AGH University of Science and Technology

Faculty of Electrical Engineering, Automatics, Computer Science and Biomedical Engineering
Department of Metrology and Electronics

Master of Science Thesis

Performance evaluation of mesh networking protocols

Porównanie parametrów protokołów komunikacji bezprzewodowej typu mesh

Filip Zajdel

Field of study:
Microelectronics in Industry and Medicine

Supervisor:
Piotr Otfinowski, PhD

Krakow, 2022

Abstract

Mesh protocols have been providing connectivity to smart homes and wireless sensing for about two decades. There is a variety of approaches to building mesh networks proposed by different research groups. Two protocols: Zigbee and Thread are discussed and compared in this thesis. Although they have similar features and behaviors, there are some significant differences in the architectures of networks built with them.

Comparing the performance of the two mentioned protocols running nRF SoCc has been a problem because method available for users of these devices had not existed. This dissertation provides an overview of these protocols alongside the performance evaluation of networks built with them. Moreover, the method of collecting performance metrics of Zigbee and Thread protocols is proposed. This method consists of using applications built with Nordic Semiconductor Connect SDK. It was used to perform the latency and throughput measurements of networks with up to 7 nodes. In a summary, the results are presented and discussed. It is concluded that Thread introduces lower latency and higher throughput than Zigbee. Although the results show Thread's advantage over Zigbee, there are other factors than performance that are important when a network protocol is chosen for a product.

Acknowledgments

I would like to express gratitude to my supervisor Piotr Otfinowski Ph.D. for his charity, patience, transparency, and immense knowledge that he has shared with me not only during this project but also for the whole period of pursing the degree in the field of microelectronics.

I am also grateful to Wojciech Bober Ph.D. for his support, ideas, time, and inspiration. His work culture, skills and knowledge has been inspiring me since I met him.

Lastly, I would like to acknowledge all Nordic Semiconductor's employees who have participated in this project for allowing me to work on this project and providing me with everything I needed.

Contents

1 Introduction **9**
1.1 Motivation 9
1.2 Problem statement 9
1.3 Objectives 9
1.4 Contribution 10

2 Fundamentals of mesh networking protocols **11**
2.1 IEEE 802.15.4 protocol 11
2.2 Introduction to Zigbee 11
2.3 Introduction to Thread 14
2.4 Thread stack layers and frame format 14
2.5 Thread device types 15
2.6 Comparison of Thread and Zigbee protocols 16

3 Research methodology **17**
3.1 Performance metrics 17
3.2 Tools used for testing 19
3.3 Test setup 20
3.4 Problems encountered 21
3.5 Thread network setup automation 23
3.6 Performance evaluation procedure 25
3.7 Test results verification 28

4 Results analysis **31**
4.1 Latency 31
4.2 Throughput 34
4.3 Performance comparison 36

5 Summary and conclusion **39**
5.1 Conclusion of performance comparison between Thread and Zigbee . . 39
5.2 Conclusion of research methodology 39
5.3 Future work 40

A Full latency measurements results **41**

Bibliography **49**

Chapter 1

Introduction

1.1 Motivation

An expanding market of IoT devices requires manufacturers and their engineers to adopt to new standards or rework the old ones. The enterprise IoT market has grown by 23% since 2021 [10]. A huge part of this business is taken by home automation products which wouldn't exist without chips combining power saving designs and wireless connectivity.

This dissertation focuses on two network protocols used mainly by home automation devices: Zigbee and Thread. Both of them are supported by Nordic Semiconductor's SoC nRF52840. Although these protocols share many features and are based on the same physical layer- IEEE 802.15.4, their designs and targets are different.

1.2 Problem statement

Zigbee as well Thread are standards designed for energy efficient low-data-rate applications. Both protocols share the same purpose and are built upon the same MAC layer. Although mesh networks using them behave similarly, their protocol stacks significantly differ.

The main focus of this thesis is to develop methods of measuring the performance of these protocols and compare them in terms of gathered results.

1.3 Objectives

The main objective is to develop the methodology for performance evaluation of Zigbee and Thread run on Nordic Semiconductor's SoCs. These measures aim to gather performance metrics, which are then used for comparison of these protocols.

1.4 Contribution

The contribution of this thesis include study of Thread and Zigbee protocols, porting and modifying benchmark and zperf application for performance measurement and results analysis.

Chapter 2

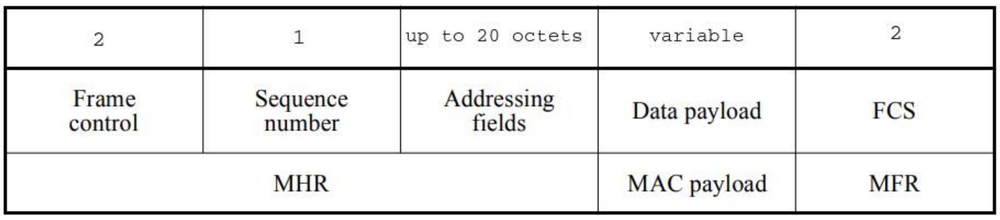
Fundamentals of mesh networking protocols

2.1 IEEE 802.15.4 protocol

The IEEE 802.15.4 serves as a physical and MAC layers for both Zigbee and Thread technologies. This thesis focuses on 2.4 GHz band, because that's the frequency the nRF 52840 SoCs operates on. But IEEE 802.15.4 supports other bands as well (868.0-868.6 MHz, 902-928 MHz)[11].

The protocol offers communication withing short range (approximately 10 meters) and the transfer rate of 250 kbit/s.

The format of an IEEE 802.15.4 frame is an important aspect to discuss. Figure 2.1 presents an IEEE 802.15.4 data frame format. This kind of frame will be used by Zigbee and Thread to send the data. The maximum length of any 802.15.4 frame is **127** bytes. Up to 25 bytes of the frame is used by Frame control, Sequence number, Addressing and FCS fields, leaving up to **102** bytes for the payload.



2	1	up to 20 octets	variable	2
Frame control	Sequence number	Addressing fields	Data payload	FCS
MHR			MAC payload	MFR

Figure 2.1: An IEEE 802.15.4 Data Frame format, from [2].

2.2 Introduction to Zigbee

Zigbee is a software networking protocol that provides connectivity over an ISM radio bands. It is used mainly for home automation, medical device data collection and other domains that require low-power short range wireless connectivity. The protocol defines a set of layers built upon IEEE 802.15.4 radio protocol, which serves as a physical and partly as a MAC layers[13][12].

2.2.1 Device roles[1]

3 device roles are defined by Zigbee standard. These are: coordinator, router and (sleepy) end device. Depending on a role, a Zigbee node carries different responsibilities which are especially important at a low layer of the stack.

The network coordinator is responsible for starting and forming the network. Every network has exactly one coordinator and is assigned a randomly generated Pan ID, which is its 16 bit unique identifier. The coordinator chooses the most optimal channel to operate on and sets up the network to use it. Any device eager to join the network must operate on the same channel and be let in by the coordinator. The process of joining devices that hadn't yet been the members of the network is called **association**. When device is associated, it received an unique 16 bit random ID is also called short or network address.

Routers, as well as the network coordinator, participate in associating new devices into the network. Before they are allowed to do so, they have to be associated into the network as well. When a router becomes a member of the network it acts as a proxy between different devices in the network. They relay packets from one node to another. Although routers take part in associating new devices they are not allowed to let a new devices into the network. In fact they extend coordinator's range and take over the initial part of associating procedure (sending Beacons and Association Response packets)[13]. When a router is responsible for associating a new device it becomes its **parent** and the new node becomes a **child** of the former. It is worth to mention that network coordinator behaves as a router in addition to its functions mentioned before.

In the Zigbee network, packets are sent asynchronously. Imagine using Zigbee for switching the lights off or moving the blinds aside. These are clearly asynchronous signals so the network has to be. For that reason, routers must stay on all the time, but the nodes which fulfil the sleepy end device role don't have to. These devices are designed to operate on batteries and exchange little data. End devices don't route packets and communicate with other nodes through their parent. The data from other devices are not sent to them directly as well. They request their parent for the data in order to get any. There are two types of end devices: sleepy and non sleepy. The former turn off their radio during periods of inactivity to lower the power consumption[5], whereas the latter listens all the time.

2.2.2 Security

Every Zigbee network employs the Trust Center device, which is usually glued together with the network coordinator, but can be implemented as a separate device

[1]Please note that roles descriptions in this section are relevant for centralized security model (2.2.2).

as well. If the trust center is altogether with network coordinator, then a security is referred as *centralized*. In the opposite case its *distributed*[4].

Trust center is an entity responsible for securing the communication within the network by generating and distributing the keys to the nodes within the network. The communication inside the network is secured with 2 different types of keys: network key and link key. Whilst the network key is known by every node in the network, the link keys are unique for all of them[4].

It is worth to mention that in case of Zigbee networks built with ZBOSS stack (the one used by Nordic Semiconductor), only centralized security model is supported.

2.2.3 Zigbee stack layers

The protocol is logically divided into several software layers that serve different functions in the stack. Zigbee is aimed to run on devices with insignificant performance, thus the stack architecture is relatively simple. The stack remains an IP protocol stack. However it is specifically designed to be used altogether with IEEE 802.15.4 and the physical layer cannot be changed to another phy.

The Zigbee stack (presented in Fig. 2.2) consists of 5 layers: Phy, Mac, Network, APS, AF. Except the Application Framework (AF), the rest of layers is not discussed in this thesis. For more details about them, please refer to Zigbee Specification [13].

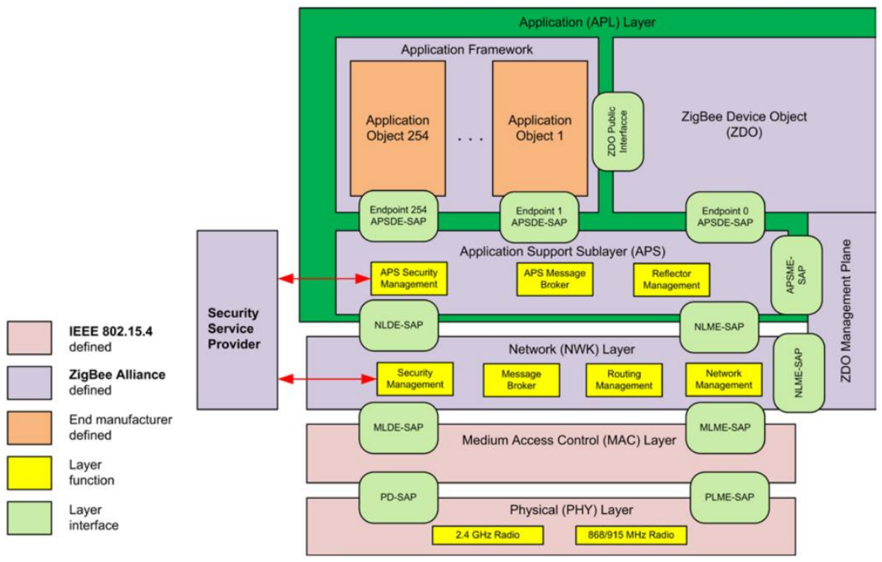


Figure 2.2: Layers of the Zigbee stacks, from [13].

