Internet of Things mesh network

Using the Thread networking protocol

Anton Alm
Abstract

This thesis summarizes my project in setting up a Thread network. The idea of this project was presented by the company ÅF in Karlstad, Sweden. ÅF wishes to upgrade their current demonstrator for IoT. The current demonstrator includes Azure Cloud component, Raspberry Pi, Bluetooth and Arduino components. The upgrade includes implementing Thread technology together with Thread verified hardware from Nordic semiconductor [1] and the Raspberry Pi Foundation [2].

Thread is an IoT mesh networking protocol that was released year 2014. Compared to Bluetooth it offers IP communication (including IPv6) combined with higher reliability, performance and security. The process of installing, compiling and configuring the Thread network is explained.

The result is an operational thread network that has sensor devices sending data to an HTTP web server, where the data is stored and monitored. Though, there are many improvements and functions that can be implemented to make this demonstrator more appealing.
Contents

1 Introduction .................................................................................................................. 1
  1.1 Applications ............................................................................................................. 2
  1.2 IoT Protocols .......................................................................................................... 3
  1.3 Thread ...................................................................................................................... 4
  1.4 Purpose ................................................................................................................... 5
  1.5 Goals ....................................................................................................................... 6
  1.6 Structure ............................................................................................................... 7

2 Background ................................................................................................................... 8
  2.1 IoT market ............................................................................................................... 8
  2.2 Thread .................................................................................................................... 9
    2.2.1 Network Architecture ....................................................................................... 10
    2.2.2 Protocol stack .................................................................................................. 12
    2.2.3 IP Addressing ................................................................................................. 14
  2.3 Constrained Application Protocol .......................................................................... 15
  2.4 Hardware ............................................................................................................... 16
    2.4.1 Hardware specification ..................................................................................... 17
    2.4.2 Hardware Usage .............................................................................................. 18

3 Design .......................................................................................................................... 19
  3.1 Network layout ...................................................................................................... 19
  3.2 Installation ............................................................................................................. 20
    3.2.1 Software development kit ................................................................................. 20
    3.2.2 Border router .................................................................................................. 20
    3.2.3 Network Co Processor ...................................................................................... 20
    3.2.4 Sensor devices ................................................................................................. 21
    3.2.5 Web Server ...................................................................................................... 21
  3.3 Network configuration ............................................................................................. 22
    3.3.1 Border router .................................................................................................. 22
    3.3.2 Network co-processor ...................................................................................... 22

4 Implementation ............................................................................................................ 23
  4.1 Network settings and commissioning ...................................................................... 23
    4.1.1 Network co-processor ...................................................................................... 24
    4.1.2 Sensor device .................................................................................................... 24
4.2 Sensor devices .................................................................................................................26
  4.2.1 Main function and global variables ............................................................................26
  4.2.2 Timer, buttons and CoAP messages .........................................................................29

4.3 Web server ..............................................................................................................34
  4.3.1 Port forwarding ......................................................................................................34
  4.3.2 Crosscoap ...........................................................................................................34
  4.3.3 Database .............................................................................................................35
  4.3.4 PHP & HTML ....................................................................................................36

5 Results .........................................................................................................................40
  5.1 Packet trace analysis ..............................................................................................41
  5.2 Database visualization .............................................................................................44

6 Evaluation ......................................................................................................................45
  6.1 Goals .........................................................................................................................45
  6.2 Future work ...............................................................................................................46
  6.3 Problems ..................................................................................................................47

7 Summary and Conclusions ..........................................................................................48

References .........................................................................................................................50
List of figures

Figure 1: ÅF's IoT demonstrator ................................................................. 5
Figure 2: An example of a Thread network architecture............................. 10
Figure 3: HTTP and CoAP protocol stacks ............................................. 15
Figure 4: The Nordic Semiconductor nRF52840 Development Kit ............. 16
Figure 5: The Raspberry Pi Model 3B ....................................................... 16
Figure 6: The programmable buttons on the nRF52840 DK ..................... 17
Figure 7: The Raspberry Pi and development kit forming the Border router. .. 18
Figure 8: An overview of the network layout ........................................... 19
Figure 9: DK's connected to Embedded Studio and J-Link on the laptop .......... 21
Figure 10: Network co-processor status .................................................. 22
Figure 11: Network co-processor status .................................................. 24
Figure 12: Joining the network as a sensor device ................................... 24
Figure 13: Configuration settings and testing the network for sensor device ..... 25
Figure 14: Main function ......................................................................... 26
Figure 15: Global variables ..................................................................... 27
Figure 16: Function thread_instance_init ................................................ 27
Figure 17: Function thread_coap_init ...................................................... 28
Figure 18: Function thread_bsp_init ........................................................ 28
Figure 19: Function thread_coap_utils_temp_timer_init ......................... 29
Figure 20: Function temp_timer_handler ................................................. 29
Figure 21: Function thread_coap_utils_send_message ............................ 31
Figure 22: Start of function bsp_event_handler ....................................... 32
Figure 23: End of function bsp_event_handler ........................................ 33
Figure 24: MYSQL table to store incoming data from sensor devices .......... 35
Figure 25: HTML and PHP to continuously update page and set style .......... 36
Figure 26: Parse incoming data and storing to database ........................... 37
Figure 27: Select button data from database and visualize in table ............. 38
Figure 28: Select temperature data from database and visualize in graph ...... 39
Figure 29: tcpdump with parameters “-nnvXXSs” on border router at port 5683 .... 41
Figure 30: tcpdump with parameters “-nnvXXSs” on web server at port 5683 ... 42
Figure 31: tcpdump with parameters “-nnvXXSs” on web server at port 80. ....... 43
Figure 32: Last 8 messages received on web server from button actions............. 44
Figure 33: Last 20 messages received on web server with temperature values. .. 44
List of tables

Table 1: The OSI-model for the Thread protocol stack..............................................12
Table 2: Hardware specification..............................................................................17
Abbreviations

IoT  Internet of Things
RPi  Raspberry Pi
DK   Development Kit
HW   Hardware
IPv6 Internet Protocol Version 6
PANID Personal Area Network Identification
XPANID Extended Personal Area Network Identification
CSMA Carrier Sense Multiple Access
Kbps  Kilobits per second
KBps  Kilobytes per second
6LowPAN IPv6 Over Low Power Wireless Personal Networks
UDP  User Datagram Protocol
DTLS Datagram Transport Layer Security
CoAP Constrained Application Protocol
SoC   System on Chip
CLI   Command line interface
USB   Universal Serial Bus
IEEE  Institute of Electrical and Electronics Engineers
IETF  Internet Engineering Task Force
RFC  Request for comments
GPIO General-purpose input/output
NAT   Network address translation
REST  Representational State Transfer
LR-WPAN Low-Rate Wireless Personal Area Network
1 Introduction

The rapid advancement of computers has broadened the scope for computer applications. The first computer came out in the 1940’s. It weighed 30 tons and was used as a basic calculator [3]. The computer had 64 KB ROM-memory and an operation speed of 0.043MHz. Today, a computer with a weight of a few grams [4] has 256KB RAM-memory and 1MB ROM-memory with a computation speed of 64MHz for a fraction of the price. Due to these advancements, a computer can theoretically be installed and used with every day applications for a low cost.

The advancement of computer networking has also rapidly increased the scope of computers, allowing for world-wide communication. Year 1965, the first two computers communicated with each other over a wired connection at MIT Lincoln Lab. In 1974 the first Internet Service Provider (ISP) was created, leading the way for global networking, with e-mail released shortly thereafter. Year 1991 the World Wide Web went public, creating a standard for websites. Year 1998 gave birth to Wi-Fi, a technology for radio wireless local area networking. Wireless networking meant computers could communicate all over the world, without the need for physical wire-connections [5].

The combination of computer hardware and networking advancements has given birth to the Internet of Things (IoT), with the purpose of letting everyday objects collect and exchange data over the internet with the use of small sensor devices. The range for IoT applications is large. By installing sensor devices on objects such as lights, fridges, heaters, fans, etc. they can be remotely monitored and controlled. They can also communicate with each other to create dependencies and requirements.
1.1 Applications

The possibility of useful applications for IoT is unlimited. The current limit is technology, but with continued advancement, more and more objects can be reliably connected to the internet with a low cost and low energy consumption. Here are a few examples of IoT applications:

A) Short range applications: Smart home - Installing sensor devices in home appliances.
   1) Creating control between devices within the network. For example, the air condition should be active only when the temperature reaches a certain limit and all the windows are closed. This can be achieved by creating dependencies between the devices in the network. If the criteria are met, the air conditioner will receive a message and activate. This is useful for saving energy spent on the air conditioning.
   2) The fridge alerts when a certain temperature is reached. This is useful to preserve the food in the fridge. If the fridge stops working, the owner will be noticed directly with an alert.
   3) Remote control of the car heater and lock. Information about car mileage, location, etc.
   4) Smart-thermostat that allows for remote control of the temperature, and automatically adjusts the temperature depending on activity and preference.
   5) Lights that activate when movement is detected, and the sun has gone down.

