detection of the observer devices. Observers in the BLE standard only listen to messages and never send messages. If the devices never enter a connected state and send out a frame with their unique ID as the source address, the device is not captured and hence does not appear in the topology. Since the address is never captured by a sniffing device, if the network topology and hence all involved devices are not known in advance, devices are missing. But even if all device IDs are known the approach does not necessary visualize all devices. In case a malicious device listens to the broadcasting address and can intercept all traffic. This device cannot be detected by this approach if only BLE traffic is captured. But it can be assumed, that the device at some point communicates with other devices to share the intercepted data. When also capturing other traffic (network), the malicious device should be captured and the unique IP or MAC addresses can be compared with all known devices.

	Small	Medium	Big
Devices	7'563	82'919	70'636
Ind. nw	374	7'587	3'615
Devices / ind. nw	20,22	10,93	19,54
pps	17,17	41,76	7,68

Table 6.8: Network Information Comparison

Network Comparison The next information to analyse are some general network level measurements as seen in Table 6.8. For the small and big experiment, the average device per individual network is 20. For the medium length experiment, the average device count is around half as the others, namely 11. A possible explanation is that a lot more point to point communication was captured, as there were more devices in the reach of the sniffer. If a lot of random addressing devices were captured and start broadcasting, the number of formed individual networks stays the same, but the devices count increases and therefore the average device count decreases. While experiment 1 and 3 so far have a similar distribution of measurements, they also differ in network pps. While the shortest run experiments has a pps of 17,17 the longest has a network pps of 7,68. This can be explained by the fact that the experiment 3 was run trough a whole day, hence over night when BLE communication is assumed to be dropping. When checking Wireshark, 438'751 out of the 672'472 (65%) were captured within the first 12 hours of the experiment. In contrast the pps of the medium length experiment with 41,76 is about double the pps of the small experiment which were run both during the day. Since in experiment 2 the pps also increases, the assumption of more device activity is further backed. The reason is, that even if a lot of random addressing would happen, the exchange of these addresses would have to lead to a doubling in pps, which is unlikely. But given that according to the frame type distribution for the medium results, more device scanning is happening. This can also mean, that a lot more devices needed to be detected or that a address has changed and rejoins a network. Hence a combination of both is assumed for the difference in results of the medium data set. Lastly, there are 89'406 individual device IDs captured over all experiments. Further a total of 1'078 devices occurred in all three data sets. It is assumed, that these are fixed devices in reach of the sniffer or general addresses such as the broadcasting address "ff:ff:ff:ff:ff:ff".

	analyze_csv()		calculate_node_pps()		visualize_topology()	
	total	n _m	total	n _d	total	n _d
Exp 1.	1,28 s	0,0182 ms	17,5 ms	0,0023 ms	3,61 s	0,4773 ms
Exp 2.	11,9 s	0,0127 ms	154 ms	0,0019 ms	972 s	11,72 ms
Exp 3.	6,81 s	0,0101 ms	204 ms	0,0029 ms	419 s	5,931 ms

Table 6.9: Run Time Analysis

6.3.2 Evaluation 3: Run Time Algorithm

In order to check the run time of the algorithm, for each data set the time is recorded per method in Table 4.1 and displayed in Table 6.9. The first row lists the methods to measure. The total column refers to the execution time needed for running this method. The n gives the term which is dominating and the value is calculated per one execution per input. As a reminder experiment 1 has $n_m = 70'194$ and $n_d = 7'563$, experiment 2 has $n_m = 977'671$ and $n_d = 82'919$ and experiment 3 has $n_m = 672'472$ and $n_d = 70'636$. The execution time of the method *visualize_topology()* is dependent two terms, n_d and n_f . The term n_f refers to the number of frame types, which is 7. Hence this method is dominated by n_d for all experiments. For the method *analyze_csv()*, the dominating term is the number of messages recorded and breaks down to around 0,014 ms per executed row $((0,0182 + 0,0127 + 0,0101) / 3)$. The *calculate_node_pps()* is dependent on the number of devices and is the quickest method with around 0,0024 ms per execution. For the these two methods, the deviation of the individual values is rather small as it is less than 0,001ms. This means, that the execution time of the two methods grow linear with the size of the input. The last time recorded is for the *visualize_topology()* method. The execution time for each n_d has a difference of over 11 ms for the smallest to the biggest input. The run time of this method hence does not grow linearly dependent on the input size, but increases with the size of the input.