Application Framework layer and Zigbee Cluster Library

The top layer of the Zigbee stack - the Application Framework - is the one that users mostly interact with. This building block provides services for the application entities to operate. The AF exposes 256 (254 available to the user) placeholders to host different applications objects. These placeholders are called **endpoints** are given unique ids (0-255).

A specific feature of the Zigbee protocol is that applications are standardized. The set of available applications are specified in Zigbee Cluster Library specification. An entity representing single functionality of a device is called **cluster**. An application consists of a set of clusters which fulfils more complex functionality.[3]

2.2.4 Zigbee frame format

Each layer in the Zigbee stack consumes some part of the frame, decreasing the available space for the higher layers. Just like IEEE 802.15.4 protocol's necessary fields occupy 25 bytes of the frame, Zigbee layers (NWK, APS, ZCL) takes 23 bytes of the remaining length. The application data must fit into **79** bytes long field.

2.3 Introduction to Thread

Thread is an open standard designed for low-power wireless communication between performance-constrained devices at lower distances (10m). Its use cases are similar to the ones of Zigbee (2.2), though Thread is an IP-based.

2.4 Thread stack layers and frame format

Although Thread stack's layers perform analogical function to the layers in Zigbee's stack, the protocols used on each layer are not defined by the standard. Thread takes advantage of IEEE 802.15.4 as a MAC and physical layer as well. The network management and routing is held by 6LoWPAN which is a protocol enabling to run IPv6 over low-power wireless networks. It is responsible for compression and fragmentation of IPv6 packets so they can fit into an IEEE 802.15.4 frame[1].

On the top of 6LoWPAN, a transport layer protocols are used. In this thesis UDP protocol was utilized, but TCP can run it as well. This layer provides a convenient interface based on sockets for application layer protocols. These are not covered in this thesis, because none of them were used.

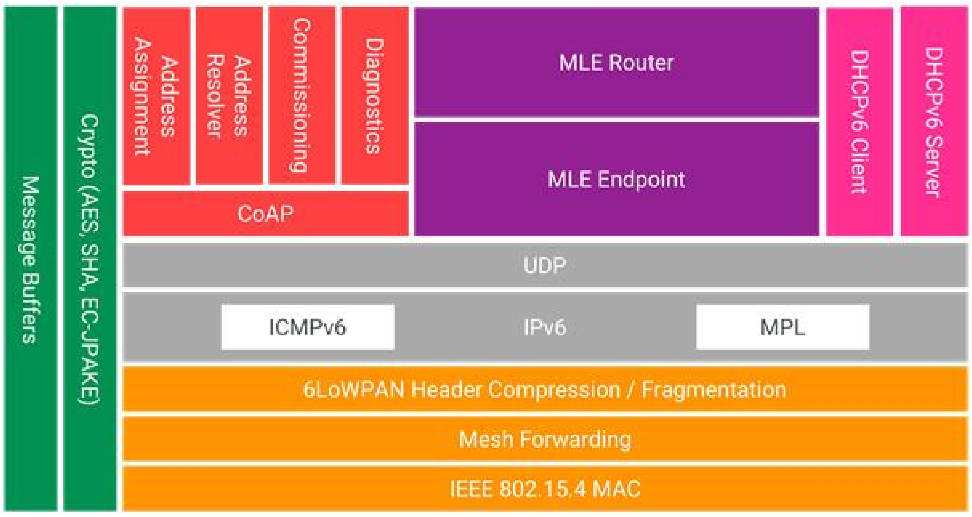


Figure 2.3: Thread stack architecture, from openthread.io.

An available payload size of an UDP packet depends on how much the IPv6 header can be compressed. In the project, a maximum size of **78** bytes was used. The compression of the header depends on on a variety of factors that include: type of IPv6 addressing, 802.15.4 addressing mode and the availability of the network context[8].

2.5 Thread device types

There are 4 device types in Thread[7]. These are border routers, routers, router-eligible devices and sleepy end devices.

The border router is a device acting as a gateway between Thread and another adjacent IP-based network. It can, for example, connect the Thread with WiFi or Ethernet-based networks. There is no limitation on the number of border routers in the network.

Another device type is the router, which is responsible for providing routing services to network devices. They always remain on and are not allowed to sleep. Besides passing the data from one node to another, they participate in joining and securing the communication in the network[9].

The next type, derived from the router, is called the router-eligible end device. A node of this type acts as a router, but due to special network topology or conditions, it can act as an end device. They have limited functions compared to router devices. They don't provide joining or security services for other devices and are limited to forwarding packets[7].

Similar to Zigbee, Thread has sleepy end devices. These can only communicate with their parents. Sleepy end devices' functions are limited to hosting and manipulating the data. They can sleep and thus save energy[7].

2.6 Comparison of Thread and Zigbee protocols

Although Thread and Zigbee serve similar purposes, their networking stack architectures differ significantly. Protocol layers in Zigbee are designed especially for this stack which can make it more robust and efficient. Thread stack consists of a set of different networking protocols which makes it more versatile compared to Zigbee. Being IP-based is a big advantage of Thread, enabling Thread devices to be available on the Internet with lower cost and effort than Zigbee devices.

However, Thread does not specify the application layer and any functionality and device behaviors need to be implemented. Zigbee, on the other hand, features ZCL and Home Automation profile which give fundamental functionalities out of the box. These functions are sometimes sufficient when a simple product like a light bulb or windows cover controller is developed. Zigbee runs devices with less performance than Thread, which is advantageous in terms of device fabrication cost.

Chapter 3

Research methodology

3.1 Performance metrics

Latency

The delay between packet's transmission from one node to another is one of the most crucial network performance parameters. Although IEEE 802.15.4 networks are not designed for high data rates, the latency is a valuable metric which depends on a variety of factors.

The latency is computed as a half of Round Trip Time (RTT)- the full delay between sending the packet from one node and receiving it from the destination node. Figure 3.1 depicts the round trip of a packet.

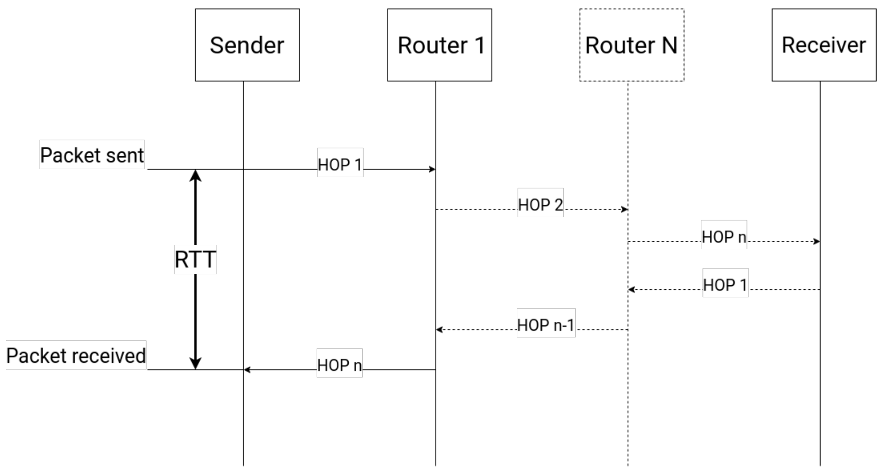


Figure 3.1: A packet's round trip

In this thesis, RTT was determined by dedicated applications running on a nodes in the network (either Zigbee or Thread). Additionally, the RTT was selectively calculated using packets timestamps from traffic dump collected with Wireshark (sec.

3.2.2). However, the latter was used rarely and mainly for ensuring the calculations correctness performed by applications. These are described in details in sections 3.2.2 and 3.2.3.

One of aspects, which latency depends most on, is the network topology. In this thesis, the latency was measures in the relationship to the number of hops the packet must go through from the sender node to the destination.

Another factor affecting the time of packet delivery is length of sent payload data. Experiments performed for this thesis used different lengths of data- from 0 to 79 bytes. This parameter has a direct impact on the packet's time in the air. At this point, sending an empty payload (which length equals to 0 bytes) can seemingly be pointless. However, term **payload length** refers to the length of data sent through an application layer and shouldn't be confused with a IEEE 802.15.4 payload. That said, a packet carrying an empty payload can be valuable as well. An exemplary use case for such a packet is sending a notification utilizing only an application layer header without any payload.

Throughput

The overall effective data transmission rate is referred as **throughput** and expressed in kilobits per second (**kbps**). In this project, throughput was calculated as a ratio of successfully exchanged data between the sender and received and the overall time of a single experiment. Each run of the experiment was approximately 10 seconds long and about 150 packets were transmitted from the sender. Figure 3.2 depicts the methodology of throughput calculation.

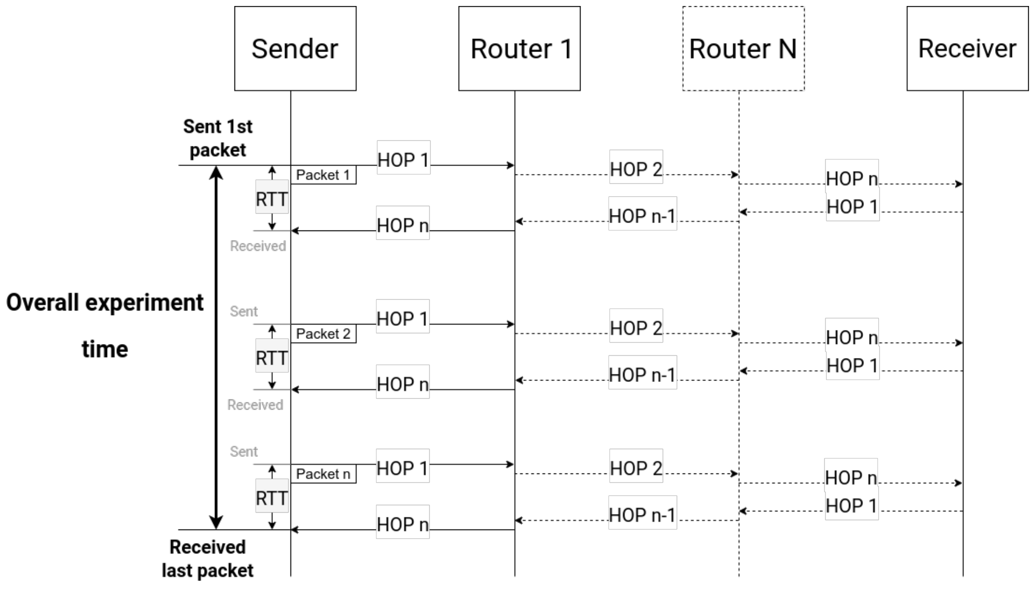


Figure 3.2: An overall experiment time calculation.