B) Long range applications: Keeping track of objects, such as key-chains and animals. For example:
   1) Connecting a sensor device on the necklace of a pet. This means you can keep track of your pet wherever its location, where there is supported radio communication. The sensor device can for example alert if the pet’s location is too far from home, or in dangerous territory.
   2) Installing sensor devices on rat traps. When a rat is caught in the trap, a message is sent to the owner so that the rat can be collected, and the rat trap reset. Maybe the owner wants to place more rat traps in that specific location due to the high rate of success.
3) Installing sensors on dustbins. The amount of waste thrown varies with season. People might create more waste at Christmas, summer vacation etc. and less waste at other times of the year. This means the dustbin collectors know how much waste is in every dustbin, and where they’re located. An efficient route can be calculated and sent to the dustbin collectors depending on the amount of waste and where the dustbins are located. This might reduce the time and distance for collecting garbage.

### 1.2 IoT Protocols

The technical term for IoT is as follows: The network of physical devices and other items embedded with electronics, software, sensors and connectivity which enable these devices to connect, collect and exchange data [6]. The sensor devices typically require being completely wireless. This means receiving energy from batteries and communicating through wireless networks.

Wi-Fi’s area of use is centred on home-devices such as computers, tablets and phones whom require a high data-rate for streaming movies and playing games etc. This means Wi-Fi focuses on maximizing throughput and high uptime, which is not energy efficient. Sensor devices usually send small amounts of data with a long time-span between the messages. For example, a temperature sensor might send an 8-bit temperature value once every hour. Therefore, a networking protocol prioritizing low-data rate with low energy consumption would be a better fit for sensor devices. IEEE 802.15.4 (LR-WPAN) [7] is a technical standard which defines the operation of low-rate wireless communication.

There are many IoT protocols and networks, depending on the application. Some sensor devices might require communication with other devices within a specific network, while other sensors can be stand-alone devices sending data to the closest radio receiver. Some devices might have to communicate with computers or phones over IP, etc. Therefore, different protocols and networks will be suited for different applications.
Devices within a home-network might contain sensitive data or require limited access from outside of the network. For example, unauthorized people should not be able to monitor or control the security system of the home, turn on devices etc. A high priority in such a network will be security. A simple stand-alone temperature sensor in a swimming pool might only require read-only limitations. If the temperature value gets into the wrong hands, it might be unhararmful. Devices in a home-network don’t require as long of a range (a few meters), as it can send data to the home-router. Standalone devices outside might require long range (a few kilometres) to send information to the closest radio receiver.

There are multiple IoT protocols and networks. Some of the currently (2019) popular ones are: Bluetooth, ZigBee, Z-Wave, NFC and the newer protocol Thread [8].

1.3 Thread

Thread was released year 2014 and has the support of many large companies [9]. It supports IP communication, which simplifies development and makes it compatible with other IP-devices. What makes Thread unique is the offer of IPv6 communication combined with high reliability and security [10]. Thread is the chosen networking protocol for this assignment. The choice was made by ÅF and the reasoning behind it is to test the compatibility and features of newer IoT protocols. The support from large companies, together with IPv6 support and high priority regarding security were some of the main features that attracted this specific protocol.
1.4 Purpose

The purpose for this project is to update ÅFs [11] IoT demonstrator with the Thread networking protocol and Thread certified hardware. The current demo-box contains a Raspberry Pi, Bluetooth and Arduino components and Azure Cloud component for storage and presentation. When the Thread network is completed, it is to be implemented in the demo-box. Figure 1 shows the demonstrator with its current components. The plan is to exchange the wireless communication protocol Bluetooth for the wireless networking protocol Thread and preferably integrate it with the current cloud network interface.

*Figure 1: ÅF’s IoT demonstrator.*
1.5 Goals

The goals for this project are:

- Test hardware and software compatibility with Thread – Thread is a rather new protocol and has gained popularity recently. ÅF have no prior testing or experience with Thread. Due to this, testing of hardware and software is required to see how functional the protocol is in practice.

- Set up a Thread mesh network with devices communicating through IPv6 – Currently, most of the internet communication is done through IPv4. Like previous goal, this is to test the network capabilities and configurations together with IPv6. Also, a mesh network must be enabled for Thread to be in use and send information to a web server or cloud.

- Implement code to the single-chip devices – To test features offered by Thread, software must be implemented on the devices sending sensor data. The plan is to install synchronous and asynchronous communication between the sensor devices and a web server.

- Test communication between Thread network and web server – Testing what type of web server works well together with Thread, and how developed Thread is on the application layer.

- Connect Thread network with Cloud network – If above goal works well, the plan is to integrate Thread to a cloud network, preferably the one already in use in the demo-box (Cloud Azure). This goal also has the purpose of testing support from different cloud interfaces to the Thread protocol.

- Implement Thread with ÅFs current IoT demonstrator – Exchanging required hardware and software in the demonstrator for Thread to be installed and working.
1.6 Structure

This chapter described the purposes and goals and gave a short introduction of IoT and Thread. Chapter 2 provides further information and choices regarding Thread. It discusses the current IoT market and the fundamentals for Thread, such as hardware, networking architecture, protocol stack and communication protocol. Chapter 3 gives an overview of the chosen network layout, installations and configurations for this project. Chapter 4 reviews the implementation of code on the devices to be able to run and communicate over Thread. Chapter 5 shows the result of the project, including a packet trace analysis and a showcase of the data sent from the Thread network, stored in the database of the web server. Chapter 6 gives an evaluation of this project. This includes reviewing which goals were achieved, what work is to be done by future developers and what problems arose during the project. Finally, chapter 7 is a conclusion of the project.
2 Background

This chapter provides information and choices regarding Thread. Section 2.1 describes the current IoT market, why the Thread protocol was created and what it offers. Section 2.2 describes the fundamentals of Thread, such as the network architecture, protocol stack and IPv6. Section 2.3 describes the application CoAP, a lightweight web networking protocol used with Thread. Section 2.4 describes the hardware specifications and usage within Thread.

2.1 IoT market

The current IoT protocols (Z-wave, ZigBee, Bluetooth) are not IP-based which makes communication with IP-based devices unnecessarily complicated. Year 2014, several companies recognized the need for a new and better way to connect IoT devices with the IP. Together, they formed the Thread Group to develop Thread. Members of the Thread group include companies such as Nordic Semiconductor, Google’s Nest Labs, Yale Security, Silicon Labs, Samsung electronics, ARM and more [9].

Thread is a mesh networking protocol that mainly focuses on connecting home applications with each other and the internet. The Thread group aims for a functionality and end user experience similar to that of Wi-Fi. Users can communicate with a Thread network directly from their own devices (smartphone, tablet, computer) via their ordinary home Wi-Fi network, alternatively using a cloud-based application with a gateway.

Both Thread and Zigbee run on already existing standards from IEEE’s [12] and IETF [13] therefor Thread and ZigBee have formed an alliance to combine ZigBee’s application library with Thread’s IP-networking protocol [14]. Year 2017, Thread and ZigBee announced the availability of Dotdot, an application layer interoperability. This allow devices from different networks (Wi-Fi, Thread, Bluetooth, Zigbee) to communicate with each other using the same protocol on an applications-layer level [15].
2.2 Thread

Thread offers a mesh networking protocol with the ability to communicate through IPv6 within the Thread network, and IPv4/IPv6 to devices outside of the network, based on existing IETF standards. Thread offers advanced encryption, manual authentication, multiple redundancy alternatives, high scalability, low energy usage and wireless device-to-device and device-to-cloud communication.

Thread runs on top of 6LowPAN [16] and every device has a unique global IPv6 addresses. This allows customers and devices direct access to the devices without using NAT. This requires an IPv6 compatible network. The network router has a global IPv6 and IPv4 address for increased compatibility with other networks over the internet. If IPv4 communication is used, a NAT transition will allow IPv6 communication within the Thread network and IPv4 communication outside the Thread network.
2.2.1 Network Architecture

To allow multiple nodes to communicate under safe and efficient circumstances, some sort of network is preferred. Figure 2 is an example of a thread network architecture. The architecture for the Thread network dynamically adjusts depending on the number of devices in the network. If there is only one thread router, the network forms a star-architecture. If there are more than one thread routers, the network forms a mesh-architecture, as seen in Figure 2. The Thread network offers different roles for the devices depending on their hardware, software and purpose. If the requirements are met, roles can and will change dynamically depending on the demand in the network.

