

Evaluating Thread protocol in the framework of Matter

Andreu Ortega Blasi

Master in Advanced Telecommunications, Internet of Things specialization,
at Universitat Politècnica de Catalunya

Supervisor: Josep Paradells Aspas

Universitat Politècnica de Catalunya
September 2021



Abstract

Internet of Things is a technology which produced one of the biggest, and with most impact on the society, change in connectivity and automation solutions. It has a lot of new features and advantages, but also has its constraints. For this purpose, many standard have been developed, and IoT solutions designed.

A Home Automation system is an application of IoT. It consist of IP-enabled embedded devices connected to the Internet using IPv6. The technology improved when IETF designed 6LoWPAN as a interface to link IPv6 to IEEE 802.15.4, a low-power wireless network standard

Untill now, the IoT market is very dispersed with many solutions based in different standards, which make them incompatible between themselves. Appart from that, Home Automation technologies and solutions do not meet the requirements of low power, IP-based, security and friendly use. In order to profit from, and accelerate it, the IoT technology emergence, a unification of the standards used and the requirements needed by the different IoT solutions must be designed.

The goal of this thesis was to study a new Home Automation standard called Matter. Matter is an application layer standard which aims to simplify the development for manufacturers and increase compatibility for the consumers, enabling communication across smart home devices, mobile apps, and cloud services. As Matter was to be released in the summer of 2021 but was delayed, instead of that, the Thread architecture has been studied. Thread is, with Wi-Fi, the core of the operational communications supported by Matter.

Thread is a simplified, IPv6-based mesh networking architecture developed for efficient communication between energy-constrained devices around the home. As Thread is a royalty-free but closed-documentation standard, Nest Labs developed an open source implementation Called OpenThread.

This study is composed by an overview of the architecture, a detailed explanation of each layer of the stack, and a implementation of the network through OpenThread. Furthermore, some test will be evaluated to provide to the reader an introduction to some of the Thread functionalities, specially related with routing.

Resumen

El Internet de las Cosas (IoT) es una tecnología que ha provocado uno de los mayores cambios, y con mayor impacto en la sociedad, en cuanto a soluciones de conectividad y automatización. Tiene muchas nuevas características y ventajas, pero también sus limitaciones. Para ello, se han desarrollado muchos estándares y se han diseñado diversas soluciones.

Un sistema de automatización del hogar es una aplicación de IoT. Consiste en dispositivos incrustados compatibles con IP conectados a Internet mediante IPv6. La tecnología mejoró cuando IETF diseñó 6LoWPAN como interfaz para enlazar IPv6 a IEEE 802.15.4, un estándar para redes inalámbricas de bajo consumo.

Hasta ahora, el mercado IoT está muy disperso con muchas soluciones basadas en distintos estándares, que las hacen incompatibles entre sí. Aparte de esto, las tecnologías y soluciones de automatización del hogar no cumplen los requisitos de bajo consumo, basado en IP, seguridad y de fácil uso. Con el fin de aprovechar y acelerar la emergencia de la tecnología IoT, debe unificarse los estándares utilizados y los requisitos que necesitan las diferentes soluciones IoT.

El objetivo de esta tesis era estudiar un nuevo estándar de automatización del hogar llamado Matter. Matter es un estándar de la capa de aplicación que tiene como en la nube. Como Matter debía lanzarse en verano del 2021 pero se retrasó, en su lugar, se ha estudiado la arquitectura Thread. Thread es, con Wi-Fi, el núcleo de las comunicaciones operativas soportadas por Matter.

Thread es una arquitectura de red de malla simplificada basada en IPv6 desarrollada para una eficiente comunicación entre dispositivos con poca energía en casa. Dado que Thread es un estándar de documentación sin derechos de autor pero cerrado, Nest Labs desarrolló una implementación de código abierto llamada OpenThread.

Este estudio está compuesto por una descripción general de la arquitectura, una explicación detallada de cada capa de la pila y una implementación de la red mediante OpenThread. Además, se evaluará alguna prueba para ofrecer al lector una introducción a algunas de las funcionalidades de Thread, especialmente relacionadas con el enrutamiento.

Resum

L'Internet de les Coses (IoT) és una tecnologia que ha provocat un dels canvis més grans, i amb més impacte a la societat, en quant a solucions de connectivitat i automatització. Té moltes noves característiques i avantatges, però també té les seves limitacions. Amb aquesta finalitat, s'han desenvolupat molts estàndards i s'han dissenyat diverses solucions.

Un sistema d'automatització de la llar és una aplicació d'IoT. Consisteix en dispositius incrustats compatibles amb IP connectats a Internet mitjançant IPv6. La tecnologia va millorar quan IETF va dissenyar 6LoWPAN com a interfície per enllaçar IPv6 a IEEE 802.15.4, un estàndard per a xarxes sense fil de baix consum.

Fins ara, el mercat IoT està molt dispers amb moltes solucions basades en diferents estàndards, que les fan incompatibles entre elles. A part d'això, les tecnologies i solucions d'automatització de la llar no compleixen els requisits de baix consum, basat en IP, seguretat i de fàcil ús. Per tal d'aprofitar i accelerar l'emergència de la tecnologia IoT, s'ha d'unificar els estàndards utilitzats i els requisits que necessiten les diferents solucions IoT.

L'objectiu d'aquesta tesi era estudiar un nou estàndard d'automatització de la llar anomenat Matter. Matter és un estàndard de la capa d'aplicació que té com en el núvol. Com Matter s'havia de llançar a l'estiu del 2021 però es va retardar, en comptes d'això, s'ha estudiat l'arquitectura Thread. Thread és, amb Wi-Fi, el nucli de les comunicacions operatives suportades per Matter.

Thread és una arquitectura de xarxa de malla simplificada basada en IPv6 desenvolupada per a una comunicació eficient entre dispositius amb poca energia a casa. Com que Thread és un estàndard de documentació sense drets d'autor però tancat, Nest Labs va desenvolupar una implementació de codi obert anomenada OpenThread.

Aquest estudi està compost per una descripció general de l'arquitectura, una explicació detallada de cada capa de la pila i una implementació de la xarxa mitjançant OpenThread. A més, s'avaluarà alguna prova per oferir al lector una introducció a algunes de les funcionalitats de Thread, especialment relacionades amb l'enrutament.

Acknowledgements

First and foremost I would like to thank my supervisor Josep Paradells, for advising and helping me with the thesis. He helped me to obtain the devices for testing and gave me documentation to fulfill all the studies done. Also I would like to thank my University for being the promotor of this thesis and provided me easy acces to the documentation.

I would like to thank as well to some professors such as Ramon Bragós or Ramon Morros for helping me in finding the right choice when deciding the thesis.

Also I would like to thank to my friends and family. In special I would like to thank my parents, who supported me to do this thesis while being at home, which was my lab as well, during this strange pandemic times.

Contents

Abstract	I
Resumen	II
Resum	III
Acknowledgements	IV
Contents	VI
List of figures	VIII
List of tables	IX
1. Motivation	1
1.1. Matter	1
2. Thread	4
2.1. Introduction to Thread	4
2.2. Technical Overview	5
2.3. Thread Network Topology	6
2.4. Physical Layer and Data Link Layer: IEEE 802.15.4	9
2.4.1. Physical Link Layer	9
2.4.2. Data Link Layer	11
2.5. Network Layer	11
2.5.1. IPv6	11
2.5.2. 6LoWPAN	14
2.5.3. Routing Protocol: RIP and RIPng	16
2.5.4. MLE messaging	18
2.5.5. DTLS	22
2.6. Transport Layer: UDP and TCP	23
2.7. Application Layer	24
3. Openthread	25
3.1. Network Implementation	27
3.2. Routing layer implementation	28
3.3. DTLS implementation	30
3.4. Application layer implementation: CoAP	30

3.5. CLI Commnads	31
4. Hardware test environment	38
4.1. Zolertia devices	38
4.1.1. Zolertia Re-Mote	38
4.1.2. Zolertia Firefly	40
4.2. TI CC2531EMK Sniffer Tool: ICQUANZX CC2531EMK	42
5. Setup, tests and performances	43
5.1. Test Scenario installation and Configuration	43
5.1.1. Zolertia devices configuration	43
5.1.2. Configuring the sniffer	46
5.2. Tests	47
5.2.1. Set up a Thread Network	47
5.2.2. Network Interaction	49
5.2.3. Visibility of the network	56
5.2.4. End to end connectivity	57
6. Conclusions	60
6.1. General Conclusion	60
6.2. Future work	61
6.2.1. Thread	61
6.2.2. OpenThread	61
6.2.3. This thesis	62

List of Figures

1.1.	Architecture Overview	1
1.2.	Matter detailed functionalities	2
2.1.	Thread Members	4
2.2.	Thread stack layer levels	6
2.3.	Thread stack layer levels	7
2.4.	802.15.4 frame	9
2.5.	802.15.4 PHY table summary	10
2.6.	802.15.4 PHY table summary	11
2.7.	IPv6 frame	12
2.8.	IPv6 newtwork example	13
2.9.	General Format of a 6LoWPAN Packet	14
2.10.	6LoWPAN headers	15
2.11.	6LoWPAN Packet Containing IPv6 Payload with Compressed IPv6 Header	15
2.12.	6LoWPAN Packet Containing Mesh Hader for Layer 2 Forwarding, a Fragmentation Header and a Compression Header	16
2.13.	6LoWPAN Packet Representing Subsequent Fragments that Do Not Contain Any Information about the IPv6 Header	16
2.14.	RIPng frame format	17
2.15.	MLE frame format	18
2.16.	TLV frame format	20
2.17.	Extended TLV frame format	20
2.18.	DTLS Timer Basic Concept	23
3.1.	SoC architecture	25
3.2.	Overview of the Openthread Modules	27
3.3.	Overview of the implemented IPv6 Module	27
3.4.	Overview of the implemented MLE Module	28
3.5.	Number of child defined in the Openthread API	28
3.6.	Some router parameters in the Openthread API	29
3.7.	Link Quality definition in the Openthread API	29
3.8.	MLE command types	29
3.9.	Overview of the DTLS module	30
3.10.	Example of the bufferinfo command output	31
3.11.	Example of the channel command output	31
3.12.	Example of the counters command output	32
3.13.	Example of the counters <name> command output	33

3.14. Example of the discover command output	34
4.1. Zolertia Re-Mote	39
4.2. Zolertia Firefly	40
4.3. Zolertia Firefly connected by its usb adaptor	41
4.4. ICQUANZX CC2531EMK kit	42
5.1. Code to compile the CC2538 example	44
5.2. Code to compile the CC2538 example	44
5.3. Window asking where to connect the plugged device	44
5.4. Command to flash the image into the device	44
5.5. Image flashing process	45
5.6. Command to start the cli console	45
5.7. Serial port setup configuration	45
5.8. Configuration window for adjusting the sniffing parameters	46
5.9. Additional commands to set up the pip	46
5.10. Decryption key configuration	47
5.11. Dataset creation and submission	48
5.12. Ipv6 interface and Thread deployment commands	48
5.13. Procedure to attach a new device into an existing Thread network .	49
5.14. Discovery command output	49
5.15. Leader device multicasting the network information	49
5.16. Details of the information provided by the MLE Advertisement mes- sage	50
5.17. Details of the initial MLE Link Request message	50
5.18. Details of the information provided by the MLE Link request . . .	50
5.19. Process of attachment, initial interaction	51
5.20. Details of the TLVs provided by the MLE Parent Resonse	51
5.21. Details of the TLVs provided by the MLE Child Id Request	52
5.22. Process of attachment, the parent allocates the new device into the network as its child	53
5.23. Details of the new routing table of the leader, through a MLE Ad- vertisement message	54
5.24. Overview of the attachment of the 3rd device (1)	54
5.25. Overview of the attachment of the 3rd device (2)	55
5.26. Routing table of the leader broadcasted in a MLE Advertisement message	55
5.27. Routing table of the leader broadcasted in a MLE Advertisement message	56
5.28. Frames captured by the sniffer without knowing the decryption key	57
5.29. Leader Routing Table and ping example	58
5.30. Ping process	58
5.31. Leader Routing Table and ping example	59
5.32. Leader Routing Table	59
5.33. Ping command from Leader to Router	59

List of Tables

- 2.1. Overview of the Thread Specification 6
- 2.2. MLE Commands description table 19
- 2.3. TLV value field description table 22

Chapter 1

Motivation

1.1. Matter

As already mentioned, the Home Automation technology needs a unification of the standards used and the requirements needed. Moreover it should be able to use Internet to access to a big ammount of services already available there. For this purpose, new architectures and standards have appeared in the lasts years.

The goal of this thesis was to study a new application layer protocol (fig 1.1) called Matter (formely Project Connected Home over IP, or Project CHIP) [2], which aims to enable communication across smart home devices, mobile apps, and cloud services and to define a specific set of IP-based networking technologies for device certification. It is developed by the Connectivity Standard Alliance (CSA, formely Zigbee Alliance) working group and their plan is to make it a royalty-free standard. At January 2020, the CSA opened the Matter Working Group and started the process of drafting Matter.

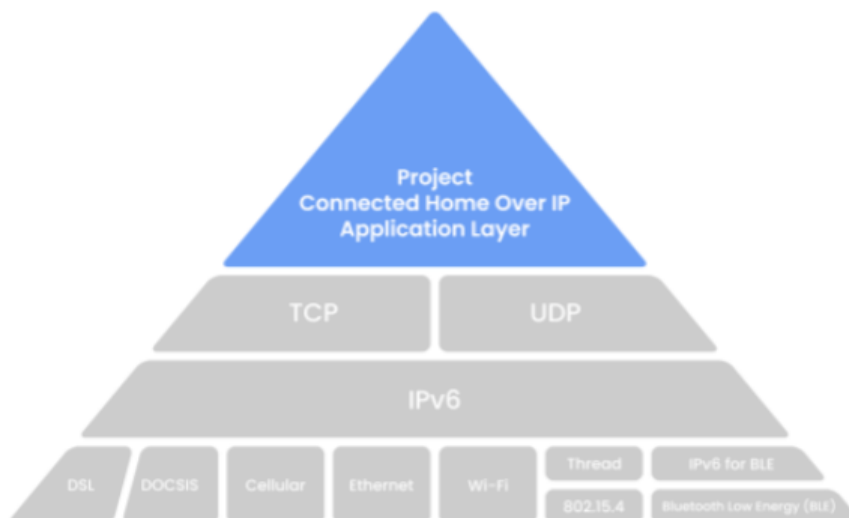


Figure 1.1: Architecture Overview

Matter will be deployed on devices and controllers to help achieve the interoperability architectural goal. It will initially support Wi-Fi and Thread for core, operational communications and Bluetooth Low Energy (BLE) will be used for device commissioning and setup.

The application layer can be broken down into seven main features (fig 1.2) Which will be detailed in a list below [1].



Figure 1.2: Matter detailed functionalities

- **Application:** High order business logic of a device. It will be mainly related with the functionalities the device provides.
- **Data Model:** Structured by the various functionalities of the devices.
- **Interaction Model:** Represents a set of actions that can be performed on the devices to interact with it. These actions operate on the structures defined by the data model.
- **Action Framing:** Once an action is constructed using the Interaction Model, it is framed into a prescriptive packed binary format to enable being well represented over the different devices of the network.
- **Security:** An encoded action frame is then sent down to the Security Layer to encrypt and sign the payload to ensure that data is secured and authenticated by both sender and receiver of a packet.
- **Message Framing and Routing:** the Message Layer constructs the payload format with required and optional header fields; which specify properties of the message as well as some routing information.
- **IP Framing and Transport Management:** Finally, the frame is sent to the underlying transport protocol for IP management of the data.

The plan of CSA was to have it released by the summer of 2021, but it was delayed. As it is not released yet and the documentation is not still ready, I had to change the thesis goal. As the routing architecture used by Matter is Thread (the other that can be used is Wi-Fi, which is already widely known), the goal will be changed to study Thread. Furthermore, as I am studying the routing architecture, the network messaging can be studied using software which I am already familiar, such Wireshark.

Chapter 2

Thread

2.1. Introduction to Thread

This chapter introduces the Thread networking architecture. Not only there is described an overview of it, but also a deeper insight into the implementation and technical details will be provided here.

The Thread protocol was created by a group, formed in 2014, of allied companies, called the "Thread Group". This alliance was composed by the following companies (fig 2.1): ARM Holdings, Big Ass Solutions, NXP Semiconductors/Freescale, Google-subsidiary Nest Labs, OSRAM, Samsung, Silicon Labs, Somfy, Tyco International, Qualcomm, and the Yale lock company. Then it has grown adding as members some of the bigger companies in the world such as Intel or Amazon.



Figure 2.1: Thread Members

The Thread architecture [7] is an open standard for wireless communication providing a native IP solution for reliable, low-power, secure, device-to-device, application agnostic communication. It is the premier IPv6 based solution running on existing and broadly supported IEEE 802.15.4 radio technology. For this reason, ZigBee and other application protocols can run over Thread networks

making it suitable for many IoT devices.

Thread networks use 6LoWPAN to be able to convert IPv6 packets into IEEE 802.15.4 frames. Therefore, Thread devices are IP-addressable with cloud access and a protection based in a AES-128 encryption with replay protection. It always must be protected, there is no way to deploy a Thread network without security.

2.2. Technical Overview

The protocol has some main features, documented in the specifications file [8], to be highlighted:

- It provides a simple network installation, start up and operation: The simple protocols for forming, joining, and maintaining Thread Networks, which will be discussed later, allow systems to self-configure, dynamically optimize and heal.
- Secure: devices cannot join a Thread Network unless they are authorized and have the credentials, and all network communication is encrypted and secure.
- Small and large networks: the thread network layer is designed to optimize the network operation based on the expected use, for a home automation, the network will be smaller but in commercial networks the number of devices can reach up to thousands.
- Range: the network covers a sufficient range, with typical devices, to cover a normal home. The Backbone Border Routers (BBRs) unify various Thread networks into a single IPv6 subnet for the commercial market. Spread spectrum technology is used at the physical layer to provide good immunity to interference.
- No single point of failure: the protocol is designed to provide to the network auto-configuring and self-healing, so will continue to provide secure and reliable communication even if individual devices fail.
- Low power: using suitable duty cycles, the host devices can operate for several years.
- Built on open and proven standards: the Thread specification uses well defined standards from the IEEE and IETF which make it transparent and with information accessible for everyone.
- Application-layer agnostic: Thread is a networking layer solution based on IPv6. Any low bandwidth application layer that can run over IPv6 can run over Thread, and multiple application layers can share the same network.

The following table describes the main technical aspects of the Thread Network:

Specification	Thread Data
Topology	Mesh Network
Range	About 20 to 30 meters
Max Devices connected	32 Routers with up to 511 end devices per router
Operating band	2.4GHz (The ISM unlicensed band)
Spread Spectrum	Direct sequence spread spectruim (DSSS)
Throughput	250kbps
Security	AES-128 encryption (Private Key cryptography)
Modulation	O-QPSK

Table 2.1: Overview of the Thread Specification

The Thread stack only defines the Network and Transport layers, as shown in the figure 2.2 and relies in the IEEE 802.15.4 protocol for the Physical and Mac layers. Then, applications such as Zigbee can work on top of the Thread network.

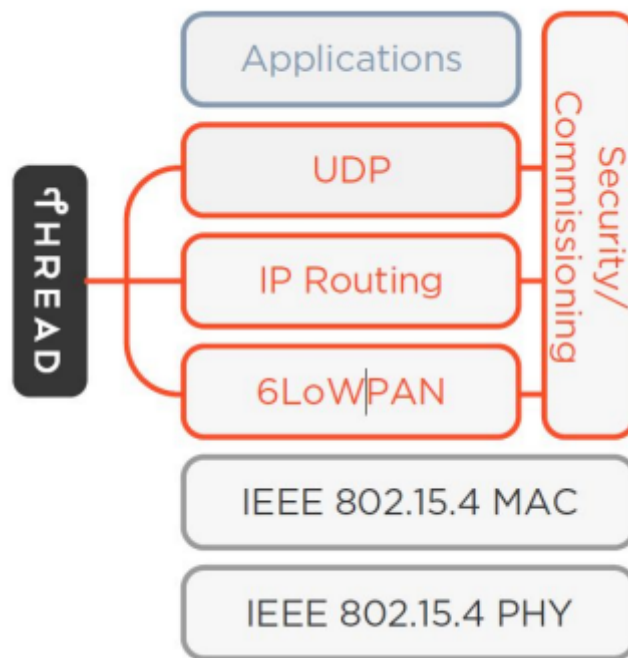


Figure 2.2: Thread stack layer levels

2.3. Thread Network Topology

As described previously, the Thread network has a mesh topology, where the Thread users can communicate with it from their own electronic devices (computer, tablet or smartphone) via Wi-Fi or using cloud-based applications.

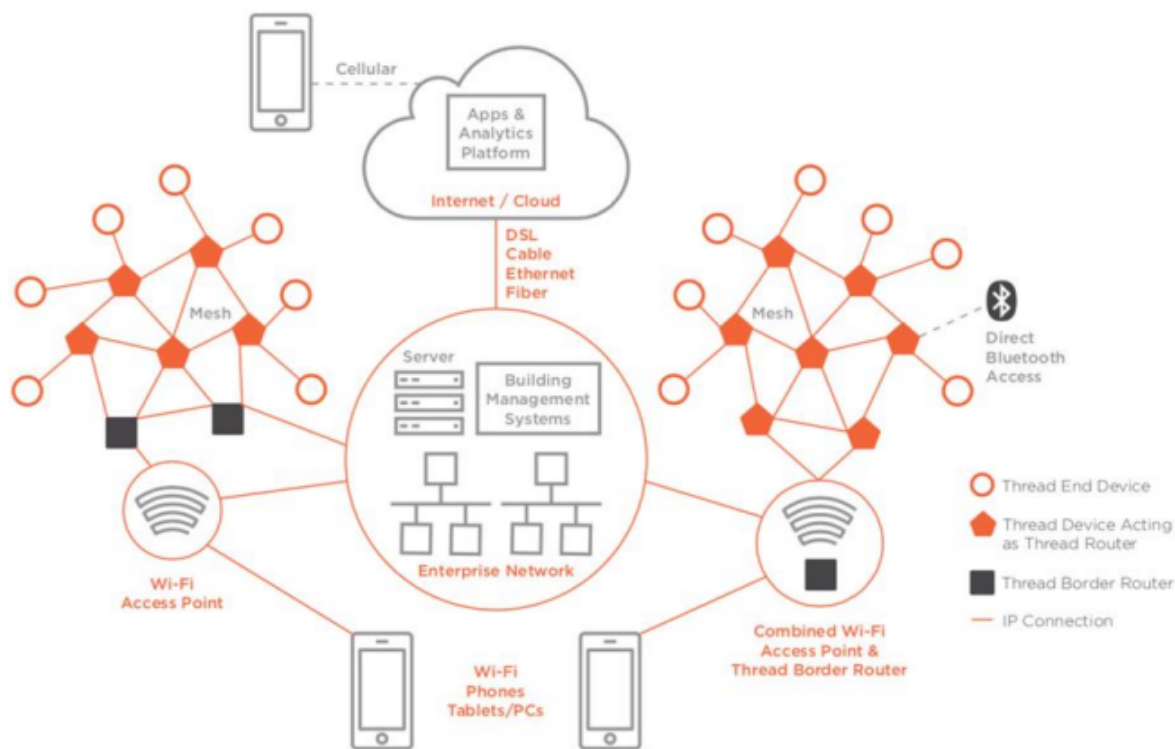


Figure 2.3: Thread stack layer levels

The network contains different types of devices [8], each of them having their specific role to obtain the desired performance. The following list describes all of them:

- **Border Routers:** they provide connectivity from the Thread network to adjacent network on other physical layers (for example, Wi-Fi or Ethernet). They are gateways which handle the connection between Thread networks and non-Thread networks. Border routers provide services for devices within the Thread network. There may be several Border Routers in one Thread network.
- **Thread Router:** it provides routing services to the Thread devices in the network. They also provide joining and security services for devices trying to join the network. These devices are not designed to sleep and they can be downgraded to REEDs (Router-eligible End Devices, which will be discussed later).
- **Leader:** this role is an additional, elected, one of one router in a Thread network. The leader takes certain decisions such as allowing REEDs to upgrade to router. If the Leader of the network fails, another router will be dynamically selected.
- **Router-eligible End Device (REED):** they have the capability to become Routers but due to the network topology or conditions they are not acting as

Routers. REEDs do not forward messages neither provide joining or security services for other devices in the network. The network manages and promotes router-eligible devices to routers if necessary without user interaction.

- End Device: they are similar to REEDs but they do not have the capability to be promoted to a Router. They can be either full end devices (FEDs) or minimal end devices (MEDs) MEDs do not need to explicitly synchronize with their parent to communicate. Sleepy end devices communicate only through their Thread Router parent and cannot relay messages for other devices.
- Sleepy End Device (SED): they communicate only through their Parent Router and cannot forward messages for other devices. It has its radio turned off during idle periods and wakes periodically to communicate with its parent.
- Synchronized Sleepy End Device (SSED): as SEDs, they cannot forward messages for other devices. It has its radio turned off during idle periods and wakes periodically to communicate with its parent.
- Bluetooth End Device (BED): they communicate only through their Parent Router, which is a Bluetooth LE Bridge Router. Unlike other Thread devices, these communicate over a Bluetooth Low Energy link, and not IEEE 802.15.4.

The newest version of the protocol, Thread 1.2, provided a new capability to control directly using non-IP technologies like Bluetooth, through the Bluetooth LE Bridge Routers, the BEEDs. These devices can be operated independently from the enterprise network. When these devices are controlled via Bluetooth, its new status is immediately available to other Thread devices or the rest of the enterprise network, as they still form a part of the overall IPv6 infrastructure.

The sleepy devices spend most of their time in sleep mode, with duty cycles which let them operate in low power mode. They can just communicate to the parent router and do not forward messages to other nodes. The parents must hold the messages until the devices wake up to poll for data or send it. Their duty cycle for these devices can be summarized in a few points:

- Wake up from sleep mode.
- Perform the startup operations and the radio initialization.
- They start in receive mode and check if they are clear to transmit.
- When they are able, they change to transmit mode and they send the data.
- They wait to receive the acknowledgment of their parent.
- Finally, they go back to sleep mode.

2.4. Physical Layer and Data Link Layer: IEEE 802.15.4

Thread stack PHY and MAC layer can use both the IEEE 802.15.4-2006 and IEEE 802.15.4-2015 versions of the specification. The IEEE 802.15.4 was a standard developed for low-rate control and monitoring applications, which first version was released in 2003. It defines the physical layer (PHY) and medium access control (MAC) layer specifications for low data rate wireless personal area networks (LR-WPANS).

These networks are commonly limited to a low range, more or less 10 meters, and require a very basic infrastructure. Their nodes are low in complexity, energy constrained and their data rate demand is low (for example, wireless sensors).

In order to achieve the desired performance in LR-WPANS, the IEEE 802.15.4 protocol uses a very small, 127 bytes at the PHY layer, frame in order to limit the Bit Error Rate (BER) when using energy constrained devices.

Then, the MAC layer payload can be at the lowest 88 bytes, depending on the security options and addressing type as illustrated in 2.4:

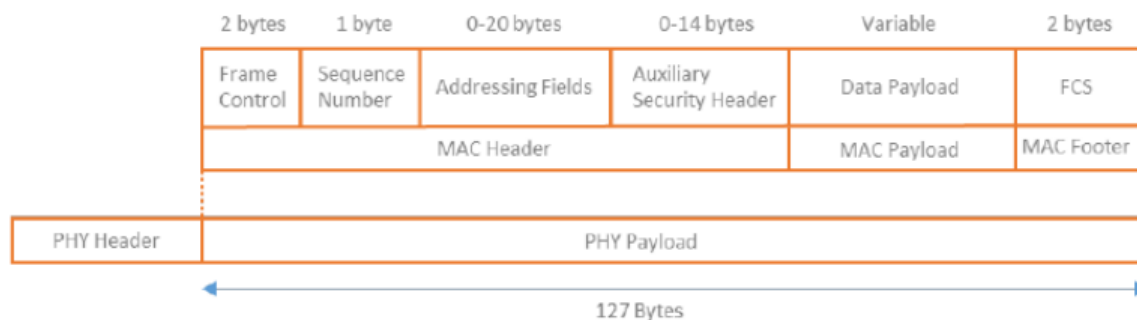


Figure 2.4: 802.15.4 frame

2.4.1. Physical Link Layer

IEEE 802.15.4 PHY provides the interface between the Data Link layer and the physical radio channel. The general characteristics of this standard is provided in the following table:

Frequency band	Number of channels	Spreading technique	Modulation	Symbol rate per channel (kbaud)	Bit rate per channel (kbps)
868 MHz	1	Binary DSSS	BPSK	20	20
915 MHz	10	Binary DSSS	BPSK	40	40
2.4 GHz	16	16-array DSSS	O-QPSK	62.5	250

Figure 2.5: 802.15.4 PHY table summary

Specifically, in the Thread specification, IEEE 802.15.4 PHY operates in the 2.4 GHz band, that can be used worldwide. The whole 2.4 GHz band is composed of different partitions, called channels (from 2.4 GHz to 2.483 GHz, each one in a higher band). The channels are numbered from channel #11 channel #26, as the initial 11 channels (0-10) are occupied by the other 802.15.4 Phy interfaces operating in the other existing bands.

There are two main services provided: the PHY data transmission service, enabling transmissions and receptions of PHY protocol data units (PPDUs) across the physical radio channel; and the interface for the PHY layer management entity (PMLE) which offers access to every physical layer management function and maintains a database of information on related personal area networks (Service Access Point, SAP). The standard also supports important functions like Energy Detection, Link Quality Indicator, channel selection or Clear Channel Assessment. Each PPDU, contains a preamble field, which designed for acquisition of symbol and chip timing, a packet length field, and a payload field, or PHY service data unit.

The IEEE 802.15.4 payload length can vary from 2 to 127 bytes and the PHY Header is composed by 4 bytes for the Preamble, 1 byte for the Start Packet Delimiter and 1 byte for Length Field.

The link quality is based on the link margin on incoming messages from the neighboring devices This incoming link margin is mapped to a link quality from 0 to 3. A value of 0 means unknown or infinite cost. The link margin is a measure of RSSI (Received Signal Strength Indication), a Signal to Noise Ratio based parameter that compares the received signal with the noise floor.

Table 2. Link Quality and Link Cost

Link Margin	Link Quality	Link Cost
None (~0 dB)	0	unknown / infinite
Poor	1	4
Reasonable	2	2
Good	3	1

Figure 2.6: 802.15.4 PHY table summary

2.4.2. Data Link Layer

The IEEE 802.15.4 MAC layer is used for basic message handling and congestion control. This MAC layer includes a CSMA-CA (Carrier Sense Multiple Access-Collision Avoidance) mechanism to allow multiple Thread devices to utilize the shared bandwidth by waiting for a clear channel before transmission, as well as a link layer to handle retries and acknowledgement of messages for reliable transmission of individual messages. Encryption, authentication and replay protection are also used to provide secure and reliable end-to-end communications in the network.

The features of the MAC sublayer are beacon management, channel access, Guaranteed Time Slots (GTS) management, frame validation, acknowledged frame delivery, association, and disassociation. In addition, the MAC sublayer provides hooks for implementing application-appropriate security mechanisms.

IEEE 802.15.4 MAC layer supports two modes:

- Beacon Mode: In this mode the PAN (Personal Area Network) coordinator generates beacons periodically to synchronize the nodes associated with it. Two beacons conform the superframe structure which is sent to the child.
- Non-Beacon Mode: In this mode, the data is transmitted using unslotted CSMA/CA algorithm. No beacons are generated, so the PHY layer now provides a higher network scalability but cannot guarantee the delivery in time of the data.

As the Thread networks topology is based on a mesh topology, for scalability purposes, it is used the Non-Beacon Mode.

2.5. Network Layer

2.5.1. IPv6

As already mentioned, Thread devices support IPv6 (Internet Protocol version 6) addressing architecture. It is the most recent version of the Internet Protocol,

developed by the IETF (Internet Engineering Task Force). Provides an identification and location system for all the devices which have internet connection and are suitable for that version. DHCPv6 is used for router address assignment.

The IPv6 frames are 40 bytes long and contains an address field of 128 bits, so this allow 2^{128} addresses, much more than what IPv4 and its 32 bits addresss field provided.

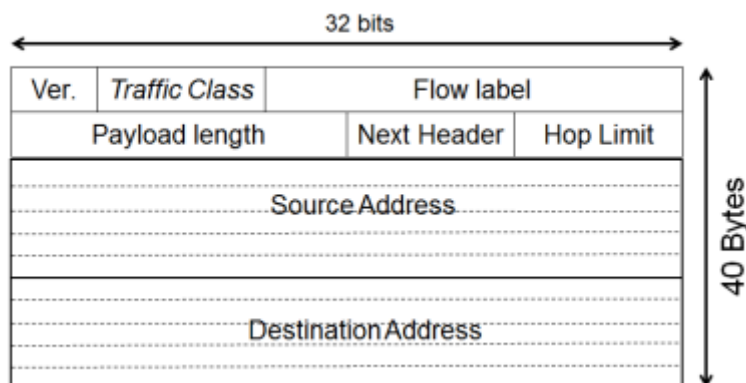


Figure 2.7: IPv6 frame

There are different type of IPv6 addresses which are classified with the following types:

- Unicast: used to send a packet from the source to a single destination. They are the commonest ones and we will talk more about them and their sub-classes.
- Multicast: used to send a packet from the source to several destinations. This is possible by means of multicast routing that enable packets to replicate in some places.
- Anycast: used to send a packet from the source to the nearest destination from a set of them.
- Reserved: Addresses or groups of them for special uses, for example addresses to be used on documentation and examples.

Then, there are some unicast addresses which the Thread users take profit from them:

- ULA (Unique Local Address): They are intended for local communications, usually inside a single site, they are not expected to be routable on the global Internet but used only inside a more limited environment.
- Global Unicast: Equivalent to the IPv4 public addresses, they are unique in the whole Internet and can be used to send a packet from one site to any destination in Internet.

Thread devices configure one or more ULAs or GUA addresses. They also support multicast addresses such as link-local all node multicast (for referring to all the nodes in a Thread network, with the notation "FF02::01") and link local all-router multicast (for referring to all the routers in a Thread network, with the notation "FF02::02").

The high-order bits of an IPv6 address are referred to the network and the rest are referred to the particular addresses in that network. Thus, all the addresses in one network have the same first N bits, which are called the "prefix". In IPv6 the prefix is 64 bits. The device starting the network picks a "/64" prefix which is used through all the network. This prefix is a Locally Assigned Global ID. The devices in the Thread networks use its Extended MAC address to derive its interface identifier and from this configure a link local IPv6 address with the prefix FE80::0/64.

Once the network is created, the thread can also contain one or more Border Routers which can have, optionally, a prefix to generate additional GUAs. Then, each end device joining the Thread network is assigned to a 16-bit short address. For Routers, this address is assigned using the high bits in the address field with the lower bits set to 0, indicating a Router address. Children are then allocated a 16-bit short address using their Parent's high bits and the appropriate lower bits for their address, allowing any other device in the Thread network to understand the Child's routing location simply by using the high bits of its address field.

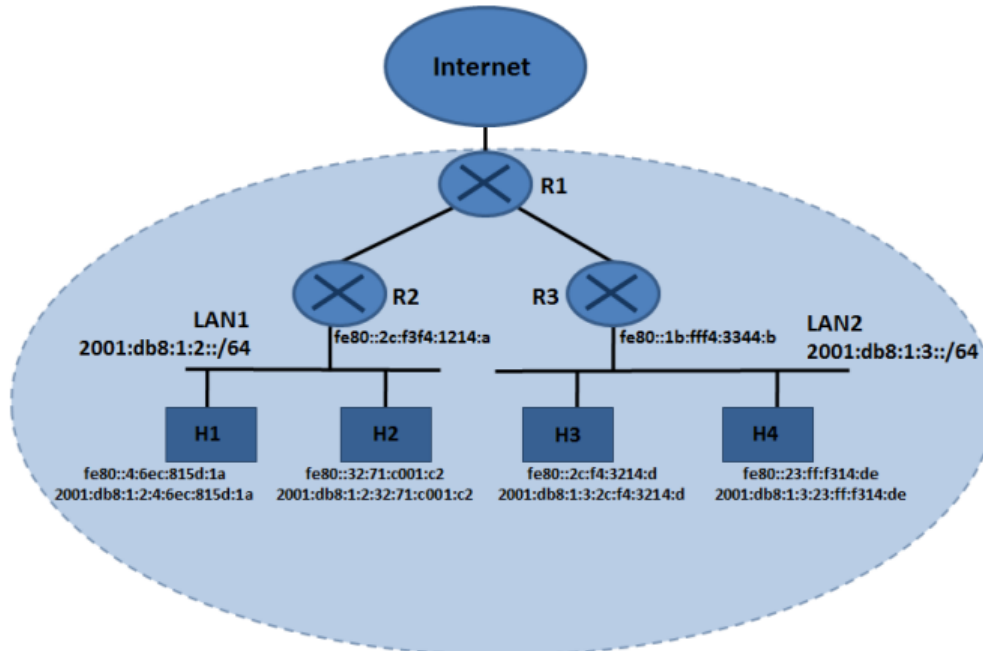


Figure 2.8: IPv6 newtwork example

Devices support the ICMPv6 (Internet Control Message Protocol version 6) protocol and ICMPv6 error messages, as well as the echo request (ping) and echo

reply messages.

Thread Devices use UDP (User Datagram Protocol) [13] for messaging between devices for mesh establishment and maintenance. Thread Networks also support TCP (or any other IPv6-based transport protocol) [3] for application layer communication.

2.5.2. 6LoWPAN

6LoWPAN defines de standard for IPv6 communications over the IEEE 802.15.4 standard. It acts as an adaptation layer between both standards. The data frames coming from internet have tipical size of 1280 bytes (IPv6 MTU). The 6LoWPAN sublayer provides a reliable way to adapt these big frames into smaller ones which are better fitted for energy constrained and basic devices.



Figure 2.9: General Format of a 6LoWPAN Packet

In order to adapt to the 127 reduced syze of the PHY 802.15.4 size (and usually just 81 bytes, which can be at maximum 102 bytes, for payload), 6LoWPAN provides a set of functionalities which will be detailed now:

- IPv6 packet fragmentation and reassembly: IPv6 specification establishes that the minimum MTU that a link layer should offer to the IPv6 layer is 1280 bytes. The protocol data units may be as small as 81 bytes in IEEE 802.15.4. For this purpose, a fragmentation of this packers is done. The fragmented packets are carried in frames containing the fragmentation header. Then, on the receiver, these packets are reassembled in order to obtain the original IPv6 frame. They also contain some specific fields for the reassembly process: it has a datagram size field for knowing the real size of the packet before fragmentation (as already mentioned, in IPv6 frames this would be 40 bytes); a datagram tag, common to all the fragmented frames, to recognize from which packet they were; and a datagram offset field that is used to order the fragments received. For this reason, it does not require to receive all the fragments in order.
- IPv6 header compression: as already exposed, the big IPv6 headers must be converted to smaller headers to minimize the overhead. 6LoWPAN offers a way to reduce the header's size which take advantage of cross-layer redundancies between protocols such as source and destination addressing, payload length, traffic class and flow labels. Thread utilizes IPHC (Improved Header Compression) for compressing IPv6 headers and NHC(Next Header Compression) for UDP headers. The 40 bytes size of the IPv6 header are compressed down to 2 bytes and the8 bytes UDP header is compressed

down to 4 bytes. The frames are marked with a field to identify if it has been compressed or not.

- Link layer packet forwarding: Thread uses IP routing to forward packets. The IP routing table is maintained with each destination and the next hop to it. The 6LoWPAN mesh header is used to do link level next hop forwarding based on the IP routing table information. Thread uses 6LoWPAN mesh headers for next hop forwarding. These nodes do not require to have reachability in order to communicate because the mesh headers have an "Originator" field and a "Final Destination" field and each node looks in their table which is the way to achieve the destination hop by hop. Each header in the header stack contains a header type followed by zero or more header fields. When more than one LoWPAN header is used in the same packet they must appear in the following order: Mesh Addressing Header, Broadcast Header, and Fragmentation Header. This order must be followed always.

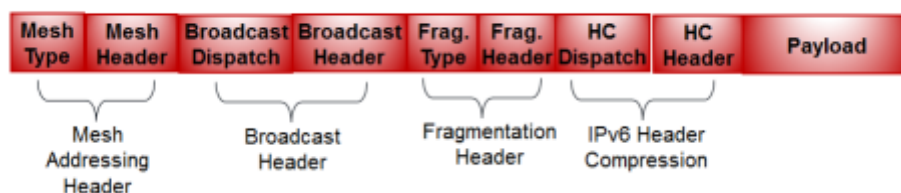


Figure 2.10: 6LoWPAN headers

From all of them, the Thread standard uses the following types of 6LoWPAN headers:

- Mesh Header: used for link layer forwarding.
- Fragmentation Header: used for fragmenting the IPv6 packet into several 6LoWPAN packets.
- Header Compression Header: used for IPv6 headers compression.

The following are examples of 6LoWPAN packets sent over the air:



Figure 2.11: 6LoWPAN Packet Containing IPv6 Payload with Compressed IPv6 Header

In the figure above (2.11), the 6LoWPAN payload is composed of the compressed IPv6 header and the rest of the IPv6 payload



Figure 2.12: 6LoWPAN Packet Containing Mesh Hader for Layer 2 Forwarding, a Fragmentation Header and a Compression Header

In the figure above (2.12), the 6LoWPAN payload contains the IPv6 header and part of the IPv6 payload

The rest of the payload will be transmitted in subsequent packets perthe format in the following figure (2.13).



Figure 2.13: 6LoWPAN Packet Representing Subsequent Fragments that Do Not Contain Any Information about the IPv6 Header

2.5.3. Routing Protocol: RIP and RIPng

Thread networks' routing is made to be a shortest-path any-to-any routing. The Thread routing protocol is a simple distance vector routing protocol. The main goal of the protocol is ot maximize the amount of routing information that can be communicated in a single message. More precissely it uses a similar version of the Routing Information Protocol (RIP).

RIP came to be one of the earliest efforts in the field of dynamic routing protocols and dated to the 1970s. Later in the 1990s a new version, named RIPv2, was released defining as well the old version as RIPv1. Also in the mid-1990s, the process of defining IPv6 was drawing toward completion at least for the original IPv6 standards the RIPv3 or RIPng. RIP is one of the first routing protocols implemented on TCP/IP.

Routing information is sent through the network using UDP. Each router that uses this protocol has limited knowledge of the network around it. This simple protocol uses a hop count mechanism to find an optimal path for packet routing. A maximum number of 16 hops are used to avoid routing loops. However, this parameter limits the size (2^{16}) of the networks that this protocol can support. The popularity of this protocol is largely due to its simplicity and its easy configurability. However, its disadvantages include slow convergence times, and its scalability limitations. Therefore, this protocol works best for small scaled networks. Simple distance vector algorithm:

- Provides next-hop information about all router nodes.
- Highly compressed protocol format: one byte per destination packets.
- No reactive route discovery by devices.

- Child ID encodes parent router ID. Route is known when address is known.
- Point to point routes always available to every router.

The frame format can be observed in the next figure 2.14:

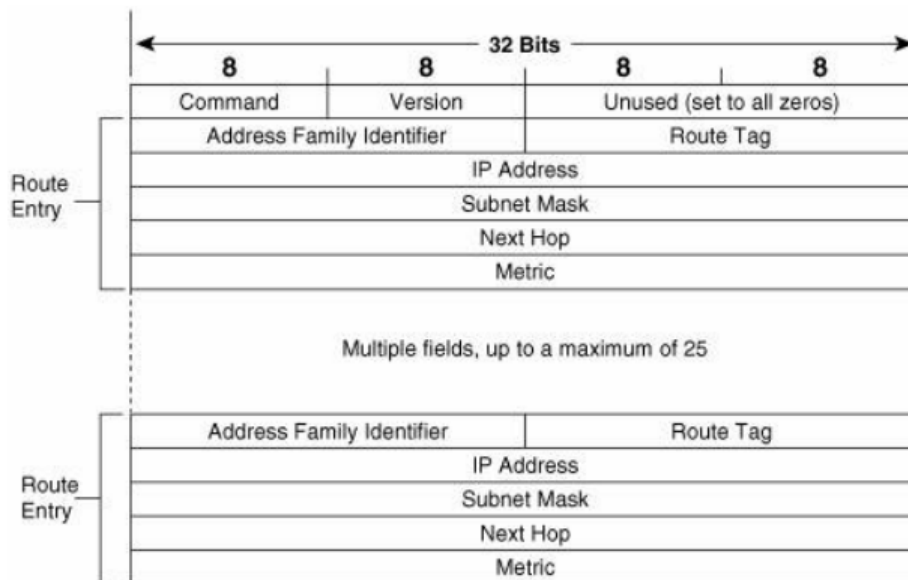


Figure 2.14: RIPng frame format

Thread uses a distance-vector routing protocol similar to RIPng, but with very compact message formats. All Routers exchange with other Routers their cost of routing to other Routers in the Thread Network in a compressed format using MLE (Mesh Link Establishment) messages.

In the Thread networks, there are some guidelines for the router selection:

- There is a limit of 32 active routers to reduce bandwidth and RAM consumption.
- There are 64 router addresses to allow timing out and reassignment.
- REED behave as end devices, but listen to routing messages.
- Routers are selected automatically from router eligible end devices (REED). They can be downgraded to end devices if the number of routers increase.

And then the Leader decisions in a network are (which are chosen dynamically and autonomously):

- Assign router IDs.
- Assign 6LoWPAN contexts.
- Collate border router information.

- Assembled network data is distributed using MLE advertisements.
- All routers store the network data, only the leader can make changes to it.

2.5.4. MLE messaging

MLE operates below the routing layer. MLE messages are used for establishing and configuring secure radio links, detecting neighboring devices, and maintaining routing costs between devices in the Thread Network. MLE is also used to distribute configuration values that are shared across the Thread Network such as the channel and the Personal Area Network ID (PAN ID). These messages are forwarded with controlled flooding as specified by Multicast Protocol for Low-power and Lossy Networks (MPL).

MLE messages consider an asymmetric link costs when linking devices. Asymmetric link qualities are common in IEEE 802.15.4 networks. To ensure two-way messaging is reliable, as the quality in one-way can be significantly different to the other, Thread Devices consider bidirectional link quality.

Their format, which can be observed in fig ?? is the following: a byte that indicates if they are secured ("0") or not ("255"), then if it is secured it includes an Auxiliary Header, a Command and the Message Integrity Code (MIC). If it is not secured, then it just includes the Command field. The Auxiliary header contains information required for security processing, including a Security Control field, a Frame Counter field, and a Key Identifier field. The MIC field is used for authentication.

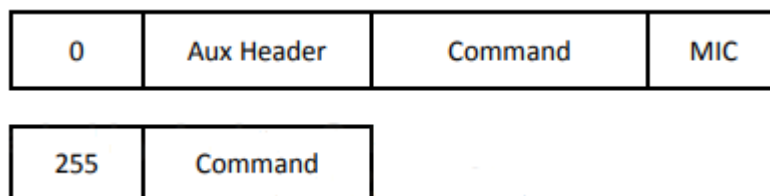


Figure 2.15: MLE frame format

The command field consists of a command type and some TLV (Time-Length-Value) fields in order to achieve the goal of each functionality. The different command types will be described in the table below:

Command Type (identifier)	Definition
Link Request (0)	A request to establish a link to a neighbor.
Link Accept (1)	Accept a requested link.
Link Accept and Request (2)	Accept a requested link and request a link with the sender of the original request.
Link Reject (3)	Reject a link request.
Advertisement (4)	Inform neighbors of network information and a device's link state.
Update (5)	Not used in Thread Networks.
Update Request (6)	Not used in Thread Networks.
Data Request (7)	A request, typically containing a TLV Request TLV that indicates which TLV(s) are being requested.
Data Response (8)	A response to a request, containing whatever TLVs were requested.
Parent Request (9)	A multicast request used to find neighboring devices that can act as a Parent.
Parent Request (10)	Response to Parent Request, identifying a potential Parent.
Child ID Request (11)	Request for a Child ID sent by a device to a Router or Router-Eligible End Device (REED).
Child ID Response (12)	Response from a Router to a device assigning it a 16-bit network ID.
Child Update Request (13)	Request by Child to update parameters on Parent.
Child Update Response (14)	Response from Parent on Child request to update parameters.
Announce (15)	A multicast message used to notify neighboring devices of the Thread Network's current Channel, PAN ID, and Active Timestamp.
Discovery Request (16)	A multicast message used to discover networks.
Discovery Response (17)	Response from a device on the Thread Network to a Discovery request.

Table 2.2: MLE Commands description table

The TLV fields are parameters designed to provide a set of commands to use according to the routing features that Thread have. As the name itself tells, the TLV frame can be fragmented into an 1-byte Type field, which tells which is the TLV used; 1-byte Length field, which tells the Value field's length; and the bytes (usually 2 to 4 bytes) corresponding to the Value field, as seen in the figure 2.16. As the Length field is 8 bits long, it gives up to 254 bytes for the Value Field.

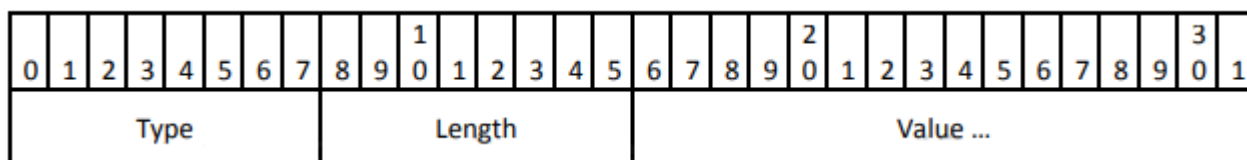


Figure 2.16: TLV frame format

There is also an extended format which allows for a higher length of the Value field as the Length field is 2 bytes long (2.17).

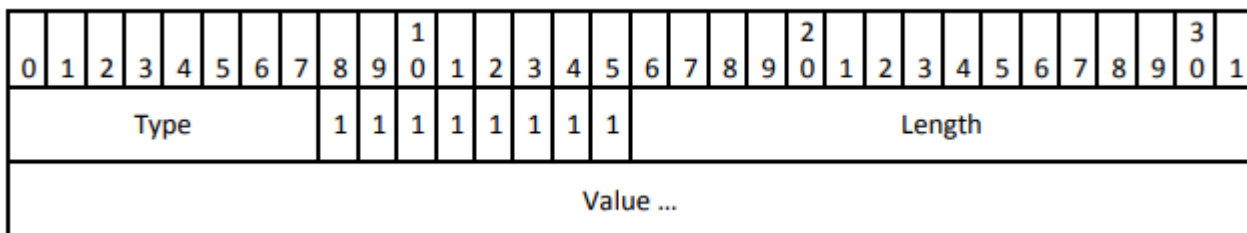


Figure 2.17: Extended TLV frame format

These commands are explained in the table below:

TLV (identifier)	Definition
Source Address TLV (0)	Contains the sender's 16 MAC address.
Mode TLV (1)	Contains a byte string representing the mode in which this link is used by the source of the message: Receiver, Secure, Device Type, Network Data.
Timeout TLV (2)	32-bit length field to fix the expected maximum interval between.
Challenge TLV (3)	A 4-byte field, at least, providing a random value which must be included in the reply MLE message to prove security against recorded messages.
Response TLV (4)	It has the byte string copied from a Challenge TLV.
Link-layer Frame Counter TLV (5)	Contains the sender's current outgoing link-layer Frame Counter, encoded as an 32-bit unsigned integer.
Link Quality TLV (6)	Not used in Thread Networks.
Network Parameter TLV (7)	Not used in Thread Networks.
Mle Frame Counter (8)	Contains the sender's current outgoing MLE Frame Counter, encoded as an 32-bit unsigned integer.

Route 64 TLV (9)	A multicast request used to find neighboring devices that can act as a Parent.
Address 16 TLV (10)	Contains a 16-bit MAC address. It is sent from a Parent to a new Child to assign it an address.
Leader Data TLV (11)	Contains the sender's current Network Leader Data.
Network Data TLV (12)	Contains the sender's current Network Data.
TLV Request TLV (13)	Contains a list of TLV codes that the sender is requesting.
Scan Mask TLV (14)	Contains flags that indicate the types of devices that are to respond to a multicast Request (Routers or End devices).
Connectivity TLV (15)	Shows how well is the sender connected to other Routers and Children.
Link Margin TLV (16)	Contains the sender's link margin in dB for the destination.
Status TLV (17)	Contains the status response to a request.
Version TLV (18)	Contains the version number of the Thread protocol implemented by the sender as an unsigned 16-bit integer.
Addresses Registration TLV (19)	Contains zero or more addresses that have been configured by the source of the MLE message that contains it. It is used for registration of valid unicast addresses.
Channel TLV (20)	Contains the channel page and channel of an adjacent Thread Network Partition operating on a different Active Operational Dataset.
PAN ID TLV (21)	Contains the PAN ID of an adjacent Thread Network Partition operating on a different Active Operational Dataset.
Active Timestamp TLV (22)	Contains an Active Timestamp.
Pending Timestamp TLV (23)	Contains a Pending Timestamp.
Active Operational Dataset TLV (24)	Contains the sender's Active Operational Dataset.
Pending Operational Dataset TLV (25)	Contains the sender's Pending Operational Dataset.

Thread Discovery TLV (26)	Contains a series of Mesh Commissioning TLVs for Thread network discovery on IEEE 802.15.4 interfaces.
---------------------------	--

Table 2.3: TLV value field description table

2.5.5. DTLS

The Datagram Transport Layer Security (DTLS) [5] is a communications protocol which provides security to datagram-based applications. It is designed to prevent eavesdropping, tampering, or message forgery. The DTLS protocol is based on the stream-oriented Transport Layer Security (TLS) protocol and is intended to provide similar security guarantees. The DTLS protocol datagram preserves the semantics of the underlying transport and the application does not suffer from the delays associated with stream protocols, but has to deal with packet reordering, loss of datagram and data larger than the size of a datagram network packet.

There are two main areas that unreliability creates problems for TLS:

- The traffic encryption layer does not allow individual packets to be decrypted, there are two inter-record dependencies:
 - The cryptographic context is chained between records.
 - A Message Authentication Code (MAC) that includes a sequence number provides anti-replay and message reordering protection, but this fields are implicit in the records.
- The handshake layer breaks if messages are lost because it depends on them being transmitted reliably for these two reasons:
 - It is a lockstep cryptographic handshake, which requires that the messages have to be transmitted and received in a defined order, causing a problem with potential reordering and message loss.
 - Fragmentation can be a problem because the handshake messages are potentially larger than any given datagram.

To solve the issue of packet loss DTLS employs a simple retransmission timer. This concept is illustrated in the figure 2.18. The client is expecting to see the HelloVerifyRequest message from the server. If the timer expires then the client knows that either the ClientHello or the HelloVerifyRequest was lost and retransmits.

Reordering is solved by giving each handshake message a specific sequence number used to determine if it has received the next message in the sequence. If the message is the next one then the peer processes it, otherwise it queues it up for future handling when message's individual sequence number is reached.

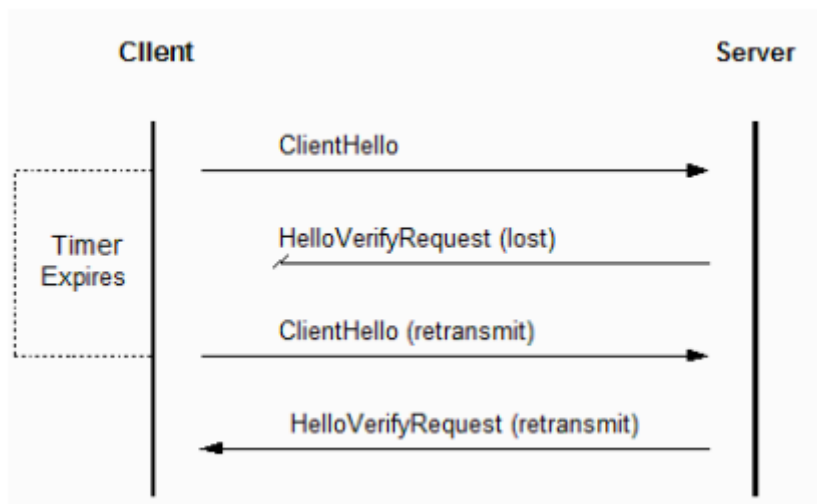


Figure 2.18: DTLS Timer Basic Concept

Thread uses the DTLS protocol for internet-grade end-to-end security when commissioning, and may use CASE, PASE, TLS or other encryption methods, based on the used application layer.

From the point of view of joining new devices into a network, the possession of the network key is used to discriminate between an authenticated and authorized Thread Device and the joining device (in its initial state). The network-wide key, along with other network parameters, is delivered securely to a joining device. This way, the network key is never exposed in the clear on a wireless link.

2.6. Transport Layer: UDP and TCP

Thread Devices use UDP (User Datagram Protocol) for messaging between devices for mesh establishment and maintenance. Thread Networks also support TCP (or any other IPv6-based transport protocol) for application layer communication.

The UDP protocol is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol (IP) is used as the underlying protocol. It provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. That is why it is needed the DTLS in the Thread architecture, as it is mainly based in UDP messaging. Applications requiring ordered reliable delivery of streams of data should use the Transmission Control Protocol (TCP).

TCP provides reliable, ordered, and error-checked delivery of a stream of data, by also providing fragmentation of reassembly of them, between applications running on hosts communicating via an IP network. TCP detects network congestion

and traffic load balancing problems, requests re-transmission of lost data, re-arranges out-of-order data and even helps minimize network congestion to reduce the occurrence of the other problems. If the data still remains undelivered, the source is notified of this failure.

2.7. Application Layer

This layer specifies the shared protocols and interface methods used by the hosts of a network. The Thread specification defines standard methods for forming and joining a network (called commissioning) and custom applications are responsible for interoperability. Thread does, however, provide these basic application services:

- **UDP messaging:** UDP offers a way to send messages using a 16-bit port number and an IPv6 address. UDP is a simpler protocol than TCP and has less connection overhead (for example, UDP does not implement keep-alive messages). For this reason, UDP enables a faster, higher throughput of messages and reduces the overall power budget of an application. UDP also has a smaller code space than TCP, which leaves more available flash on the chip for custom applications.
- **Multicast messaging:** Thread provides the ability to broadcast messages, that is, sending the same message to multiple nodes on a Thread network. Multicast allows a built-in way to talk to neighbor nodes, routers, and an entire Thread network with standard IPv6 addresses.
- **Application layers using IP services:** Thread allows the use of application layers such as UDP and CoAP (Constrained Application Protocol) to allow devices to communicate interactively over the Internet. Non-IP application layers will require some adaptation to work on Thread.

Chapter 3

Openthread

Nest Labs, Inc. (acquired by Google in the beginning of 2014) released OpenThread in May 2016. OpenThread [11] is an open source implementation of the Thread networking protocol (based in the Thread 1.1.1 specification). The purpose is to make the Nest products more broadly available to accelerate the development of products for the connected home. It is operative system (OS) and platform agnostic, with a radio abstraction layer and a small memory footprint, making it highly portable. It supports both system-on-chip (SoC) and network co-processor (NCP) designs.

Along with Nest, ARM, Atmel, Dialog Semiconductor, Qualcomm Technologies, Inc. and Texas Instruments Incorporated are contributing to the ongoing development of OpenThread. In addition, OpenThread can run on Thread-capable radios and corresponding development kits from silicon providers like NXP Semiconductors and Silicon Labs.

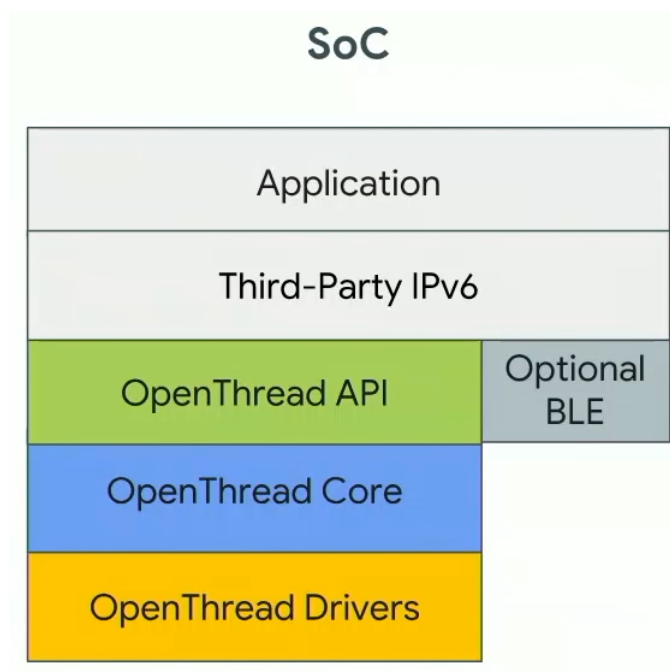


Figure 3.1: SoC architecture

OpenThread implements all Thread networking layers including:

- IEEE 802.15.4 with MAC security.
- IPv6 and 6LoWPAN.
- Mesh Link Establishment and Mesh Routing.
- Key management.
- Definitions in code of specific roles in Thread including:
 - Leader.
 - Router.
 - End Device.
 - Border Router.
- UDP packet compression.
- A CoAP implementation.

It is mostly written in C++. All the code is available at the [Openthread Github repository](#) and can be run in a variety of Software platforms and SoCs development boards including:

- Dialog DA15000.
- Nordic Semiconductor nRF52840.
- Texas Instruments CC2538, CC2650 and CC2652 .
- Zolertia RE-Mote and Firefly.
- Silicon Labs EFR32.
- NXP KW41Z.
- Qorvo GP712.
- POSIX Emulation.

In the next sections, some of the most important functions implemented will be explained.

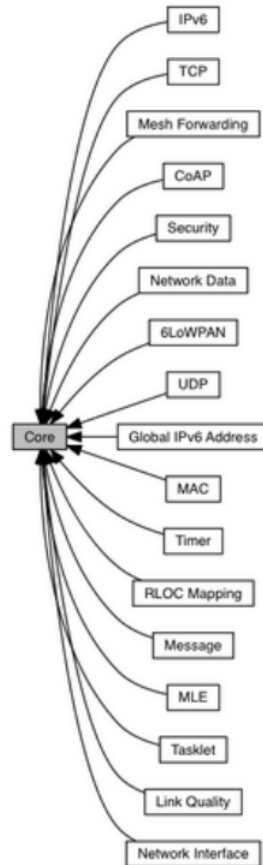


Figure 3.2: Overview of the Openthread Modules

3.1. Network Implementation

The Openthread API offers an IPv6 implementation which includes definitions for the IPv6 network layer. The IPV6 module (fig 3.3) have different submodules implemented: the ICMPv6 implementation, the network interfaces, the multicast protocol and the IPv6 implementation itself.

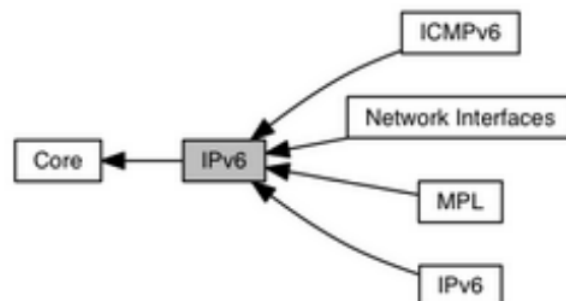


Figure 3.3: Overview of the implemented IPv6 Module

All the files concerning the network implementation can be found in *src/core/net* folder and the IPv6 main file can be found in the folder *src/core/net/ipv6.cpp*.

3.2. Routing layer implementation

OpenThread implement MLE to propagate the Routing table information and RIPng to process information and maintain routing tables.

The MLE module provides the MLE functionality required for the Thread Router and Leader roles. It contains the Router module, to manage the roles already mentioned, the Core and the TLVs module. The Type Length Value (TLV) module includes definitions for generating and processing MLE TLVs.

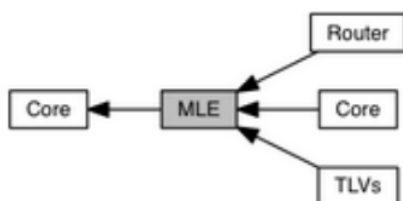


Figure 3.4: Overview of the implemented MLE Module

All the routing implementations are found in the *src/core/thread*. MLE main file is *src/core/thread/mle.cpp* and contains all the functions and references to implement MLE functionalities. The Routing parameters are defined in the file named *src/core/thread/mle_types.hpp*. Some examples are: the number of child allowed by a router (FIG), the maximum number of allowed Routers ($OPENTHREAD_CONFIG_MLE_MAX_ROUTERS$ is defined as 32 in a more general configuration file) in (FIG) or the definition of the different levels of link quality (FIG).

```

constexpr uint16_t kMinChildId = 1; ///< Minimum Child ID
constexpr uint16_t kMaxChildId = 511; ///< Maximum Child ID
  
```

Figure 3.5: Number of child defined in the Openthread API

```

constexpr uint8_t kMaxRouterId          = OT_NETWORK_MAX_ROUTER_ID; ///< Max Router ID
constexpr uint8_t kInvalidRouterId      = kMaxRouterId + 1;        ///< Value indicating incorrect Router ID
constexpr uint8_t kMaxRouters           = OPENTHREAD_CONFIG_MLE_MAX_ROUTERS;
constexpr uint8_t kMinDowngradeNeighbors = 7;

constexpr uint8_t kNetworkIdTimeout     = 120; ///< (in sec)
constexpr uint8_t kParentRouteToLeaderTimeout = 20; ///< (in sec)
constexpr uint8_t kRouterSelectionJitter = 120; ///< (in sec)

constexpr uint8_t kRouterDowngradeThreshold = 23;
constexpr uint8_t kRouterUpgradeThreshold   = 16;

```

Figure 3.6: Some router parameters in the Openthread API

```

constexpr uint8_t kLinkQuality3LinkCost = 1;          ///< Link Cost for Link Quality 3
constexpr uint8_t kLinkQuality2LinkCost = 2;          ///< Link Cost for Link Quality 2
constexpr uint8_t kLinkQuality1LinkCost = 4;          ///< Link Cost for Link Quality 1
constexpr uint8_t kLinkQuality0LinkCost = kMaxRouteCost; ///< Link Cost for Link Quality 0

```

Figure 3.7: Link Quality definition in the Openthread API

The command types are defined in `src/core/thread/mle.hpp` (FIG).

```

kCommandLinkRequest          = 0,  ///< Link Request
kCommandLinkAccept          = 1,  ///< Link Accept
kCommandLinkAcceptAndReject = 2,  ///< Link Accept and Reject
kCommandLinkReject          = 3,  ///< Link Reject
kCommandAdvertisement        = 4,  ///< Advertisement
kCommandUpdate              = 5,  ///< Update
kCommandUpdateRequest       = 6,  ///< Update Request
kCommandDataRequest         = 7,  ///< Data Request
kCommandDataResponse        = 8,  ///< Data Response
kCommandParentRequest       = 9,  ///< Parent Request
kCommandParentResponse      = 10, ///< Parent Response
kCommandChildIdRequest      = 11, ///< Child ID Request
kCommandChildIdResponse     = 12, ///< Child ID Response
kCommandChildUpdateRequest  = 13, ///< Child Update Request
kCommandChildUpdateResponse = 14, ///< Child Update Response
kCommandAnnounce            = 15, ///< Announce
kCommandDiscoveryRequest    = 16, ///< Discovery Request
kCommandDiscoveryResponse   = 17, ///< Discovery Response
kCommandLinkMetricsManagementRequest = 18, ///< Link Metrics Management Request
kCommandLinkMetricsManagementResponse = 19, ///< Link Metrics Management Response
kCommandLinkProbe           = 20, ///< Link Probe
kCommandTimeSync            = 99, ///< Time Sync (when OPENTHREAD_CONFIG_TIME_SYNC_ENABLE enabled)

```

Figure 3.8: MLE command types

3.3. DTLS implementation

The DTLS implemented (FIG) is part of the MeshCoP module and defines all the related messages to implement the DTLS functionalities in OpenThread stack. It is the file named *src/core/meshcop/dtls.hpp*.

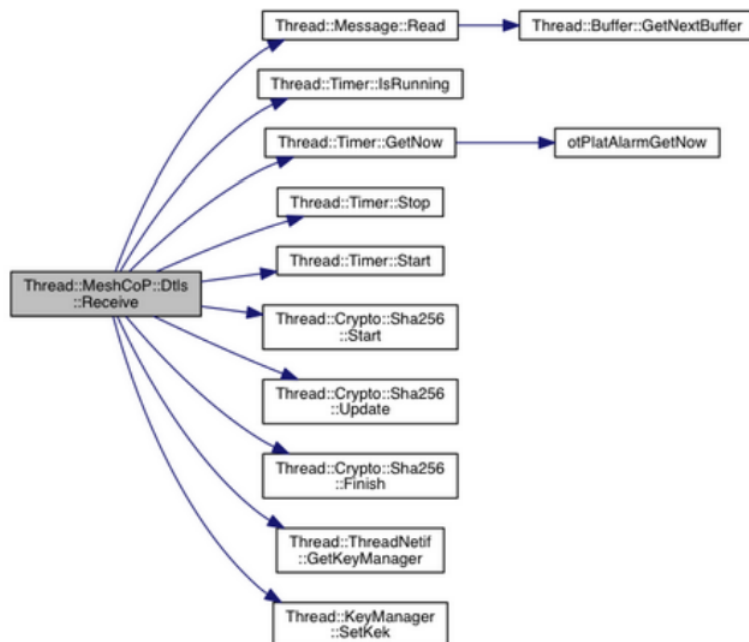


Figure 3.9: Overview of the DTLS module

3.4. Application layer implementation: CoAP

The OpenThread standard offers a CoAP API that allows application to use the same CoAP implementation to send/receive CoAP messages. It also offers an API for communications over CoAP and DTLS.

CoAP is an application layer protocol that is intended for use in resource-constrained internet devices, such as WSN nodes. CoAP is designed to easily translate to HTTP for simplified integration with the web, while also meeting specialized requirements such as multicast support, very low overhead and simplicity.

CoAP can run on most devices that support UDP. CoAP makes use of two message types, requests and responses, using a simple binary base header format. The base header may be followed by options in an optimized Type-Length-Value format. CoAP is by default bound to UDP and optionally to DTLS (Datagram Transport Layer Security), providing a high level of communications security.

The CoAP Message Types can be either confirmable (CON), non-confirmable (NON). Confirmable messages require an ACK, while non-confirmable messages

don't. If we don't need reliability, we use NON, for example, a sensor broadcasting data and if we need reliability, we use CON, for example, issuing a GET to a server.

3.5. CLI Commnads

The OpenThread CLI exposes configuration and management APIs via a command line interface. Use the CLI to play with OpenThread, which can also be used with additional application code.

Some of the CLI available commands are:

- `bufferinfo`: shows the current buffer information.

```
> bufferinfo
total: 44
free: 44
6lo send: 0 0
6lo reas: 0 0
ip6: 0 0
mpl: 0 0
mle: 0 0
arp: 0 0
coap: 0 0
coap secure: 0 0
application coap: 0 0
```

Figure 3.10: Example of the `bufferinfo` command output

- `channel`:
 - `channel`: get the IEEE 802.15.4 channel value.
 - `channel <channel>`: set the IEEE 802.15.4 channel value.

```
> channel
11
```

Figure 3.11: Example of the `channel` command output

- `child`:
 - `child list`: prints a list of the attached child IDs.
 - `child table`: prints a trable of the attached children.

- child max:
 - child max: get the Thread maximum available connected children.
 - child max <count>: set the Thread maximum available connected children.

- childtimeout:
 - child timeout: get the Thread Child Timeout value.
 - child timeout <timeout>: set the Thread Child Timeout value.

- commissioner:
 - commissioner start <provisioningURL>: Start the Commissioner role.
 - commissioner stop: Stop the Commissioner role.
 - commissioner joiner add <hashmacaddr> <psdk>: add a joiner entry.
 - commissioner joiner remove <hashmacaddr>: remove a joiner entry.
 - commissioner provisioningurl <provisioningUrl>: set the provisioning URL.
 - commissioner energy <mask> <count> <period> <scanDuration> <destination>: send a *MGMT_ED_SCAN* message.
 - commissioner panid <panid> <mask> <destination>: send a *MGMT_PANID_QUERY* message.
 - commissioner sessionid: Get current commissioner session id.

- contextreusedelay:
 - contextreusedelay: get the *CONTEXT_ID_REUSE_DELAY*.
 - contextreusedelay <delay>: set the *CONTEXT_ID_REUSE_DELAY*.

- counter:
 - counter: get the supporter counter names.

```
> counters
ip
mac
mle
Done
```

Figure 3.12: Example of the counters command output

- counter <name>: get the counters value.

```

> counters ip
TxSuccess: 0
TxFailed: 0
RxSuccess: 0
RxFailed: 0
Done
> counters mac
TxTotal: 0
  TxUnicast: 0
  TxBroadcast: 0
  TxAckRequested: 0
  TxAked: 0
  TxNoAckRequested: 0
  TxData: 0
  TxDataPoll: 0
  TxBeacon: 0
  TxBeaconRequest: 0
  TxOther: 0
  TxRetry: 0
  TxErrCca: 0
  TxErrBusyChannel: 0
RxTotal: 0
  RxUnicast: 0
  RxBroadcast: 0
  RxData: 0
  RxDataPoll: 0
  RxBeacon: 0
  RxBeaconRequest: 0
  RxOther: 0
  RxAddressFiltered: 0
  RxDestAddrFiltered: 0
  RxDuplicated: 0
  RxErrNoFrame: 0
  RxErrNoUnknownNeighbor: 0
  RxErrInvalidSrcAddr: 0
  RxErrSec: 0
  RxErrFcs: 0
  RxErrOther: 0
Done

```

Figure 3.13: Example of the counters <name> command output

- dataset (set of configuration properties of the Thread network):
 - dataset active: active operational dataset.
 - dataset channel <channel>: set channel.
 - dataset commit <dataset>: commit operational dataset buffer to active/pending operational dataset.
 - dataset panid <extpanid>: set panid.
 - dataset networkkey <networkkey>: set network key.
- delaytimermin:
 - delaytimermin: get the minimal delay timer.
 - delaytimermin <delaytimermin>: set the minimal delay timer.
- discover:

- discover: perform an MLE Discovery operation.
- discover <channel>: The channel to discover on. If no channel is provided, the discovery will cover all valid channels.

```
> discover
| 3 | Network Name      | Extended PAN      | PAN | MAC Address      | Ch | dBm | LQI |
+---+-----+-----+---+-----+---+---+---+
| 0 | OpenThread        | dead00beef00cafe | ffff | f1d92a82c8d8fe43 | 11 | -20 | 0   |
Done
```

Figure 3.14: Example of the discover command output

- eui64: Get the factory-assigned IEEE EUI-64.
- extaddr:
 - extaddr: get the IEEE 802.15.4 Extended Address.
 - extaddr <extaddr>: set the IEEE 802.15.4 Extended Add
- extpanid:
 - extpanid: get the Thread Extended PAN ID value.
 - extpanid <extpanid>: set the Thread Extended PAN ID value.
- factoryreset: Delete all stored settings, and signal a platform reset.
- hashmacaddr: Get the HashMac address.
- ifconfig:
 - ifconfig: Show the status of the IPv6 interface.
 - ifconfig up: Bring up the IPv6 interface.
 - ifconfig down: Bring down the IPv6 interface.
- ipaddr:
 - ifconfig: List all IPv6 addresses assigned to the Thread interface.
 - ifconfig up: Add address to the Thread interface.
 - ifconfig down: Delete address from the Thread interface.
- ipmaddr:
 - ipmaddr: List all IPv6 multicast addresses.
 - ipmaddr add <ipmaddr>: Subscribe the Thread interface to the IPv6 multicast address.

- `ipmaddr del <ipaddr>`: Unsubscribe the Thread interface to the IPv6 multicast address.
- `linkquality`:
 - `linkquality <extaddr>`: Get the link quality on the link to a given extended address.
 - `linkquality <extaddr> <linkquality>`: Set the link quality on the link to a given extended address.
- `masterkey`
 - `masterkey`: get the Thread Master Key value.
 - `masterkey <key>`: set the Thread Master Key value.
- `mode`:
 - `mode`: get the Thread Device Mode value.
 - `mode [rsdn]`: set the Thread Device Mode value.
 - r: rx-on-when-idle.
 - s: Secure IEEE 802.15.4 data requests.
 - d: Full Function Device
 - n: Full Network Data
- `netdataregister`: register local network data with Thread Leader.
- `networkidtimeout`:
 - `networkidtimeout`: get the *NETWORK_ID_TIMEOUT* used in the Router.
 - `networkidtimeout <timeout>`: set the *NETWORK_ID_TIMEOUT* used in the Router role.
- `networkname`:
 - `networkname`: get the Thread Network Name.
 - `networkname <name>`: set the Thread Network Name.
- `panid`:
 - `panid`: get the IEEE 802.15.4 PAN ID value.
 - `panid <panid>`: set the IEEE 802.15.4 PAN ID value.
- `parent`: get the information for a Thread Router as parent.
- `ping <ipaddr> [size] [count] [interval]`: Send an ICMPv6 Echo Request.
- `releaserouterid <routerid>`: Release a Router ID that has been allocated by the device in the Leader role.

- reset: signal a platform reset.
- rloc16: get the Thread RLOC16 value.
- route:
 - route remove <prefix>: Invalidate a prefix in the Network Data.
 - route add <prefix> [s] [prf]: Add a valid prefix to the Network Data
- router:
 - router list: List allocated Router IDs.
 - router <id>: Print diagnostic information for a Thread Router.
 - router table: Print table of routers.
- routerrole:
 - routerrole: Indicates whether the router role is enabled or disabled.
 - routerrole enable: enable the router role.
 - routerrole disable: disable the router role.
- routerupgradethreshold:
 - routerupgradethreshold: get the *ROUTER_UPGRADE_THRESHOLD* value.
 - routerupgradethreshold <threshold>: set the *ROUTER_UPGRADE_THRESHOLD* value.
- scan: perform an IEEE 802.15.4 Active Scan.
- singleton: Return true when there are no other nodes in the network, otherwise return false.
- state:
 - state: return the current state.
 - state <mode>: tries to switch to the State (detached, child, router, leader).
- thread:
 - thread start: Enable Thread protocol operation and attach to a Thread network.
 - thread stop: Disable Thread protocol operation and detach from a Thread network.
- version: print the build version information
- whitelist:

- whitelist: List the whitelist entries.
- withelist enable: Enable MAC withelist filtering.
- withelist disable: Disable MAC withelist filtering.
- withelist add <extaddr>: Add an address to the withelist.
- withelist remove <extaddr>: Remove address from the withelist.
- withelist clear: Clear all entries from the withelist.

Chapter 4

Hardware test environment

Once all the theory is explained, and before tests are performed and the results are exposed, a Thread network must be created. For doing it, different thread devices are needed, which are detailed in the following sections.

Zolertia devices (Re-Mote and Firefly) are the ones used for creating the Thread Networks. Other devices were tested, like MakerDiary Thread devices, but they were not configured successfully.

4.1. Zolertia devices

Zolertia is a catalan company who develop IoT hardware and software solutions. They provide some development tools to set up and configure Thread networks like Zolertia Re-Mote and Zolertia Firefly. Both will be discussed in a detailed way in the following sections.

4.1.1. Zolertia Re-Mote

The Zolertia Re-Mote is a lightweight and powerful Internet of Things hardware development platform to enable any idea to be connected to the Internet, providing a seamless connectivity for most indoor and outdoor applications, running forever on batteries. The RE-Mote includes a multiband antenna to start sending data from the start.



Figure 4.1: Zolertia Re-Mote

It provides the following features [16]:

- ISM 2.4-GHz IEEE 802.15.4 and Zigbee compliant radio.
- ISM 863-950-MHz ISM/SRD band IEEE 802.15.4 compliant radio.
- ARM Cortex-M3 32 MHz clock speed, 512 KB flash and 32 KB RAM (16 KB retention).
- AES-128/256, SHA2 Hardware Encryption Engine.
- ECC-128/256, RSA Hardware Acceleration Engine for Secure Key Exchange.
- User and reset button.
- Consumption down to 150 nA using the shutdown mode.
- Programming over BSL without requiring to press any button to enter boot-loader mode.
- Built-in battery charger (500 mA), facilitating Energy Harvesting and direct connection to Solar Panels and to standards LiPo batteries.
- Wide range DC Power input: 3.3-16 V.
- Small form-factor (73 x 40 mm).
- MicroSD (over SPI).
- On board RTCC (programmable real time clock calendar) and external watchdog timer (WDT).
- Programmable RF switch to connect an external antenna either to the 2.4 GHz or to the Sub 1 GHz RF interface through the RP-SMA connector.

- Supported in Open Source OS as Contiki, RIOT and OpenWSN.

And with those, it can perform the following functionalities:

- Two radios, compatible with trending protocols such as Thread and SiG-FOX, to use both residential/indoor and long-range applications, on top of very well supported protocols like 6LoWPAN and IEEE 802.15.4. The maximum range is between 100 meters and 20 km, with highly configurable radio parameters such as modulation, data rate, transmission power, etc.
- Ultra-low power operation, ranging from 1uA to 150nA.
- Protected communication with on-board hardware security (SHA2, AES-128/256, ECC-128/256 and RSA for secure key exchange).
- Real Time clock capabilities to develop applications based on real time information
- Available interfaces and connectors to plug-in directly any different sensors (analogue and digital).
- Benefit from a wide-world community of developers due to being supported in Open Source OS.

4.1.2. Zolertia Firefly

The Zolertia Firefly is a breakout board designed to inspire, build and develop from scratch any Internet of Things application. The Firefly can be seen as the "small brother" of the RE-Mote, with a slick design and a lower cost.



Figure 4.2: Zolertia Firefly

It provides the following features [14]:

- ISM 2.4-GHz IEEE 802.15.4 and Zigbee compliant radio.
- ISM 863-950-MHz ISM/SRD band IEEE 802.15.4 compliant radio.
- ARM Cortex-M3 with 512KB flash and 32KB RAM (16KB retention), 32MHz.
- On-board printed PCB sub-1GHz antenna, ceramic antenna for 2.4GHz interface. Alternatively u.FL for sub-1GHz external antenna.
- AES-128/256, SHA2 Hardware Encryption Engine.
- ECC-128/256, RSA Hardware Acceleration Engine for Secure Key Exchange.
- Compatible with breadboards and protoboards.
- On-board CP2104/PIC to flash over PCB USB.
- User and reset buttons.
- RGB LED to allow more than 7 colour combinations.
- Small form factor (68x25mm) including USB.
- 3.3VDC C support for 2xAA/AAA and Coin Cell batteries.

So, the hardware of the Zolertia Firefly is more basic than the Hardware Zolertia Re-Mote development board. The improvements are the size reduction and the easy way to connect, through the usb adaptor to the pc.

The firefly can be programmed and debugged over JTAG and USB. The board has a CP2104 USB to serial converter with a PIC, it allows to program the CC2538 without having to manually to put the device in bootloader mode.



Figure 4.3: Zolertia Firefly connected by its usb adaptor

4.2. TI CC2531EMK Sniffer Tool: ICQUANZX CC2531EMK

The ICQUANZX CC2531EMK kit provides one CC2531 USB Dongle and documentation to support a PC interface to 802.15.4 / ZigBee applications. The dongle can be plugged directly into your PC and can be used as an IEEE 802.15.4 packet sniffer or for other purposes.



Figure 4.4: ICQUANZX CC2531EMK kit

The dongle has 2 LEDs, two small push-buttons and connector holes that allow connection of external sensors or devices. The dongle also has a connector for programming and debugging of the CC2531 USB controller.

It comes preprogrammed with firmware such that it can be used as a packet sniffer device.

Chapter 5

Setup, tests and performances

This chapter will cover the different hardware and software configurations are applied to be able to perform all the desired tests, such as a basic connectivity between devices, thus creating a small Thread network. In addition, we will check the visibility of the nodes connected to the same network and also the ones that are not joined. The next step will be the revision of the routing tables. Finally, through different captures, we will analyze the behavior after some changes of scenario.

5.1. Test Scenario installation and Configuration

As the goal of this section is to create a Thread network to evaluate, a configuration and installation of the test must be done in the available devices. It consist on flashing images with the pre-configured examples to emulate the desired scenario for doing the test. In our case, it will be to allow the cli interaction through the command window.

Moreover, it is needed to configure a Thread sniffer, in order to adquire and visualize the Thread messages between devices, and configure Wireshark to plot directly from the data provided by the sniffer.

5.1.1. Zolertia devices configuration

In order to set up a Thread network, the Zolertia devices have to be configured and programmed by a flashing image. In order to do it, a preconfigured virtual enviroment will be used, called Contiki, provided by the course *IoTinfivedays*. Contiki is an open source operating system for the Internet of Things, it connects tiny low-cost, low-power microcontrollers to the Internet.

Contiki provides powerful low-consumption Internet communication, it supports fully standard IPv6 and IPv4, along with the recent low-power wireless standards: 6LoWPAN, RPL, CoAP. With Contiki's ContikiMAC and sleepy routers, even wireless routers can be battery-operated.

A virtual machine with Contiki installed will be used. It can be downloaded freely from this web site: <https://sourceforge.net/projects/zolertia/files/>. It contains also the CC2538 compiler which will allow to interact with the device by the cli command console.

```
./bootstrap
make -f examples/Makefile-cc2538 clean && make -f
examples/Makefile-cc2538
```

Figure 5.1: Code to compile the CC2538 example

And the image file will be stored in *openthread/output/cc2538/bin*.

```
arm-none-eabi-objcopy -O binary output/cc2538/bin/ot-cli-ftd
output/cc2538/bin/ot-cli-ftd.bin
```

Figure 5.2: Code to compile the CC2538 example

Then, the Zolertia board can be plugged into the computer. A window will prompt asking where the device is connected and it must be chosen to connect it to the virtual machine.

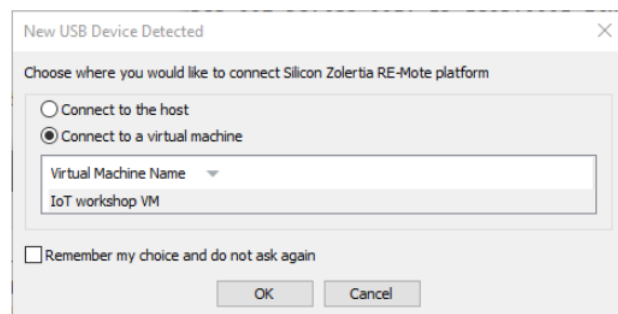


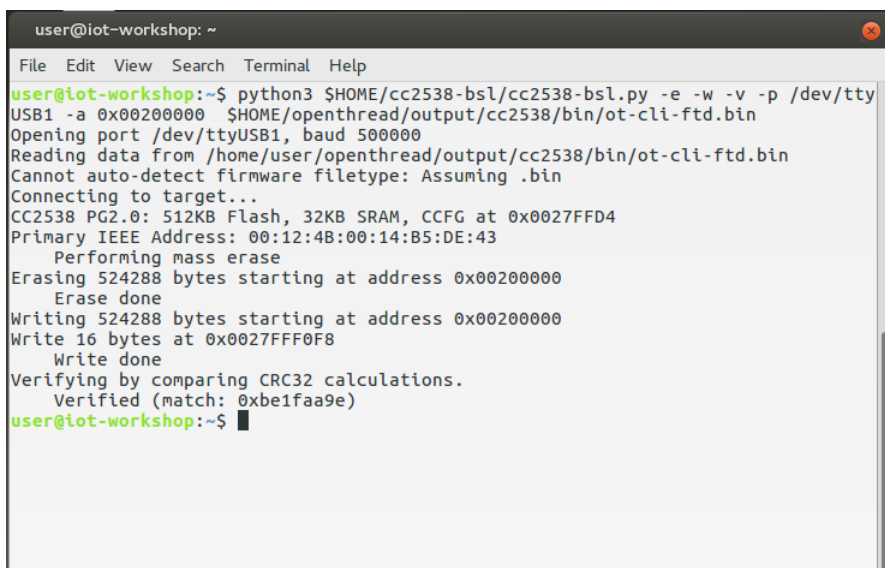
Figure 5.3: Window asking where to connect the plugged device

For installing the image in any device, the following command must be used (taking in account the usb where is connected the board):

```
python3 $HOME/cc2538-bsl/cc2538-bsl.py -e -w -v -p /dev/ttyUSB0
-a 0x00200000 $HOME/openthread/output/cc2538/bin/ot-cli-ftd.bin
```

Figure 5.4: Command to flash the image into the device

The instalation process, if concluded successfully, will notify that it was able to write the image into the memory and the checksum obtained matched, as shown in the figure below.



```

user@iot-workshop: ~
File Edit View Search Terminal Help
user@iot-workshop:~$ python3 $HOME/cc2538-bsl/cc2538-bsl.py -e -w -v -p /dev/tty
USB1 -a 0x00200000 $HOME/openthread/output/cc2538/bin/ot-cli-ftd.bin
Opening port /dev/ttyUSB1, baud 500000
Reading data from /home/user/openthread/output/cc2538/bin/ot-cli-ftd.bin
Cannot auto-detect firmware filetype: Assuming .bin
Connecting to target...
CC2538 PG2.0: 512KB Flash, 32KB SRAM, CCFG at 0x0027FFD4
Primary IEEE Address: 00:12:4B:00:14:B5:DE:43
Performing mass erase
Erasing 524288 bytes starting at address 0x00200000
Erase done
Writing 524288 bytes starting at address 0x00200000
Write 16 bytes at 0x0027FFF0F8
Write done
Verifying by comparing CRC32 calculations.
Verified (match: 0xbe1faa9e)
user@iot-workshop:~$

```

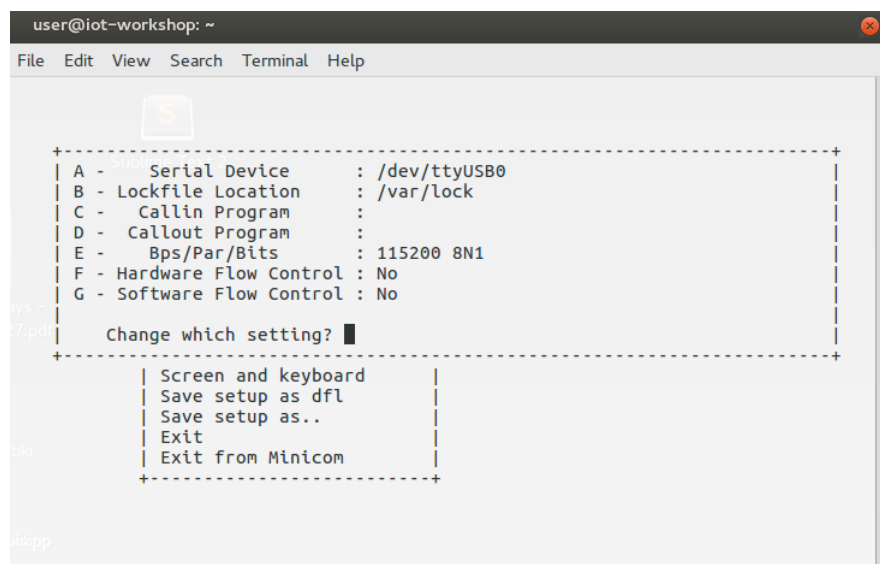
Figure 5.5: Image flashing process

Then, the only thing left to do is call the appropriate command to enable the cli console interaction:

```
sudo minicom -b 115200 -D /dev/ttyUSBx -s
```

Figure 5.6: Command to start the cli console

It will prompt a settings menu where we need disable, in the *serialportsetup* submenu, the *HardwareFlowControl* option (fig 5.7). Then the configuration and communication with the Zolertia device through the cli interface will be available.



```

user@iot-workshop: ~
File Edit View Search Terminal Help
$
+-----+
| A - Serial Device      : /dev/ttyUSB0 |
| B - Lockfile Location  : /var/lock    |
| C - Callin Program    :              |
| D - Callout Program   :              |
| E - Bps/Par/Bits      : 115200 8N1   |
| F - Hardware Flow Control : No       |
| G - Software Flow Control : No       |
+-----+
Change which setting? █
+-----+
| Screen and keyboard | |
| Save setup as dfl   | |
| Save setup as..    | |
| Exit                | |
| Exit from Minicom  | |
+-----+

```

Figure 5.7: Serial port setup configuration

5.1.2. Configuring the sniffer

For setting up the sniffer, the software needed can be downloaded the Texas Instrument web site [10]. It is a IEEE 802.15.4 MAC software stack used to sniff IEEE 802.15.4 frames. It is also needed to install the Microsoft Visual C++ and the drivers of the usb dongle. Finally, in order to visualize the frames, Wireshark must be installed in our Windows device.

Once everything is installed, the TiWsPc (Texas Instruments Wireshark Packet Converter) program must be available in our machine. Then, then only thing remaining is to configure the TiWSPc to user our sniffer device to get the IEEE 802.15.4 frames. If everything is installed well, TiWsPC will recognize the CC2538 usb dongle. It must be configured to sniff in the same channel where the Thread network is working (I will set it to 26 in my particular case, as this one is the channel with less interference due to WiFi users).

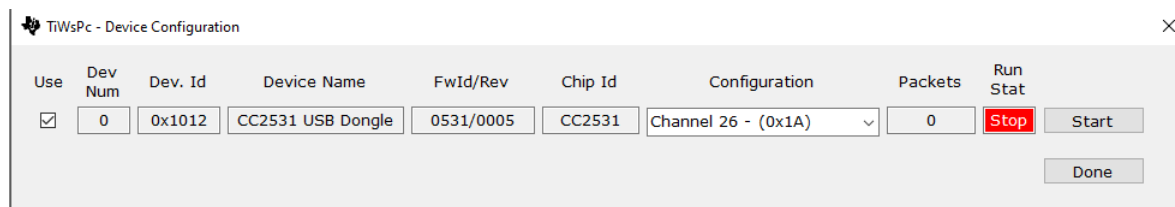


Figure 5.8: Configuration window for adjusting the sniffing parameters

Next, a pipe for connecting the sniffer with Wireshark must be created. First, a shortcut in our desktop will be created. Then, some commands must be added to enable it properly, as shown in FIG:

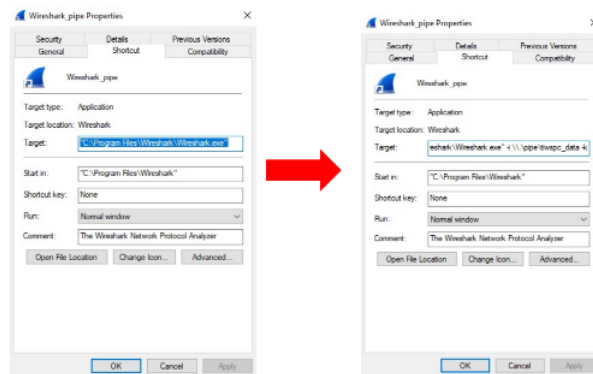


Figure 5.9: Additional commands to set up the pip

Finally, the program will show if its working properly. Then, clicking the *start* button in the TIWPC program will enable, when we open Wireshark through the link created. to capture and show the IEEE 802.15.4 frames.

In order to decrypt all the network messaging, it is also needed to add a configuration field in the Wireshark. It is found in *Preferences* Wireshark's option,

then in the navigation bar *Protocols* must be selected to expand it and search for *IEEE802.15.4*. There, a decryption key must be provided, adding a new field with the index of the key and the key of the network 5.10 (which is known and can be found in the configuration dataset of the network leader deviiifconfe).

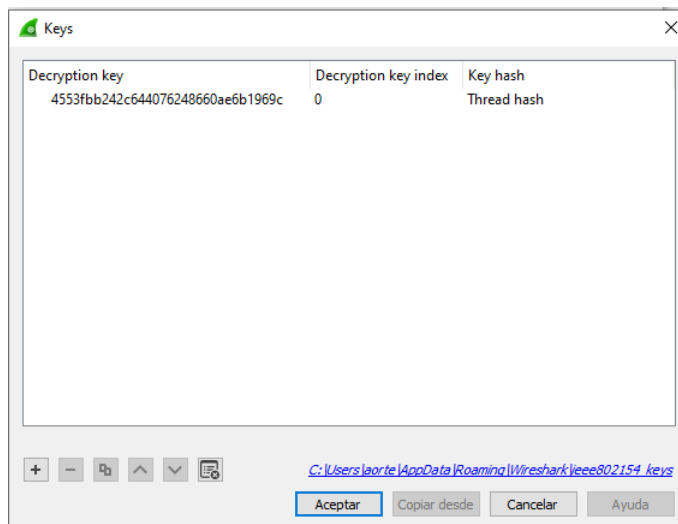


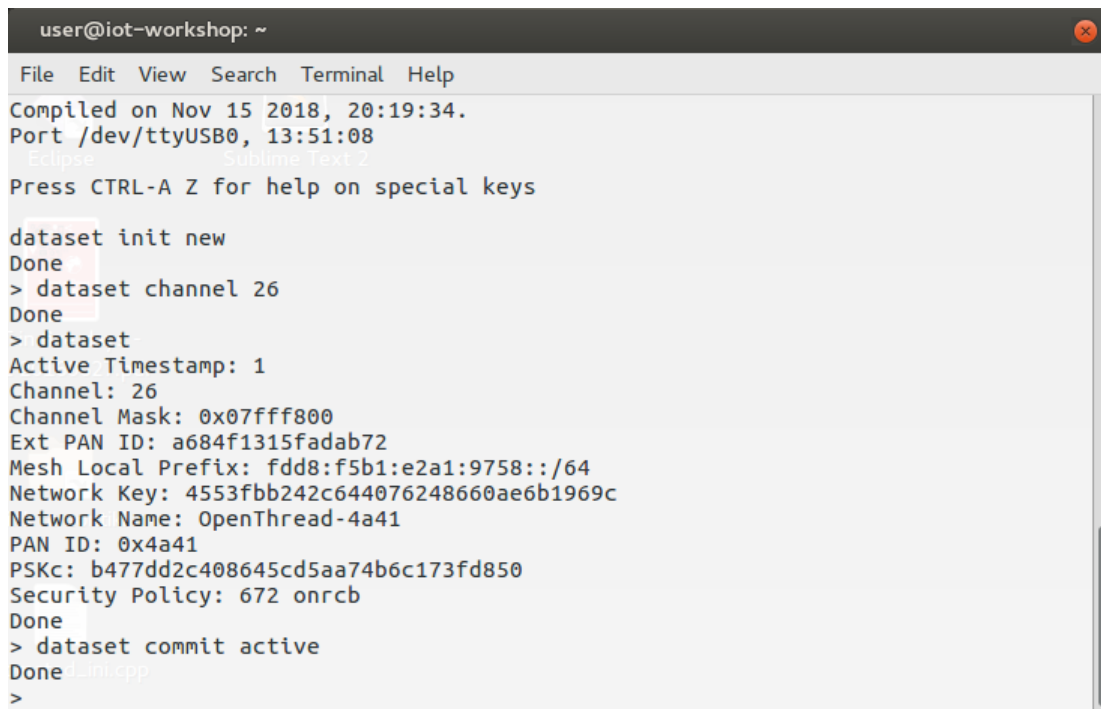
Figure 5.10: Decryption key configuration

5.2. Tests

5.2.1. Set up a Thread Network

The goal of the initial test is to set up a Thread network and observe in Wireshark the interaction between the devices and the network's performance under different conditions, such as the attachment of a new device or a device disconnection.

In order to create the network, the first Zolertia device must create a new dataset with the configuration needed for the network creation. It will consist on adjusting the channel where the network will operate, in my case, to be able to capture the frames with the sniffer it will be the 26th channel; and commit it. Whenever the device is connected, it will use the configuration provided (5.11). The network key that appears here it will be used to configure the new devices in order to attach it to the network as well as to configure Wireshark to unmask all the network's communication.



```
user@iot-workshop: ~  
File Edit View Search Terminal Help  
Compiled on Nov 15 2018, 20:19:34.  
Port /dev/ttyUSB0, 13:51:08  
Eclipse Sublime Text 2  
Press CTRL-A Z for help on special keys  
  
dataset init new  
Done  
> dataset channel 26  
Done  
> dataset  
Active Timestamp: 1  
Channel: 26  
Channel Mask: 0x07fff800  
Ext PAN ID: a684f1315fadab72  
Mesh Local Prefix: fdd8:f5b1:e2a1:9758::/64  
Network Key: 4553fbb242c644076248660ae6b1969c  
Network Name: OpenThread-4a41  
PAN ID: 0x4a41  
PSKc: b477dd2c408645cd5aa74b6c173fd850  
Security Policy: 672 onrcb  
Done  
> dataset commit active  
Done  
>
```

Figure 5.11: Dataset creation and submission

Then, the IPv6 interface must be enabled and the Thread network is ready to be deployed (5.12). As it is the first device, it will become the leader of the network.

```
> ifconfig up  
Done  
> thread start  
Done  
> state  
leader  
Done
```

Figure 5.12: Ipv6 interface and Thread deployment commands

In order to attach a new device, the process is similar to creating one. But in that case the dataset will be created with just the channel and the network key (fig 5.13). Once the Thread interface is enabled, this new device will attach to the existing network and obtain the information of it through the devices already deployed. Then, the last thing to do is to select the role (Router or Child), according to the functionality desired for the new device.

```

dataset channel 26
Done
> dataset networkkey 4553fbb242c644076248660ae6b1969c
Done
> dataset commit active
Done
> ifconfig up
Done
> thread start
Done
> state
detached
Done
> state router
Done
> state
router
Done
> state child
Done

```

Figure 5.13: Procedure to attach a new device into an existing Thread network

For doing this initial test, two devices will be attached to the network (apart from the leader): one will be a router and one will be a child.

Finally, to prove that the devices are connected in the same network, a discovery of the channel used is done:

```

> discover 26
+-----+-----+-----+-----+-----+-----+-----+-----+
| J | Network Name | Port | Extended PAN ID | PAN | MAC Address | Ch | dBm |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 0 | OpenThread-4a41 |  | a684f1315fadab72 | 4a41 | ae96413135e5bf3f | 26 | -46 |
| 0 | OpenThread-4a41 |  | a684f1315fadab72 | 4a41 | c2bd092dfa221fbc | 26 | -55 |
Done

```

Figure 5.14: Discovery command output

As the information table provides, there are 2 devices, apart from the leader who is doing the discovery, and they belong to the same Personal Area Network.

5.2.2. Network Interaction

In this subsection the network interaction will be analyzed through the Wireshark captures showing the sniffed the Thread messages.

At the beginning, there was just one device. It continuously keeps announcing the network information to all neighbors, shown in figure 5.15, and its links with other devices (like a routing table) through MLE advertisement messages (fig 5.16).

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	fe80::7c00:42bc:ec35:fa0	ff02::1	MLE	129	Advertisement
2	21.882158	fe80::7c00:42bc:ec35:fa0	ff02::1	MLE	129	Advertisement
3	53.413830	fe80::7c00:42bc:ec35:fa0	ff02::1	MLE	129	Advertisement

Figure 5.15: Leader device multicasting the network information

```

  ▾ Mesh Link Establishment
    Security Suite: 802.15.4 Security (0x00)
    > Auxiliary Security Header
      Command: Advertisement (4)
    ▾ TLV (Source Address = c4:00)
      Type: Source Address (0)
      Length: 2
      Address: c400
    ▾ TLV (Leader Data)
      Type: Leader Data (11)
      Length: 8
      Partition ID: 0x029c8adb
      Weighting: 64
      Data Version: 52
      Stable Data Version: 114
      Leader Router ID: 49
    ▾ TLV (Route64)
      Type: Route64 (9)
      Length: 10
      ID Sequence: 98
      Assigned Router ID Mask: 0000000000004000
    ▾ Routing Table Entry: 0x01 (49)
      00.. .... = Neighbor Out Link Quality: 0
      ..00 .... = Neighbor In Link Quality: 0
      .... 0001 = Router Cost: 1

```

Figure 5.16: Details of the information provided by the MLE Advertisement message

Once a new device wants to be attached to the existing network it will start sending a MLE Link Request (5.17) message where it will ask, as it is a request, for different TLV (5.18) in order to obtain the Routing Information through the Route64 TLV and the MAC addressing through the Address16 TLV.

```

6 147.493212  fe80::ac96:4131:35e5:bf3f  ff02::2  MLE  121 Link Request

```

Figure 5.17: Details of the initial MLE Link Request message

```

  ▾ Mesh Link Establishment
    Security Suite: 802.15.4 Security (0x00)
    > Auxiliary Security Header
      Command: Link Request (0)
    > TLV (Version = 3)
    ▾ TLV (TLV Request)
      Type: TLV Request (13)
      Length: 2
      Type: Address16 (10)
      Type: Route64 (9)
    > TLV (Challenge = 9e5ec7c1c61823ee)

```

Figure 5.18: Details of the information provided by the MLE Link request

Then it will start sending MLE Parent Request to start the handshake. Whenever the Leader (and the only one in the network till now) realized of a new entry, it announces through an MLE Announce message. Then, the routers and REEDs that receive the child request (and the leader) will respond with a MLE Parent Response message. This process can be seen in fig 5.19.

6	147.493212	fe80::ac96:4131:35e5:bf3f	ff02::2	MLE	121 Link Request
7	149.344182	fe80::ac96:4131:35e5:bf3f	ff02::2	MLE	123 Parent Request
8	150.095904	fe80::ac96:4131:35e5:bf3f	ff02::2	MLE	123 Parent Request
9	151.632973	fe80::ac96:4131:35e5:bf3f	ff02::2	MLE	123 Parent Request
10	152.382319	fe80::ac96:4131:35e5:bf3f	ff02::2	MLE	123 Parent Request
11	153.633702	fe80::ac96:4131:35e5:bf3f	ff02::2	MLE	123 Parent Request
12	154.963718	fe80::ac96:4131:35e5:bf3f	ff02::1	MLE	138 Announce
13	154.982981	fe80::7c00:42bc:ec35:fa0	ff02::1	MLE	138 Announce
14	154.997349	fe80::7c00:42bc:ec35:fa0	fe80::ac96:4131:35e5:bf3f	MLE	143 Announce
15	155.000389			IEEE 802.15.4	65 Ack
16	155.859172	fe80::ac96:4131:35e5:bf3f	ff02::2	MLE	123 Parent Request
17	156.162826	fe80::7c00:42bc:ec35:fa0	fe80::ac96:4131:35e5:bf3f	MLE	177 Parent Response
18	156.166954			IEEE 802.15.4	65 Ack

Figure 5.19: Process of attachment, initial interaction

The MLE Parent Response will contain the following TLV (fig 5.20):

- Source Address TLV.
- Leader Data TLV.
- Link-layer Frame Counter TLV.
- MLE Frame Counter TLV.
- Response TLV.
- Challenge TLV, which is the same that the parent has received from the attaching device.
- Link Margin TLV.
- Connectivity TLV.
- Version TLV.

```

▼ Mesh Link Establishment
  Security Suite: 802.15.4 Security (0x00)
  > Auxiliary Security Header
    Command: Parent Response (10)
  > TLV (Source Address = c4:00)
  > TLV (Leader Data)
  > TLV (Link Layer Frame Counter = 2000)
  > TLV (MLE Frame Counter = 2072)
  > TLV (Response = 92b58e627238f91d)
  > TLV (Unknown (86))
  > TLV (Challenge = f9453dc7df620b03)
  > TLV (Link Margin = 41dB)
  > TLV (Connectivity)
  > TLV (Version = 3)

```

Figure 5.20: Details of the TLVs provided by the MLE Parent Resonse

Then, based on each device's connectivity, through the Connectivity TLV, the new device will decide where to attach and who will become his parent. It will

consider the two-way link quality, the parent priority and the connectivity data in that order of priority when deciding. Then, the new device unicast a MLE Child ID Request to the chosen parent requesting the following TLV (fig 5.21):

- Response TLV.
- Link-layer Frame Counter TLV.
- MLE Frame Counter TLV.
- Model TLV.
- Timeout TLV.
- Version TLV.
- Address Registration TLV.
- TLV Request TLV: Address16 (Network Data and/or Route).
- Active Timestamp TLV.
- Pending Timestamp TLV.

```

▼ Mesh Link Establishment
  Security Suite: 802.15.4 Security (0x00)
  > Auxiliary Security Header
    Command: Child ID Request (11)
  > TLV (Response = f9453dc7df620b03)
  > TLV (Link Layer Frame Counter = 1001)
  > TLV (MLE Frame Counter = 1059)
  > TLV (Mode = 0f)
  > TLV (Timeout = 240)
  > TLV (Version = 3)
  > TLV (TLV Request)

```

Figure 5.21: Details of the TLVs provided by the MLE Child Id Request

If the Child ID Request does not contain an MLE Active Timestamp TLV the Parent must include an Active Operational Dataset TLV in the Child ID Response. If the Parent has a valid Pending Operational Dataset and the Child ID Request does not contain a MLE Pending Timestamp TLV, or if contains an MLE Pending Timestamp TLV that does not match the Parent's Pending Timestamp, the Parent must include a Pending Operational Dataset TLV in the Child ID Response.

Now, the parent will answer with a MLD Child ID Response where it will include:

- Source Address TLV.
- Leader Data TLV.

- Address16 TLV
- Network Data TLV
- Route64 TLV
- Address Registration TLV.
- Active Operational Dataset TLV.
- Pending Operational Dataset TLV.

So the child can now be allocated in the network. The parent will add this new device to its Child Table. To end with the attachment, the child will ask for creating a link between itself and the Leader through a Link Request MLE message, starting a new handshake with a Challenge TLV. As the links are just one way, the Leader will repply accepting that link and will request to create its own link to the child (MLE Link Accept and Request message). After the acceptance of this last link, the attachment will be finished.

19	156.610218	fe80::ac96:4131:35e5:bf3f	fe80::7c00:42bc:ec35:fa0	MLE	148 Child ID Request
20	156.613417			IEEE 802.15.4	65 Ack
21	156.633186	7e:00:42:bc:ec:35:0f:a0	ae:96:41:31:35:e5:bf:3f	6LoWPAN	186 Data, Dst: ae:96:41:31:35:e5:bf:3f, Src: 7e:00:42:bc:ec:35:0f:a0
22	156.637602			IEEE 802.15.4	65 Ack
23	156.653457	fe80::7c00:42bc:ec35:fa0	fe80::ac96:4131:35e5:bf3f	MLE	180 Child ID Response
24	156.657681			IEEE 802.15.4	65 Ack
25	159.160312	::ff:fe00:c401	::ff:fe00:c400	UDP	116 61631 → 61631 Len=29
26	159.162488			IEEE 802.15.4	65 Ack
27	159.174491	::ff:fe00:c400	::ff:fe00:c401	UDP	112 61631 → 61631 Len=25
28	159.176539			IEEE 802.15.4	65 Ack
29	159.192359	fe80::ac96:4131:35e5:bf3f	ff02::2	MLE	134 Link Request
30	159.206332	fe80::ac96:4131:35e5:bf3f	ff02::1	MLE	138 Announce
31	159.405713	fe80::7c00:42bc:ec35:fa0	fe80::ac96:4131:35e5:bf3f	MLE	164 Link Accept and Request
32	159.409425			IEEE 802.15.4	65 Ack
33	159.426352	fe80::ac96:4131:35e5:bf3f	fe80::7c00:42bc:ec35:fa0	MLE	151 Link Accept
34	159.429647			IEEE 802.15.4	65 Ack
35	159.766604	fe80::7c00:42bc:ec35:fa0	ff02::1	MLE	130 Advertisement
36	160.035269	fe80::ac96:4131:35e5:bf3f	ff02::1	MLE	130 Advertisement

Figure 5.22: Process of attachment, the parent allocates the new device into the network as its child

Then, it can be seen that the routing table of each device has been updated (fig 5.23). It can be observed that the routing table includes a new device with his link quality information appart form itself.

```

  Mesh Link Establishment
    Security Suite: 802.15.4 Security (0x00)
  > Auxiliary Security Header
    Command: Advertisement (4)
  > TLV (Source Address = c4:00)
  > TLV (Leader Data)
  > TLV (Route64)
    Type: Route64 (9)
    Length: 11
    ID Sequence: 114
    Assigned Router ID Mask: 0080000000004000
  > Routing Table Entry: 0xf1 (8)
    11.. .... = Neighbor Out Link Quality: 3
    ..11 .... = Neighbor In Link Quality: 3
    .... 0001 = Router Cost: 1
  > Routing Table Entry: 0x01 (49)
    00.. .... = Neighbor Out Link Quality: 0
    ..00 .... = Neighbor In Link Quality: 0
    .... 0001 = Router Cost: 1

```

Figure 5.23: Details of the new routing table of the leader, through a MLE Advertisement message

After attaching the first device, a new device is attached. The process is the same but now it must interact with the two devices already deployed to decide which one will be his Parent, and to create links. In the routing table of the Leader, a new device can be observed.

52	275.750210	fe80::c0bd:92d:fa22:1fbc	ff02::2	MLE	121 Link Request
53	277.922627	fe80::c0bd:92d:fa22:1fbc	ff02::2	MLE	123 Parent Request
54	278.674354	fe80::c0bd:92d:fa22:1fbc	ff02::2	MLE	123 Parent Request
55	280.222425	fe80::c0bd:92d:fa22:1fbc	ff02::2	MLE	123 Parent Request
56	280.972778	fe80::c0bd:92d:fa22:1fbc	ff02::2	MLE	123 Parent Request
57	282.224109	fe80::c0bd:92d:fa22:1fbc	ff02::2	MLE	123 Parent Request
58	283.554593	fe80::c0bd:92d:fa22:1fbc	ff02::1	MLE	138 Announce
59	283.573875	fe80::ac96:4131:35e5:bf3f	ff02::1	MLE	138 Announce
60	283.588151	fe80::ac96:4131:35e5:bf3f	fe80::c0bd:92d:fa22:1fbc	MLE	143 Announce
61	283.591190			IEEE 802.15.4	65 Ack
62	283.609196	fe80::7c00:42bc:ec35:fa0	fe80::c0bd:92d:fa22:1fbc	MLE	143 Announce
63	284.345941	fe80::c0bd:92d:fa22:1fbc	ff02::2	MLE	123 Parent Request
64	284.611626	fe80::ac96:4131:35e5:bf3f	fe80::c0bd:92d:fa22:1fbc	MLE	177 Parent Response
65	284.615754			IEEE 802.15.4	65 Ack
66	284.798270	fe80::7c00:42bc:ec35:fa0	fe80::c0bd:92d:fa22:1fbc	MLE	177 Parent Response
67	284.802397			IEEE 802.15.4	65 Ack
68	285.097681	fe80::c0bd:92d:fa22:1fbc	fe80::ac96:4131:35e5:bf3f	MLE	148 Child ID Request
69	285.100881			IEEE 802.15.4	65 Ack
70	285.120701	ae:96:41:31:35:e5:bf:3f	c2:bd:09:2d:fa:22:1f:bc	6LoWPAN	186 Data, Dst: c2:bd:09:2d:fa:22:1f:bc, Src: ae:96:41:31:35:e5:bf:3f
71	285.125117			IEEE 802.15.4	65 Ack
72	285.140921	fe80::ac96:4131:35e5:bf3f	fe80::c0bd:92d:fa22:1fbc	MLE	181 Child ID Response
73	285.145177			IEEE 802.15.4	65 Ack
74	285.834199	fe80::7c00:42bc:ec35:fa0	ff02::1	MLE	130 Advertisement
75	285.850713	fe80::c0bd:92d:fa22:1fbc	fe80::7c00:42bc:ec35:fa0	MLE	136 Link Request
76	285.853528			IEEE 802.15.4	65 Ack
77	285.866526	fe80::7c00:42bc:ec35:fa0	fe80::c0bd:92d:fa22:1fbc	MLE	141 Link Accept
78	285.869502			IEEE 802.15.4	65 Ack

Figure 5.24: Overview of the attachment of the 3rd device (1)

107	340.746415	fe80::c0bd:92d:fa22:1fbc	ff02::2	MLE	134	Link Request
108	341.058262	fe80::7c00:42bc:ec35:fa0	fe80::c0bd:92d:fa22:1fbc	MLE	164	Link Accept and Request
109	341.061974			IEEE 802.15.4	65	Ack
110	341.079438	fe80::c0bd:92d:fa22:1fbc	fe80::7c00:42bc:ec35:fa0	MLE	151	Link Accept
111	341.082733			IEEE 802.15.4	65	Ack
112	341.300355	fe80::c0bd:92d:fa22:1fbc	ff02::1	MLE	131	Advertisement
113	341.317870	fe80::ac96:4131:35e5:bf3f	fe80::c0bd:92d:fa22:1fbc	MLE	139	Link Request
114	341.320782			IEEE 802.15.4	65	Ack
115	341.334444	fe80::c0bd:92d:fa22:1fbc	fe80::ac96:4131:35e5:bf3f	MLE	151	Link Accept
116	341.337740			IEEE 802.15.4	65	Ack

Figure 5.25: Overview of the attachment of the 3rd device (2)

```

  ▾ Mesh Link Establishment
    Security Suite: 802.15.4 Security (0x00)
  > Auxiliary Security Header
    Command: Advertisement (4)
  > TLV (Source Address = 78:00)
  > TLV (Leader Data)
  ▾ TLV (Route64)
    Type: Route64 (9)
    Length: 12
    ID Sequence: 133
    Assigned Router ID Mask: 0080000200004000
  > Routing Table Entry: 0xf1 (8)
  > Routing Table Entry: 0x01 (30)
  > Routing Table Entry: 0xf1 (49)

```

Figure 5.26: Routing table of the leader broadcasted in a MLE Advertisement message

To end with this first test, the last device attached (the child) will be disconnected, and the performance of the network will be analysed in Wireshark.

When the leader does not receive any broadcast MLE Advertisement message of the child, after a timeout. it will start asking for routing information to the Router (the parent of the disconnected child). After some retries, it will conclude that the device has been detached from the network. So it will update his routing tables as shown in the figure 5.27.

```
▼ Mesh Link Establishment
  Security Suite: 802.15.4 Security (0x00)
  > Auxiliary Security Header
    Command: Advertisement (4)
  > TLV (Source Address = 78:00)
  > TLV (Leader Data)
  ▼ TLV (Route64)
    Type: Route64 (9)
    Length: 12
    ID Sequence: 133
    Assigned Router ID Mask: 0080000200004000
  > Routing Table Entry: 0xf1 (8)
  > Routing Table Entry: 0x01 (30)
  > Routing Table Entry: 0xf1 (49)
```

Figure 5.27: Routing table of the leader broadcasted in a MLE Advertisement message

If the Leader is the one which is detached, another Router will assume the role.

5.2.3. Visibility of the network

In this short test, the purpose is to observe what would see a device which can't decypher the MLE messages, because it does not know the network key. Taking in account the same example used previously, now the decryption key added to Wireshark has been deleted.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	fe80::7c00:42bc:ec35:fa0	ff02::1	MLE	129	
2	21.882158	fe80::7c00:42bc:ec35:fa0	ff02::1	MLE	129	
3	53.413830	fe80::7c00:42bc:ec35:fa0	ff02::1	MLE	129	
4	90.696814	fe80::7c00:42bc:ec35:fa0	ff02::1	MLE	129	
5	131.952127	fe80::7c00:42bc:ec35:fa0	ff02::1	MLE	129	
6	147.493212	fe80::ac96:4131:35e5:bf3f	ff02::2	MLE	121	
7	149.344182	fe80::ac96:4131:35e5:bf3f	ff02::2	MLE	123	
8	150.095904	fe80::ac96:4131:35e5:bf3f	ff02::2	MLE	123	
9	151.632973	fe80::ac96:4131:35e5:bf3f	ff02::2	MLE	123	
10	152.382319	fe80::ac96:4131:35e5:bf3f	ff02::2	MLE	123	
11	153.633702	fe80::ac96:4131:35e5:bf3f	ff02::2	MLE	123	
12	154.963718	fe80::ac96:4131:35e5:bf3f	ff02::1	MLE	138	
13	154.982981	fe80::7c00:42bc:ec35:fa0	ff02::1	MLE	138	
14	154.997349	fe80::7c00:42bc:ec35:fa0	fe80::ac96:4131:35e...	MLE	143	
15	155.000389			IEEE 802.15.4	65	Ack
16	155.859172	fe80::ac96:4131:35e5:bf3f	ff02::2	MLE	123	
17	156.162826	fe80::7c00:42bc:ec35:fa0	fe80::ac96:4131:35e...	MLE	177	
18	156.166954			IEEE 802.15.4	65	Ack
19	156.610218	fe80::ac96:4131:35e5:bf3f	fe80::7c00:42bc:ec3...	MLE	148	
20	156.613417			IEEE 802.15.4	65	Ack
21	156.633186	7e:00:42:bc:ec:35:0f:a0	ae:96:41:31:35:e5:b...	IEEE 802.15.4	186	Data,
22	156.637602			IEEE 802.15.4	65	Ack
23	156.653457	7e:00:42:bc:ec:35:0f:a0	ae:96:41:31:35:e5:b...	IEEE 802.15.4	180	Data,
24	156.657681			IEEE 802.15.4	65	Ack
25	159.160312	0xc401	0xc400	IEEE 802.15.4	116	Data,

```

<
> Frame 205: 140 bytes on wire (1120 bits), 140 bytes captured (1120 bits) on interface \\.\pipe\tiwspc_data, id 0
> Internet Protocol Version 4, Src: 192.168.1.3, Dst: 192.168.1.3
> User Datagram Protocol, Src Port: 17754, Dst Port: 17754
> ZigBee Encapsulation Protocol, Channel: 26, Length: 80
> IEEE 802.15.4 Data, Dst: ae:96:41:31:35:e5:bf:3f, Src: 7e:00:42:bc:ec:35:0f:a0
> 6LoWPAN, Src: fe80::7c00:42bc:ec35:fa0, Dest: fe80::ac96:4131:35e5:bf3f
> Internet Protocol Version 6, Src: fe80::7c00:42bc:ec35:fa0, Dst: fe80::ac96:4131:35e5:bf3f
> User Datagram Protocol, Src Port: 19788, Dst Port: 19788
▼ Mesh Link Establishment
  Security Suite: 802.15.4 Security (0x00)
  > Auxiliary Security Header
  > [Expert Info (Warning/Undecoded): No encryption key set - can't decrypt]
  > Data (33 bytes)

```

Figure 5.28: Frames captured by the sniffer without knowing the decryption key

In the figure above, the sniffer could not know the purpose of each message, the type of MLE message is, and the information provided in the TLV. So with this simple example, it is demonstrated that the MLE fields are secure end to end. In addition, sniffers are configured to capture the frames of one determined 802.15.4 PHY channel. If the network communication goes through one of the other 802.15.4 PHY channels, it could not get them.

5.2.4. End to end connectivity

The last test that will be detailed is the end to end connectivity through a multihop routing. For this purpose, a Thread network composed by one Leader and two Routers will be created. The routing table of the leader will be studied and ping messages will be sent from the Leader to one of the Routers, proving how it reaches to the destination and it replies.

Finally, in order to see the smart routing process that the Thread architecture has, a new Router will be added into the existing network. The restructuration of the routing tables will be discussed, and then an older router will be detached, to observe the same performance of auto reconfiguration of the network.

So, when the environment is already set, a Thread network composed by a Leader and two Routers is deployed, the leader has the routing table shown in the figure

below. It also included an example of a Ping command, used to send 5 ICMP echo request messages, sent by the Leader, to the Router identified as 36 (routing through '63' is indicated when the receiver Router is itself).

```

router table
| ID | RLOC16 | Next Hop | Path Cost | LQ In | LQ Out | Age | Extended MAC | Link |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 8 | 0x2000 | 63 | 0 | 0 | 0 | 0 | ae96413135e5bf3f | 0 |
| 30 | 0x7800 | 36 | 2 | 3 | 3 | 21 | 0e6acdfec8bd66b | 1 |
| 36 | 0x9000 | 30 | 2 | 2 | 1 | 0 | c2bd092dfa221fbc | 1 |

Done
> ping fe80:0:0:c0bd:92d:fa22:1fbc 10 5 2
18 bytes from fe80:0:0:c0bd:92d:fa22:1fbc: icmp_seq=30 hlim=64 time=25ms
18 bytes from fe80:0:0:c0bd:92d:fa22:1fbc: icmp_seq=31 hlim=64 time=25ms
18 bytes from fe80:0:0:c0bd:92d:fa22:1fbc: icmp_seq=32 hlim=64 time=26ms
18 bytes from fe80:0:0:c0bd:92d:fa22:1fbc: icmp_seq=33 hlim=64 time=25ms
18 bytes from fe80:0:0:c0bd:92d:fa22:1fbc: icmp_seq=34 hlim=64 time=25ms
5 packets transmitted, 5 packets received. Packet loss = 0.0%. Round-trip min/avg/max = 25/25.200/26 ms.
Done

```

Figure 5.29: Leader Routing Table and ping example

As can be observed, the ICMP echo request message was delivered successfully and a ICMP echo reply was returned from the Router to the Leader. The ping's output also provides information about the average Round-trip time, the time to reach the destination and come back with the reply, of 26.2 ms.

In Wireshark the interaction between the devices was demonstrated as 5 the different ICMP echo requests and replies can be seen easily (fig 5.30).

195 741.425686	fe80::ac96:4131:35e5:bf3f	ff02::1	MLE	131 Advertisement
196 764.124552	fe80::c6a:cdfe:cc8b:d66b	ff02::1	MLE	131 Advertisement
197 771.953109	fe80::ac96:4131:35e5:bf3f	ff02::1	MLE	131 Advertisement
198 791.136166	fe80::ac96:4131:35e5:bf3f	fe80::c0bd:92d:fa22...	ICMPv6	114 Echo (ping) request id=0x000a, seq=30, hop limit=64 (reply in 200)
199 791.138277		IEEE 802.15.4		65 Ack
200 791.148773	fe80::c0bd:92d:fa22:1fbc	fe80::ac96:4131:35e...	ICMPv6	114 Echo (ping) reply id=0x000a, seq=30, hop limit=64 (request in 198)
201 791.150884		IEEE 802.15.4		65 Ack
202 793.136168	fe80::ac96:4131:35e5:bf3f	fe80::c0bd:92d:fa22...	ICMPv6	114 Echo (ping) request id=0x000a, seq=31, hop limit=64 (reply in 204)
203 793.138280		IEEE 802.15.4		65 Ack
204 793.148773	fe80::c0bd:92d:fa22:1fbc	fe80::ac96:4131:35e...	ICMPv6	114 Echo (ping) reply id=0x000a, seq=31, hop limit=64 (request in 202)
205 793.150885		IEEE 802.15.4		65 Ack
206 795.137265	fe80::ac96:4131:35e5:bf3f	fe80::c0bd:92d:fa22...	ICMPv6	114 Echo (ping) request id=0x000a, seq=32, hop limit=64 (reply in 208)
207 795.139376		IEEE 802.15.4		65 Ack
208 795.149990	fe80::c0bd:92d:fa22:1fbc	fe80::ac96:4131:35e...	ICMPv6	114 Echo (ping) reply id=0x000a, seq=32, hop limit=64 (request in 206)
209 795.152102		IEEE 802.15.4		65 Ack
210 797.136371	fe80::ac96:4131:35e5:bf3f	fe80::c0bd:92d:fa22...	ICMPv6	114 Echo (ping) request id=0x000a, seq=33, hop limit=64 (reply in 211)
211 797.148980	fe80::c0bd:92d:fa22:1fbc	fe80::ac96:4131:35e...	ICMPv6	114 Echo (ping) reply id=0x000a, seq=33, hop limit=64 (request in 210)
212 797.151092		IEEE 802.15.4		65 Ack
213 797.849069	fe80::c0bd:92d:fa22:1fbc	ff02::1	MLE	131 Advertisement
214 798.998410	fe80::c6a:cdfe:cc8b:d66b	ff02::1	MLE	131 Advertisement
215 799.136272	fe80::ac96:4131:35e5:bf3f	fe80::c0bd:92d:fa22...	ICMPv6	114 Echo (ping) request id=0x000a, seq=34, hop limit=64 (reply in 217)
216 799.138384		IEEE 802.15.4		65 Ack
217 799.149193	fe80::c0bd:92d:fa22:1fbc	fe80::ac96:4131:35e...	ICMPv6	114 Echo (ping) reply id=0x000a, seq=34, hop limit=64 (request in 215)
218 799.151385		IEEE 802.15.4		65 Ack

Figure 5.30: Ping process

Now, a new router has been attached and the table has been reconfigured. The same Ping example has been tested successfully. The following figure shows how the routing information has changed and adapted to the current situation: Router 49 has been added and can be reached through Router 30. At the same time, Router 30 has changed his routing topology and now can be accessed over the Router 49 instead of Router 36 of the previous test.


```

> router table
| ID | RLOC16 | Next Hop | Path Cost | LQ In | LQ Out | Age | Extended MAC | Link |
+---+-----+-----+-----+-----+-----+---+-----+----+
| 8 | 0x2000 | 63 | 0 | 0 | 0 | 0 | ae96413135e5bf3f | 0 |
| 30 | 0x7800 | 49 | 1 | 3 | 3 | 5 | 0e6acdfec8bd66b | 1 |
| 36 | 0x9000 | 30 | 2 | 2 | 1 | 3 | c2bd092dfa221fbc | 1 |
| 49 | 0xc400 | 30 | 1 | 3 | 3 | 3 | 7e0042bcec350fa0 | 1 |
+---+-----+-----+-----+-----+-----+---+-----+----+
Done
Link to config
> ping fe80:0:0:0:c0bd:92d:fa22:1fbc 10 5 2
18 bytes from fe80:0:0:0:c0bd:92d:fa22:1fbc: icmp_seq=41 hlim=64 time=25ms
18 bytes from fe80:0:0:0:c0bd:92d:fa22:1fbc: icmp_seq=42 hlim=64 time=25ms
18 bytes from fe80:0:0:0:c0bd:92d:fa22:1fbc: icmp_seq=43 hlim=64 time=27ms
18 bytes from fe80:0:0:0:c0bd:92d:fa22:1fbc: icmp_seq=44 hlim=64 time=25ms
18 bytes from fe80:0:0:0:c0bd:92d:fa22:1fbc: icmp_seq=45 hlim=64 time=25ms
5 packets transmitted, 5 packets received. Packet loss = 0.0%. Round-trip min/avg/max = 25/25.400/27 ms.
Done

```

Figure 5.31: Leader Routing Table and ping example

Finally, the Router 30 was detached, and after a while, the routing was reconfigured as shown in the following figure:

```

router table
| ID | RLOC16 | Next Hop | Path Cost | LQ In | LQ Out | Age | Extended MAC | Link |
+---+-----+-----+-----+-----+-----+---+-----+----+
| 8 | 0x2000 | 63 | 0 | 0 | 36 | 0 | ae96413135e5bf3f | 0 |
| 36 | 0x9000 | 49 | 4 | 2 | 49 | 1 | c2bd092dfa221fbc | 1 |
| 49 | 0xc400 | 36 | 4 | 3 | 3 | 19 | 7e0042bcec350fa0 | 1 |
+---+-----+-----+-----+-----+-----+---+-----+----+

```

Figure 5.32: Leader Routing Table

So, to reach the Router 36 now it must go through the Router 49 instead of the previously used Router 30. The same Ping command already used was tested and was able to reach Router 36 and come with a response as the command's output explains.

```

> ping fe80:0:0:0:c0bd:92d:fa22:1fbc 10 5 2
18 bytes from fe80:0:0:0:c0bd:92d:fa22:1fbc: icmp_seq=46 hlim=64 time=26ms
18 bytes from fe80:0:0:0:c0bd:92d:fa22:1fbc: icmp_seq=47 hlim=64 time=25ms
18 bytes from fe80:0:0:0:c0bd:92d:fa22:1fbc: icmp_seq=48 hlim=64 time=25ms
18 bytes from fe80:0:0:0:c0bd:92d:fa22:1fbc: icmp_seq=49 hlim=64 time=25ms
18 bytes from fe80:0:0:0:c0bd:92d:fa22:1fbc: icmp_seq=50 hlim=64 time=25ms
5 packets transmitted, 5 packets received. Packet loss = 0.0%. Round-trip min/avg/max = 25/25.200/26 ms.
Done

```

Figure 5.33: Ping command from Leader to Router

Chapter 6

Conclusions

6.1. General Conclusion

In this thesis, a general study of the Thread Protocol has been provided. Moreover, the Open Source Thread (the 1.1.0 version) implementation, OpenThread has also been studied. Last but not least, a Thread network has been implemented to observe its performance.

In summary, Thread is built upon a foundation of existing standards and can fulfill the requirements of low power, resilience, ip-based, security and friendly use. On top of that, the application layer runs independently supporting a wide range of applications which can use Thread for the interaction of their devices.

With the cli interface, it is very easy to set up and observe the different functionalities Thread offers by using the Openthread API. Not only a Thread network has been created easily, but also a basic network interaction, through some commands such as the Discover command and the Wireshark, has been tested. In overall, Thread is one of the leading protocols for LR-WPAN networks.

Eventhough the overall feedback was good, there were some problems related with flashing the examples into different devices, such as MakerDiary and also some Zolertia devices were not working well or the documentation to do it was confusing or outdated. This prevented me to reach the nexts steps into studying that protocol. Also the scarcity of the devices or the malfunctioning of some of them prevented to have a more broad overview of the technology.

For doing all these studies and tests, a lot of documentation has been used. The Thread documentation is very messy and even the specification file is very difficult to follow. Also, it is said that all the protocols, such as the IEEE 802.15.4 involved are open so Thread is built on top of open and accessible protocols, but then I see that I need an IEEE subscription to see them, which may not be affordable for everyone. I could get acces thanks to the UPC ebib functionallity which let me dive in these files.

6.2. Future work

6.2.1. Thread

Thread must keep evolving, including new features and functionalities. Till now, Thread has achieved the following features: it is IP-based mesh networking solution that is secure, reliable, scalable and optimized for low power operation. In addition to it, the newest version provided support for management of the network via bluetooth. Nevertheless it needs a tunneling solution until ISPs provide native IPv6 to the home.

Not only in the technical aspects, and something which is found in other IoT solutions, Thread has to improve. Eventhough seems very easy to build the network, for non-engineer users remains quite complicated. In order to extend the reachability of the technology, it must provide very simple solutions for unexperienced users.

Another thing to consider for improving in Thread, and also in almost each IoT solution, is the cost of the devices. The Thread certified products [6] are very expensive and creating a home automated network requires a very high investment, which are not affordable for the majority of the population.

All users are looking for saving money and commodity, which a smart network provides both. IoT solutions have a great future and for me, eventhough I am not an expert, the key improvements are the cost of the devices involved and the complexity of creating an automated network.

As already mentioned, the Thread documentation must be reworked. Now there are many documents explaining different things and is not even clear in the main specification file. In order to attract more users (and developers for the Thread implementations), the Thread Group must work in producing a more ordered and clean documentation.

6.2.2. OpenThread

OpenThread must always seek for addapting to the newest versions of the Thread protocol. For example, now it must update to support the Bluetooth managment, as it does Thread 1.2.0.

As it is an Open Source implementation, all the users are free to add their improvements. In the future, it may have a lot of functionalities which will improve the performance of Openthread, thus adding also new features.

As Thread Group's objective is to make Thread widely available, the Open Source implementations such as OpenThread, which they directly depend on how many users are involved in developing it, would be very benefited from the nature and goal of this protocol.

6.2.3. This thesis

This thesis has provided not only a summary of the theory of how the Thread protocol work, but also has shown some practical implementations. Eventhough, it is just an introduction and it needs to be more developed. Different studies are still needed to test its performance, such as obtaining the metrics of the messaging in the network or testing the network in different enviroments with different interferences, to test its reachability.

Many problems I had when trying to configure the different devices (for example, I was unable to configure and program the MakerDiary devices, the documentation for programming the Zolertia devices was outdated and needs a review...) and they prevented me from reaching further steps, so a very concise documentation, which in part is provided here is needed.

I hope that future students or Thread users would be able, following all the step I have described here, to continue testing and studying this protocol. This would generate feedback which could improve the performance of Thread. In my personal opinion, the development of IoT solutions, like Thread, are very important when considering that IoT solutions are one of the key technologies in our future, if not already in the present, and it will be one of the technologies more used, if not the most.

Bibliography

- [1] ALLIANCE, Connectivity S.: *Matter github*. – URL <https://github.com/project-chip/connectedhomeip#readme>
- [2] ALLIANCE, Connectivity S.: *Matter web site*. – URL <https://csa-iot.org/all-solutions/matter/>
- [3] CERF, Vint ; KAHN, Bob: *TCP*. – URL <https://datatracker.ietf.org/doc/html/rfc675>
- [4] COLINA, Antonio L. ; MARCO, Alvaro V. ; BAGULA, Zennaro A. ; PIETROSEMOLI, Ermanno: *IoT in 5 days*. 2016. – URL <http://www.iet.unipi.it/c.vallati/files/IoTinfivedays-v1.1.pdf>
- [5] GROUP, IETF: *DTLS rfc*. – URL <https://datatracker.ietf.org/doc/html/rfc6347>
- [6] GROUP, Thread: *Thread certified products*. – URL <https://www.threadgroup.org/What-is-Thread/Thread-Benefits#certifiedproducts>
- [7] GROUP, Thread: *Thread*. 2014. – URL <https://www.threadgroup.org/>
- [8] GROUP, Thread: *Thread*. 2014. – URL <https://www.threadgroup.org/support#Whitepapers>
- [9] IEEE: *802.15.4 Mac and Phy*. 2007. – URL <https://ieeexplore.ieee.org/document/4152704>
- [10] INSTRUMENTS, Texas: *CC2531EMK IEEE802.15.4 sniffer*. – URL <https://www.mouser.es/ProductDetail/Texas-Instruments/CC2531EMK?qs=Gbw5dzAYBFdEggnrw1cUyQ%3D%3D>
- [11] LAB, Nest: *OpenThread*. 2016. – URL <https://openthread.io/>
- [12] LAB, Nest: *OpenThread Github*. 2016. – URL <https://github.com/openthread>
- [13] REED, David P.: *UDP*. – URL <https://datatracker.ietf.org/doc/html/rfc768>
- [14] ZOLERTIA: *Zolertia Firefly Github*. – URL <https://github.com/Zolertia/Resources/wiki/Firefly>

-
- [15] ZOLERTIA: *Zolertia github*. – URL <https://github.com/Zolertia>
- [16] ZOLERTIA: *Zolertia Re-Mote Github*. – URL <https://github.com/Zolertia/Resources/wiki/RE-Mote>
- [17] ZOLERTIA: *Zolertia web site*. – URL <https://zolertia.io/>