3.2 Tools used for testing

3.2.1 Hardware

The platform chosen for test implementation was Nordic Semiconductor nRF52840. This SoC features 2.4 GHz radio and supports a broad variety of wireless protocols. Except for the bare Soc intended to assembly in a custom PCB, the device is manufactured in two main versions: nRF52840 DK and nRF52840 Dongle. Both of them were utilized for this thesis.

Mesh networks were deployed onto the set of 7 nrf52840 DK boards shown in the figure 3.3. All devices were connected to the workstation via USB cables.

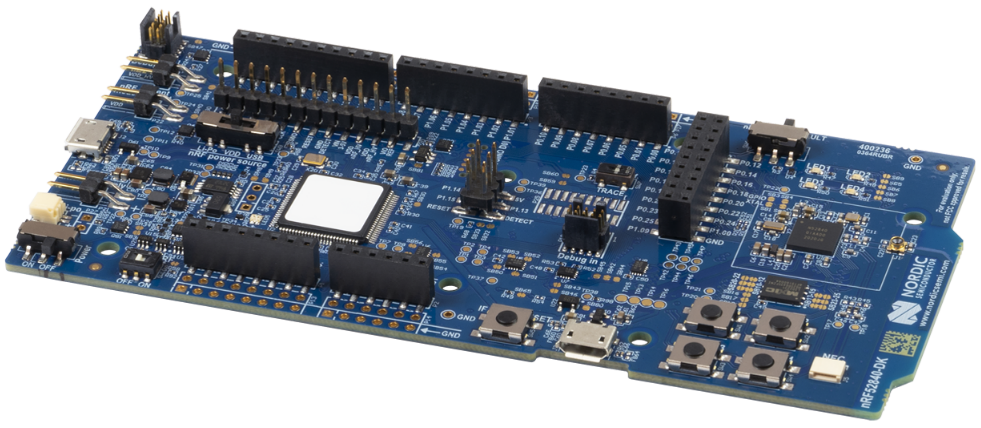


Figure 3.3: The nRF52840 DK, from nordicsemi.com.

3.2.2 Benchmark application

The most recent implementation of Zigbee and Thread protocols for nRF52840 are shipped as a components of nRF Connect SDK (abbreviated as **NCS**). The SDK does not feature any tool which can be used for benchmarking Thread and Zigbee. However, such an application existed before- in the previous Nordic Semiconductor SDK (nRF5 SDK for Thread and Zigbee) and was called **benchmark**. When implementing tests, the tool based on it was implemented and used to perform tests of a Zigbee network.

Benchmark implementation details

3.2.3 Zperf application

Since Thread protocol is an Internet oriented protocol, an existing methodologies and tools could have possibly been used to measure network performance. **Iperf** is one of the tools created for evaluating performance metrics. A sample application implementing Iperf's API is available in Zephyr RTOS and is called **Zperf**. For this project, the RTT measurement for UDP based connections and support for Thread was added to the **Zperf** application.

Zperf implementation details

3.2.4 Other tools used for diagnostics

The exploration of computer networks is usually done by eavesdropping the network link between entities. That kind of capturing packets is called **sniffing** and can be done by a special software. In this case, the traffic of Zigbee and Thread networks was sniffed by **Wireshark**.

Any software is not able to operate without a hardware. When IP networks are diagnosed, a network interface card (either wired or wireless) available in any modern computer is utilized. Unfortunately, sniffing IEEE 802.15.4 requires special network interface. In this thesis, nRF52840 Dongle flashed with a dedicated firmware was used.

3.3 Test setup

3.3.1 Network topology

As mentioned in the section 3.1, both of the measures metrics are affected by the network topology. If the trip of the packet is followed, it is not surprising that the time of its delivery between nodes is strictly bonded with the number of hops. The mesh networks routing is designed to deliver packets using the most efficient path. Although the routing algorithm differs between Zigbee and Thread, the general principle is similar. Routers choose the path to relay packets in such a way, the **cost** of the packets delivery is as small as possible [6]. That way of forming a network topology was one of the obstacles encountered during preparation for this project. An explanation and more detailed description can be found in section 3.4.

The measurement of latency and throughput in reference to the number of hops requires the network topology to provide a deterministic number of hops between given nodes. It was decided to test the network performance against 1 to 6 hops. That need could be met only by introducing a network topology similar to **daisy chain** where any node can have only 1 or 2 connections to the other nodes. Figure 3.4 presents the general idea of such a topology.

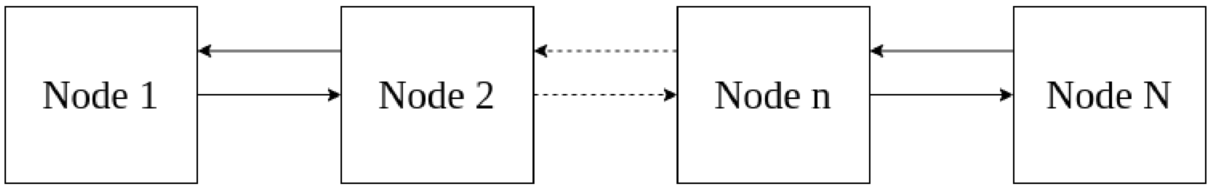


Figure 3.4: The daisy chain topology.

It is worth to mention that introduced topology is neither optimal nor energy efficient for the devices in the network. Although, it’s really hard to establish a mesh network that would allow to measure performance in the relation to the number of hops.

Setting up a mesh network that would use the topology described above was one of the problems encountered during the test preparation and has been broadly described in section 3.4.

3.4 Problems encountered

While developing the test methodology, especially the part consisting of setting up the network (3.3.1), a number of problems arose. A number of ways of solving them had been attempted before a satisfying solution was found.

3.4.1 Enforcing the network topology

The routing in mesh network is based on a cost of given paths, which relates to the quality of wireless link between devices. This gives an opportunity to build the network topology by tweaking the transmission power of devices. However, it is rarely possible to do so in practice, because 2.4GHz band is broadly used by other electronic goods present around us (for example WiFi access points, Bluetooth headphones etc.). Besides the crowded band, furnishings and other equipment introduce interference compounding the link quality. Taking into account all of these factors, it's very hard to determine the right level of amplification of each device's transceiver suitable to create a given network topology, and especially the daisy chain.

Using different levels of transmission power

Setting certain levels of transmission power was the first attempt to build the network required to conduct the tests. This method seemed to be the most appropriate for the networks which build their topology on the basis of the effectiveness of the link between nodes. At the early stage of experiment, the results were promising. However, as soon as the number of nodes in the network raises, the method stopped working as expected. Changing the transmitting power works for networks built with up to 4 devices. It becomes very hard to manage if the number of nodes exceeds that number. That limitations come from the nature of 2.4 GHz band is sensitive for interference. These are caused by furniture, the location of radio transceivers and even people moving by. It is hard to determine an appropriate power level for such a sensitive band.

Rearranging devices

Once the method of using different power settings of devices failed, the next idea was developed. The issue with the first attempt is that devices located in the same room are able to communicate with each other. Even though devices were located in different areas of the room and their transmission power was decreased, an unwanted direct links between nodes were established.

It was decided to put devices into separate rooms located on different floors to overcome that problem. It is worth to mention that at this stage of project, only Zigbee networks were taken for the tests and a variety of modifications were introduced into the benchmark application (3.2.2). An established network was managed by the Benchmark commands sent to other nodes from the device located near the workstation. The devices were located as shown in the figure 3.5 and a number of tests were conducted using this setup.

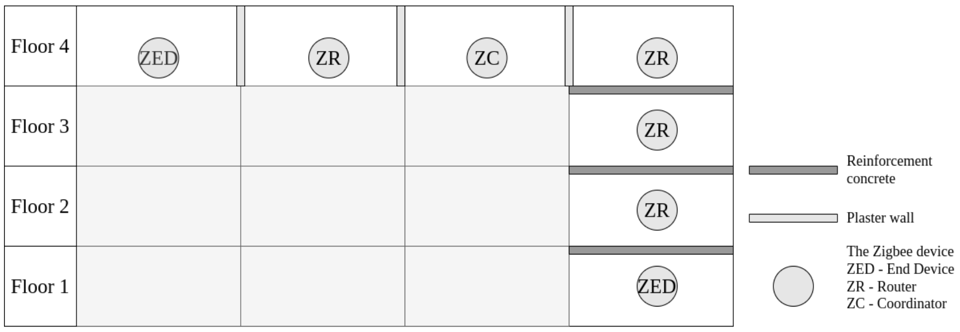


Figure 3.5: The location of devices after rearrangement.

While having that setup powered on and working for a couple of days, a new problem appeared when one node was arbitrarily switched off. A lack of one router in the network broke an established routing. Although the problematic device had been switched on back again it was hard to rebuild the old network topology. This kind of issues enforced a change into the methodology of creating a network topology to more convenient and easier to manage.

Modifying the software to limit visibility of certain devices

The described methods and their issues were finally worked around using dedicated software configuration of each node. A general concept was to limit the visibility of nodes not supposed to exchange data with a chosen device.

This method is based on **filtering** the devices by their **IEEE address**. Zigbee, as well as Thread stack have an API designed especially for this case. Although long address is not used in every transmitted packet, filtering it at the stage of associating the device into the network is sufficient to filter out all unwanted packets.

In case of Zigbee, each device was assigned a long address at a compile time. A figure 3.6 shows the chosen addressing scheme. Listing 3.1 presents a piece of program where API for assigning and filtering out addresses is used. Please note, that code which is not not related to setting and filtering the long addresses was cut off the listing.

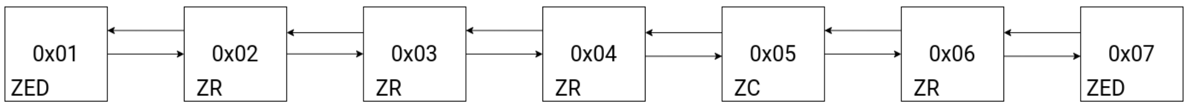


Figure 3.6: The Zigbee addressing scheme altogether with devices roles.

```c
#define DEV_LONG_ADDR                       {0,0,0,0,0,0,0,1}
#define NEIGHBOR1_LONG_ADDR                 {0,0,0,0,0,0,0,2}

zb_ieee_addr_t dev_long_addr = DEV_LONG_ADDR;
zb_ieee_addr_t neighbor_long_addr = NEIGHBOR1_LONG_ADDR;

/*  Macros, global variables and static functions declarations */

void main(void) {
    /* Initialization */

    /* Assigning an IEEE address */
    zb_set_long_address(dev_long_addr);

    /* Filtering out all devices except the one with
     * neighbor_long_addr assigned. */
    mac_add_visible_device(neighbor_long_addr);

    /* Enabling the application */
}
```

Listing 3.1: An usage of the ZBOSS API for assigning and filtering long addresses

Assigning and filtering out long addresses at a compile time requires to know them before the firmware is built and flashed onto boards. This was not problematic in this case, but it is worth to consider to extend the *shell* subsystem with commands for performing these operations at run-time.