Figure 2: An example of a Thread network architecture
**Border Routers** – Provides internet access and manages communication of IP-packets into and out of the thread network. The physical requirements for this role is a IEEE 802.3 (Ethernet) [17] or IEEE 802.11 (Wi-Fi) [18] to connect with the home-network, as well as IEEE 802.15.4 (LR-WPAN) [7] to connect with the devices in the Thread network. A thread network can have multiple border routers to increase redundancy and distribute traffic. The network should have at least two border routers to guarantee no single-point of failure policy. If the border router itself does not support LR-WPAN a network coprocessor that does can be used as support.

**Leaders** – Manages the registered routers and decides which devices should be routers by promoting or demoting sensor devices. All information present in the leader is also present in all other routers. If the leader fails, a router will adopt the leader role. Leaders are typically plugged in to the wall and require being up and running always.

**Thread routers** – Provides routing services for packets in the Thread network, by forwarding packets to their desired destination within the network. They handle security during authentication for new devices and uses a routing table to manage neighbours and child devices. The table is updated upon change in the network and used to streamline routing. Routers can either upgrade to a leader role, or downgrade to a sensor device. Routers are typically plugged in to the wall and require being up and running always.

**Child devices** – Also known as sleepy devices. These devices usually wake up, send or receive data and go back to sleep mode. If the child device is router eligible, it can be upgraded to a Thread router. Child devices share parts of their local address with their parent Thread router, for simplified routing. If the parent fails, the network will provide another parent router for the child device. Child devices usually run on batteries, and their lifetime can be several years due to their energy-saving functions.
2.2.2 Protocol stack

Table 1 shows the OSI-model used in the Thread network from the application-layer down to the physical-layer. The rows with the checkmark are the ones included in the Thread protocol stack. Thread uses a combination of already defined standards, which include CoAP, UDP, DTLS, 6LowPan, vector routing and LR-WPAN.

Table 1: The OSI-model for the Thread protocol stack.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Protocol</th>
<th>Standards and Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>CoAP</td>
<td>RFC's: 7252</td>
</tr>
<tr>
<td>Transport</td>
<td>UDP + DTLS</td>
<td>RFC's: 788, 6347, 4270, 4492v, 3315, 5007</td>
</tr>
<tr>
<td>Network</td>
<td>6LowPan + Distance Vector Routing</td>
<td>RFC's: 4944, 4962, 6775, 1059, 2080</td>
</tr>
<tr>
<td>Link</td>
<td>Low-rate WPAN</td>
<td>IEEE 802.15.4 MAC</td>
</tr>
<tr>
<td>Physical</td>
<td>Low-rate WPAN</td>
<td>IEEE 802.15.4 PHY</td>
</tr>
</tbody>
</table>

**IEEE 802.15.4 PHY** [19] – LR-WPAN connection designed for low power devices where the data rate and range are limited. The maximum data rate is 250 kbps, with a maximum range estimated at 10 meters and a frequency band of 2.4GHz, with sub bands. This physical layer is most suitable for applications involving many nodes in a network, due to the shorter range and low power usage.

**IEEE 802.15.4 MAC** [19] – This media access control layer is used for message handling. Functions include congestion control, CSMA mechanisms, resending messages, acknowledging messages as well as encrypting messages. The maximum packet size is 128 bytes.
**6LowPAN (IPv6) [16]** – Transmits and receives IPv6 packets restricting the maximum packet size to 128 bytes on 802.15.4 networks. 6LowPAN enables encapsulation and compression mechanisms for increased efficiency. Since IPv6 has a maximum packet size of 1280 bytes, the packets must be divided into smaller packets to run on LR-WPAN.

**Distance Vector Routing [20]** – Used for finding efficient routing for packet communication within the Thread network. It measures the distance by number of routers a packet must pass and determines the best route. This protocol requires that a router informs its neighbours of network topology changes periodically.

**UDP [21] + DTLS [22]** – Thread uses UDP for low amount of meta-data and its simplicity. To ensure safe and reliable data-transfer, Thread has complemented the UDP-protocol with MAC- and application-layer retries if an acknowledgement is not received. DTLS connection with a thread border router is required for a new device to join the network for increased security and works similar to a TCP-handshake.

**Application Layer** – There is no standard application layer. The customer or developer chooses or builds the application based on the capabilities and requirements for the network. Thread does though provide basic application services such as UDP, multicast and CoAP [23].
2.2.3 IP Addressing

Thread devices have local and global IPv6-address. Global addresses are generated through an ISP and are reachable world-wide, used for direct communication with devices outside of the network. Unique local addresses are also generated through an ISP and used within the Thread network to simplify routing based on the devices addresses. For example, a child node contains parts of the unique local address of its parent. Unique local addresses can be routed within one routing domain, so an ISP can choose to use them for services which can’t be publicly accessible. The prefix is fc00::/7.

Devices are assigned a link local address which is semi-randomly created within the network used for neighbour discovery, duplicate address detection and router advertisement. The prefix is fe80::/10. Devices are also assigned a multicast address, used if a message has multiple receivers in the network with the prefix ff00::/8

For IPv6/IPv4 translation, the last 32 bits used with the prefix 64:ff9b::/96 is considered an IPv4 address. This is used so messages can be sent to IPv4-addresses as well as IPv6-addresses. For example, the IPv6 address 64:ff9b::0808:0808 equals the IPv4 address 8.8.8.8.

The Thread border router has one IPv6-address and IPv4-address. If IPv6 is not compatible with the home-network router, IPv4 must be used instead. Since devices in the network don’t use IPv4 addresses, the border router uses a NAT6to4 to facilitate communication between IPv6 and IPv4.
2.3 Constrained Application Protocol

Specific network-protocols have been created to be suited for IoT devices. Some key network features for IoT devices are low energy and processing consumption for increased battery life and response-time. The recommended and compatible network protocol for Thread is CoAP.

CoAP is a machine-to-machine lightweight web application networking protocol. The format has similarities to HTTP but offers lower energy and processing consumption combined with smaller packet sizes to better fit IoT devices. CoAP runs over UDP instead of TCP for further simplification as well as compatibility with 6LowPAN. CoAP share with HTTP the architectural style that defines a set of constraints to be used for web services known as REST. Some of the REST methods are GET, POST and DELETE [24].

Figure 3 shows the application-, transport- and networking-layer protocols for HTTP to the left and CoAP to the right.

![Figure 3: HTTP and CoAP protocol stacks.](image)

To compensate for the lack of reliability using UDP, CoAP provides an optional retransmission mechanism on the application-layer that retransmits messages until an acknowledgement is received.
2.4 Hardware

There are several compatible hardware devices that are Thread certified. [25]. The choice of hardware was based on recommendations from ThreadGroup [9], OpenThread [10] and other variables such as accessibility and quality. The chosen hardware for this project are the RPi model 3B [26] shown in Figure 5, and the nRF52840 DK [27] shown in Figure 4.

![Figure 5: The Raspberry Pi Model 3B](image1)

![Figure 4: The Nordic Semiconductor nRF52840 Development Kit](image2)
2.4.1 Hardware specification

Table 2 show the relevant specifications for all the hardware used for the Thread network. The GPIO-pins can be used to connect sensors to the devices. The programmable buttons seen in Figure 6 can be used to execute commands on the sensor devices, and the LED lights to show activity. This simplifies development and offers functionality for presentation purposes, and more.

Table 2: Hardware specification

<table>
<thead>
<tr>
<th>Category</th>
<th>RPi Model 3B</th>
<th>nRF52840 DK</th>
</tr>
</thead>
<tbody>
<tr>
<td>SoC</td>
<td>Broadcom BCM2837</td>
<td>nRF52840</td>
</tr>
<tr>
<td>CPU</td>
<td>Quad core 1.2GHz 64bit</td>
<td>ARM Cortex-M4 64GHz</td>
</tr>
<tr>
<td>RAM</td>
<td>1GB</td>
<td>1MB</td>
</tr>
<tr>
<td>ROM</td>
<td>8GB Micro SD-card</td>
<td>256KB</td>
</tr>
<tr>
<td>Networking</td>
<td>Ethernet, WLAN, BLE</td>
<td>NFC, LR-WPAN, Bluetooth, BLE</td>
</tr>
<tr>
<td>GPIO</td>
<td>40-pin</td>
<td>48-pin</td>
</tr>
<tr>
<td>Ports</td>
<td>4x USB2, HDMI</td>
<td>2x Mini USB2</td>
</tr>
<tr>
<td>Battery</td>
<td>-</td>
<td>CR2032</td>
</tr>
<tr>
<td>Programmable LED-lights</td>
<td>-</td>
<td>4</td>
</tr>
<tr>
<td>Programmable Buttons</td>
<td>-</td>
<td>4</td>
</tr>
</tbody>
</table>

Figure 6: The programmable buttons on the nRF52840 DK.
2.4.2 Hardware Usage

The **border gateway** is set up with a RPi and a DK, communicating with each other over USB as shown in Figure 7. The RPi is connected to a home-router through Wi-Fi or Ethernet and oversees communication with outside networks. The DK acts as a network co-processor to the RPi and oversees communication within the network with its LR-WPAN radio.