While these numbers look feasible, the opening of the actual graph.html file can take a lot of time. For the middle sized data set with over 80'000 nodes, the loading of the topology took more than an hour. While the pyvis library has no maximum nodes to visualize in theory, in reality the network topology will look too cluttered. The cluttering can happen already with fewer nodes, as seen in the small data set. This problem can be circumvented by dropping the broadcasting mode in Wireshark or by selecting all devices to display but the broadcasting address "ff:ff:ff:ff:ff:ff". With his solution, the topology is visually appealing but information is lost, as all nodes which do not communicate otherwise are not displayed. Another possibility is to show the individual networks. These topologies can be grouped to decrease the number of resulting communication graphs.

6.4 Limitations

One limitation concerns the packet analyzer tool Wireshark. When sniffing network level traffic, such as in experiment 2, Wireshark uses a capture profile called "any". This means that more than just one protocol is captured. Since each of the protocols have a custom header containing transfer-related information, Wireshark adds its own uniform header. The information from this header is not always correct, as some data is assumed, for example the frame type is mostly data.

The data set can also be a limitation factor. For this approach to work, a data set needs to contain some addresses, as otherwise only the two unknown nodes are communicating.

Another problem is introduced by the protocols implementation guidelines. In order to make IoT networks more secure, random addressing is an option. As soon as the same device communicates under two unique IDs, two devices are recorded. Without any background knowledge of the topology and involved devices, the address changing nodes can not be identified. If the time of change is known, for example a device changes its address every day, the network can only be sniffed during a shorter time frame. This way, each device is only detected once. There are also passive devices in IoT networks, such as observer in BLE, which are never recorded. There is no possibility to detect said devices, as they never appear as a source or destination address. A similar problem comes from the custom implementation of a standard by companies. From personal experience it is known, that UWB is implemented differently than the standard suggests. Wireshark is not able to read the custom frames and only displays empty packages.

Dependent on the protocol captured, different information is collected. While one protocol works with IP addresses, another uses MAC addresses. The algorithm needs to be able to work with both addresses, hence IP and MAC addresses are treated the same. As the uniform approach leads to generalization, some protocol specific information is lost. This can be the exact role of a BLE device, as only master nodes in general are identified, and hence the broadcaster and observer role are not visualized.

The last limitation concerns the loading of the produced communication graph. For a network topology with 80'000 nodes, the loading took more than a hour (5 year old laptop). This is not user-friendly but there is no better visualization library. This means, no other library is even powerful enough to visualize 80'000 nodes and the user is still able to read any information as other tools don't allow interaction (scrolling in or out). But even if the loading took less time, the resulting topology is still not visually appealing, as there are simply too many nodes for a personal computer to fit on the display. Possible solutions include the visualization of each sub network individually as there are not connected, no information is lost.

Chapter 7

Conclusions and Future Work

In this chapter, this projects work is summarized and concluded. This includes some proposed enhancements to the visualization approach.

7.1 Conclusion

In a IoT network, different device types communicate with each other over diverse protocols. Some devices require low power consumption and have limited functionalities, such as a sensor only collecting data. To further handle and visualize the collected data, a more powerful device is needed with enough memory to process the data, namely smart-devices. Smart-devices manage and configure the end-devices, while also distributing the collected data to a data system. The data systems, in form of a server or the cloud, are used to store the sensor data and prepare for further use.

The communication types also have different requirements. On one hand, the communication between the end- and smart device is short-ranged, fast, energy-saving but limited in endpoints. This means, a smart-device can handle a limited amount of endpoints. The communication range is below a few meters but the data is transferred quickly. The protocols used on the lower level are BLE and UWB.

On the other hand is the long-range but rather slow data transfer between smart devices and data systems. This higher level communication is able so serve a high amount of endpoints and hence has a high power consumption. The mentioned standards include TCP and UDP.

The diverse requirements and characteristics of IoT networks present a unique challenge for a network detection and visualization approach. An approach needs to facilitate multiple devices types and communication standards. This project presents an approach, where diverse IoT network traffic are analyzed and the underlying topology visualized. To ensure a uniform approach, traffic of different IoT network protocols is captured and further processed.

The first experiment analyses UWB traffic. The data set was provided by another project, implementing an UWB network in a closed environment. Hence, the expected topology is known and is used to validate the visualization approach.

Experiment 2 researches two data sets which captured diverse higher level communication protocols, such as TCP and UDP. The data set was recorded by another project and comes two fold, once under attack and in a normal state. Both data sets are recorded in a lab environment. This experiment allows for an comparison between a network under attack and one that is not.

The third and last experiment takes a closer look at BLE traffic. In contrast to the other two experiments, the BLE traffic was captured by a sniffing device in the real-world. This BLE experiment further involves a time factor, as the BLE traffic was captured for different time frames. This allows for the evaluation of real world BLE traffic.

The projects approach works and visualizes the correct topology based on the data set. Yet, the resulting topology is only as good as the captured data. This means, the presence of a lot of empty source and destination addresses yields in a informative topology, as most communication will end in a unknown node. Further characteristics of some of the protocols pose a challenge, for example random addressing. Devices which change their unique ID periodically are not detectable as just one device. On the other hand, some protocols involve passive devices, meaning they wait for a certain message and don't have to be addressed directly, such as observers in BLE. Passive devices, if not directly addressed in the destination address, are not detectable.

7.2 Future Work

There are several ways to extend this project. One option involves enhancing the user interface. Currently, this approach is only usable in a python notebook. All configurations regarding devices or the path of the files needs to be entered manually into the notebook. Thus, a user interface can be added, where the files and configurations can be defined.

The next extension concerns the package capture tool, in this case Wireshark. Since this project follows a uniform and generalization approach, fields present in every protocol capture configuration are selected. However, Wireshark captures different information depending on the configured protocol, particularly for network data, as it adds a uniform link header. Hence if tools can identify the correct protocol, deeper analysis is possible as also protocol relevant information can be added. As an example, if the frame types and protocols can be trusted (link header often assumes type data) in Wireshark, then incorrect behaviour can be identified based on the topology by looking at a nodes frame type distribution. If a node communicates with TCP, then a certain distribution of frame types can be expected based on the three-way-handshake, any deviation can be interpreted as an unexpected behaviour.

The same applies to company-specific protocol implementations. As not all configurations are known, for example in real-world UWB cases, it can be unclear whether the information displayed in Wireshark is accurate. That is because Wireshark only analyzes the package structure based on the configured capture profile. If a company-specific implementation changes the order of fields in the UWB header, Wireshark may misinterpret the data. Thus, the displayed information may not be entirely reliable

In summary, if the information can be trusted, more protocol specific analysis can be executed. The project currently only executes whole network comparisons, e.g. number and ID of shared/different nodes. This can be extended by bringing the comparison on a node level where a user can compare one specific nodes information from various topologies.

The approach demonstrates the use of the tool with three different data sets. This is because, at present, different protocols require different tools for capture. If a user wishes to combine multiple protocols, the captured traffic from multiple Pcap files can be concatenated into a single csv file. This resulting larger csv file can then be input into the code. If the addresses of connecting nodes are captured (e.g., smart and end devices), the smart device will likely appear in two csv files: one in the lower-level (BLE or UWB) and one in the network data set. This indicates the presence of a connecting node between the higher and lower levels of communication within the topology.

Bibliography

- [1] S. Madakam, R. Ramaswamy, and S. Tripathi, "Internet of things (iot): A literature review," *Journal of Computer and Communications*, vol. 3, pp. 164–173, 04 2015.
- [2] P. Gokhale, O. Bhat, and S. Bhat, "Introduction to iot," *International Advanced Research Journal in Science, Engineering and Technology*, vol. 5, no. 1, pp. 41–44, 2018.
- [3] S. Al-Sarawi, M. Anbar, K. Alieyan, and M. Alzubaidi, "Internet of things (iot) communication protocols," 2017.
- [4] A. Khanna and S. Kaur, "Internet of things (iot), applications and challenges: a comprehensive review," *Wireless Personal Communications*, vol. 114, pp. 1687–1762, 2020.
- [5] G. Nunberg, "The advent of the internet," *12th April, Courses*, 2012.
- [6] M.-O. Pahl and S. Liebold, "A modular distributed iot service discovery," pp. 448–454, 2019.
- [7] G. Fortino, C. Savaglio, G. Spezzano, and M. Zhou, "Internet of things as system of systems: A review of methodologies, frameworks, platforms, and tools," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 51, no. 1, pp. 223–236, 2021.
- [8] "Internet of things (iot) communication protocols." <https://ieeexplore.ieee.org/abstract/document/8717904>. (Accessed: 2023-06-15).
- [9] "Europeâs internet of things policy | shaping europeâs digital future.." <https://digital-strategy.ec.europa.eu/en/policies/internet-things-policy>. (Accessed: 2023-06-15).
- [10] "Owasp internet of things | owasp foundation.." <https://owasp.org/www-project-internet-ofthings/>. (Accessed: 2023-06-15).
- [11] T. Haramaki and H. Nishino, "A network topology visualization system based on mobile ar technology," pp. 442–447, 2015.
- [12] S. H. Kunugi Yuya and K. Akio, *IoT Security Viewer System Using Machine Learning*. Springer International Publishing, 2020.
- [13] L. W. Mengnan Gao, Q. Li, and W. Chen, "Anomaly traffic detection in iot security using graph neural networks," *Journal of Information Security and Applications*, vol. 76, p. 103532, 2023.

- [14] M. Naldi, "Connectivity of waxman topology models," *Computer communications*, vol. 29, no. 1, pp. 24–31, 2005.
- [15] N. Kojić, I. Reljin, and B. Reljin, "A neural networks-based hybrid routing protocol for wireless mesh networks," *Sensors*, vol. 12, no. 6, pp. 7548–7575, 2012.
- [16] L. Saino, C. Cocora, and G. Pavlou, "A toolchain for simplifying network simulation setup.," *SimuTools*, vol. 13, pp. 82–91, 2013.
- [17] J. Ali, R. H. Jhaveri, M. Alswailim, and B.-h. Roh, "Escalb: An effective slave controller allocation-based load balancing scheme for multi-domain sdn-enabled-iot networks," *Journal of King Saud University-Computer and Information Sciences*, vol. 35, no. 6, p. 101566, 2023.
- [18] M. Sanni, A. Hashim, F. Anwar, G. Ahmed, and S. Ali, "How to model wireless mesh networks topology," vol. 53, no. 1, 2013.
- [19] T. N. Alrumaih and M. J. Alenazi, "Genind: An industrial network topology generator," *Alexandria Engineering Journal*, vol. 79, pp. 56–71, 2023.
- [20] A. Ulmer, D. Sessler, and J. Kohlhammer, "Netcapvis: Web-based progressive visual analytics for network packet captures," pp. 1–10, 2019.
- [21] D. Arendt, D. Best, R. Burtner, and C. Paul, "Cyberpetri at cdx 2016: Real-time network situation awareness," pp. 1–4, 10 2016.
- [22] P. Desai, A. Sheth, and P. Anantharam, "Semantic gateway as a service architecture for iot interoperability," 10 2014.
- [23] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges," *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497–1516, 2012.
- [24] K. Panigrahi. <https://www.tutorialspoint.com/difference-between-wifi-and-bluetooth>, 2022. (Accessed: 2023-05-15).
- [25] S. Flynn. <https://www.techtarget.com/iotagenda/tip/A-look-at-the-top-5-IoT-device-types>, 2022. (Accessed: 2023-05-15).
- [26] "Part 4 : A walk through internet of things (iot) basics.." <https://opentechdiary.wordpress.com/2015/07/18/part-4-a-walk-through-internet-of-things-iot-basics/>. Accessed: 2023-08-29.
- [27] P. Dudhe, N. Kadam, R. M. Hushangabade, and M. S. Deshmukh, "Internet of things (iot): An overview and its applications," in *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, pp. 2650–2653, 2017.
- [28] B. Crow, I. Widjaja, J. Kim, and P. Sakai, "Ieee 802.11 wireless local area networks," *IEEE Communications Magazine*, vol. 35, no. 9, pp. 116–126, 1997.

- [29] “Ieee standard for low-rate wireless networks - redline,” *IEEE Std 802.15.4-2020 (Revision of IEEE Std 802.15.4-2015) - Redline*, pp. 1–1294, 2020.
- [30] <https://learn.saylor.org/mod/book/view.php?id=32876chapterid=20347>. (Accessed: 2023-05-15).
- [31] P. Mandl, *TCP und UDP Internals: Protokolle und Programmierung*. Springer Fachmedien Wiesbaden, 2018.
- [32] M. Mullins, “Exploring the anatomy of a data packet,” 2001.
- [33] K. R. Fall and W. R. Stevens, *TCP/IP illustrated, volume 1: The protocols*. addison-Wesley, 2011.
- [34] <https://www.computerweekly.com/de/definition/Client-Server>, 2023. (Accessed: 2023-05-15).
- [35] W. Jia and W. Zhou, *Distributed Network Systems: From Concepts to Implementations*. Network Theory and Applications, Springer US, 2006.
- [36] R. Yager and J. Espada, *New Advances in the Internet of Things*. Studies in Computational Intelligence, Springer International Publishing, 2017.
- [37] <https://www.geeksforgeeks.org/types-of-network-topology/>, 2023. (Accessed: 2023-05-15).
- [38] D. Serpanos and M. Wolf, *Internet-of-Things (IoT) Systems: Architectures, Algorithms, Methodologies*. Springer International Publishing, 2017.
- [39] N. K. Gupta, *Inside Bluetooth low energy*. Artech House, 2016.
- [40] N. Hlapisi, “Exploring the basics of bluetooth low energy: A beginners guide to ble,” 2023.
- [41] “Bluetooth® low energy packet types.” <https://microchipdeveloper.com/wireless:ble-link-layer-packet-types>, 2021.
- [42] M. Galinski, “Introduction to bluetooth low energy advertisements,” 2023.
- [43] “Bluetooth® low energy packet types.” <https://microchipdeveloper.com/wireless:ble-link-layer-address>, 2021.
- [44] “Privacy in ble.” <https://lpccs-docs.renesas.com/Tutorial-DA145x-BLE-Security/privacy.html>, 2022.
- [45] “Ble link layer roles and states.” <https://microchipdeveloper.com/wireless:ble-link-layer-address>, 2021.
- [46] C. Gupta and G. Varshney, “An improved authentication scheme for ble devices with no i/o capabilities,” *Computer Communications*, vol. 200, pp. 42–53, 2023.
- [47] “Bluetooth® low energy gap roles.” <https://microchipdeveloper.com/wireless:ble-gap-roles>, 2021.

- [48] “IEEE standard for low-rate wireless networks—amendment 1: Enhanced ultra wide-band (UWB) physical layers (PHYs) and associated ranging techniques,” IEEE, C/LM - LAN/MAN Standards Committee, 2020.
- [49] D. Coppens, E. De Poorter, A. Shahid, S. Lemey, and C. Marshall, “An overview of ultra-wideband (UWB) standards (IEEE 802.15. 4, FiRa, Apple): Interoperability aspects and future research directions,” *arXiv preprint arXiv:2202.02190*, 2022.
- [50] “IEEE standard for low-rate wireless networks,” IEEE, C/LM - LAN/MAN Standards Committee, 2020.
- [51] W. Cao, W. Xia, D. Sun, F. Yan, L. Shen, Y. Zhang, and Y. Gao, “A hierarchical ble mesh network for iot and performance analysis,” pp. 01–06, 2021.
- [52] M. Cermak and D. Sramkova, “Granef: Utilization of a graph database for network forensics,” in *SECRYPT*, pp. 785–790, 2021.
- [53] L. Diederichsen, K.-K. R. Choo, and N.-A. Le-Khac, “A graph database-based approach to analyze network log files,” pp. 53–73, 12 2019.
- [54] J. Uhlir, M. Holkovič, and V. Rusák, “Pcapfunnel: A tool for rapid exploration of packet capture files,” in *2021 25th International Conference Information Visualization (IV)*, pp. 69–76, 2021.
- [55] “Ubertooth one.” https://ubertooth.readthedocs.io/en/latest/ubertooth_one.html, 2023. (Accessed: 2023-09-01).
- [56] “Build guide.” https://ubertooth.readthedocs.io/en/latest/build_guide.html, 2023. (Accessed: 2023-09-01).
- [57] “Linux cooked-mode capture (sll).” <https://wiki.wireshark.org/SLL.md>, 2020.
- [58] <https://networkx.org/>, 2023. (Accessed: 2023-05-15).
- [59] <https://github.com/apache/superset>, 2023. (Accessed: 2023-05-15).
- [60] <https://matplotlib.org/>, 2023. (Accessed: 2023-05-15).
- [61] <https://www.analyticsvidhya.com/blog/2020/06/guide-geospatial-analysis-folium-python>, 2023. (Accessed: 2023-05-15).
- [62] <https://pyvis.readthedocs.io/>, 2023. (Accessed: 2023-05-15).
- [63] C. Eder, “Design and evaluation of ultra-wideband (uwb) architectures with a focus on privacy-preserving characteristics,”
- [64] W. martin, “Bluetooth technology protecting your privacy,” 4 2015. (Accessed: 2023-09-28).

Glossary

AI Artificial Intelligence. 1, 17

BLE Bluetooth Low Energy. 2, 5, 10–17, 20, 21, 39, 45, 64, 65, 67–69, 71, 73, 85–87, 89, 91–93

CRC Cyclic Redundancy Check. 11

csv Comma-separated Values. 22, 23, 34, 36, 41, 42, 44, 48, 93, 106

DDoS Distributed Denial of Service. 27, 43

FFD Full-function Device. 15, 16

HTML Hypertext Markup Language. 106

HTTP Hypertext Transfer Protocol. 17

IoT Internet of Things. 1–5, 9, 10, 15, 17, 19–21, 39, 43, 65, 89, 91

IP Internet Protocol. 8, 21, 43, 44, 82, 86, 89

IPv4 Internet Protocol Version 4. 8, 23

IPv6 Internet Protocol Version 6. 8

LR-WPAN Low-rate Wireless Personal Area Network. 14

MAC Medium Access Control. 8, 11, 12, 15, 21, 23, 43, 44, 69, 82, 86, 89

MFR MAC Footer. 15

MHR MAC Header. 15

MITM Man-in-the-Middle. 43, 61, 82

PAN Personal Area Network. 15, 16

Pcap Package Capture Dump File Format. 17, 22, 41, 44, 48, 74, 75, 93

PcapNg Package Capture next Generation Dump File Format. 106

PDU Protocol Data Unit. 11, 12

PHR Physical Header. 15

PHY Physical. 15

pps Packages per second. 22, 23, 25–27, 29–32, 38, 47, 49–51, 54–63, 65, 67–69, 71, 72, 74–77, 80–84, 87

PSDU Physical Service Data Unit . 15

RFD Reduced-function Device. 15, 16

SHR Synchronization Header. 15

SLL Linux cooked-mode Capture. 80

TCP Transmission Control Protocol. 5–8, 22, 80, 91–93

UDP User Datagram Protocol. 5–8, 23, 91, 92

USB Universal Serial Bus. 20

UWB Ultra-Wideband. 5, 14–17, 20–22, 39, 41, 42, 47–49, 52, 53, 73, 74, 78, 79, 89, 91–93

Wi-Fi Wireless Fidelity. 5

WLAN Wireless Local Area Network. 5, 10, 21, 43, 57, 60, 61, 80

WPAN Wireless Personal Area Network. 5

List of Figures

2.1	IoT Network [22]	4
2.2	UDP Header Structure (based on [32; 30])	6
2.3	TCP Header Structure (based on [32; 30])	7
2.4	Frame Encapsulation (based on [32; 30])	8
2.5	BLE Frame Structure [41]	11
2.6	BLE Device Addresses [43]	12
2.7	BLE Topology Options [47]	13
2.8	UWB Frame Structure [50]	15
2.9	UWB Topology Options [50]	16
4.1	Captured Parameters	20
4.2	Network Source Address	22
4.3	UWB Source Address	22
4.4	Example Captured Package	23
4.5	Example Activity Timeline	33
4.6	Topology in the Passive Localization System [63]	40
4.7	Experiment 1: Captured UWB Packages [63]	41
4.8	Experiment 1: Rows with empty "Source" and "Destination" field	41
4.9	Experiment 1: Adapted Data Set Fields	42
4.10	Experiment 2.2: Identification IP Address	44
5.1	UWB Time Configuration	48

5.2	Result 1.1: Network Level	49
5.3	Result 1.1: Node and Edge Information	50
5.4	Result 1.1: "UNKNOWN" Nodes Topology Retrieval	51
5.5	Result 1.1: Node Activity Timeline	53
5.6	Result 1.2: Network Level	54
5.7	Result 1.2: Node Activity Timeline	56
5.8	Result 2.1: Network Topology	58
5.9	Result 2.1: Network Node	58
5.10	Result 2.1: "UNKNOWN_DESTINATION" node	59
5.11	Result 2.1: Network Level	61
5.12	Result 2.2: Network with Malicious Nodes	63
5.13	Result 3.1: Network Level	65
5.14	Result 3.1: Network Topology without Broadcast	67
5.15	Result 3.2: Network Level	68
5.16	Result 3.2: Node Activity Timeline	70
5.17	Result 3.3: Network Level	71
6.1	Evaluation 1.1: Networks and Device Count Validation	75
6.2	Evaluation 1.2: Sub Network Node	77
6.3	Evaluation 1.2: Visualization Approach	78
6.4	Evaluation 2.2: IP sharing of malicious Nodes	82
A.1	Result 2.2: Node Activity Timeline	107
A.2	Result 2.2: Node Activity Timeline with devices_to_select = ['00:0c:29:5b:73:5b','00:0c:29:b8:b1:74']	

List of Tables

4.1	Time Complexity Analysis	36
5.1	Experiment 1: Analysis	48
5.2	Experiment 2: Analysis	57
5.3	Experiment 3: Analysis	64
6.1	Evaluation 1.1: Node Information Validation	74
6.2	Evaluation 1.1: Edge and Node Information Validation	75
6.3	Evaluation 1.1: Special Nodes Validation	76
6.4	Devices highlighted in 'darkgreen'	81
6.5	Devices highlighted in 'yellow'	81
6.6	Frame Type Distribution	86
6.7	Device Behaviour Distribution	86
6.8	Network Information Comparison	87
6.9	Run Time Analysis	88

Appendix A

Appendix

All additional material used in this project is stored in a Github repository under the following link: https://github.com/AndrinaVincenz/IoTNetworks_MA. This includes a python notebook with methods and the visualization approach, the BLE, UWB and network data set and all generated topology graphs. As most graphs and data sets are too big for Github, an additional drive folder containing all data is found under the following link: https://drive.google.com/drive/folders/130WzaDoh0q-Tt_H4LJ1JRP AE-HjLI28J?usp=sharing Additionally, a .zip version of the repository is handed in together with this thesis.

A.1 Data Sets

File-path: ExperimentResults/Exp1_UWB/Datasets

Description: Results of experiment 1 as PcapNg and csv files.

File-path: ExperimentResults/Exp2_NW/Datasets

Description: Results of experiment 2 as PcapNg and csv files.

File-path: ExperimentResults/Exp3_BLE/Datasets

Description: Results of experiment 3 as PcapNg and csv files.

A.2 Communication Graphs

File-path: ExperimentResults/Exp1_UWB/Topology_graphs

Description: Topology graphs of experiment 1 as HTML files.

File-path: ExperimentResults/Exp2_NW/Topology_graphs

Description: Topology graphs of experiment 2 as HTML files.

File-path: ExperimentResults/Exp3_BLE/Topology_graphs

Description: Topology graphs of experiment 3 as HTML files.

A.3 Installation Guidelines

Configuration of input files captured as pcap in Wireshark:

1. Columns in following order: Nr., Time, Source, Destination, Protocol, Length, Info
2. Time configuration: Since start of Recording in milliseconds
3. Download as csv

Configuration in Python Notebook:

1. Define file paths to csv
2. Define file names to distinguish topology graph as string
3. Define nodes to show in topology
4. Define which node groups to highlight

A.4 Timelines of Experiment 2.2



Figure A.1: Result 2.2: Node Activity Timeline



Figure A.2: Result 2.2: Node Activity Timeline with `devices_to_select = ['00:0c:29:5b:73:5b', '00:0c:29:b8:b1:74']`