In case of OpenThread stack, options needed for long addresses management are built into its CLI subsystem. Addresses assignment as well as filtering was done during device's setup by issuing certain commands. These are presented in the listing 3.2 and the full process of preparing the network is covered by the section 3.5.

```
uart$: ot extaddr 0000000000000001
uart$: ot macfilter addr allowlist
uart$: ot macfilter addr add 0000000000000002
```

Listing 3.2: Shell OpenThread commands for assigning and filtering IEEE addresses

Commands issued in the OpenThread shell cause a similar effect to a direct usage of an API in case of Zigbee.

3.5 Thread network setup automation

As a matter of Thread network being based on the CLI interface it was relatively easy to incorporate an automation in this field. Every step of the network setup is covered by a single command issued to the device by the Python program. Devices are flashed with the same firmware and connected to the USB hub before the script is run. When the network is up and running, the nodes are ready to run performance tests, which is done manually.

The Python script was created in relatively generic way enabling to work with any number of devices. It is based on loading the configuration files given by a command line parameters. These are then parsed accordingly to a fixed json schema (listing 3.4). It is required to bond a configuration file with the board id, which is used by the script to find an appropriate serial port to establish a connection with.

```
def main():
    logging.basicConfig(format="%(levelname)s: %(message)s",
       level=logging.DEBUG)

    configs = sys.argv[1:]
    thread_devices = []

    for config in configs:
        if ".json" not in config:
            logging.error(f"{config} is not a .json file")
            exit(-1)

    for config in configs:
        thread_devices.append(ThreadDevice(config))

    logging.info(f"Found {len(thread_devices)} Thread device(s)")
    for dev in thread_devices:
        logging.info(f"Setting up {dev}")
        dev.device_setup()

    for dev in thread_devices:
        _ = input(f"Press any key to start {dev.config.board_id} on
           {dev._find_tty()}")
        dev.network_up()
```

Listing 3.3: The main function of a script for Thread network setup

```
ot_config_schema = {
    "type" : "object",
    "properties" : {
        "board_id" : {"type" : "number"},
        "baudrate" : {"type" : "number"},
        "panid" : {"type" : "string"},
        "factoryreset" : {"type" : "boolean"},
        "channel" :  {"type" : "number"},
        "networkkey" : {"type" : "string"},
        "extaddr" : {"type" : "string"},
        "allowlist" : {
            "type" : "array",
            "items" : {"type": "string"}
        },
        "zperf_role" : {"enum" : ["client", "server"] }
    }
}
```

Listing 3.4: The json schema used to parse boards configuration files.

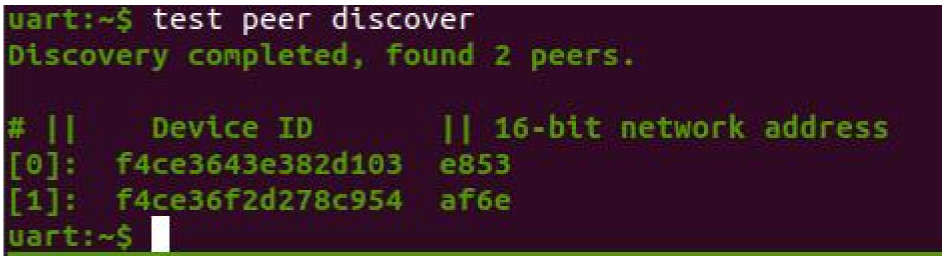
3.6 Performance evaluation procedure

3.6.1 Zigbee evaluation

The evaluation process begins with setting up the Zigbee network. Devices used to test are flashed with the firmware modified as described in 3.4.1 to establish a desired topology. Once the network is ready, nodes eligible to take part in the test are discovered by issuing a benchmark command on one of the border devices:

```
uart$: test peer discover
```

As a result of this command, a list of devices shows up (figure 3.7).



```
uart:~$ test peer discover
Discovery completed, found 2 peers.

# ||    Device ID       || 16-bit network address
[0]:  f4ce3643e382d103  e853
[1]:  f4ce36f2d278c954  af6e
uart:~$
```

Figure 3.7: A result of peer discovery command.

In the next step, a node from the peers listed in 3.7 is chosen:

```
uart$: test peer select 1
```

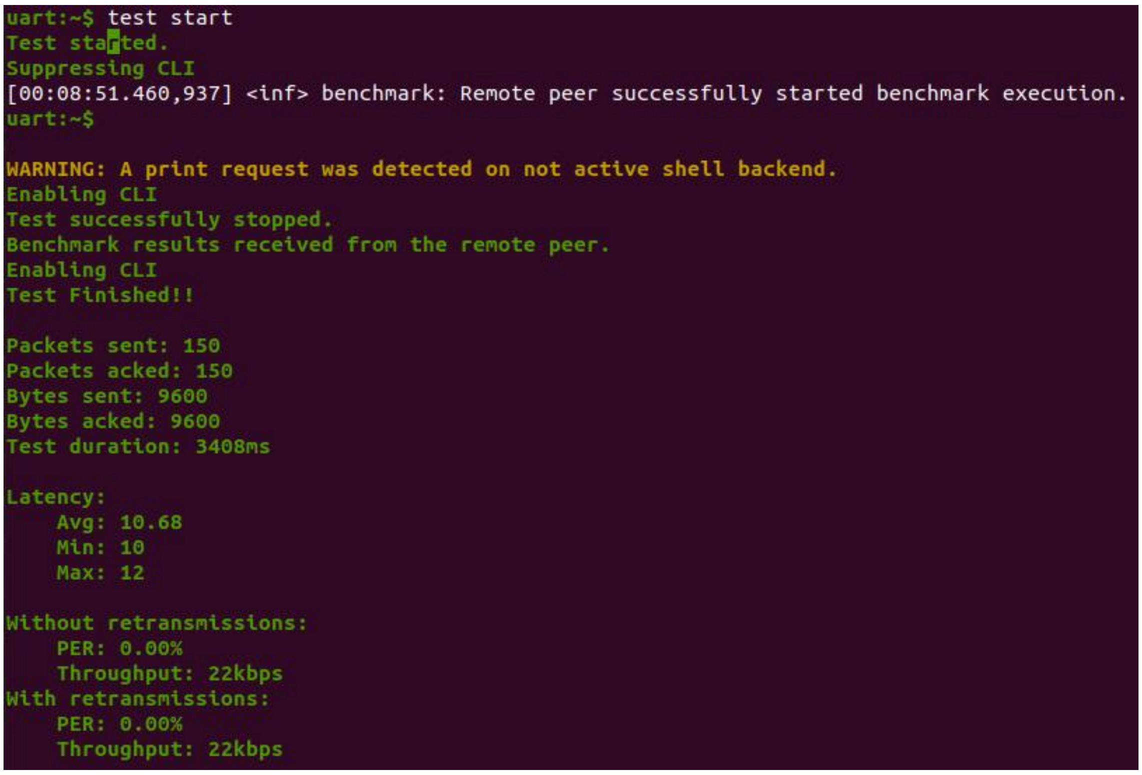
Then the test configuration is chosen. In the majority of tests executed for the project, tests were configured to exchange 150 packets in the mode echo. The length of the data was changed in the consecutive tests to cover different cases.

```
uart$: test configure mode echo
uart$: test configure length 64
uart$: test configure count 150
```

The final step is to run the test and wait until it finishes.

```
uart$: test start
```

The command line remains frozen for the time of test execution. As soon as the test ends, the results are given and the command line is unblocked. An exemplary test results are presented in fig. 3.8.



```
uart:~$ test start
Test started.
Suppressing CLI
[00:08:51.460,937] <inf> benchmark: Remote peer successfully started benchmark execution.
uart:~$

WARNING: A print request was detected on not active shell backend.
Enabling CLI
Test successfully stopped.
Benchmark results received from the remote peer.
Enabling CLI
Test Finished!!

Packets sent: 150
Packets acked: 150
Bytes sent: 9600
Bytes acked: 9600
Test duration: 3408ms

Latency:
    Avg: 10.68
    Min: 10
    Max: 12

Without retransmissions:
    PER: 0.00%
    Throughput: 22kbps
With retransmissions:
    PER: 0.00%
    Throughput: 22kbps
```

Figure 3.8: Zigbee benchmark test results.

3.6.2 Thread evaluation

Thread evaluation process is started with setting up the Thread network as section 3.5 describes. As a next step, the node chosen to perform tests with is set up as a zperf server. Zperf needs to be configured with an ip address of a node:

```
uart$: ot ipaddr
fdde:ad00:beef:0:0:ff:fe00:0
```

```
fdde:ad00:beef:0:558:f56b:d688:799
fe80:0:0:0:f3d9:2a82:c8d8:fe43
Done
uart$: zperf setip fdde:ad00:beef:0:558:f56b:d688:799 64
Done
```

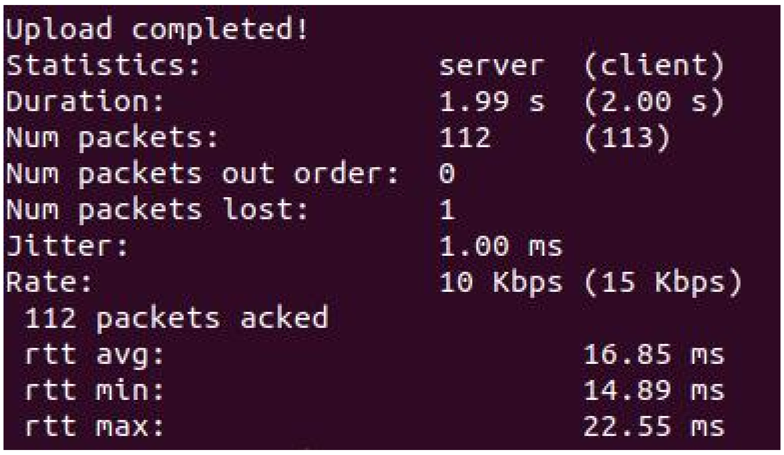
Then, udp server is started:

```
uart$: zperf udp download 5001
```

Finally, another node is configured with ip address as well and starts uploading the data onto the previously configured server node:

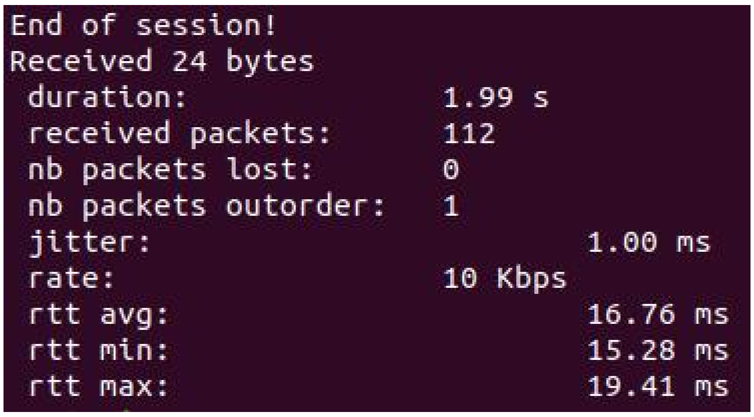
```
uart$: ot ipaddr
fdde:ad00:beef:0:0:ff:fe00:0
fdde:ad00:beef:0:558:f56b:d682:795
fe80:0:0:0:f3d9:2a82:c8d8:fe43
Done
uart$: zperf setip fdde:ad00:beef:0:558:f56b:d682:795 64
Done
uart$: zperf udp upload fdde:ad00:beef:0:558:f56b:d688:799 5001 10 50
    1M
```

The command line is frozen for the time of test execution. At the end of run, results show up in the output of both devices.



```
Upload completed!
Statistics:             server  (client)
Duration:               1.99 s  (2.00 s)
Num packets:            112     (113)
Num packets out order:  0
Num packets lost:       1
Jitter:                 1.00 ms
Rate:                   10 Kbps (15 Kbps)
 112 packets acked
 rtt avg:                       16.85 ms
 rtt min:                       14.89 ms
 rtt max:                       22.55 ms
```

Figure 3.9: Results on the client side.



```
End of session!
Received 24 bytes
 duration:               1.99 s
 received packets:       112
 nb packets lost:        0
 nb packets outorder:    1
 jitter:                         1.00 ms
 rate:                   10 Kbps
 rtt avg:                        16.76 ms
 rtt min:                        15.28 ms
 rtt max:                        19.41 ms
```

Figure 3.10: Results on the server side

3.7 Test results verification

The performance metrics are computed and obtained from the nodes residing inside the tested networks. It should not be assumed that the results are legitimate without analyzing them. The Wireshark program altogether with nRF 802.15.4 Sniffer were used for verification of the gathered results correctness. This method was used mainly during development of testing software to make sure the algorithm used for calculating the metrics was correct.

Please note that despite benchmark verification is described in this section, zperf was tested using similar scenario. The verification test run was executed in the following stages:

1. Flash two nRF52840 DKs with benchmark software.
2. Start recording the network activity with Wireshark and nRF Sniffer.
3. Establish a Zigbee network consisting of newly flashed devices.
4. Configure the test run with the following parameters:
   - Test mode: Echo
   - Packets count: 150
   - Packet length: 64 bytes
5. Execute the test.
6. Compare the outcome calculated by benchmark with the results from Wireshark.

The 3.11 figure presents the setup used for the verification altogether with annotated network addresses of devices in the Zigbee network.

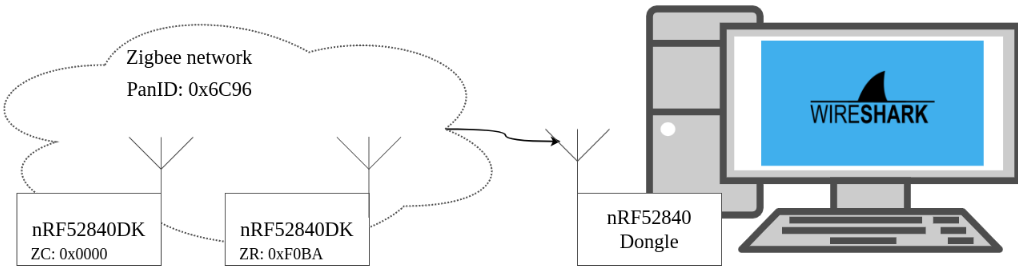
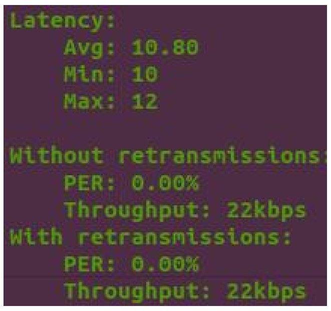


Figure 3.11: The network and workstation setup for test results verification.

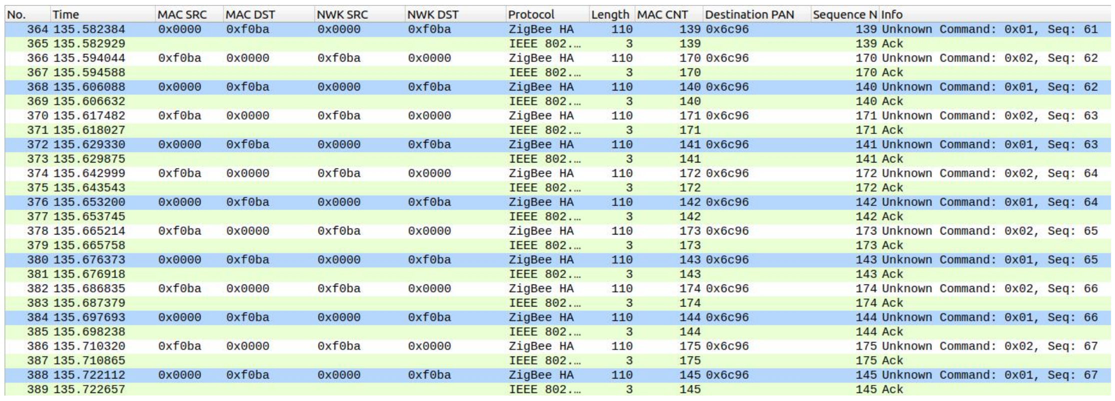
When the test finished, benchmark gave results shown in the figure 3.12 and the packet stream (shown in the fig. 3.13) was collected.



```
Latency:
    Avg: 10.80
    Min: 10
    Max: 12

Without retransmissions:
    PER: 0.00%
    Throughput: 22kbps
With retransmissions:
    PER: 0.00%
    Throughput: 22kbps
```

Figure 3.12: The results gathered computed by the benchmark.



No.	Time	MAC SRC	MAC DST	NWK SRC	NWK DST	Protocol	Length	MAC CNT	Destination PAN	Sequence N	Info
364	135.582384	0x0000	0xf0ba	0x0000	0xf0ba	ZigBee HA	110	139	0x6c96	139	Unknown Command: 0x01, Seq: 61
365	135.582929					IEEE 802.…	3	139		139	Ack
366	135.594044	0xf0ba	0x0000	0xf0ba	0x0000	ZigBee HA	110	170	0x6c96	170	Unknown Command: 0x02, Seq: 62
367	135.594588					IEEE 802.…	3	170		170	Ack
368	135.606088	0x0000	0xf0ba	0x0000	0xf0ba	ZigBee HA	110	140	0x6c96	140	Unknown Command: 0x01, Seq: 62
369	135.606632					IEEE 802.…	3	140		140	Ack
370	135.617482	0xf0ba	0x0000	0xf0ba	0x0000	ZigBee HA	110	171	0x6c96	171	Unknown Command: 0x02, Seq: 63
371	135.618027					IEEE 802.…	3	171		171	Ack
372	135.629330	0x0000	0xf0ba	0x0000	0xf0ba	ZigBee HA	110	141	0x6c96	141	Unknown Command: 0x01, Seq: 63
373	135.629875					IEEE 802.…	3	141		141	Ack
374	135.642999	0xf0ba	0x0000	0xf0ba	0x0000	ZigBee HA	110	172	0x6c96	172	Unknown Command: 0x02, Seq: 64
375	135.643543					IEEE 802.…	3	172		172	Ack
376	135.653200	0x0000	0xf0ba	0x0000	0xf0ba	ZigBee HA	110	142	0x6c96	142	Unknown Command: 0x01, Seq: 64
377	135.653745					IEEE 802.…	3	142		142	Ack
378	135.665214	0xf0ba	0x0000	0xf0ba	0x0000	ZigBee HA	110	173	0x6c96	173	Unknown Command: 0x02, Seq: 65
379	135.665758					IEEE 802.…	3	173		173	Ack
380	135.676373	0x0000	0xf0ba	0x0000	0xf0ba	ZigBee HA	110	143	0x6c96	143	Unknown Command: 0x01, Seq: 65
381	135.676918					IEEE 802.…	3	143		143	Ack
382	135.686835	0xf0ba	0x0000	0xf0ba	0x0000	ZigBee HA	110	174	0x6c96	174	Unknown Command: 0x02, Seq: 66
383	135.687379					IEEE 802.…	3	174		174	Ack
384	135.697693	0x0000	0xf0ba	0x0000	0xf0ba	ZigBee HA	110	144	0x6c96	144	Unknown Command: 0x01, Seq: 66
385	135.698238					IEEE 802.…	3	144		144	Ack
386	135.710320	0xf0ba	0x0000	0xf0ba	0x0000	ZigBee HA	110	175	0x6c96	175	Unknown Command: 0x02, Seq: 67
387	135.710865					IEEE 802.…	3	175		175	Ack
388	135.722112	0x0000	0xf0ba	0x0000	0xf0ba	ZigBee HA	110	145	0x6c96	145	Unknown Command: 0x01, Seq: 67
389	135.722657					IEEE 802.…	3	145		145	Ack

Figure 3.13: A Wireshark output showing a fragment of the captured packet stream.

A table containing the packets which comes as an output of Wireshark enables to compute a delay between packets. This is done by using timestamp values saved in the "Time" column. The difference between the transmission timestamps of consecutive data packets is interpreted as the latency.

The packet dump was exported into the .csv file and loaded into a spreadsheet for further analysis. Calculated latency measures for this run, alongside with benchmark results are shown in the table 3.1.

Testing tool	Latency			Throughput [kbps]
	Average [ms]	Minimum [ms]	Maximum [ms]	
Benchmark	10.80	10.00	12.00	22.00
Wireshark	11.01	10.31	12.76	22.70

Table 3.1: A comparison of latency and throughput measurements.