The **sensor devices** consist of DKS. They use LR-WPAN radio to communicate within the network. These devices can be completely wireless.

*Figure 7: The Raspberry Pi and development kit forming the Border router.*
3 Design

This chapter gives a fundamental view of the network and how to set it up. Section 3.1 describes the network layout and the purpose of every component in the layout. Section 3.2 describes required or recommended installations to set up the Thread network and web server. Section 3.3 describes network configurations required for devices to join and communicate in the Thread network.

3.1 Network layout

Figure 8 shows an overview of the network layout. The chosen design for this project is to be compatible with the functions provided by Thread. That is, get information to and from the Thread network to a server on the internet, using the safety and reliability offered by Thread.

*Figure 8: An overview of the network layout.*
The web server is an Apache2 HTTP-server running a MySQL database. It uses Crosscoap for CoAP-HTTP translation. All data sent from sensor devices is shown in tables and graphs on the web server. Sensor devices can be leaders, routers or child devices. The sensor devices send the temperature value of their SoC to the server synchronically with use of a timer. The four programmable buttons on the sensor devices are used to activate and deactivate the timer, as well as increase or decrease the timer interval. The web server is noted when a button is pressed.

3.2 Installation

3.2.1 Software development kit
The programs used to compile and run the software on the network co-processor and sensor devices are Embedded Studio and J-link. Embedded Studio is an IDE application for embedded c/c++ programming. J-Link is a SDK required to debug and run code on the DKs. Both programs are compatible with Windows, macOS and Linux.

3.2.2 Border router
Nordic Semiconductor provides an image file that is compatible with the RPi model 3B. The image file is flushed to the microSD card of the RPi. The operating system is Raspbian, running on GNU/Linux. Libraries included are Open Thread’s open-source implementation of the Thread networking protocol as well as their open-source Thread border router. Alternatively, the border router can manually be compiled from OpenThread’s own guide. For communication to the border router, the SSH client PuTTY is used which allows for world-wide remote access and control.

3.2.3 Network Co Processor
Nordic Semiconductor provide ready to run example code for the network co-processor. The network co-processor and border router communicate through the spinel protocol on tunnel-mode. Network configurations and settings are managed through the wpantund network interface from the border router CLI. A USB connection is used to transfer the code to the network co-processor. When the code is installed and transferred, connect it to the RPi and reboot.
3.2.4 Sensor devices

Nordic offers example code for different applications [41]. The code running on the sensor devices is modified code mostly from the CoAP-cloud example. A USB connection is used to transfer the code as shown in Figure 9. The code is explained in detail in section 4.2.

![Figure 9: DK's connected to Embedded Studio and J-Link on the laptop.](image)

3.2.5 Web Server

The web server of choice is Apache2 HTTP server for being the most popular web server for over 20 years [28]. MYSQL is used as a database to store incoming data from sensor devices. Communication between the web server and the database is done with the programming language PHP [42]. Since the sensor devices send data through the CoAP-protocol and Apache uses HTTP, a CoAP-HTTP proxy is required to translate messages between the two protocols [30]. The web server runs on Ubuntu 18.04 [43].

The installations required are apache2 [28], MySQL server [29], php [42] and cross-coap [30]. The code is explained in detail in section 4.3.
3.3 Network configuration

For devices to join and communicate in the Thread network some network configuration are required. All network configurations are done from a separate windows laptop with remote control through PuTTY.

3.3.1 Border router

Nordics border router image file is ready-to-run. Use the system central management tool to check if necessary border router libraries are running.

3.3.2 Network co-processor

Settings are configured with the `wpantund` interface through the RPi CLI. Figure 10 shows the network configuration for the network co-processor. The channel determines what frequency to be used within the network. All channels are close to the chosen frequency band, differentiating with a small range. For communication in the Thread network, devices need to use the same channel. The XPAN ID together with the network name is a unique identifier for the network and is used in case of name collision. The PAN ID is a unique identifier for communication within the Thread network and used in case of multiple Thread networks within radius running on the same channel. Figure 10 shows the status for the network co-processor.

```
pi@raspberrypi:~ $ sudo wpantctl status
wpan0 => {
   "NCP:State" => "associated",
   "Daemon:Enabled" => true,
   "NCP:Version" => "OPENTHREAD/g591ec62; NRF52840; Sep 19 2018 11:42:29",
   "Daemon:Version" => "0.08.00d ( ; Feb 23 2018 13:17:33)",
   "Config:NCP:DriverName" => "spinel",
   "NCP:HardwareAddress" => [B42D363E889862B0]
   "NCP:Channel" => 11,
   "Network:NodeType" => "leader",
   "Network:Name" => "NordicOpenThread",
   "Network:XPANID" => "0xDEAD00BEEF00C4F",
   "Network:PANID" => "0xABCDEF",
   "IPv6:LinkLocalAddress" => "fe80::e40e:08:4ef7:2bf1",
   "IPv6:MeshLocalAddress" => "fdde:ad00:beef:0:f519:35ec:137b:e4f3",
   "IPv6:MeshLocalPrefix" => "fdde:ad00:beef::/64",
   "com.nestlabs.internal:Network:AllowingJoin" => false
}
```

*Figure 10: Network co-processor status*
4 Implementation

This chapter describes the implementation of the design described in previous chapter. Section 4.1 describes the commands to configure devices and add them to the network. Section 4.2 describes the code running on the sensor devices, and section 4.3 describes the configuration and code running on the web server.

4.1 Network settings and commissioning

This section describes the commands to configure devices and add them to the network. Configuration is done through the Open Thread CLI commands [44], through a PuTTY COM-port configuration with J-Link [45]. Subsection 4.1.1 shows how to add sensor devices from the border router and subsection 4.1.2 shows how to join the network from the sensor devices.
4.1.1 Network co-processor

Figure 11 shows the commands to add a device to the network. `sudo wpanctl commissioner -e` enables commissioning. Sudo wpanctl commissioner -a PSK EUI64 will add the device. PSK is a pre-shared key, and EUI is an extended unique identifier, which is unique for every device and generated based on the devices MAC-address.

```
pi@raspberrypi:~ $ sudo wpanctl commissioner -e
Commissioner command applied.
pi@raspberrypi:~ $ sudo wpanctl commissioner -a NORD1C 0b53903f166cfaf2
Joiner added.
pi@raspberrypi:~ $  
```

*Figure 11: Network co-processor status*

4.1.2 Sensor device

Figure 12 shows the commands for joining a network with a sensor device. First, the device gets its EUI which is used by the network co-processor in Figure 11. When the network co-processor has added the sensor device, the sensor device must join the network with the command `joiner start PSK`. Joining a network takes a few seconds, because of handshakes and other necessary functions required. The response is either accepted or rejected.

```
> eui64
0b53903f166cfaf2
Done
> joiner start NORD1C
Done
```

*Figure 12: Joining the network as a sensor device.*

When the sensor device is added to the network, it needs to use the correct configuration and parameters. Figure 13 show the configuration settings for the network. First, we choose a temporary network name `temporaryName`. This name will change to the network name NordicOpenThread of the border router if everything is working. The PAN ID and channel are set identical to the network it wants to join. Autostart can be enabled so that the PuTTY connection to the sensor device can be skipped at restart. Use the command `ifconfig up` to enable networking, and
the command `thread start` to enable communication within the Thread network. If the device joins a network, the network name will change to that of the joined network. The state of the device will depend on network demands and device specifications. This sensor device is router eligible, and is set to the router state. To confirm that communication from the child device is correctly routed through the thread network to the internet, the router role is disabled and the devices goes into child state. When in child state, communication will be routed to it’s parent (a leader or router device) before entering the border router, which is how information should travel from a child device. When entering child state, the device can ping a server on the internet. If the device receives a response, communication from sensor devices within the thread network to the internet works.

```plaintext
  > networkname temporaryName
  Done
  > panid 0xabcd
  Done
  > channel 11
  Done
  > autostart enable
  Done
  > ifconfig up
  Done
  > thread start
  Done
  > networkname
  NordicOpenThread
  Done
  > state
  router
  Done
  > routerrole disable
  Done
  > state
  child
  Done
  > ping 64:ff9b::0808:0808
  > 16 bytes from 64:ff9b::0:0:0:0:0:0:808:808: icmp seq=6 hlim=117 time=47ms
```

*Figure 13: Configuration settings and testing the network for sensor device.*
4.2 Sensor devices

This section describes the code running on the sensor devices. The functions described below are either created by me (commented with +) or created by Nordic Semiconductor in one of their example codes. Subsection 4.2.1 describes the functions in the main loop, seen on Figure 14. Subsection 4.2.2 describes the functions for the timer, button interrupts and CoAP messages.