The results shown differ for all metrics between the benchmark outcome and the Wireshark output based calculations. These margins are caused mainly by following factors:

- The sniffer does not run the network stack which processes the packets before passing them to the application layer. Packets are delivered and marked with a timestamp as soon as they are received.
- The sniffer is not located in the exact place as either of tested boards.
- The latency in benchmark application is computed as a difference between the time when packet is scheduled to send by the stack and the time of response reception. In the method utilizing packets sniffing, the latency is expressed as a delay between consecutive packets with data. The difference can come from a time needed by the stack to process the scheduled packet.

After summing up above factors, the results yield by the benchmark application were considered valid and the software was allowed for future use for Zigbee performance evaluation. A similar analysis had been conducted for zperf application and similar results were obtained.

Chapter 4

Results analysis

The results of performance metrics measurements (3.1) of Zigbee and Thread networks are presented and discussed throughout this chapter. Tests were executed on networks consisting of 7 devices and followed the procedures described in chapter 3.

4.1 Latency

4.1.1 Latency in Thread networks

All Thread latency measurement results are gathered in the table A.1.

Relation between latency and the number of hops

Let's first have a look at the plots of latency in Thread network (Fig. 4.1).

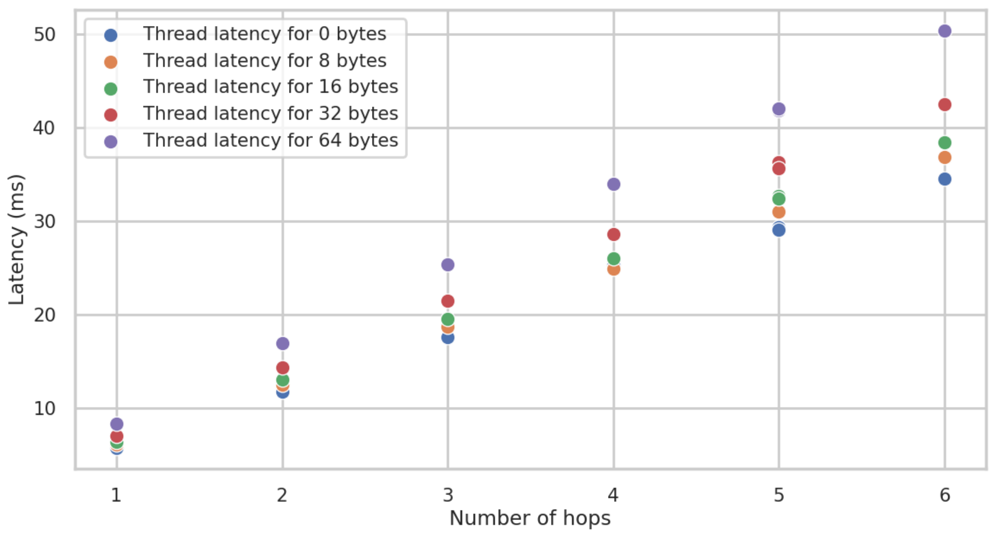


Figure 4.1: Latency in Thread network dependent on the number of hops.

The number of hops, a packet must be passed through is the most significant factor affecting the latency. It is clearly noticeable, that additional hop adds roughly the same amount of time to the overall packet delivery delay. It means that the latency is affected the most by the routing inside the network. To sum up, **the relation between latency and the number of hops can be expressed as a product of single hop delay and the number of hops.**

Relation between latency and payload length

The length of a packet contributes to the packet delivery delay as well as the number of hops. The former relation is shown in 4.2.

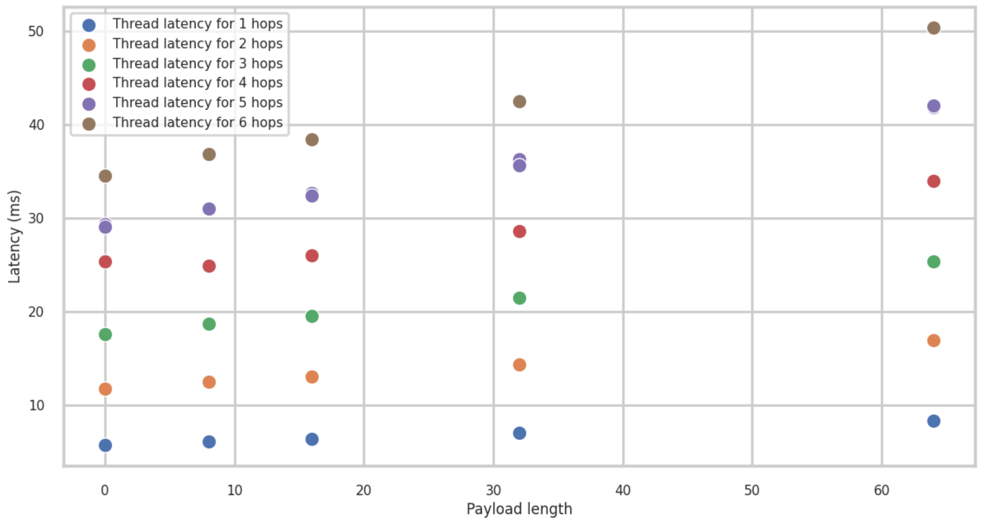


Figure 4.2: Latency in Thread network dependent on the payload length.

In this case, increasing the payload length does not affect the latency as much as routing. However, the delay of 64 bytes long payload is a latency of 0 bytes long payload doubled. Looking at the plot and the table of results (A.1), **the single hop latency can be estimated between 5 and 9 milliseconds**.

Fragmentation

Although not covered by this experiment, an important thing to mention is the packet fragmentation. When the length of a payload carried by the application layer exceeds 79 bytes (Zigbee) or 80 bytes (Thread, using UDP as transport layer protocol), the application payload is divided into multiple packets. In this case the time of packet delivery seen by the application layer is increased proportionally to the number of packets needed to carry the split payload.

4.1.2 Latency in Zigbee networks

The relation of latency and the number of hops in the Zigbee network is shown in Fig. 4.3. It is similar to the one determined for Thread network. The latency is equals the product of number of hops and single hop delay. The principle applied for Thread, is relevant for Zigbee as well.

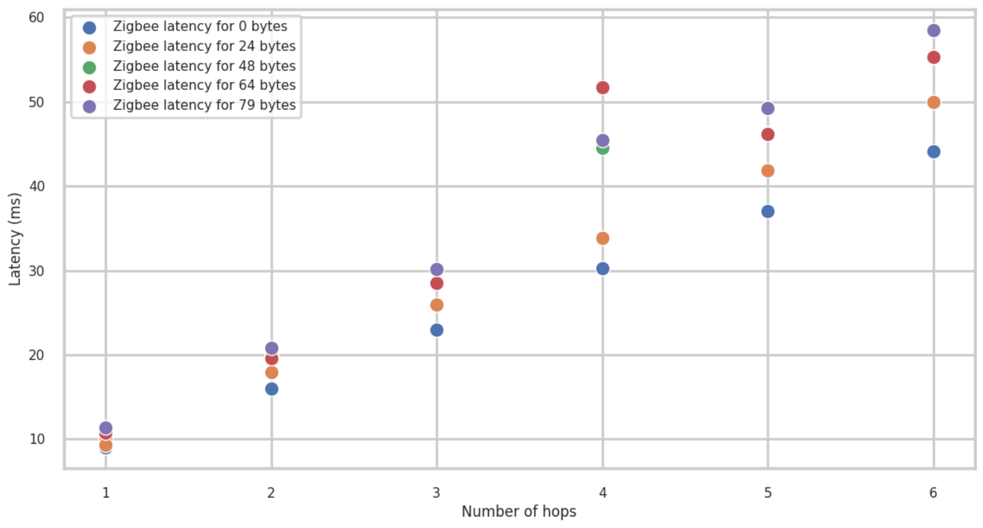


Figure 4.3: Latency in Zigbee network dependent on the number of hops.

The single hop delay in Zigbee differs from the one estimated for Thread. In this case the latency varies between **9 and 11.5 milliseconds**, which is higher than for Thread. The relation between latency and the payload length is shown in Figure 4.4.

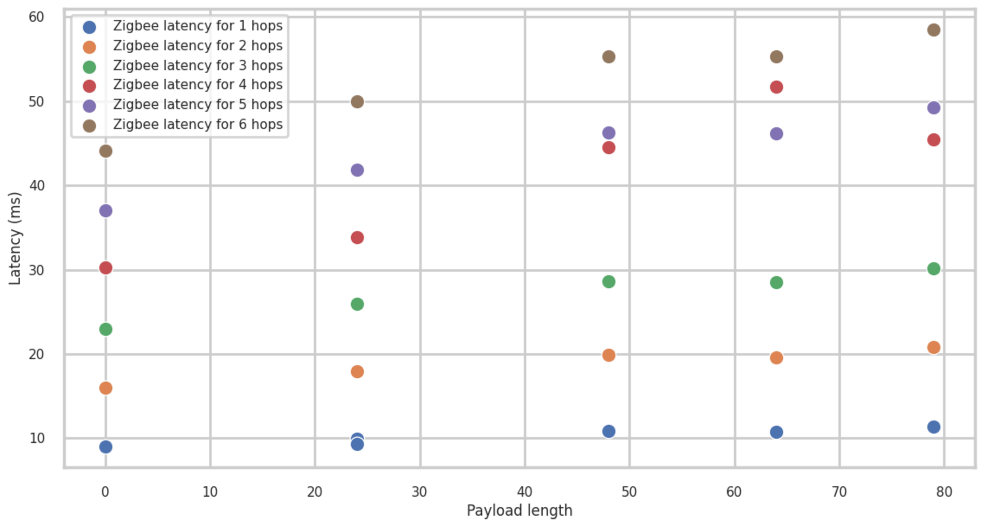


Figure 4.4: Latency in Zigbee network dependent on the payload length.

4.2 Throughput

Throughout the experiments, throughput was computed as a ratio of sent bites and the full time of test execution. This is expressed as

$$Throughput = \frac{payload\_length \cdot 8}{execution\_time}[kbps]. \tag{4.1}$$

Before diving into the results, some conclusions can be noticed by looking at the equation 4.1. Throughput depends on two factors: length of the payload and time of test execution. When payload length increases, throughput increases as well. If execution time is higher, throughput decreases.

The execution time of test consists of the latency multiplied by a number of sent packets and time spent on data processing. The latter includes delays introduced by: buffers copying, packet scheduling and the overall network stack processing.

4.2.1 Throughput in Thread networks

Following plots (4.5, 4.6) show the throughput in Thread network relatively to the number of hops or payload length. As determined at the beginning of this section, throughput is directly proportional to the payload length and inversely proportional to the latency (which depends directly on the number of hops).