4.2.1 Main function and global variables.

Figure 14 shows the code in the main function and the functions inside it. Rows 3-11 initiates required functions such as a logger, a scheduler, a timer, thread, CoAP and so on. Rows 14-21 checks if there are any tasks scheduled for execution. If not, the sensor device will enter sleep mode (if activated) and receive an interrupt when a task needs to be executed. Figure 15 shows the global variables added. Variable temp_timer_interval sets the starting interval for the temperature timer in milliseconds. Variable temp_timer_active shows if timer is activated or not. List server_ipv4_address stores the address of the server, and variable node_name gives the sensor device a “human-readable” name.

```c
1. int main(int argc, char * argv[])  
2. { 
3.     log_init(); 
4.     scheduler_init(); 
5.     timer_init(); 
6.     nrf_temp_init(); 
7.     thread_instance_init(); 
8.     thread_coap_init(); 
9.     thread_bsp_init(); 
10.    thread_coap_utils_temp_timer_init(); // + 
11.    while (true) 
12.        { 
13.            thread_process(); 
14.            app_sched_execute(); 
15.        } 
16.        if (NRF_LOG_PROCESS() == false) 
17.            { 
18.                thread_sleep(); 
19.            } 
20.    } 
21. }
```

*Figure 14: Main function.*
Figure 15: Global variables

Function log_init (see Figure 16) initiates the log module. The command NRF_LOG_INFO(*char) outputs a debug string on the CLI of JLinkRTTViewer.exe software from J-Link [32]. Function scheduler_init initiates a scheduler which places upcoming tasks to execute. Function timer_init initiates the timer, and function nrf_temp_init loads values to the temp module so the temperature value can be fetched.

Function thread_instance_init initiates Thread and configures settings for the sensor device, as seen in Figure 16. Row 5 disables sleepy mode, so the sensor device always is enabled (for development purposes) and row 6 commissions the device automatically if already accepted to a network. Function thread_init initiates Thread with chosen PAN ID, channel, network name, and other configurations. Function thread_cli_init initiates a command line interface (used in Figure 13). This connection can be used to set PAN ID, channel name, network name, and other parameters when the sensor device is up and running [44]. Function thread_state_changed_callback_set sets functions that are executed when the state of the node changes, from for example a child device to a router device.

```c
1. static void thread_instance_init(void)  
2. {                                           
3.     thread_configuration_t thread_configuration = 
4.         {                                   
5.             .role                             = RX_ON_WHEN_IDLE,  
6.             .autocommissioning = true,  
7.         };                              
8. }                                          
9. thread_init(&thread_configuration);        
10. thread_cli_init();                       
11. thread_state_changed_callback_set(thread_state_changed_callback); 
12. }
```

Figure 16: Function thread_instance_init.
Function thread_coap_init in Figure 17 chooses CoAP configuration and enables CoAP. Sensor devices act as CoAP clients so coap_client_enabled = true.

```
1. static void thread_coap_init(void)
2. {
3.   thread_coap_configuration_t thread_coap_configuration =
4.   {
5.     .coap_server_enabled = false,
6.     .coap_client_enabled = true,
7.     .coap_cloud_enabled = false,
8.     .configurable_led_blinking_enabled = false,
9.   };
10.   thread_coap_utils_init(&thread_coap_configuration);
11. }
```

Figure 17: Function thread_coap_init.

Function thread_bsp_init initiate the Thread board support package in Figure 18, which include functions for the LED lights and buttons to activate and function correctly.

```
1. static void thread_bsp_init(void)
2. {
3.   uint32_t error_code = bsp_init(BSP_INIT_LEDS | BSP_INIT_BUTTONS, bsp_event_handler);
4.   APP_ERROR_CHECK(error_code);
5.   error_code = bsp_thread_init(thread_ot_instance_get());
6.   APP_ERROR_CHECK(error_code);
7. }
```

Figure 18: Function thread_bsp_init.
Function `thread_coap_utils_temp_timer_init` initiates a timer to be used for sending temperature values synchronically, as shown in Figure 19. Row 3 creates an application timer. Row 4 inputs a timer identifier, row 5 tells the timer to repeat itself until halted, and row 6 inputs the function to be executed when the timer goes off.

```c
1. void thread_coap_utils_temp_timer_init() // +
2. {
3.     uint32_t error = app_timer_create(
4.         &m_temp_timer,
5.         APP_TIMER_MODE_REPEATED,
6.         temp_timer_handler
7.     );
8.     ASSERT(error == NRF_SUCCESS);
9. }
```

*Figure 19: Function thread_coap_utils_temp_timer_init.*

### 4.2.2 Timer, buttons and CoAP messages

Figure 20 shows the function executed when the temperature timer goes off. Row 3 logs the function name for debugging purposes, row 4-6 allocates a buffer, fetches the temperature value from the SoC and converts it to a string. Row 7 sends a CoAP message where `info="temp"` and `data=buffer`.

```c
1. static void temp_timer_handler(void * p_context) // +
2. {
3.     NRF_LOG_INFO("temp_timer_handler");
4.     char buffer [BUFFER_SIZE];
5.     uint32_t temp = nrf_temp_read();
6.     sprintf(buffer, "%d", temp);
7.     thread_coap_utils_send_message(thread_ot_instance_get(), "temp", buffer); // +
8. }
```

*Figure 20: Function temp_timer_handler.*
Figure 21 shows function `thread_coap_utils_send_message`. Nordic semiconductor provides a similar function towards a cloud interface of their choice. This function is slightly modified to send messages to my private Apache HTTP server. The function receives the input parameters: `p_instance`, used to allocate a CoAP message. Variable `node_info` is used for sorting purposes on the web server, and variable `node_data` represents the data sent to the server. Row 9 initiates a CoAP header with `OT_COAP_TYPE_CONFIRMABLE` activating the reliable data transfer CoAP offers and `OT_COAP_CODE_POST` setting the method as POST. Row 10 sets a 2-byte token ID required for reliable transfer protocol. Row 11 sets the URI-path. Row 20 stores the name, info and data into a buffer. Row 22 puts the payload equal to the buffer. Rows 28-36 stores the IPv6 address of the server. The first four bytes (6400:9bff::/96) does an IPv6/IPv4 translation where the last 32 bits (83.254.63.90) is the IPv4-address. Rows 36-39 sets interface, port-number and stores that in struct `OtMessageInfo`. Row 40 sends the CoAP request.
void thread_coap_utils_send_message(otInstance * p_instance, char* node_info, char* node_data)
{
  otError error = OT_ERROR_NONE;
  otMessage * p_message;
  otMessageInfo message_info;
  otCoapHeader header;
  do
  {
    otCoapHeaderInit(&header, OT_COAP_TYPE_CONFIRMABLE, OT_COAP_CODE_POST);
    otCoapHeaderGenerateToken(&header, 2);
    UNUSED_VARIABLE(otCoapHeaderAppendUriPathOptions(&header, ""));
    UNUSED_VARIABLE(otCoapHeaderSetPayloadMarker(&header));
    p_message = otCoapNewMessage(p_instance, &header);
    if (p_message == NULL)
      NRF_LOG_INFO("Failed to allocate message for CoAP Request\n");
    break;
  }
  while (false);

  char buffer[BUFFER_SIZE];
  int size = sprintf(buffer, "name=%s\ninfo=%s\ndata=%s", node_name, node_info, node_data);
  if (size <= 0) NRF_LOG_INFO("Failed to sprintf");
  error = otMessageAppend(p_message, buffer, size);
  if (error != OT_ERROR_NONE) { 
    NRF_LOG_INFO("Failed to append message\n");
    break;
  }
  m_coap_status.peer_address.mFields.m16[0] = 0x6400;
  m_coap_status.peer_address.mFields.m16[1] = 0x9bff;
  m_coap_status.peer_address.mFields.m32[1] = 0;
  m_coap_status.peer_address.mFields.m32[2] = 0;
  m_coap_status.peer_address.mFields.m8[12] = server_ipv4_address[0];
  m_coap_status.peer_address.mFields.m8[13] = server_ipv4_address[1];
  m_coap_status.peer_address.mFields.m8[14] = server_ipv4_address[2];
  m_coap_status.peer_address.mFields.m8[15] = server_ipv4_address[3];
  memset(&message_info, 0, sizeof(message_info));
  message_info.mInterfaceId = OT_NETIF_INTERFACE_ID_THREAD;
  message_info.mPeerPort = OT_DEFAULT_COAP_PORT;
  message_info.mPeerAddr = m_coap_status.peer_address;
  error = otCoapSendRequest(p_instance,
    &message_info,
    p_message,
    NULL,
    p_instance);
  if (error != OT_ERROR_NONE && p_message != NULL)
    NRF_LOG_INFO("Failed to send CoAP Request: %d\n", error);
  otMessageFree(p_message);
  NRF_LOG_INFO("CoAP message sent");
}
Figure 22 shows the beginning of the function `bsp_event_handler`. This function is activated when one of the four buttons are pressed. Button 0 activates or deactivates the temperature timer. When any button is pressed, a CoAP message is sent to the server with `info="btn"` and `data` representing which button was pressed and what function it has, as seen on row 11 and row 21. When the temperature timer is active, it will send the temperature value of the SoC every `temp_timer_interval` milliseconds. All `NRF_LOG_INFO` commands are used for debugging purposes.

```c
1. static void bsp_event_handler(bsp_event_t event) // +
2. {
3.     switch (event)
4.     {
5.         case BSP_EVENT_KEY_0:
6.             NRF_LOG_INFO("Button 0 pressed.");
7.             if(!temp_timer_active)
8.                 {
9.                     temp_timer_active = 1;
10.                    thread_coap_utils_send_message(thread_ot_instance_get(), "btn", "Button 0: Timer activated.");
11.                   NRF_LOG_INFO("Timer is activated");
12.                    thread_coap_utils_activate_temp_timer(temp_timer_interval);
13.                   break;
14.             }
15.         }
16.         else
17.             {
18.                 temp_timer_active = 0;
19.                 NRF_LOG_INFO("Timer is deactivated");
20.                 thread_coap_utils_send_message(thread_ot_instance_get(), "btn", "Button 0: Timer deactivated.");
21.                 thread_coap_utils_disable_temp_timer();
22.                 break;
23.             }
24.     }
25. }
```

*Figure 22: Start of function bsp_event_handler.*
Figure 23 show the actions for button 2 and button 3 (button 1 has no function). Button 2 will decrease the timer interval by 1000 milliseconds at a minimum value of 2000 millisecond. Button 3 will increase the timer interval by 1000 milliseconds. If the timer is activated when the timer interval is increased or decreased, the timer will restart with the new timer interval as seen on rows 11-15 and rows 29-33.

```c
1.     case BSP_EVENT_KEY_2:
2.         {
3.             NRF_LOG_INFO("Button 2 pressed.");
4.             temp_timer_interval = temp_timer_interval - 1000;
5.             if(temp_timer_interval < 2000) temp_timer_interval = 2000;
6.             char buffer [BUFFER_SIZE];
7.             sprintf(buffer, "Button 2: Timer interval set to %d seconds.", temp_timer_interval/1000);
8.             thread_coap_utils_send_message(thread_ot_instance_get(), "btn", buffer);
9.             if(temp_timer_active)
10.                 {
11.                     thread_coap_utils_disable_temp_timer();
12.                     thread_coap_utils_activate_temp_timer(temp_timer_interval);
13.                 }
14.             break;
15.         }
16.     case BSP_EVENT_KEY_3:
17.         {
18.             NRF_LOG_INFO("Button 3 pressed.");
19.             temp_timer_interval = temp_timer_interval + 1000;
20.             char buffer [BUFFER_SIZE];
21.             sprintf(buffer, "Button 3: Timer interval set to %d seconds.", temp_timer_interval/1000);
22.             thread_coap_utils_send_message(thread_ot_instance_get(), "btn", buffer);
23.             if(temp_timer_active)
24.                 {
25.                     thread_coap_utils_disable_temp_timer();
26.                     thread_coap_utils_activate_temp_timer(temp_timer_interval);
27.                 }
28.             break;
29.         }
30.     default:
31.         return;
32.     }
```

Figure 23: End of function bsp_event_handler.
4.3 Web server

Data from the sensor devices are sent to a web server which collects, stores and presents the data. The web host as of January 2019 is a personal hosting by me due to ease of development. To use another server, update the IP-address in Figure 15. This section describes the configuration and code running on the web server. Subsection 4.3.1 describes the port forwarding required to receive incoming data. Subsection 4.3.2 describes the CoAP to HTTP translation component Crosscoap. Subsection 4.3.3 describes the database used to store sensor data, and subsection 4.3.4 describes the code running the web server.

4.3.1 Port forwarding

Routers which automatically blocks incoming data require port forwarding. This allows remote computers to connect to a specific computer or service within a private local-area network. The ports opened to allow incoming data from sensor devices are:

- Port 80 TCP: Apache service.
- Port 22 TCP: Remote access through PuTTy.
- Port 5683 UDP: Crosscoap service.

4.3.2 Crosscoap

The Crosscoap component is running on the same computer as the web server and acts as a translation service between CoAP and HTTP. It can run on a separate computer. Crosscoap is downloaded from [30]. The programming language Go [46] must be installed and a $GOPATH must be set. To build the crosscoap executable, use the following command:

```
go get https://github.com/ibm-security-innovation/crosscoap
```

The executable should be placed in $GOPATH/bin.

To start crosscoap, use the following command.

```
./crosscoap -listen 0.0.0.0:5683 -backend http://127.0.0.1:80
```

This specific command tells crosscoap to listen for incoming CoAP messages on port 5683 and forward them to the local HTTP server on port 80. It also listens for incoming HTTP messages from port 80 and forwards them to port 5683, supporting two-way communication.
4.3.3 Database
To store incoming data a MYSQL database is used. Figure 24 shows the table used to store data. The table includes five fields. Id stores a unique identifier, for sorting and counting purposes. Name stores the name of the node. Data stores the value, for example the temperature or other sensor data. Info stores information about the data, for example the info might be “temperature”, or “button pressed”. Date stores the date and time when the message was received. This data is presented with graphs and tables on the web site.

```
1. CREATE TABLE nodes (  
2.   id INT(7) UNSIGNED AUTO_INCREMENT PRIMARY KEY,  
3.   name VARCHAR(100) NOT NULL,  
4.   data VARCHAR(100) NOT NULL,  
5.   info VARCHAR(100) NOT NULL,  
6.   date TIMESTAMP CURRENT_TIMESTAMP  
7. )
```

*Figure 24: MYSQL table to store incoming data from sensor devices.*
4.3.4 PHP & HTML

Data is sent continuously to the web server from multiple sensor devices. For simplified presentation, automatic page update is implemented. For better visualization, table style is configured. Figure 25 shows the HTML and PHP code to update the page every two seconds and set the style for tables.

```php
1. 1. <!DOCTYPE html>
2. 2.
3. 3. <?php
4. 4.
5. 5. $page = $_SERVER['PHP_SELF'];
6. 6. $sec = "2";
7. 7.
8. 8. ?>
9. 9.
10. 10. <html>
11. 11. <head>
13. 13. <style>
14. 14. table, th, td
15. 15. {
16. 16.    border: 1px solid black;
17. 17.    border-collapse: collapse;
18. 18. }
19. 19.
20. 20. th, td
21. 21. {
22. 22.    padding: 5px;
23. 23. }
24. 24.
25. 25. th
26. 26. {
27. 27.    text-align: left;
28. 28. }
29. 29. </style>
30. 30. </head>
31. 31. </html>
```

*Figure 25: HTML and PHP to continuously update page and set style.*
Figure 26 shows how data is parsed from incoming post requests and stored into the database. Row 2 is used to parse incoming data to the \$_POST array. CoAP-messages lack a context-type and results in the \$_POST array being empty, if Row 2 is not executed. Rows 9-11 stores the incoming data connected to the keys: name, info and data to local arrays. Row 15 connects to the database with the variables stored in rows 4-7. This will directly connect to the specific database we want to use, where the tables nodes exist. Rows 21-25 will store data to the database if none of the fields are empty. This is to avoid storing empty fields.

```php
<?php
parse_str(file_get_contents("php://input"), \$_POST);

$conn = new mysqli($servername, $username, $password, $database);
if ($conn->connect_error) {
    die("Connection failed: ". $conn->connect_error);
}
if(!empty($name) && !empty($data) && !empty($info))
{
    $sql = "INSERT INTO nodes (name, info, data)
VALUES ('$name', '$info', '$data');
if ($conn->query($sql) === TRUE)
{
    echo "New record created successfully\n"
} else
{
    echo "Error: ". $sql . "<br>" . $conn->error;
}
}
```

*Figure 26: Parse incoming data and storing to database.*
Figure 27 shows how data is selected from the database and printed in tables and graphs. Rows 1-5 select all the data where the info is equal to btn. Data with this information is sent when a button is pressed on the sensor devices. All button actions are to be stored in the table shown on rows 9-28. The synchronous temperature values sent from the sensor devices are displayed in graph.php, implemented on row 38.

```php
$sql = "SELECT name, info, data, date
FROM nodes
WHERE info='btn'
ORDER BY id DESC LIMIT 5"

$result = $conn-&gt;query($sql);

if ($result-&gt;num_rows &gt; 0)
{
    echo "<table style="width:100\%">";
    echo "<tr>";
    echo "<th>Date</th>";
    echo "<th>Name</th>";
    echo "<th>Button data</th>";
    echo "</tr>";
    while ($row = $result-&gt;fetch_assoc())
    {
        if ($row["info"] == 'btn')
        {
            echo "<tr>";
            echo "<td>" . $row["date"] . "</td>";
            echo "<td>" . $row["name"] . "</td>";
            echo "<td>" . $row["data"] . "</td>";
            echo "</tr>";
        }
    }
    echo "</table>";
}
echo "</tr>";
$conn-&gt;close();

<img src="graph.php" alt="Graph over node temperatures">
```

Figure 27: Select button data from database and visualize in table.
Jpgraph [47] is used to implement a graph with PHP to visualize the temperature values sent from the sensor devices. Figure 28 shows how data is selected from the database and stored in the arrays used for the graph. Data where info is equal to temp is shown in the graph. This is the data of sensor devices SoC temperature. The code is designed to work with only two sensor devices (A and B) for presentation purposes but can be altered to automatically show data from new devices added to the network.

```php
1. $datay1 = array();
2. $datay2 = array();
3. $datax = array();
4. $sql = "SELECT name, data, date, info FROM nodes WHERE info='temp' ORDER BY date DESC LIMIT 20";
5. $result = $conn->query($sql);
6. if ($result->num_rows > 0)
7. {
8.   $iterator = $result->num_rows-1;
9.   while($row = $result->fetch_assoc())
10.   {
11.     if($row["name"] == 'A')
12.     {
13.       $data_celcius = $row["data"]/4;
14.       $datay1[$iterator] = $data_celcius;
15.       $datay2[$iterator] = NULL;
16.       $date = substr($row["date"], -8);
17.       $datax[$iterator] = $date;
18.       $iterator = $iterator - 1;
19.     }
20.     }
21.   } else if($row["name"] == 'B')
22.   {
23.     $data_celcius = $row["data"]/4;
24.     $datay2[$iterator] = $data_celcius;
25.     $datay1[$iterator] = NULL;
26.     $date = substr($row["date"], -8);
27.     $datax[$iterator] = $date;
28.     $iterator = $iterator - 1;
29.   }
30. }
31. }
```

Figure 28: Select temperature data from database and visualize in graph.
5 Results

This chapter describes the results of the project design and implementation. Overall, the Thread network is operational and sends sensor data with the CoAP protocol and translates it to HTTP, so the data can be collected by a HTTP web server.

Any type of sensor can be connected to one of the GPIO pins (see Table 2) of the DK with addition to small software configurations. The network can send data from anywhere in the world where there is an internet connection and electricity, to a remote web server.

Section 5.1 traces a packet from a sensor node to the web server with tcpcapd to show an example of how a data packet travels from the Thread network to a remote web server. Section 5.2 visualizes the database with tables and graphs to simplify the representation of data sent from the Thread network to the web server.
5.1 Packet trace analysis

Figure 29, Figure 30 and Figure 31 shows an internet packet routing from a sensor device, through the border router and arriving on the web server.

```
22:05:13.763271 IP (tos 0x0, ttl 61, id 0, offset 0, flags [DF], proto UDP (17), length 83) 192.168.0.15.5683 > 83.252.63.90.5683: [udp sum ok] UDP, length 55 0x0000: 0004 0001 0006 5827 c801 365a 0000 0000 ......'.6...... 0x0010: 4500 0003 0000 4000 3d11 e5eb c0a8 000f e..S...=.......... 0x0020: 53fd 3f5a 1633 1633 003f 8b7b 4202 a035 s.y.R.3.3..(B.5 0x0030: 98fb b0ff 6e61 6d65 3d42 2669 6e66 6f3d ...name=B&info= 0x0040: 6274 6e26 6461 7461 3d42 7574 746f 6c20 btn&data=Button. 0x0050: 303a 2035 696d 6572 2061 6374 6976 6174 O:Timer.activated 0x0060: 6564 2e
```

Figure 29: tcpdump with parameters “-nnnXXs” on border router at port 5683.

Figure 29 shows an internet packets going through the border router of the Thread network. The first segment of Figure 29 shows the incoming packet from a sensor device. The message was sent at the time 22:05:13 as an IPv6 packet. The packet is of type UDP and has a payload of 63 bytes. The payload is shown below as hexadecimal representation to the left and ASCII representation to the right. The name of the sensor device is B, the information sent from the device is btn which tells the server that a button was pressed on the sensor device. The data equals Button 0: Timer activated, which is the data represented in Figure 32. If the information of the packet was temp instead of btn the data would instead be displayed in Figure 33 as a temperature value. After they payload length, we have the IP-address of where the packet was sent. The address df11:22::ce24:9866:f76c:73e9.5683 is a local IPv6 address from one of the sensor devices in the network. The .5683 at the end of the IP address shows which port is being used to transfer the packet (standard CoAP port). After the > symbol is the IPv6 address for the destination of the packet. The
The destined address is 64:ff9b::53fd:3f5a which represents the IPv4-address 83.253.63.90 and is the address of the web server.

The second segment in Figure 29 shows the packet leaving the border router. The time is still 22:05:13 as the routing of packets is a fast process. The packet is now an IPv4 packet instead of IPv6. This is because the border router encapsulates the IPv6 address as an IPv4 address for increased compatibility with other networks and routers. An IPv6 packet would for example not be supported in my home-network nor ÅF’s network. The protocol is still UDP but due to encapsulation the payload is increased from 63 to 83 bytes. The source address is 192.168.0.15.5683 which is the local IPv4 address of the border router, still using the port 5683. The destination address has been translated to 83.253.63.90, which is the IPv4 representation of 64:ff9b::53fd:3f5a. At this point, the message has left the border router of the Thread network and travels to the home-network router, to continue its destination through the internet to the destined web server located on address 83.253.63.90.

Figure 30 shows the packet arriving to the server computer. When a packet with the CoAP format arrives at port 5683 the application Crosscoap translates it into an HTTP packet and forwards it to the desired address and port, which in this case is localhost:80 as the web-server runs on the same computer as Crosscoap, listening on port 80. The figure below only shows the packet arriving to crosscoap, not passed to the web server. This is shown in the next figure.

![tcpdump output](image)

*Figure 30: tcpdump with parameters “-nnvvXXSs” on web server at port 5683.*
Figure 31 shows the packet arriving to the HTTP-server on port 80. Crosscoap has translated the packet format from CoAP to HTTP and forwarded it to the web server. This changes the protocol from UDP to TCP and modifies the metadata and payload format to fit the HTTP format. The source address of the packet is 127.0.0.1.44808 which is the local host address (itself, as Crosscoap is on the same computer) using a temporary arbitrary port (44808) to locally transfer the message to port 80 on localhost. The checksum is incorrect, which seems to be a problem with Crosscoap as the same issue appears when virtual CoAP messages are sent through Crosscoap (with the tool curl). The specifics of the problem are not identified. The REST type for the message is POST with the HTTP 1.1 protocol. The user-agent is Crosscoap/1.0. The content length is increased from 83 bytes to 205 bytes due to the higher packet size demand for HTTP.

The message has now travelled from a sensor node in the Thread network, through the border router, home-router, the internet and finally arrived at the web server address on port 80 as an HTTP message. These messages are displayed in tables and graphs on the next chapter.
5.2 Database visualization

Figure 32 shows a table of the latest 8 packets received on the web server from button actions \((\text{info}=\text{btn})\) on sensor devices in the Thread network. The button that increases the temperature interval was pressed rapidly for a few seconds with bad wireless coverage. If all packets were delivered successfully, they would be received within short time interval and mostly in ascending order. The packets shown in Figure 32 are received unordered and sometimes delayed. This is because packets are being resent with CoAP’s reliable data transfer protocol. All packets were delivered.

<table>
<thead>
<tr>
<th>Date</th>
<th>Name</th>
<th>Button data</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-01-03 01:37:54</td>
<td>A</td>
<td>Button 3: Timer interval set to 8 seconds.</td>
</tr>
<tr>
<td>2019-01-03 01:37:54</td>
<td>A</td>
<td>Button 3: Timer interval set to 9 seconds.</td>
</tr>
<tr>
<td>2019-01-03 01:37:54</td>
<td>A</td>
<td>Button 3: Timer interval set to 7 seconds.</td>
</tr>
<tr>
<td>2019-01-03 01:37:54</td>
<td>A</td>
<td>Button 3: Timer interval set to 6 seconds.</td>
</tr>
</tbody>
</table>

*Figure 32: Last 8 messages received on web server from button actions.*

Figure 33 shows the latest 20 packets received on the web server with \(\text{info}=\text{temp}\). The y-axes of the graph represent the temperature in Celsius and the x-axis the time when the server received the data. Node A (colour blue) has a timer interval of 15 seconds and Node B (colour grey) has a timer interval of 3 seconds. This causes node B to send 5 times as much data than node A. The SoC temperature of Node B was slightly heated before execution and the temperature drops approximately 1 degree Celsius over the time span of 50 seconds as it gets cooled. All packets are delivered.

*Figure 33: Last 20 messages received on web server with temperature values.*
6 Evaluation

This chapter evaluates the project. It compares the set goals to reached goals, discusses potential future work and the problems that arose during the project.

6.1 Goals

This section compares to the set goals (section 1.5) to the accomplished goals for this project. The completed goals are:

- Test hardware and software compatibility with Thread – The nRF52840 development kit works fine with Thread. The nRF52840 Dongle should therefore work fine if it’s programmed through the nRF52840 development kit (connected via SWD-pins for GPIO). The Raspberry Pi model 3B also works fine with Thread.
- Set up a Thread mesh-network with devices communicating through IPv6 – Nordic semiconductors border router image file and Thread example code works fine and performs as expected. There were some software fixes released during the project that fixed a few issues. IPv6 communication works fine within the network and has yet to be tested outside of the Thread network due to lack of IPv6 compatible routers and networks connected to with border router. The border router instead encapsulates IPv6 packets as IPv4 packets.
- Implement code to the single-chip devices – Segger embedded studio together with J-Link works as expected. J-Link offers a lot of tools to supply debugging.
- Test communication between Thread network and web server – The border router and Crosscoap sends and receives messages to the web-server as expected.

The uncompleted goals whom can be implemented by future developers are, but not excluded to (more about this in future work):

- Connect Thread network with Cloud network – Currently there is not too much support for CoAP with Thread on cloud interfaces. Azure Cloud does not support CoAP. Due to time constraints the network is not connected to a cloud interface.
• Implement Thread network with ÅF’s current IoT demonstrator - This was not completed due to lack of cloud interfaces with CoAP together with Azure Cloud as well as lack of time to exchange and integrate hardware.

6.2 Future work

There are a lot of interesting ideas that can be implemented to the IoT Thread demonstrator. Here are some recommended features:

• Implementing and installing Thread on ÅF’s current IoT demo-box. This includes exchanging some hardware in the box to Thread-certified hardware.

• Connect Thread network with a CoAP Cloud interface rather than a private HTTP web server. The reason for this is the increased safety, reliability, ease, compatibility etc. that comes with a remotely hosted CoAP Cloud server.

• Improved web-to-device communication. Currently, the web server can only send acknowledgements to the sensor devices. Further functions could be implemented, for example receiving CoAP post messages from web server to triggering the LED lights on the sensor devices.

• Implementing some sort of voice recognition with Thread could be a neat function.

• Implement code on nRF52840 Dongle. They are a lot smaller, but unfortunately do not run on batteries.

• Several sensors can be connected to the GPIO-pins so further data than the SoC temperature can be sent.
6.3 Problems

Many problems and issues arose throughout the project. This chapter will discuss the problems that might be of use for future development. Irrelevant problems caused by human error will not be mentioned.

- To program the dongles with Nordic’s Thread libraries they must be connected to the development kit through SWD-pins as shown in the following guide [48]. The reason for this is that the dongles are not programmable by them self, as they have no support for J-Link. This means the devices cannot be programmed, debugged or make use of other software that J-Link supports without the support of the DK.

- The Thread commissioning application [49] did not work properly. When trying to connect the application to the border router, the application replied with empty password input even though the password was correctly entered. Alternatively, nodes can be added directly from the network co-processor (RPi CLI) as done in this project.

- When using CoAP’s reliable data transfer (OT_COAP_TYPE_CONFIRMABLE, see Figure 21, row 9) the messages are sent multiple times even though the server has received the message. It was working properly until recently. The reason for the problem is unknown, but all messages received at the web server from Crosscoap generate invalid checksums, which might be the cause of this issue. The same problem arose when sending virtual CoAP messages with the application curl (instead of sending messages from sensor devices), so the problem seems to be with Crosscoap. Messages are sent correctly with OT_COAP_TYPE_NON_CONFIRMABLE, but then messages might not arrive to the server due to packet losses.
7 Summary and Conclusions

During half-time work for 16 weeks, I have had a lot of fun and learnt a lot. The width of this project has made it very educational. The work has included a wide spectrum of tasks, such as: Experimenting with different single chip computers. Programming Realtime embedded single chip computers with my favourite programming language C. Setting up a web server with an SQL database, port forwarding, packet tracing and of course setting up a functional IoT Thread mesh network.

Since Thread is a relatively new IoT protocol, there was a lack of documentation which meant some problems were hard to overcome. It was also hard to differentiate between commercial statements and objective information. For example, at first it appeared the nRF52840 dongles were stand alone compatible with Thread. Therefore, ÅF bought a few dongles but no DKs. Further research revealed that the dongles needed to be attached to a nRF52840 DKs to be programmed. OpenThread’s [36] and Nordic Semiconductor’s tutorials [50] for setting up a Thread network required a few nRF DKs, and I had only one at the start of the project. I tried to set up the Thread network with only one DK but to no avail. Two more DKs were ordered a few weeks into the project. This caused some delays.

Working on this project alone was hard at times. It would have been nice to have someone to spontaneously discuss ideas and problems with, also distribute the load so more work could have been done in total.

I would say Threads future is unclear. The majority of the IoT market is currently using ZigBee, so working together with them on the application layer protocol might be a wise choice [14]. The network requires a gateway (border router) to properly function and IPv6 is still not supported enough for pure IPv6 communication. Neither my home network or ÅF’s network supports IPv6, for example. Other features are can also be problematic. For example, if a battery driven child device upgrades its role to a Thread router, the battery life time would significantly decrease.
Thread would benefit from future development and support for IPv6. Currently, no country exceeds more IPv6 traffic than 15% of total traffic [51]. This means that the support of IPv6 is rather insignificant. The lack of support on the application layer is also currently an obstacle. The current cloud storage in the demo-box (Azure cloud) does for example not support CoAP.

I would like to thank the following people for the help and support given through the project: My supervisor at Karlstad university, Stefan Alfredsson. My two supervisors at ÅF, Per Finnstam and Mathias Björklund. My roommate at ÅF, Christoff Markusson, whom also took time from his schedule to help, and ÅF for giving me this project.
References


[Accessed 07 01 2019].

[Accessed 07 01 2019].

[Accessed 07 01 2019].

[Accessed 07 01 2019].

[Accessed 07 01 2019].

Available: https://www.zigbee.org/zigbee-for-developers/dotdot/.
[Accessed 07 01 2019].

[16] “IETF, 6LowPAN,” [Online].
[Accessed 07 01 2019].

[Accessed 22 01 2019].

[Accessed 22 01 2019].

[19] “IETF, IPv6 over 802.15.4,” [Online].


[Accessed 07 01 2019].

[Accessed 07 01 2019].

[Accessed 07 01 2019].

Available: https://www.nordicsemi.com/eng/Products/nRF5-SDK-for-Thread.
[Accessed 07 01 2019].

[34] “OpenThread, GitHub,” [Online].
[Accessed 07 01 2019].

[Accessed 07 01 2017].

[Accessed 07 01 2019].

[37] “PuTTY,” [Online].
Available: https://www.putty.org/.
[Accessed 07 01 2019].

[38] “Nordicsemi Infocenter, Thread examples,” [Online].
[Accessed 07 01 2019].

[40] “Openthread Github, wpantund,” [Online].
[Accessed 07 01 2019].

[41] “Nordicsemi, Thread examples,” [Online].
index.jsp?topic=%2Fcom.nordic.infocenter.thread_zigbee.v2.0.0%2Findex.html.
[Accessed 07 01 2019].

[Accessed 07 01 2019].

Available: https://wiki.ubuntu.com/BionicBeaver/
[Accessed 07 01 2019].

[44] “GitHub, OpenThread CLI,” [Online].
Available: https://github.com/
openthread/openthread/blob/master/src/cli/README.md.
[Accessed 07 01 2019].

[Accessed 07 01 2019].

Available: https://golang.org/.
[Accessed 07 01 2019].

Available: https://jpgraph.net/
[Accessed 07 01 2019].

Available: https://electronut.in/nrf52840-thread/.
[Accessed 07 01 2019].

[Accessed 07 01 2019].

[Accessed 22 01 2019].

[Accessed 22 01 2019].

[52] “Silicon Labs, Thread fundamentals,” [Online].
[Accessed 07 01 2019].

[Accessed 07 01 2019].

[Accessed 07 01 2019].