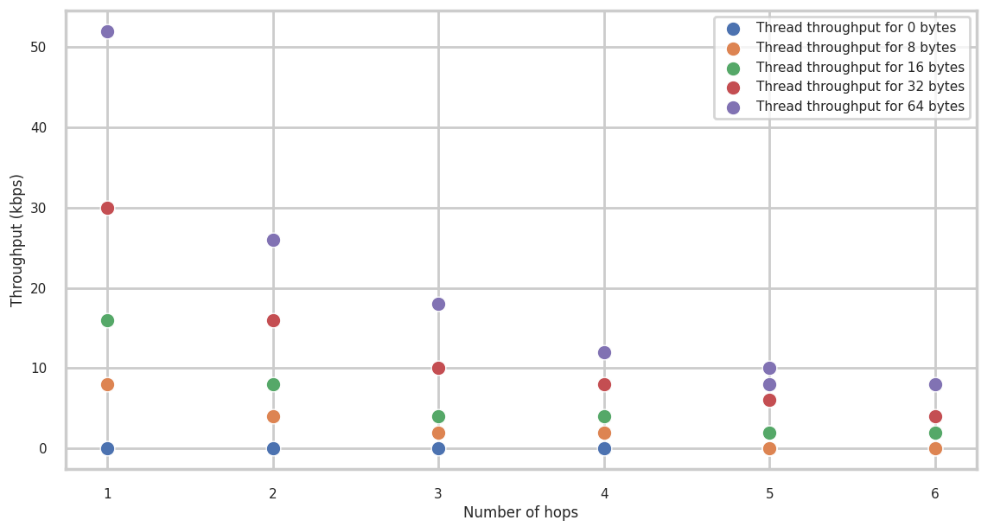


Figure 4.5: Throughput in Thread network dependent on the number of hops.

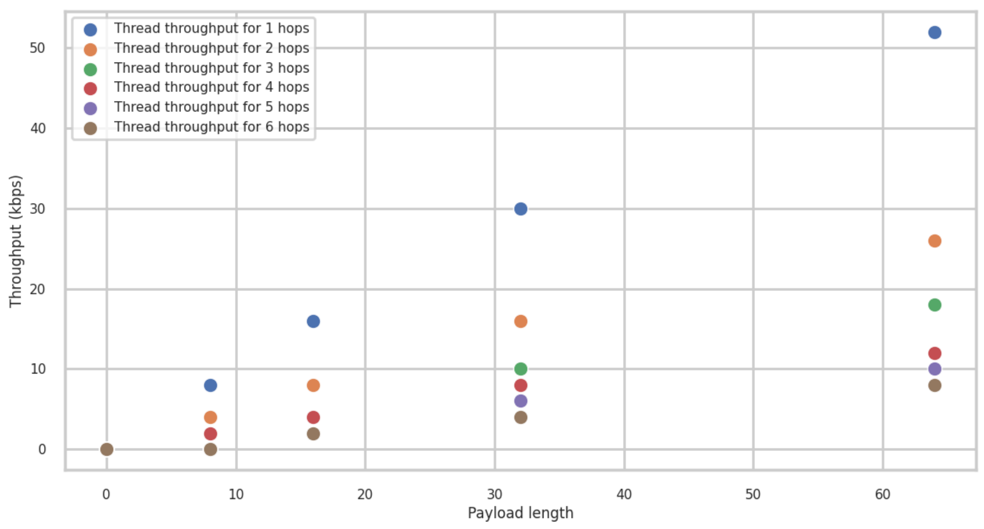


Figure 4.6: Throughput in Thread network dependent on the payload length.

The minimum throughput in Thread network for non-zero bytes long payload and 1 hop is **4**kbps. The maximum of **52kbps** was obtained for 1 hop and 64 bytes long payload. It is worth to mention that every case which uses 0 bytes long payload yields 0 kbps payload.

4.2.2 Throughput in Zigbee networks

The results of throughput measurements of Zigbee network are depicted in figures 4.7 and 4.8. The dependencies of throughput and packet length or execution time are the same as for Thread. The minimum value of throughput (0kbps) as well as for Thread is received when 0 bytes long payload is sent. The minimum value for non-zero bytes long payload (received for 8 bytes and 1 hop) is **3** kbps. The maximum throughput in Zigbee network, received for 1 hop and 64 bytes long payload equals **44kbps**.

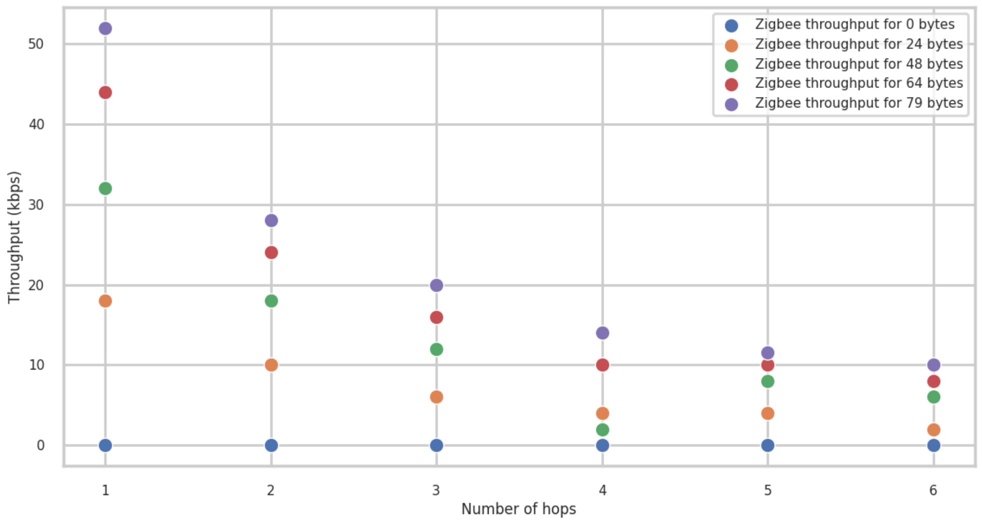


Figure 4.7: Throughput in Zigbee network dependent on the number of hops.

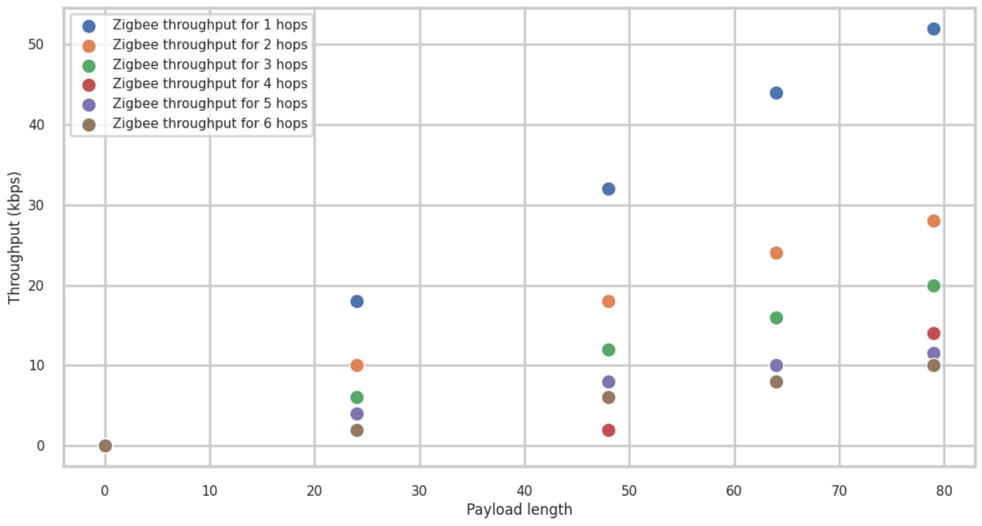


Figure 4.8: Throughput in Zigbee network dependent on the payload length.

4.3 Performance comparison

As shown in sections 4.1 and 4.2, the two fundamental metrics of network protocol performance obey the similar principles. It is not surprising, because both tested technologies are built upon the same MAC layer. This implies that none of them cannot go beyond IEEE 802.15.4 constraints.

For latency, as well as for throughput measurements, Thread network performs better. Figure 4.9 presents latency of 24 bytes long payload for these two protocols and the throughput is shown in 4.10. The tests showed that Thread is significantly faster than Zigbee.

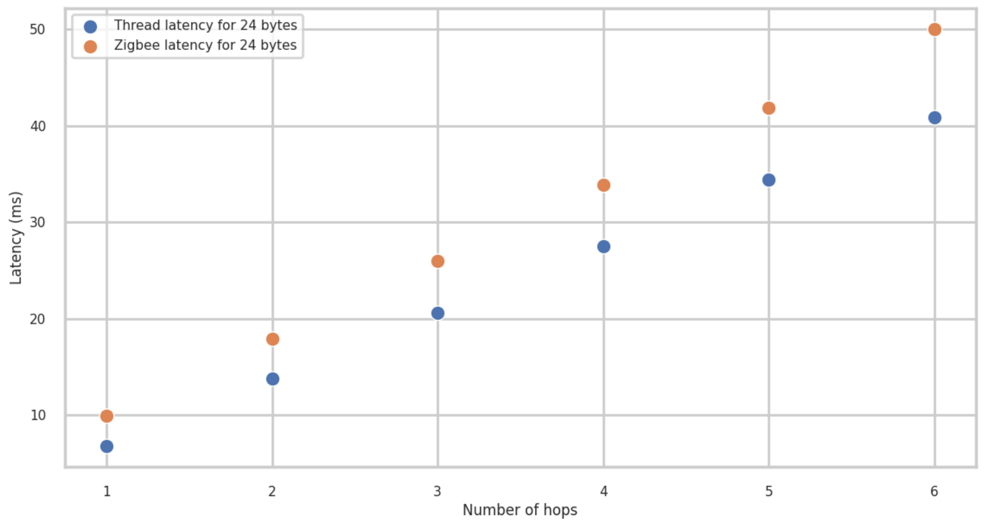


Figure 4.9: Latency comparison betwen Thread and Zigbee.

The comparison between most representative results alongside with calculated differences are shown in the table 4.1.

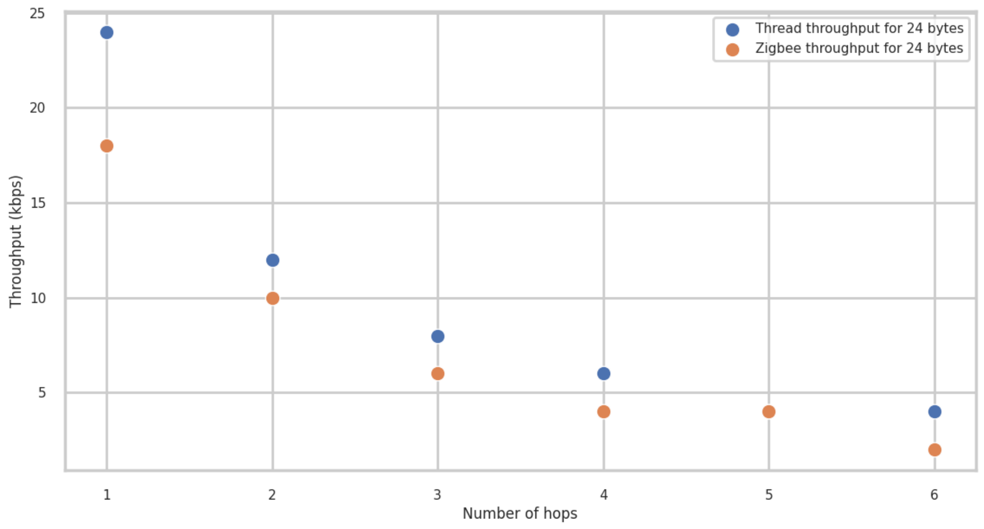


Figure 4.10: Throughput comparison between Thread and Zigbee.

Metric	Thread	Zigbee	Difference
0 bytes, 1 hop			
Latency (ms)	5.74	9.00	3.26 (56.8%)
Throughput (kbps)	0	0	(0%)
24 bytes, 1 hop			
Latency (ms)	6.76	9.95	3.19 (47.2%)
Throughput (kbps)	24	18	6 (33.3%)
64 bytes, 1 hop			
Latency (ms)	8.33	10.76	2.43 (29.2%)
Throughput (kbps)	52	44	8 (18.2%)

Table 4.1: Comparison of latency and throughput for chosen number of hops and payload length between Thread and Zigbee.

The performance differences between Zigbee and Thread are significantly higher for shorter payload lengths. Taking into account that tests were executed on the same hardware and both protocols use IEEE 802.15.4 MAC layer, it can be assumed that for a packet of given length, the time in the air is the same. That said, the factor affecting the packets delay in Zigbee network is probably associated with data processing. Working on a packet in Zigbee takes longer than in Thread.

Chapter 5

Summary and conclusion

5.1 Conclusion of performance comparison between Thread and Zigbee

Both Zigbee and Thread are popular standards for various applications. Although performance comparison showed that Thread has lower latency and higher throughput when compared to Zigbee, there are other important factors that need to be taken into consideration while choosing a technology for the product. Zigbee has been in the market for the longer period than Thread and has been adopted by many vendors. On the other hand, Thread enables for easier integration with Internet, which can be an asset for some applications.

Both protocols require little computing power which is good in terms of energy consumption. However, Zigbee required less RAM than Thread.

To sum up, a technology choice should not be taken based just on the performance. The purposes of the two compared protocols are slightly different. When designing a product, one must choose the protocol that matches the product specification and enables its further development and certification.

5.2 Conclusion of research methodology

The proposed methodology relies on testing applications which exchange a packets stream between chosen nodes in the network. Both programs compute the packets delay, throughput and some other statistics which were not used in the experiment.

Among other objectives, this thesis aimed to develop such a methodology which could be used in the future for testing new versions of protocols. Moreover, the NCS framework hasn't included such a programs before and it can be a good opportunity for Nordic Semiconductor's customers to test the provided technology on their own.

5.3 Future work

Keeping the software up to date with constantly changing libraries is challenging and that is one of the goals for the future. Moreover, the developed methodology should be shared to broader community and tested against setup in the environment, where mesh networks are usually located.

Appendix A

Full latency measurements results

Figure A.1: Thread latency measurements results.

Number of hops	Packet length	Min	Max	Average
1	0	4.5	65.2	5.74
1	8	4.85	61.72	6.13
1	16	5.26	61.065	6.38
1	24	5.49	114.04	6.76
1	32	5.905	61.52	7.05
1	40	6.13	67.84	7.40
1	48	6.405	68.355	7.64
1	56	6.91	68.54	8.05
1	64	7.05	64.665	8.33
1	72	7.55	69.38	8.66
1	74	7.475	66.465	8.77
2	0	9.975	63.735	11.75
2	8	10.985	66.025	12.55
2	16	11.395	66.54	13.08
2	24	12.02	67.425	13.78
2	32	12.495	69.395	14.38
2	40	13.275	70.175	15.12
2	48	13.655	70.54	15.69
2	56	14.615	73.56	16.46
2	64	14.965	73.835	16.98
2	72	16.05	72.525	17.74
3	0	15.165	70.02	17.62
3	8	16.785	69.76	18.67
3	16	17.285	109.86	19.58
3	24	18.735	69.7	20.62
3	32	19.085	72.815	21.45
3	40	20.26	72.905	22.70
3	48	21.07	77.435	23.51
3	56	22.505	72.43	24.53
3	64	23.48	74.28	25.43
3	72	24.185	77.42	26.58
3	74	24.7	77.39	26.76

Figure A.2: (cont.) Thread latency measurements results.

Number of hops	Packet length	Min	Max	Average
4	0	20.46	72.6	25.34
4	8	22.32	76.015	24.95
4	16	23.085	74.735	26.05
4	24	24.565	78.17	27.48
4	32	25.985	77.925	28.62
4	40	27.695	80.15	30.18
4	48	28.44	81.615	31.34
4	56	30.315	82.7	32.64
4	64	31.095	87.72	34.00
4	72	33.065	82.32	35.17
4	74	33.705	123.61	35.70
5	0	26.76	76.92	29.34
5	0	25.875	115.185	29.12
5	8	28.425	118.175	31.00
5	8	28.075	79.055	31.00
5	16	29.11	86.33	32.67
5	16	29.11	122.45	32.45
5	24	31.34	123.855	34.43
5	32	32.82	85.645	36.31
5	32	32.3	123.47	35.70
5	40	35.43	40.51	37.27
5	40	35.125	124.295	37.71
5	48	36.755	89.095	39.07
5	48	36.42	126.495	39.37
5	56	37.945	92.36	42.34
5	56	38.25	96.34	40.80
5	64	39.425	46.81	41.90
5	64	39.52	100.2	42.03
5	72	41.53	96.6	44.34
5	72	41.18	100.295	44.15
5	72	41.165	95.15	44.83
5	74	42.11	114.3	44.57

Figure A.3: (cont.) Thread latency measurements results.

Number of hops	Packet length	Min	Max	Average
6	0	32.53	39.015	34.59
6	8	34.665	42.555	36.90
6	16	37.61	39.305	38.47
6	24	38.205	46.14	40.89
6	32	40.02	45.685	42.48
6	40	42.48	50.655	44.83
6	48	44.08	48.49	46.43
6	56	46.14	51.91	48.67
6	64	47.955	54.58	50.41
6	72	50.11	106.14	53.45
6	74	50.565	103.56	53.74

Figure A.4: Zigbee latency measurements results.

Number of hops	Packet length	Min	Max	Average
1	0	8.0	22.0	9.00
1	8	8.0	23.0	9.36
1	16	9.0	13.0	9.53
1	24	9.0	23.0	9.95
1	24	8.0	21.0	9.36
1	32	9.0	26.0	10.22
1	40	10.0	25.0	10.55
1	48	10.0	25.0	10.84
1	56	10.0	25.0	11.22
1	64	10.0	24.0	10.76
1	74	10.0	24.0	11.14
1	76	10.0	13.0	11.20
1	78	10.0	27.0	11.31
1	79	10.0	25.0	11.36
2	0	14.0	31.0	15.98
2	8	15.0	46.0	16.78
2	16	15.0	32.0	17.23
2	24	16.0	36.0	17.91
2	32	17.0	33.0	18.48
2	40	17.0	35.0	19.23
2	48	18.0	37.0	19.89
2	56	19.0	36.0	20.43
2	64	18.0	36.0	19.61
2	74	19.0	35.0	20.48
2	76	19.0	50.0	20.60
2	78	19.0	37.0	20.76
2	79	19.0	39.0	20.85
3	0	21.0	38.0	23.00
3	8	22.0	52.0	24.19
3	16	23.0	56.0	24.81
6	64	52.0	144.0	55.29
6	79	55.0	108.0	58.53

Figure A.5: (cont.) Zigbee latency measurements results.

Number of hops	Packet length	Min	Max	Average
3	24	24.0	47.0	25.93
3	32	25.0	46.0	26.68
3	40	26.0	43.0	27.84
3	48	27.0	45.0	28.63
3	56	27.0	53.0	29.65
3	64	26.0	61.0	28.50
3	74	28.0	53.0	29.70
3	76	28.0	50.0	29.99
3	78	28.0	43.0	30.11
3	79	28.0	43.0	30.21
4	0	28.0	82.0	30.27
4	12	28.0	123.0	35.11
4	24	31.0	73.0	33.86
4	36	33.0	79.0	35.56
4	48	34.0	290.0	44.51
4	64	34.0	409.0	51.75
4	79	37.0	350.0	45.51
5	0	34.0	114.0	37.05
5	12	37.0	108.0	39.95
5	24	39.0	80.0	41.86
5	36	41.0	114.0	44.11
5	48	43.0	79.0	46.33
5	64	43.0	75.0	46.16
5	79	46.0	151.0	49.30
6	0	41.0	133.0	44.17
6	12	44.0	155.0	47.30
6	24	47.0	139.0	50.03
6	36	48.0	145.0	52.17
6	48	52.0	128.0	55.28

Bibliography

[1] M. Christiano. The new wireless thread network protocol. Sept. 2015.

[2] IEEE Computer Society. Part 15.4: Wireless medium access control (mac) and physical layer (phy) specifications for low-rate wireless personal area networks (lr-wpans). Specification, Zigbee Alliance inc., Oct. 2003. https://people.iith.ac.in/tbr/teaching/docs/802.15.4-2003.pdf.

[3] Nordic Semiconductor. Adding ZCL Clusters to application.

[4] V. Rudresh. Zigbee security: Basics (part 2). November 2017.

[5] Silicon Labs. Zigbee sleepy end device. Technical report, Silicon Labs, 2019. https://raw.githubusercontent.com/wiki/MarkDing/IoT-Developer-Boot-Camp/files/ZB-2019Q4-ZMGC-Training/Zigbee-Sleepy-End-device.pdf.

[6] Texas Instruments. Exploring Thread and Zigbee for home and building automation.

[7] Thread Group. Thread stack fundamentals. Technical report, Thread Group, July 2015. https://www.silabs.com/documents/public/white-papers/Thread-Stack-Fundamentals.pdf.

[8] Thread Group. Thread usage of 6lowpan. Technical report, Thread Group, July 2015. https://www.silabs.com/documents/public/white-papers/Thread-Usage-of-6LoWPAN.pdf.

[9] Thread Group. Thread network fundamentals. White Paper, Thread Group, May 2020.

[10] P. Wegner. Global IoT market size.

[11] Wikipedia. IEEE 802.15.4.

[12] Wikipedia. Zigbee.

[13] Zigbee Alliance. Zigbee. Specification, Zigbee Alliance inc., Aug. 2015. https://zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf.