



**UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH**

ANALYZING AND EVALUATING NETWORK PROTOCOLS IN IoT



December, 10, 2014

**BARCELONA SCHOOL OF INFORMATICS (FIB)
TECHNICAL UNIVERSITY OF CATALONIA (UPC)
UNIVERSITAT POLITÈCNICA DE CATALUNYA**

**MASTER IN COMPUTER ARCHITECTURE, NETWORKS AND
SYSTEMS**

AUTHOR: FARNOOSH FAROKHMANESH
farnoosh.farokhmanesh@est.fib.upc.edu
farnoosh_farokhmanesh@yahoo.com

Supervisor: Jose Maria Barcelo Ordinas
joseb@ac.upc.edu

ACKNOWLEDGEMENTS

I am particularly grateful to my supervisor Prof. Jose Maria Barcelo for supervising my thesis. I would like to express my sincere appreciation to my Prof Jorge Garcia Vidal for the guidance and his useful advices. Actually I feel lucky to have them as my tutor and Prof. and working with them and benefiting from their experience was a great opportunity for me to develop my technical and personal aspects.

Special thank my family and all my friends for the support, encouragement and confidence they gave to me during my Master Degree. I would like to thank everyone in two divisions for creating a harmony and friendly environment to work in.

ABSTRACT

In recent years, there has been an increasing interest in "Internet of Things ". The future sensor networks are envisioned as comprising heterogeneous devices assisting to a large range of applications. Enabling small wireless sensors as easy to access as web servers is increasingly gaining attraction in our day to day lives. At this time, the development of the Constrained Application Protocol (CoAP) combining with 6LowPAN has made it possible to provide wireless sensor nodes with web service functionalities.

This thesis is aimed at evaluation of the performance of an HTTP/TCP and CoAP/ UDP [23] network protocols over IEEE 802.15.4 communication protocol based on Contiki 2.7 and sensor nodes Zolertia Z1[16]. WSN architecture consists of a Server, RPL Border Router, and Client. As our implementation targets real deployment, we faced some issues regarding different technologies, supported protocols, suitable hardware selection and also environment obstacles.

The evaluation was carried out on the basis of Transmissions delay and Response time. All experiments were conducted considering a client querying an embedded server to discovery of "Hello-World" resource hosted by a constrained serve. The response time was calculated for the case where the server was at different distances with different transaction length.

The test and experimental results have shown that CoAP/UDP work properly rather than HTTP/TCP, based on IEEE 802.15.4, since the CoAP/UDP protocol have a poor effect on the current WSNs bandwidth and it could enhance network performance like packet reception ratio, throughput and could save more energy

Contents

INTRODUCTION	1
1.1 MOTIVATION AND PROBLEM STATEMENT	3
1.2 THESIS OBJECTIVE.....	4
1.3 THESIS STRUCTURE.....	4
2 FUNDAMENTALS OF WIRELESS SENSOR NETWORKS	5
2.1 WIRELESS SENSOR NETWORKS	5
2.1.1 CHARACTERISTICS OF WSN	5
2.2 WIRELESS SENSOR NODES ARCHITECTURE.....	7
2.2.1 ZOLERTIA Z1.....	8
2.3 IEEE 802.15.4 - COMMUNICATIONS PROTOCOL	11
2.3.1 PHYSICAL LAYER.....	13
2.3.2 MAC LAYER	13
2.4 6LOWPAN ADAPTATION LAYER	13
2.4.1 HEADER COMPRESSION	14
2.4.2 FRAGMENTATION	15
2.5 IPV6/RPL NETWORK LAYER	16
2.5.1 IPV6	16
2.5.2 RPL (ROUTING PROTOCOL FOR LOW POWER AND LOSSY NETWORKS) ..	16
2.6 UDP.....	18
2.7 RESTFUL ARCHITECTURE.....	18
2.8 CoAP.....	19
2.8.1. CoAP FEATURES	20
2.8.2 CoAP IMPLEMENTATIONS.....	21
2.8.3 MESSAGE LAYER MODEL	23
2.8.4 REQUEST/RESPONSE LAYER MODEL.....	23
2.8.5 URI SCHEME	26
2.8.6 BLOCKWISE TRANSFER.....	26
2.9 CONTIKI OPERATING SYSTEM.....	27
2.9.1 CONTIKI STRUCTURE.....	28
3 SYSTEM ARCHITECTURE	29
3.1 Hardware.....	29
3.1.1 Wasp mote sensor node	29
3.1.2 Z1 Zolertia	30
3.2 Challenges.....	31
3.2.1 Simulation VS. Real World Testing	31
3.2.2 Selection of the hardware.....	32
3.2.3 Selection of the Operating System.....	33
3.3 System architecture	34

3.3.1 RPL Border Router	36
3.3.2 CoAP and HTTP Server.....	36
3.3.3 CoAP and HTTP Client	37
4 EXPERIMENTAL ENVIRONMENT.....	38
4.1 Software tools	38
4.1.1 Copper plug-in	38
4.1.2 Wireshark	39
4.2 Scenarios	40
4.2.1 CoAP scenario	41
4.2.2 Coap Block-wise implementation.....	45
4.2.3 HTTP scenario	47
5 EXPERIMENTAL RESULTS	51
5.1 CoAP scenario	51
5.2 HTTP scenario	53
5.3 Network layers overhead	55
5.4 CoAP VS. HTTP.....	56
5.5 Qualitative Results	59
6 CONCLUSIONS AND FUTURE WORK	60
6.1 Conclusions.....	60
6.2 Summary conclusion.....	61
6.3 Future Work	61
BIBLIOGRAPHY AND REFERENCES	62

LIST OF FIGURES

1.1:	ARCHITECTURE OF A TYPICAL WSN.....	3
2.1:	WSN STRUCTURE.....	7
2.2:	ARCHITECTURE OF WIRELESS SENSOR NODE.....	8
2.3:	WIRELESS SENSOR NODE -ZOLERTIA Z1	9
2.4:	IP PROTOCOL STACK FOR LOW_POWER, RELIABLE WSN	11
2.5:	GENERAL PACKET FRAME FORMAT.....	12
2.6:	IEEE802.15.4 FRAME FORMAT	15
2.7:	6LoWPAN PACKET STRUCTURE OF THE FRAGMENTED PACKET	15
2.8:	IPV6HEDEAR FORMAT	16
2.9:	RPL EXAMPLE :PHYSICAL AND LOGICAL TOPOLOGY OF A NETWORK	18
2.10:	UDP SOURCE HEADER FORMAT.....	18
2.11:	ABSTRACT LAYERING OF COAP	22
2.12:	COAP RELIABLE MESSAGE TRANSMISSION	23
2.13:	UNRELIABLE MESSAGE TRANSPORT.....	23
2.14:	THE SUCCESSFUL AND FAILURE RESPONSE RESULTS OF GET METHOD	24
2.15:	GET REQUEST WITH A SEPARATE RESPONSE.....	24
2.16:	NON CONFIRMABLE REQUEST AND RESPONSE.....	25
2.17:	COAP MESSAGE FORMAT	26
2.18:	BLOCK TRANSFER OPTION	27
3.1:	WASPMOTE SENSOR.....	29
3.2:	WASPMOTE GATEWAY	30
3.3:	WIRELESS SENSORE NODE- ZOLERTIA Z1	31
3.4:	TESTBED NETWORK. HTTP AND COAP.....	35
3.5:	THE COMPLETE ARCHITECTURE.....	36
4.1:	COPPER PLUG-IN INTERFACE	38
4.2:	WIRESHARK TOOL.....	39
4.3:	REAL LABORATORY ENVIRONMENT	41

4.4: ARCHITECTURE OF LABORATORY	41
4.5: COPPER PLUG-IN	43
4.6: NETWORK LAYERS IN CoAP	44
4.7: CoAP/UDP EXCHANGING DATA	44
4.8: BLOCKWISE TRANSFER, EXCHANGING 100 BYTES TRANSACTION USING CHUNKS RESOURCE	46
4.9: BLOCKWISE TRANSFER BLOCK	46
4.10: HTTP TRANSFERRING DATA ARCHITECTURE IN WSN	48
4.11: TCP CONNECTION WITH THE TIMESTAMPTS	48
4.12: TCP3-WAY HANDSHAKE	49
4.13: HYPER TEXT TRANSFER PROTOCOL	50
5.1: CoAP TRANSMISSION DELAY VS. DISTANCE WITH DIFFERENT TRANSACTION SIZES	53
5.2: HTTP TRANSMISSION DELAY IN DIFFERENT LOCATIONS	54
5.3: HTTP PACKETS OF EXCHANGING 50BYTES TRANSACTION	55
5.4: CoAP AND HTTP LARGE DATA TRANSFERRING	57
5.5: FADING EFFECT ON HTTP AND CoAP	58

List of tables

TABLE 2.1: WSN ADVANTAGE AND DISADVANTAGE	6
TABLE 2. 2: ZOLERTIA Z1 PARAMETERS	10
TABLE 2.3: FREQUENCY BANDS AND DATA RATES	12
TABLE 3. 1: Z1 AND WASPMOTE SPECIFICATION.....	32
TABLE 3.2: COMPARISON BETWEEN TINYOS AND CONTIKI OS	33
TABLE 5.1: COMMUNICATION TIMING REQUIREMENTS FOR TRANSFERRING DATA OVER CoAP/UDP BASED ON 802.15.4	51
TABLE 5. 2: COMMUNICATION TIMING REQUIREMENTS FOR DATA TRANSFERRING OVER HTTP/TCP BASED ON 802.15.4	54
TABLE 5.3: NETWORK LAYER OVERHEAD. HTTP/TCP vs. CoAP/UDP	56
TABLE 5. 4: COMPARING CoAP AND HTTP TRANSMISSION DELAY IN TERM OF TRANSACTION SIZE	58
TABLE 5. 5: COMPARING CoAP AND HTTP TRANSMISSION DELAY IN TERM OF DISTANCE.....	58

CHAPTER 1

INTRODUCTION

The Internet of Things is a case which always has been interested for scholars and developers to work on it. The reason is that, Internet of Things is a simple idea of connecting all identified objects through wireless connection, that they could communicate with each other and be able to identify themselves to other devices [1]. By this term we can say, it's a reality that allows to connect people to their possessions, and also it provides connectivity among those possessions, in ways that will save us time and effort and enable things to be smart and work automatically. The IoT goal is not just being connected in terms of computers, tablets and smart phones, but you can imagine a world where everything is connected together with a intelligent communication among them.

In this scenario a *Thing* can be everything from cell phones, washing machines, headphones, lamps or also can be a person with a heart monitor implant, a farm animal with a biochip transponder or any other natural or man-made object. It is predictable that IoT will encompass every aspect of our lives that it would be provided by the intelligence of the embedded devices. We can easily guess how much data will be generated! IoT tries to make our life easier and safer by processing the gathered data into useful actions and controlling the connected objects. As a simple and basic example: when you are shopping, you are connected to your refrigerator and it informs you about its content and the products expiration date. Or for example you have this chance to save money by shifting your electricity use to cheaper off-peak hours.

So by IoT we can make better decision since we have more information, and we can save more time, energy and money. Monitoring our possessions and controlling them is another advantage of IoT in our life.

It's clear that each IoT device needs to utilize an IP address as a unique identifier In order to connect and integrate with the internet and automatically transfer data over a network. Internet of Things (IoT) will change the world, perhaps more profoundly than today's human-centric Internet. As huge volumes of data are being generated, determining the more feasible architecture and the most efficient network protocols used by WSN in term of saving more energy in a secure way would be some important issues.

Wireless Sensor Networks (WSNs) have been recognized as very important elements of IoT which are enabled by three trends:

- Cheaper computation (Moore's Law)
- Compact sensing (MEMS sensors)
- Wireless networking (low-power radios)

Since it is a new term, there are not much prior work can be used with WSNs, so one of the big issues would be finding out new compatible solutions in all area of the system. Significant work and careful analysis required to implement WSN because of the inherit WSN constraints.

Protocols defined for WSN must be designed in a way to achieve low energy consumption and low latency for reliable data communications with efficient node placement because of the constraints on sensor nodes such as energy, memory, computational speed and communications bandwidth.

WSNs establish a cheap sensory infrastructure to internet-connected devices distributed in different scales, from several to tens of thousands of sensor nodes with computing capability. This flexibility causes any next descendant of Internet-enabled portable object to interact easily with the “Things” and create a new IoT.

Small nodes with low power and low memory equipped one or more sensors in addition to a microcontroller, wireless transceiver, antenna, a processor and a power supplier are thought to transfer small payload such as temperature, air pollution, humidity, etc. "Sleep" mode and duty cycling mechanism are two mechanisms that could be applied to change the state of sensor from "On" to "Idle" or "Sleep" mode in order to turn on the sensor when it has some operation to do

New technology has developed in order to implement WSN applications. 6LoWPAN [2] standard, has defined by IETF to transmit IPv6 packets through computationally constraint networks. In order to integration IP of WSNs, sensor nodes implement the TCP/IP stack (or a compatible set of protocols such as 6LoWPAN [2] in 802.15.4 [18] networks), so there are direct connectivity between sensors and internet hosts.

Some of WSN examples could be Structural Health Monitoring [6], networked control, signal processing, air pollution monitoring, etc. Some of WSN application are inexpensive tool for measurements or monitoring of diverse phenomena [3], collecting data from vehicles and send them to the web portal [4], Water quality monitoring [5].

Standard based protocol stack for Wireless Sensor Networks includes 6LoWPAN, IEEE802.15.4e [18], RPL and CoAP for the management of layer-related procedures.

A typical wireless sensor network is shown in Figure 1.1. Low power sensor nodes are used to monitor physical and environmental conditions. Then, the measurements can be transmitted from server node to another one through certain route and arrive at a Border router/gateway in the WSN or end users. The Border router in Figure 1.1 is usually a node powered by wire or a base station which may communicate with the data collection center or end-users via the Internet or any type of wireless network (like WiFi, mesh networks, cellular networks, WiMAX, etc.)

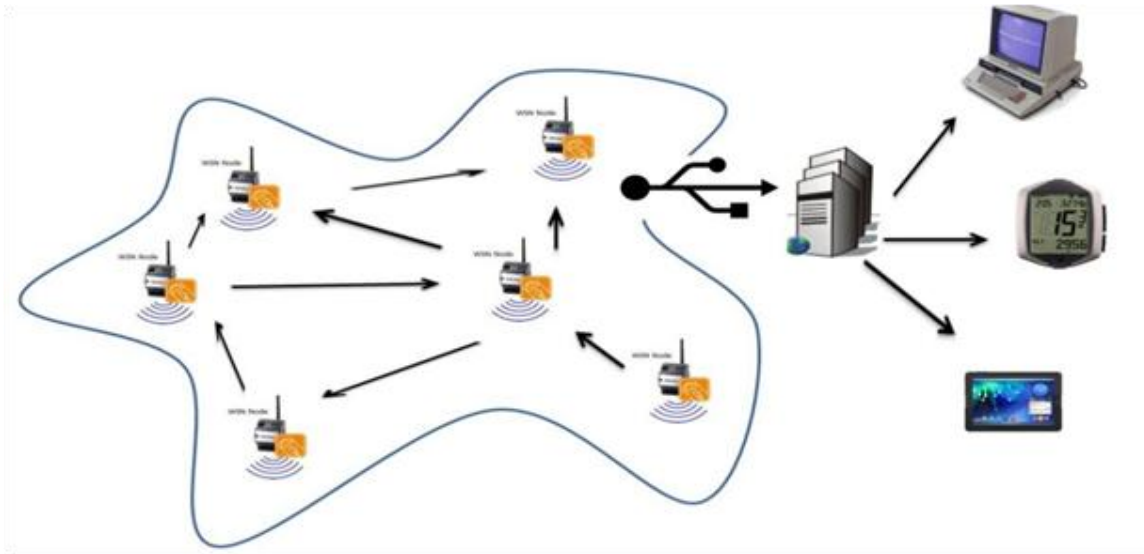


Figure 1.1 Architecture of a typical WSN

1.1 MOTIVATION AND PROBLEM STATEMENT

In WSNs the overhead imposed at the different network layers are very important in term of energy consumption due to the power limitation in wireless sensor nodes. Sensor nodes will cost much more energy for transmitting one data packet rather than processing data by itself. So Communication in sensor networks is based on very short packets to reduce the communication overhead.

Regard to this issue the WSN designers and researcher try to find out the best way of using all WSN protocols in order to optimize power consumption and increase the network performance at all levels of the protocol stack.

A sensor node will lose more energy for transmitting one data packet than processing data by itself. Furthermore, In order to reduce the transmission power consumption in sensor nodes, they have to be small in size and memory.

The retransmission energy consumption caused by network congestion, unstable channel environment or noise, makes nodes to run out of energy sooner and it's not always feasible to recharge them or replace the batteries easily. Therefore, the network lifetime of wireless sensor networks is affected by energy efficiency. As energy is a vital issue for WSNs, saving energy for the sensor nodes and making them alive as long as possible have become more and more important.

In this thesis we research the most feasible WSN network protocols to transfer data transactions

between a sensor client and a sensor server with low overhead and energy consumption in a real hardware and operating system.

1.2 THESIS OBJECTIVE

The main objective of this work is to integrate some of the proposals from the literature and standardization bodies in a real hardware and test it in order to build an architecture that can be integrated in a real WSN, an IEEE 802.15.4-based WSN testbed.

The main challenge is to define a system architecture that can be cheaply and easily deployed by users and application developers, with low programmable effort, that works and can be used for obtaining data from the sensor nodes from an outside source of the sensor network. In addition, we aim to examine the behavior of a real hardware while testing it in different environments.

1.3 THESIS STRUCTURE

Chapter 2 contains literature review of the wireless sensor networks, including wireless sensor nodes Zolertia Z1, communication protocol – IEEE 802.15.4. Section 3 is designed to explain the system architecture with applying dissimilar decisions in order to examine the functionality of system in the real hardware. Section 4 presents the different scenarios used for the experiments. Chapter 5 presents the experiments and the results. Finally chapter 6 concludes the thesis.

CHAPTER 2

FUNDAMENTALS OF WIRELESS SENSOR NETWORKS

2.1 WIRELESS SENSOR NETWORKS

This chapter briefly presents all the protocol stacks needed to implement the system architecture and WSN nodes. A number of researchers and scholars have been interested to work on Wireless sensor networks (WSNs) especially in the case of wireless communication board for the past few years. A WSN scales can vary from a few to a large number of sensor nodes. In WSN, sensor node is one of the principal components which include a transceiver, antenna, microcontroller, memory, power source and one or more sensors such as temperature sensor and humidity sensor. We used Zolertia Z1 [16] sensor nodes in this project which includes CC2420 transceiver [13], IEEE 802.15.4 compliant [17] protocol. Further, an operating system is an important part that allows applications to be efficiently implemented and makes the sensor node work.

In this project the Contiki [10] operating system implemented the protocols runs on every node. Contiki OS is developed by SICS (Swedish Institute of Computer Science) and is a small, open source operating system designed for wireless sensor networks. In this chapter, wireless sensor nodes are introduced first. After that, all the protocol stacks used in implementing the system architecture will be described. Finally, Contiki OS is described.

2.1.1 CHARACTERISTICS OF WSN

The following is a briefly description of WSN characteristics:

1) Node mobility:

It makes the network links to be formed dynamically, that more nodes can join to the network easily or disjoin when they move out of the range.

2) Unattended operation:

Ability of reconfiguration the network by the nodes without any human intervention

3) Dynamic Network Topology:

It is important to Re-deployment the network topology in case of failure of the node, failure of radio links, or arrival of some mobile obstacles

4) Limited power:

In some environment it is not easy to charge energy regularly [27]. They may, probably, be. So, energy consumption is a major issue, and should be optimized at three stages, node communication, sensing and processing to reduce the energy consumption.

5) Large scale of deployment

The large scale of WSN from hundreds or thousands of nodes and some environmental parameters like noise, dispersion, interface and available bandwidth, affects on the connection quality and may causes some disconnection between the nodes even in tiny networks [28].

Some advantages of WSN are listed in the table 2.1 [29]

Advantages	Disadvantages
No need the fixed infrastructure to set up the network	Less secure
Could be set up in the non-reachable places	Lower speed
Low cost implementation	More complex to configure than a wired network
Flexible if there is ad hoc situation when additional workstation is required	Easily affected by surroundings (walls, microwave, large distances due to signal attenuation..

Table 2.1: WSN advantage and disadvantage

A Wireless Sensor Network structure is shown in Figure 2.1

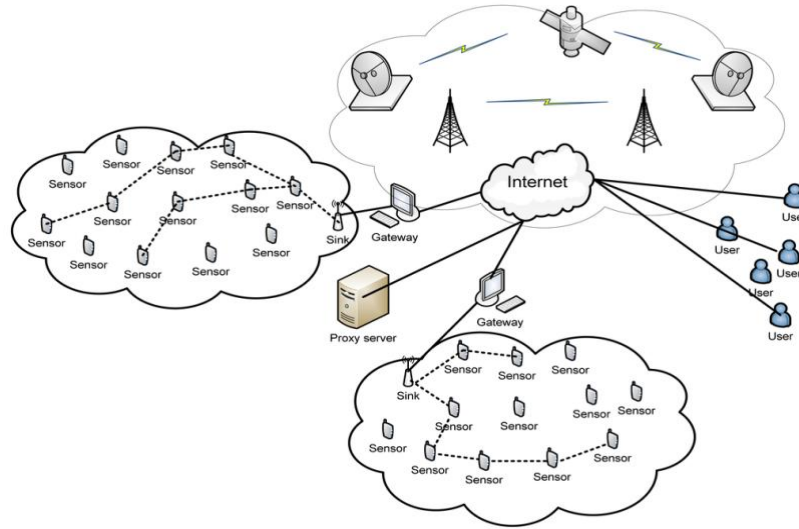


Figure 2.1: WSN structure

2.2 WIRELESS SENSOR NODES ARCHITECTURE

A sensor node, as a most important feature in WSN is a resource-constrained device with the communication capability, sensing and limiting data processing. Architecture of a sensor node involves four layers: a Processing Unit, a Power Unit, one or more Sensing Units and/or Actuating Units, and a Transceiver that they are shown in the figure 2.2.

Sensor nodes are often deployed in hostile environments or over large geographical areas, to monitor physical or environmental conditions, from indoor to outdoor.

Due to the application's and customer's requirements, sensor nodes tend to be designed to focus on lower energy consumption and easier development process for a given wireless communication range and area. Nowadays, sensor nodes are categorized in two different kinds [9]. One is ordinary sensor or non-sink nodes which are used to sense and monitor different physical phenomena like temperature, humidity etc, collect them. The other one known as a sink (gateway) node (gateway) uses radio communication to collect the data gathered by ordinary sensor and also it connects sensor networks to the external world. Sinks normally are more powerful than the ordinary nodes in terms of processing power, available storage and available energy. Some small data processing are done by ordinary sensors but high end processing is performed at sink nodes.

In this thesis project, ordinary sensor nodes are used to send the collected data to the external network through the sink node. Up to now, there exist a lot of sensor nodes designed and manufactured by different companies. A list of available sensor nodes can be found here [9]. In our project, we choose Zolertia Z1 sensor nodes. A detailed introduction to the Z1 sensor

platform is given in the following subsection.

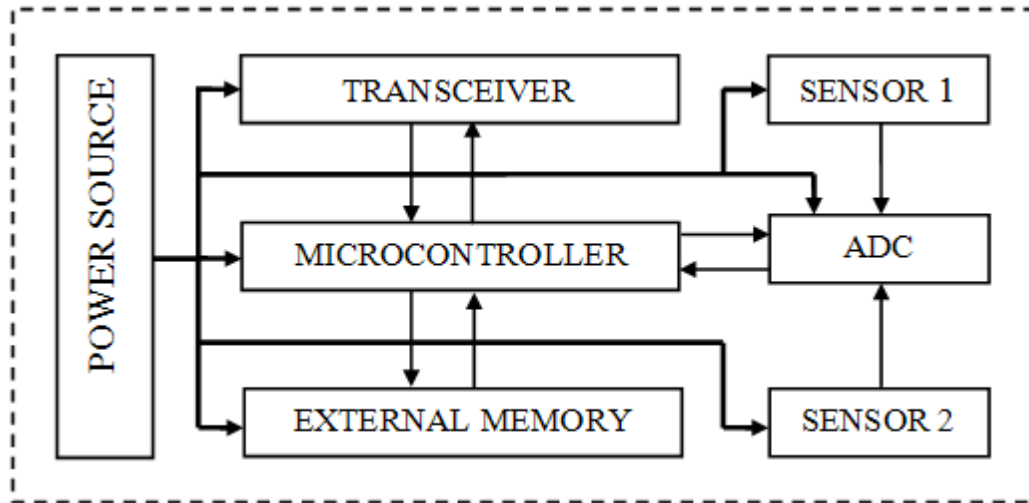


Figure 2.2: Architecture of Wireless Sensor Node

2.2.1 ZOLERTIA Z1

Z1 sensor nodes are produced by a Spanish company, Zolertia, which is located in Barcelona, Spain. A Z1 sensor node [16] (shown in Figure 2.3) is a low power WSN module that is designed as a general purpose development platform for WSN developers, researchers, enthusiasts and hobbyists. It is designed to maximum backwards compatibility with the successful Tmote-like family motes while improving the performance and maximum flexibility and expandability with regards to any combination of power supplies, sensors and connectors. The two most employed open source operating systems/ stacks by the WSN community, like TinyOS 2.x [11] and Contiki [10] are supported by Z1 sensor.

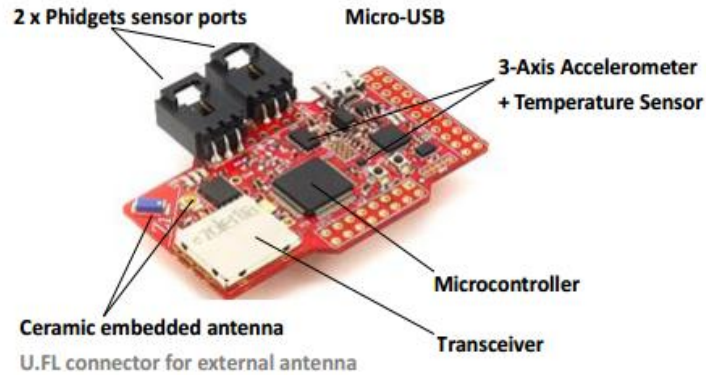


Figure 2.3: Wireless Sensor Node -Zolertia Z1

The Z1 includes the well known CC2420 transceiver, IEEE 802.15.4 and Zigbee compliant, which operates at 2.4GHz with an effective data rate of 250Kbps. Its core architecture is based upon the MSP430+CC2420 family of microcontrollers and radio transceivers by Texas Instruments. Z1 uses the MSP430F2xxx microcontroller unit (MCU) instead of the MSP430F1xxx, as is customary among other motes, like Crossbow's TelosB, Moteiv's Tmote, and alike. The inner changes between F2xxx and F1xxx devices lead to some Contiki functions available for Tmote sensor nodes are not portable to the Z1 sensor nodes that we are using [16]. In the following subsections, the Z1 sensor node will be introduced and described in the context of the present project.

2.2.1.1 Z1 SUMMARY FEATURES

A Z1 sensor node is equipped with the low power microcontroller which is MSP430F2617 made by Texas Instruments [12]. A Z1 involves of a powerful 16bit RISC CPU @16MHz clock speed, builtin clock factory calibration, 8KB RAM and a 92KB Flash memory. It also includes a CC2420 transceiver from Texas Instruments/Chipcon [13], which is IEEE 802.15.4 compliant and operates at 2.4GHz with an effective data rate of 250Kbps. The hardware selection of Z1 guarantees the maximum efficiency and robustness with low energy cost [16].

Z1 has two builtin sensors, one accelerometer sensor and one temperature sensor. The ADXL345 [14] is a small, thin, low power, 3axis accelerometer with high resolution (13bit) measurement at up to $\pm g$. Digital output data is formatted as 16bit twos complement and 16 is accessible through either a SPI (3 or 4wire) or I2C digital interface.

The TMP102 [15] is ideal for extended temperature measurement in a variety of communication, computer, consumer, environmental, industrial, and instrumentation applications. The device is specified for operation over a temperature range of -40° to $+125^{\circ}$. In addition to the built-in sensors, Z1 also supports up to four external sensors. There exist a full range of pins available for

connecting with phidgets (physical widgets) and many kinds of sensors from the third-party in an easy way.

Z1 mote comes with an integrated ceramic antenna from Yageo/Phycomp connected to the CC2420 through the C62 capacitor. Optionally, an external antenna can be connected via a u.FL connector. The CC2420 [14] is a low-cost single chip with 2.4 GHz IEEE 802.15.4 compliant RF transceiver which, designed for low power and low voltage wireless applications. The CC2420 provides extensive hardware support for packet handling, data buffering, burst transmissions, data encryption, data authentication, clear channel assessment, link quality indication and packet timing information [14]. More basic features of CC2420 are presented below.

- **CC2420 Key Features:**

- True single-chip 2.4 GHz IEEE 802.15.4 compliant RF transceiver with baseband modem and MAC support
- DSSS baseband modem with 2 MChips/s and 250 kbps effective data rate.
- Suitable for both RFD and FFD operation
- Low current consumption (RX: 18.8 mA, TX: 17.4 mA)
- Low supply voltage (2.1 – 3.6 V) with integrated voltage regulator
- Low supply voltage (1.6 – 2.0 V) with external voltage regulator
- Programmable output power
- No external RF switch / filter needed
- I/Q low-IF receiver
- I/Q direct upconversion transmitter
- Very few external components
- 128(RX) + 128(TX) byte data buffering
- Digital RSSI / LQI support
- Hardware MAC encryption (AES-128)

To summarize some principal characteristics of the Z1 are listed in Table 2.2.

Microcontroller	2nd generation MSP430™ ultra-low power 16-bit MCU 16MHz
Radio chip	250 kbps 2.4 GHz IEEE 802.15.4 compliant RF transceiver CC2420
Integrated sensors	3-Axis, $\pm 2/4/8/16$ g digital accelerometer Low-power digital temperature sensor with $\pm 0.5^\circ\text{C}$ accuracy (in $-25^\circ\text{C} \sim 85^\circ\text{C}$ range)
Supported protocol	2.4GHz IEEE 802.15.4, 6LowPAN compliant and ZigBee™ ready

Table 2. 2: Zolertia Z1 parameters

A unique global standard definition needed for the objects to could communicate with each other

through the Internet despite of depend less on their technologies and capabilities. So the basic standardization bodies have developed standards for the Internet of Things and IP protocol stack for Low-Power, Reliable Wireless Sensor Networks have defined that could be seen in Figure 2.4. The different layers are explained in the following sections.

APPLICATION	CoAP
TRANSPORT	UDP
NETWORK	IPv6/RPL
ADAPTATION	6LoWPAN Adaptation
MAC	IEEE 802.15.4
PHYSICAL	IEEE 802.15.4

Figure 2.4: IP protocol stack for Low_Power, Reliable WSN

2.3 IEEE 802.15.4 - COMMUNICATIONS PROTOCOL

IEEE 802.15.4 is a standard which specifies the physical layer and media access control for low-rate wireless personal area network (LR-WPANs). It is defined by the IEEE 802.15 working group in 2003 [17]. IEEE standard 802.15.4 aims to operate within a short range (i.e. 10 meters), with very low transmission rate of 250Kbit/s and with a reasonable battery life rather than other approaches, such as WI-FI, which offers more bandwidth and requires more power.

IEEE 802.15.4 specifications operate on several bands such as:

16 channels in the 2.4 GHZ, 10 channels in the 915GHZ and 1 channel in the 868GHZ.

Table 2.3 shows the data rate and modulation-related parameters in different frequency bands

PHY (MHz)	Frequency band (MHz)	Spreading parameters		Data parameters		
		Chip rate (kchip/s)	Modulation	Bit rate (kb/s)	Symbol rate (ksymbol/s)	Symbols
868/915	868–868.6	300	BPSK	20	20	Binary
	902–928	600	BPSK	40	40	Binary
868/915 (optional)	868–868.6	400	ASK	250	12.5	20-bit PSSS
	902–928	1600	ASK	250	50	5-bit PSSS
868/915 (optional)	868–868.6	400	O-QPSK	100	25	16-ary Orthogonal
	902–928	1000	O-QPSK	250	62.5	16-ary Orthogonal
2450	2400–2483.5	2000	O-QPSK	250	62.5	16-ary Orthogonal

Table 2.3: Frequency bands and data rates

The IEEE 802.15.4 packet consist of the 64-bit IEEE address or a short 16-bit address that is located in the destination and source addressing mode field, so the size of the packet could be different regarding to the address size.

Figure 2.5 shows the general packet frame structure of IEEE 802.15.4.

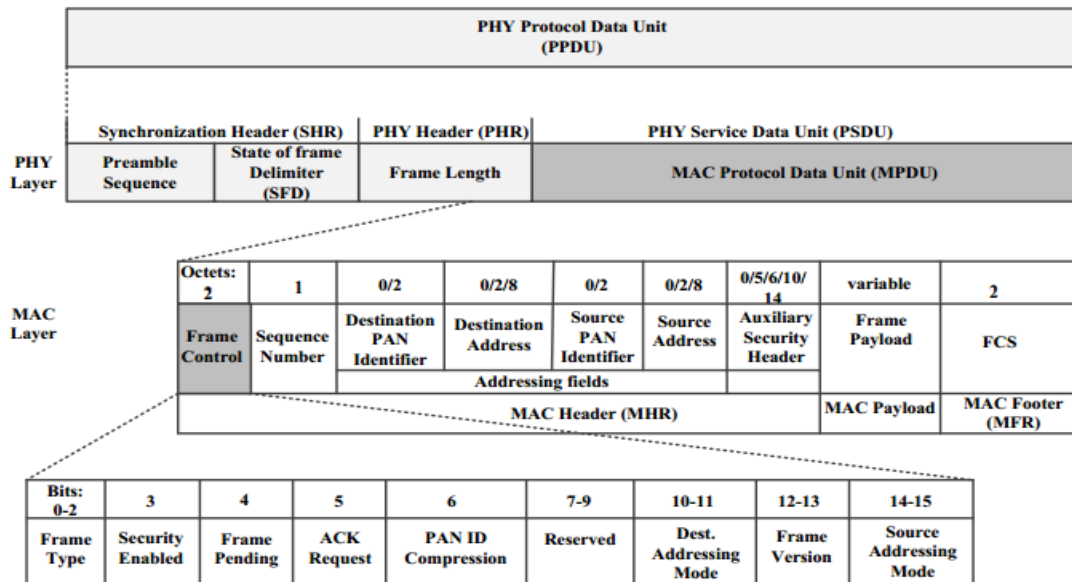


Figure 2.5: IEEE 802.15.4 General packet frame format

The frame control field defines the type of the frame (i.e. data, acknowledgement or other type) and the addressing used (16 bit or 64 bit). The sequence number is an increasing counter of the

frames transmitted by the node. The addressing fields contain the source and destination addresses. The following fields are the frame payload followed by the frame check sequence which ends the frame.

The two subsections below are dedicated to the main functions of physical (PHY) layer and media access control (MAC) layer

2.3.1 PHYSICAL LAYER

Physical layer is the initial layer in the OSI reference model used worldwide. The implementation of this layer is often termed PHY. Some special characteristics of WSN create a number of significant differences in comparison with traditional wireless network. The main different features of WSN make the transmission at shorter distances, lower data rates and node power constraints.

The physical layer requirements used in Wireless Sensor Networks differ based on the particular scenarios usage. The physical layer is an important network stack layer to reduce energy dissipation by finding the optimal transmit (relay) distance and transmit power for a given modulation scheme and a given channel model, in order to maximize network lifetime.

The physical layer consists of the basic network hardware transmission technologies of a network. The data transmission service provided by physical layer (PHY) enables accessibility to every layer management function and maintains a database of information on related personal area networks. The physical RF transceiver performs channel selection, energy and signal management functions which are managed by PHY.

2.3.2 MAC LAYER

Wireless networks must avoid collisions to ensure packets reach their destination.

The Physical Layer uses Multiple Access with Collision Avoidance (CSMA-CA) to detect whether or not another radio is transmitting and employ a method to avoid collisions. In this algorithm the MAC rst listens for energy or modulated data on the air. If none is detected, it can transmit immediately. If the channel is not clear the algorithm applies random wait times (backoffs) before retrying the transmissions. Power management is another functionality of MAC layer that allows radio devices to be turn off most of the time in order to save more energy. Contiki OS uses ContikiMAC [17] mechanism to achieve this call. The mechanism is described in subsection 2.10.

2.4 6LOWPAN ADAPTATION LAYER

6LoWPAN [19] [20] provides a means of carrying packet data in the form of IPv6 over IEEE

802.15.4 networks. Some sensor network protocols have non-IP network layer protocol such as ZigBee, where TCP/IP protocol is not used. However, future WSNs consisting of thousands of nodes and these networks may be connected to others via the internet. Hence, IPv6 over LoWPAN (6LoWPAN) is defined by Internet Engineering Task Force (IETF) [19] as a technique to apply TCP/IP into WSN [20].

Using the fact that IANA is going to run out of IPv4 addresses, and with the possible explosion of devices with allocated IP addresses, 6LoWPAN starts by using IPv6 as the basic IP format.

The 6LoWPAN group within the IETF has then defined the encapsulation and compression mechanisms and also provides the wireless sensor network (WSN) node with IP communication capabilities by introducing an adaptation layer that enables efficient IPv6 communication over IEEE 802.15.4 LoWPAN links since IPv6 requires support of packet sizes much larger than the largest IEEE 802.15.4 frame size. The MTU size for IPv6 packets over IEEE 802.15.4 is 1280 octets which were set to keep transmissions short and thereby reduce power consumption. However, a full IPv6 packet does not fit in an IEEE 802.15.4 frame.

The specification number for 6LoWPAN is RFC 4944 with the problem statement document being referred to as RFC 4919. [19]

2.4.1 HEADER COMPRESSION

802.15.4 Protocol has a maximal available payload size of 127 bytes. As seen in Figure 2.3 from the 127 bytes of the frame, 25 bytes are required for the MAC header, 40 bytes for the IPv6 header and 8 bytes for the UDP Header resulting only 54 bytes available for Payload that can be used from the application. Note that if encryption is to be added the available payload decreases further.

The maximum packet length of IEEE 802.15.4 is 127 Bytes. As seen in Figure 2.3 21 bytes of that is occupied by MAC header, 8 bytes by UDP header, 40 bytes by IPv6 header and more or less 53 bytes for the payload in full UDP/IPv6 (64-bit addressing) that is shown in Figure 2.6. In case of using TCP it consumes 20 bytes and the payload header would be 41 bytes. These numbers tell us that header compression mechanisms are needed for 6LoWPAN to be more efficient. Besides that, adding encryption to the available payload will decrease the payload size further. Actually any header fields that can be calculated from the context will be elided by header compression mechanism and it sends the remaining fields unmodified. As a simple example it can remove the Version field in the IPv6 header since it has always the same value as 6. As we can obtain the address information from the MAC header it could be removed.

Figure 2.4 Minimal UDP/IPv6LoWPAN shows the increasing in Payload size to 108 bytes by IPv6 compression header mechanism.

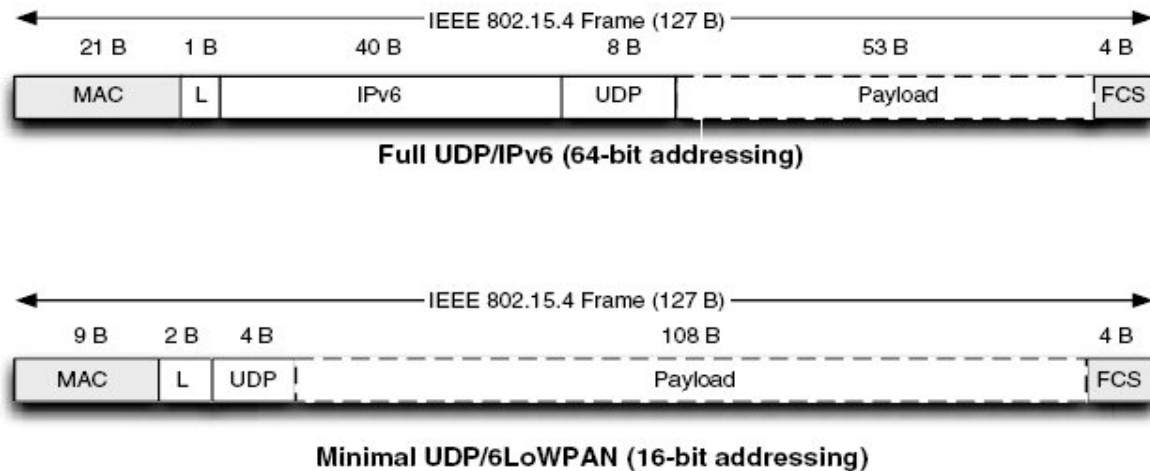


Figure 2.6: IEEE802.15.4 frame format

2.4.2 FRAGMENTATION

Fragmentation [30] is a 6LoWPAN capability that provides the transmission of IPv6 large packet that exceeds the maximum frame size of the link layer by dividing them to some smaller fragments and sending to the destination by order. Fragmentation mechanism has brings some overhead to the packets that these extra information will be removed in the receiver by de-fragmentation mechanism and the fragmented packets will be combined and delivered to the destination as a total package.

The 6LoWPAN packet structure of the fragmented packets is shown in Figure 2.7. It is used when the payload is too large to transmit by a single IEEE 802.15.4 frame. Each fragment involves a fixed-size fragment header. The remaining space of the link-layer frame is iteratively filled with the IPv6 packet content. [30]. In this mechanism the first fragment (FRAG1) is responsible of carrying the end-to-end routing information and the remaining fragments (FRAGN) to the FRAG1 will be correlated in the destination node in order to derive IP-based routing or processing decisions for these fragments.

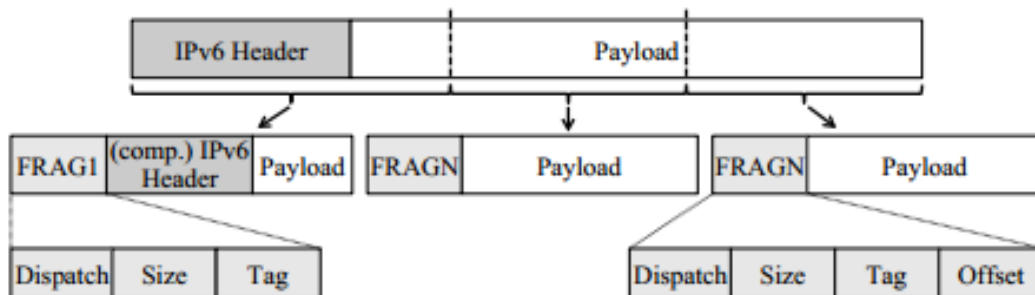


Figure 2.7: 6LoWPAN packet structure of the fragmented packet

2.5 IPV6/RPL NETWORK LAYER

2.5.1 IPV6

Nowadays all kind of objects and devices like wireless sensors try to communicate with each other through the internet [31]. In order to achieve to this goal its necessary to use IPv6 as a new version of the Internet Protocol which expands the address size from 32 bits to 128 bits. IPv6 has the capability of implementing the fragmentation mechanism at the at the endpoints instead of at the intermediate routers that is significantly simplifies the router and increases the performance. Some of the main mechanisms defined by IPv6 are Neighbor Discovery (ND), duplicate address detection (DAD), router discovery and Stateless address auto configuration (SAA).

IPv6 header format is shown in figure 2.8. Some of the main fields are:

Version: This represents the IP version number. This field's value is 6 for IPv6

Payload length: This is the length of IPv6 payload in octets which is a 16-bit unsigned integer.

Next header: This identifies the type of header immediately following the IPv6 header.

Hop limit: This specifies the maximum number of hops that a packet may take before it is discarded.

Source address: This is the IPv6 address of the originator of the packet.

Destination address: This is the IPv6 address of the intended recipients of the packet.

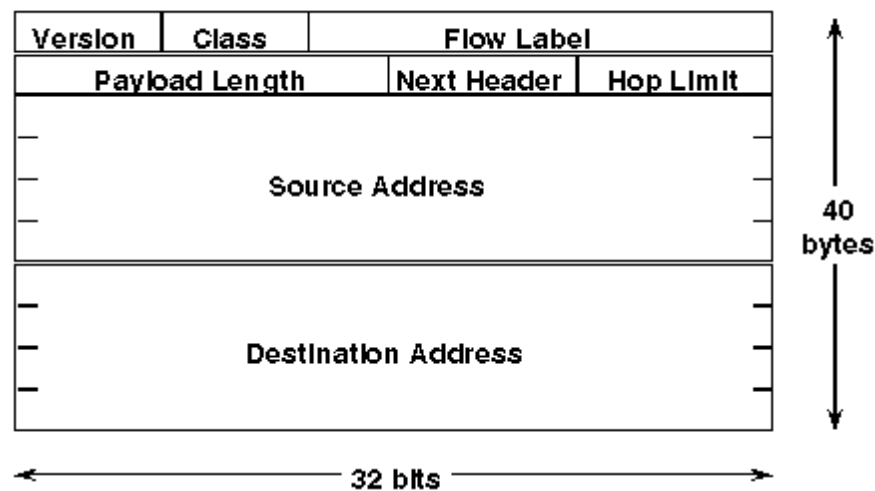


Figure 2.8: IPv6 header format

2.5.2 RPL (ROUTING PROTOCOL FOR LOW POWER AND LOSSY NETWORKS)

Designing energy efficient protocols is the most important issue in Wireless Sensor Network

(WSN) because of limitation in power and also in memory etc. And such networks are often tries to consume energy as less as possible. The IETF ROLL working

group has defined the IPv6 Routing Protocol for Low power and Lossy Networks (RPL) in order to overcome routing issues in LLNs. In order to achieve to this goal RPL uses some metrics such as latency, reliability, battery left on the sensor to compute the "best" path from a "leaf" node to a "root" node.

There are four types of control messages used by RPL protocol in order to maintenance the topology and exchanging the information that are briefly explained below:

DODAG: Destination Oriented Directed Acyclic Graph which is routed at a single destination and constructed by RPL in order to avoid any cycle in the connected nodes. The logical topology of the network will be defined by his graph.

DIO: DODAG information Object message is sent periodically by the root node down to the leaf nodes to form and maintain the DODAG graph.

DAO: DODAG Destination Advertisement Object is sent from nodes to their parents to inform their presence.

DIS: DODAG Information Solicitation message is used by nodes to enforce other nodes in the network to send DIO messages. For example a newly joined node can send a DIS message to its parent to enforce it to send back a DIO message. The other option is to wait for the periodically generated DIO message.

Rank: The Rank value of a node indicates its position with respect to the DODAG root.

DODAG CONSTRUCTION PROCESS

Nodes periodically advertise DIO messages that contain information of the graph.

In receiving such a message, a node decides whether it will join the graph or not. This decision is based on constrain that it sets to achieve the goal of having a minimized path cost toward the DODAG root node.

Nodes that join the graph are able to advertise a DIS message in order to receive back a DIO message. The result of this procedure is a DODAG graph with upward routes towards the DODAG root.

Figure 2.9 shows an example of a physical topology with the corresponding logical topology (DODAG graph).

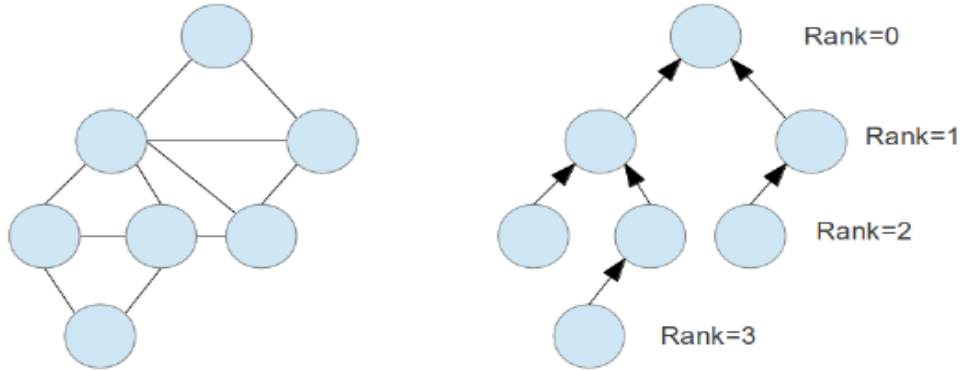


Figure 2.9: RPL example :Physical and Logical Topology of a network

2.6 UDP

User Datagram Protocol (UDP) [32] provides a procedure for application programs to send messages (datagram) to other hosts on the IP network without any prior communication set up. UDP provides no guarantees to the Upper Layer Protocol for message delivery, ordering or duplicate detection. The main reason of UDP is its simplicity and low overhead that cause UDP to be much faster than TCP.

UDP source header format is shown in Figure 2.10. Sending process and the receiving process are identified by the port number. The port number of sending process is set when it is needed. TCP and UDP use the destination port number to demultiplex incoming data from IP. The length of the UDP header is in bytes and the encapsulated data. Checksum is followed by the data are to be sent.

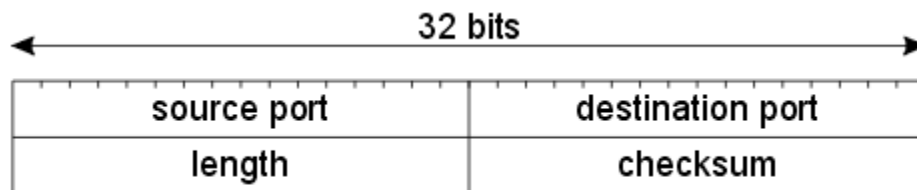


Figure 2.10: UDP source header format

2.7 RESTFUL ARCHITECTURE

REST stands for **R**epresentational **S**tate **T**ransfer [21]. It relies on a stateless, client-server, cacheable communications protocol, and in virtually all cases, the HTTP protocol [22] is used.

REST is an architecture style for designing networked applications. It uses HTTP rather than using complex mechanisms such as CORBA, RPC or SOAP to connect between machines.

The World Wide Web itself is based on HTTP, can be viewed as a REST-based architecture.

Resources are key abstraction of information in REST and they identified by uniform resource identifiers (URIs) that they get a unique, global ID and could be manipulated through their representations.

It is a collection of resources, with three defined aspects: 1) The "verbs" of the service are strictly those defined by the HTTP methods HEAD, GET, PUT, POST, and DELETE, 2) The "verbs" are used to act upon resources, and 3) resources are addressable using URLs. Get requests to retrieve a resource, PUT is used to create or update a resource with a known URI while POST is used to transfer the resource into a new state, and Delete data.

Having stateless communication between the client and the server means that the two entities can begin a communication without passing any session state. Each request from the clients must contain all the necessary information, so that the server understands the request and generates the corresponding response if any.

The main reason of using REST in IoT is the fact that it is more lightweight and. IP based networking in LLNs could enable the use of standard web service architectures without using application gateways. As a consequence, smart objects will not only be integrated with the internet but also with the Web. This integration is defined as the Web of Things (WoT). The advantage of the WoT is that smart object applications can be built on top Representational State Transfer (REST) architectures. REST architectures allow applications to rely on loosely coupled services which can be shared and reused.

2.8 CoAP

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for using with constrained nodes and constrained (e.g., low-power, lossy) networks. [23]

The limitations of the HTTP in the WoT applications on constrained devices with 8-bit microcontrollers and small amounts of ROM and RAM caused the Internet Engineering Task Force (IETF) to introduce the Constrained Application Protocol (CoAP) a web transfer protocol optimized for the constrained power and processing capabilities of WoT smart objects. The idea of Constrained RESTful Environments (CoRE) working group was to realize the REST architecture in a suitable form for the most constrained nodes and networks.

The currently available IP based application protocols such as HTTP, FTP, SOAP, etc. are not appropriate for 802.15.4 based networks, since 802.15.4 has a maximum frame size of 127 bytes. Besides, the WSN links are bandwidth constraint that these application protocols should consider

limitation. CoAP intends to avoid any complexity by running over UDP instead of TCP that has complexity in congestion control. Each GET request will be assigned by a unique Transaction ID to identify for retransmission purposes in order to be reliable.

Using CoAP limits the use of fragmentation and sparing constrained networks, like 6LoWPAN, to execute expensive fragmentation of IPv6 packets.

CoAP easily interfaces with HTTP for integration with the existing Web while meeting specialized requirements such as multicast support, very low overhead, and simplicity for constrained environments and M2M applications.

CoAP intends to keep message overhead small and bring Web-like service over low-power / loopy network, like 802.15.4, etc. The communication between HTTP devices and CoAP endpoints is via intermediary proxy that can translate from CoAP to HTTP and vice versa.

There are four types of messages used by CoAP to establish a message exchange between client and server:[33]

- Confirmable (CON) messages always carry a request or response and require an Acknowledgment.
- Non-Confirmable (NON) messages are used for regularly repeated messages and do not require an Acknowledgment (e.g. subscriptions or reading a sensor).
- Acknowledgment (ACK) messages acknowledge CON messages and must carry a response or be empty.
- Reset (RST) messages are sent in case a CON message is not received properly or some context is missing.

A CoAP message carries the request or response or would be empty. CoAP Codes is the registry to maintain the value of Code field in CoAP header. For example 0 indicates an empty message and 1-31 indicates a request message. CoAP messages are encoded in a simple binary format. A message consists of a fixed-sized CoAP Header followed by options in Type-Length-Value (TLV) format and a payload. Datagram length determines the length of the payload and when bound to the User Datagram Protocol (UDP) (the standard case), it must fit within a single datagram.

2.8.1. CoAP FEATURES

CoAP has the following main features [23]:

- Constrained web protocol fulfilling M2M requirements.
- UDP [RFC0768] binding with optional reliability supporting unicast and multicast requests.

- Asynchronous message exchanges.
- Low header overhead and parsing complexity.
- URI and Content-type support.
- Simple proxy and caching capabilities.
- A stateless HTTP mapping, allowing proxies to be built providing
- Access to CoAP resources via HTTP in a uniform way or for HTTP
- Simple interfaces to be realized alternatively over CoAP.
- Security binding to Datagram Transport Layer Security (DTLS) [RFC6347].

Now a day, tiny embedded devices are directly connected to the Internet over IP rather than using barcodes and RFID. So as most Internet applications today use web services, Internet of Things (IoT) would be the best promising applications of CoAP.

2.8.2 CoAP IMPLEMENTATIONS

Californium (Cf) , Erbium (Er) , and Copper (Cu) [44] are three implementations of CoAP developed at the Institute for Pervasive Computing, ETH Zurich:

Californium (Cf) is an Unconstrained CoAP implementation that is written in Java and focuses on scalability and usability. It implemented for

- IoT cloud services
- Stronger IoT devices (Java SE Embedded or special JVMs)

Using Californium makes users to work easily with CoAP without knowing the details of CoAP in order to provide an intuitive, easy-to-use framework to interact with CoAP endpoints or provide specific services. So, users do not need to know the internals like message retransmissions, block-wise transfers and observation handling in details.

Californium is based on a layered architecture, extending the two-layer approach described in the CoAP draft [23]. This allows an isolated implementation of different aspects such as message retransmission, transactions, and block-wise transfers specified by CoAP over several levels. The Cf architectural design of is explained in the lab report [24].

Erbium (Er) is a low-power REST Engine for Contiki that was developed together with SISC and makes low-power systems to communicate efficiently and easily with the Internet [25]. Therefore, this implementation is specialized for constrained environments as it is designed to run on small amounts of memory and low-power Central Processing Units (CPUs) or Microcontroller Units (MCUs).

Copper (Cu) is a CoAP protocol handler for Mozilla Firefox, used for Browsing and bookmarking of CoAP URIs and Interaction with resource like RESTClient or Poster. It treats tiny devices like normal RESTful Web services.

it is designed for unconstrained environments as Cf. Its ability to render a number of different content types such as JSON or the CoRE Link Format makes it a useful testing tool for application as well as protocol development[26].

The interaction model of CoAP is similar to the client/server model of HTTP. However, machine-to-machine interactions typically result in a CoAP implementation acting in both client and server roles. A CoAP request is sent by a client to perform an action (GET, POST, PUT, and DELETE) on a resource (identified by a URI) on a server. The server then sends back a response with a response code; this response may contain a resource representation.

Unlike HTTP, CoAP interaction is asynchronous over UDP. Coap reliability is handled in the application layer, using a message layer (with exponential back-off). CoAP defines four types of messages:

Confirmable (CON), NonConfirmable (NON), Acknowledgement (ACK) and Reset (RST), method codes and response codes included in some of these messages and requests or responses are carried by them. Requests can be carried in Confirmable and Non-confirmable messages, and responses can be carried in these as well as piggy-backed [] in Acknowledgement messages. The request/response interactions using Method and Response codes (see Figure 11). CoAP is however a single protocol, with messaging and request/response just features of the CoAP header.



Figure 2.11: Abstract layering of CoAP

2.8.3 MESSAGE LAYER MODEL

Figure 2.12 shows an example of reliable CoAP transmission message. Client starts to open the reliable session by sending the in a CON message with the message ID (0x8c56), to the server. Once the server get the request, prepares the response and carries it in a ACK message using the same message ID in order to detect any duplication. In this way Client will not retransmit the request when it receives the ACK message. In case the server could not prepare the proper response to the client will send Reset message (RST) instead of an Acknowledgement (ACK).

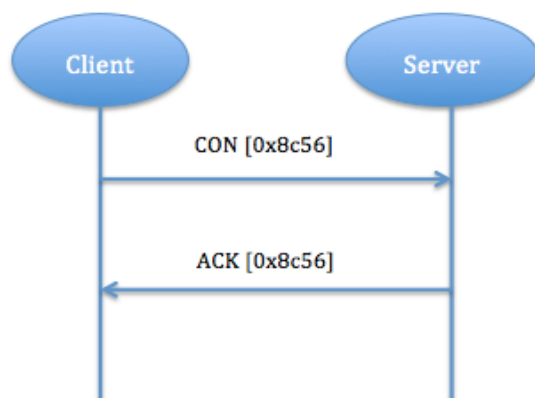


Figure 2.12: CoAP reliable message transmission

In some situations such as sending of single measurement data from a sensor in a frequent rate, it is not necessary to send a Confirmable message, and a message could be marked as “Non-confirmable” carrying the same message ID. Figure 2.13 below is such an example.

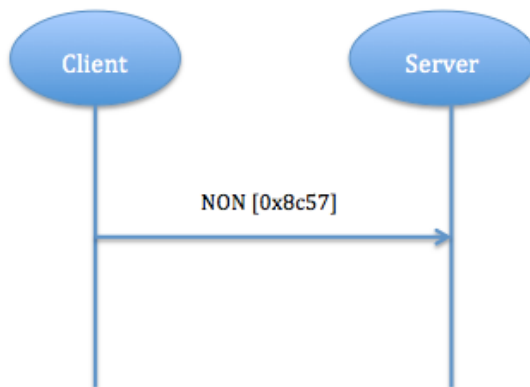


Figure 2.13: Unreliable message Transport

2.8.4 REQUEST/RESPONSE LAYER MODEL

CoAP uses three different models in order to carry the response message that is explained as following:

- 1) Piggy-backed: In this case regardless to the request type(CON or NON) client will received the response ACK with confirmable message immediately that means the ACK involves the response message in successful case, and in failure case it involves the

failure response code, shown in figure 2.14.

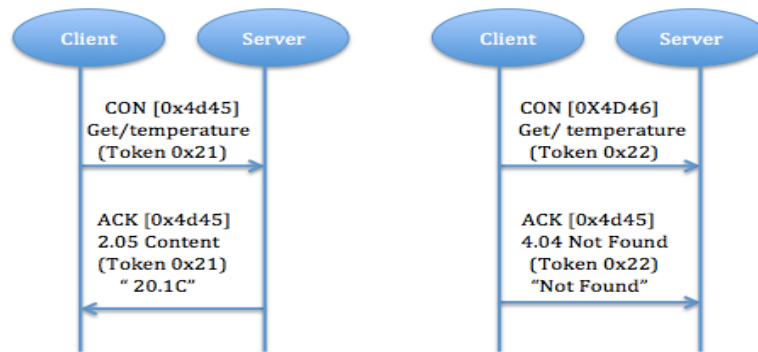


Figure 2.14: The successful and failure response results of GET method

- 2) Separate response: The server will send an empty ACK when it cannot answer immediately to the CON type message in order to avoid client to resend the request. Once server is ready to answer the client, will send a CON to the client and receive the confirmable message with ACK from the client to initiate the message transmission again. (Figure. 2.15).

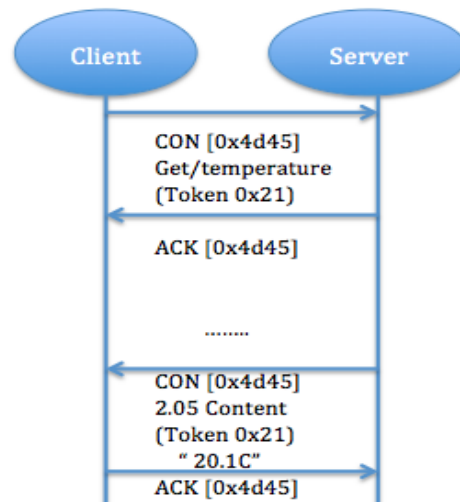


Figure 2.15: GET request with a separate response

- 3) Non confirmable request and response: In this case request and response would be sent in Non confirmable and NON type message respectively (Figure 2.16).

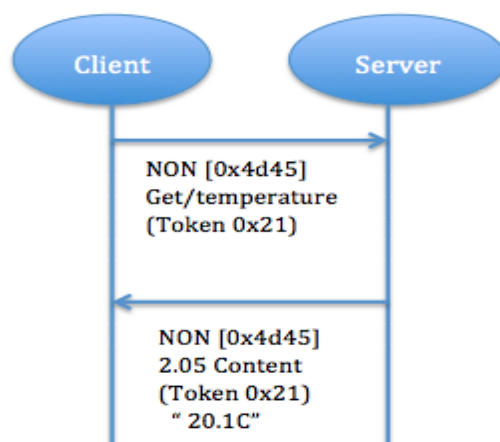


Figure 2.16: Non confirmable request and response

The message format of CoAP is based on the exchange of **messages** over UDP that is in binary form that it can be seen in Figure 2.17.

In this figure “Version” (Ver), indicates the CoAP version number that is a 2-bit unsigned integer data-type. The message type (“Confirmable” = “0”, “Non-confirmable” = “1”, “Acknowledgement” = “2” or “Reset” = “3”) is defined by “Type” (T) field. Token length (TKL) specifies the length of the Token field. Code specifies where the message takes a request (1-31) or a response (64-191) or is empty (0). In more details, for example code 1 signifies that the message is a GET request and code 2 is a POST request while codes 69 and 128 are both responses translated to “2.05 Content” and “4.00 Bad Request” respectively. Message ID is a 16-bit unsigned integer and is used for duplicate detection as mentioned above.

The Token value is used to associate requests and responses in a significant way. Options could specify the maximum age of a resource in seconds, the Internet media type identify the body of message, indicates the Internet host of a resource, the transport layer port number of the resource and more. Finally, the payload of a request is typically a representation of a resource while the payload of a response is the result of the requested action.

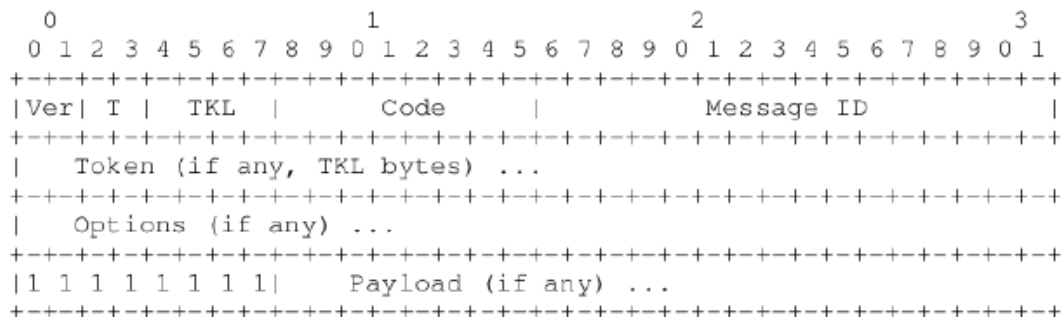


Figure 2.17: CoAP message Format

2.8.5 URI SCHEME

CoAP URI is used to identify and locate the CoAP resources. Here's an example of a CoAP URI:

coap://[fe80::c30c:0000:0000:0002]:5683/HelloWorld

Here, “[fe80::c30c:0000:0000:0002]” is the host IPv6 address, “5683” the default UDP port number used for CoAP resources and “HelloWorld” the resource representation asked by client to obtain.

2.8.6 BLOCKWISE TRANSFER

As we explained before CoAP is thought for transferring data through small constrained devices in term of memory and power, hence payload size must be reasonable to ensure it fits in a CoAP packet and could be sent without fragmentation. So the payload is considered to be as small as possible, but in some cases such as firmware updates it needs to transfer large amount of data. Block transfer is a CoAP capability applied to the payload to overcome this issue in the application layer by dividing the response to some block options automatically without involvement of the handler. The advantage of using block options could be as the following:

- No conversation state is needed between the client and server (stateless communication).
- Transfer of packets that are larger than what can be handled in lower layers (link-layer)
- Can be performed in smaller blocks.
- The transfer of each block is acknowledged, enabling retransmission if required.
- Both client and server side have a say in the block size that actually will be used.
- The resulting exchanges are easy to understand using packet analyzer tools or better debugging.

Blockwise transfer [45] could be start either by client or server. “Block1” and “Block2” are

introduced by CoAP that appertained to the request payload and to the response payload respectively.

A block transfer option is constructed of three parts shown in figure 2.18. The first field (NUM) indicates the block sequence number that “0” means it is the first block of this message. The M flag indicates the number of blocks are followed, M=0 means this is the last block of the message. The maximum size of this block defines in SZX field that is calculated using the formula $\text{Size} = 2^{(\text{Exp} + 4)}$. For example the SXZ=2 corresponds to a block size value of 64 bytes that the payload would be divided to 64 bytes and distributed to some blocks needed to transfer whole data. Note that the Block options support only a power of two block sizes from 16 to 1024 bytes.

Bytes=4	0	2
NUM	M	SXZ

Figure 2.18: Block transfer option

2.9 CONTIKI OPERATING SYSTEM

The fast growth of WSN applications in recent years to *monitor* physical or environmental conditions, such as temperature, sound, pressure, etc it brings the necessitate to improve the developments of wireless sensor network application. Since WSNs contain hundreds or thousands of cheap and small sensor devices and low power wireless communication, it is important to be able to dynamically download code into the network. So middleware and simple operating systems for Wireless Sensor Network application are essentially needed.

Depend on the application requirements that have to be fulfilled the hardware and software chosen for the particular application could be different. For example a microcontroller as a CPU used by hardware, is not very powerful because the main focus of those motes. So the operating systems such as Windows or Linux or operating systems for powerful embedded systems like smart phones could not be run on the sensor motes. One of the most critical software that we have to select for the application is Operating System. There are a lot of operating systems developed for sensor nodes. Two of the most known sensor node operating systems are Contiki[34] and TinyOS [36].

In this project we used Contiki, a lightweight operating system developed at the Swedish

Institute of Computer Science by Dunkels et al. [35]. It supports for dynamic loading and replacement of individual programs and services. It has possibility of multi-threading atop of an event driven kernel. A typical Contiki configuration needs 2 kilobytes of RAM and 40 kilobytes of ROM.

2.9.1 CONTIKI STRUCTURE

Contiki is an open source Operating System written in the C programming language. The main system parts are: kernel, libraries, program loader, and a set of processes.

Contiki provides micro IP and Rime communication stacks:

Micro IP is a small RFC-compliant TCP/IP stacks called uIP which provides Internet communication abilities to Contiki while Rime is a lightweight communication stack developed especially for low-power radios.

Rime is a set of communication means, e.g. unicast, broadcast, trickle that used when bandwidth is at a premium or where the full IPv6 networking stack is overkill. Rime can send a message to all neighbors or to a specified neighbor, as well as more complex mechanisms such as network flooding and address-free multi-hop semi-reliable scalable data collection.

Contiki supports TCP/IP networking using uIP TCP/IP stack which provides TCP, UDP, IP, and ARP protocols.

Contiki Libraries

The convenience libraries prepared by Contiki to manage the memory and linked list operations are explained briefly.

Timer library to set, reset and restart timers, also it checks the timer expiration. The application is not done automatically.

Memory block management provides set of functions for managing a set of memory blocks of fixed size.

Linked list library provides a set of functions for manipulating linked lists.

In order to reduce consuming of energy, Contiki uses sleep mode instead of using power saving, while scheduler will be in charge of energy saving by putting any specific peripheral hardware, CPU and Microcontroller inside of sleep mode where there is no more queue.[34]

Chapter 3 System architecture

3.1 Hardware

As we explained in section 2, sensor node is one of the most important elements in WSN since Wireless Sensor Networks consist of hundreds to thousands of sensor nodes to sense or measure physical data of the area to be monitored. Researchers always have been trying to find the best and significant way to decrease the power consumption of the sensor nodes while increasing of efficiency of this important element because high consumption is a crucial issue towards to the sensor nodes. Still there is lack of diversity of sensor nodes to support this area.

Wireless sensor networks are used and designed according to the specific application. Therefore the selection of required motes will be based on the application requirements'. Zolertia and Wasp mote are two different available sensor nodes that will be presented in the next subsection.

3.1.1 Wasp mote sensor node

Wasp mote (Figure 3.1) is an open source wireless sensor platform launched by Libelium in November 2009 [37].

It is included of six connectors that enable sensor probes to be attached to the platform easily. There are more than 60 sensors available for Wasp mote. Humidity, temperature, vehicle detection, radiation, current, liquid and luminosity are some examples of Wasp mote sensors.



Figure 3.1: Wasp mote sensor

Libelium tries to save battery when it is not transmitting data by using the sleep features and cyclic waking up. It uses Over The Air Programming (OTAP) [41] in order to upgrade the firmware wirelessly. Wasmote hardware includes an ATmega1281 Microcontroller with 14 MHz frequency, a 8KB SRAM, 4KB of EEPROM and 28KB of Flash Memory. It also accepts SD Cards up to 2GB. XBee communication module (Figure 3.2) is used for Wasmote to exchange the messages which offers the IEEE 802.15.4 connectivity in the 2.4GHz ISM band. Node discovery and Duplicated packet detection are other functionalities provided by this mote.

Wasmote wireless interfaces are as follow:

3G, GPRS, 868 and 900MHz are used for long range, ZigBee, 802.15.4 and WiFi for medium range and RFID, NFC and Bluetooth 4.0 for short range.

3.1.1.1 Wasmote Gateway

Wasmote Gateway will be used to gather the data collected by Wasmote sensors and store into the receiving equipment which can be a PC with Linux, Windows or Mac OS, or any device compatible with standard USB connectivity. The gateway offers a USB A plug connector, so the receiving device must have a USB A receptacle. Wasmote Gateway is shown in figure 3.2.



Figure 3.2: Wasmote Gateway

3.1.2 Z1 Zolertia

Z1 [16] (Figure 3.3) sensor node that is explained in section 2, supports WSN protocols such as 2.4GHz IEEE 802.15.4, 6LowPAN compliant and ZigBee™ ready and Contiki OS which is our selected sensor node for this project.

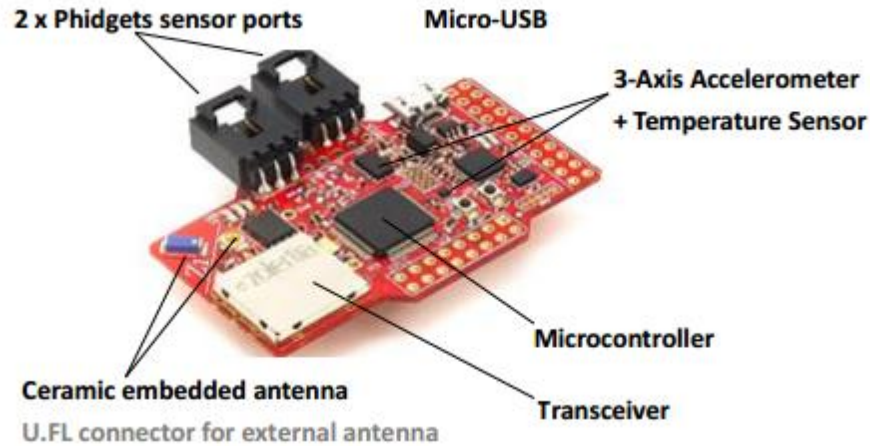


Figure 3.3: Wireless Sensor Node- Zolertia Z1

3.2 Challenges

Lack of robustness of protocols at all WSN levels make it difficult to be adopted really easy in the real-world and may make some challenges.

Up to now not so many related researches have been focused on implementing CoAP and HTTP in the networks using Z1 motes that are based on IEEE 802.15.4 and Contiki. Sometimes the problems we met were not presented before. Thus, it often took a lot of time to locate the problems, figure out the causes and work out the solutions.

The most important challenges that we encountered in order to carry on in this project, are listed in below.

3.2.1 Simulation VS. Real World Testing

In order to make up both practical and theoretical context to be significant in a real world, I carry on my project on a systematic and capable method. However when working with real-world experiments or deployments, many new issues arise.

There are many experiments and researches focused on implementing of simulation mode such as [42], but in simulation mode there is no any impact of real environment such as noises that are made by different sources or physical obstacles. Furthermore, simulation results are typically not reliable to operational networks due to the variety of different technologies used in a Wireless Sensor Network leading to very inconsistent network structures. For example the most frequent issue that I faced through carrying of this project was inability of battery energy power towards to the transmitting data across of devices in terms of long distance while in simulation mode, we

do not encounter with these kinds of issues. So the simulation is normally simple and convenient and the results may not be accurate for system evaluation.

3.2.2 Selection of the hardware

List of the specifications of the Z1 and Wasp mote nodes is shown in table 3.1.

Features	Z1	Wasp mote
Microcontroller	MSP430(F2617)	ATmega1281
Voltage	1.8 - 3.6V	3.3 - 4.2V
Operating Frequency	16MHz	14.75MHz
RAM	8KB	8KB
ROM	-	4KB
Flash	92KB	128KB
Power Consumption		
On mode	10mA	15mA
Sleep/Standby mode	0.5 μ A	55 μ A
Off Mode	0.1 μ A	0 μ A
Tx Mode	17.4mA	0.33mA

Table 3. 1: Z1 and Wasp mote specification

As a result Z1 seems like a fairly good hardware in implementation of our project objectives, because it is more energy efficient rather than Wasp mote, and it comes out as the best available option which has the ability to support a wide range of WSN network applications, also it is IEEE 802.15.4 compliant. In Addition Z1 is designed based on a well-established platform to run that it is supported by Contiki OS and is easier to run rather than Wasp mote, since there is no operating system running on Wasp mote's microcontroller, just the bootloader which allows uploading the code through the USB.

However the limitation of the Z1 memory size and processor's capacity could be evaluated as an important issue towards to the running of applications. This issue sometimes leads to insufficient storage of some essential information that does not allow us to implement some applications on that mote. Therefore we had to disable some functionality of some applications to solve this issue. For example, first, the idea was to run Radio Duty Cycle [38] mechanism in our project to save more energy, but due to the Z1 memory limitations (8Kb ram) we could not run that mechanism because RDC requires [43] additional program and RAM to store neighbor information. And we had to disable RDC in border-router/project-conf.h in Contiki to prevent any overload in the system.

Another most important issue was the limitation of msp430-gcc 4.4.5 compiler in the current 12.04 LTS Instant Contiki for compiling the programs over the 64kb. As my project data exceeded of 64kb and the compiler in Contiki does not support data regions above the 64kb, I have limited the resources from Contiki on the Z1 mote in order to avoid the risk of stack overflows.

3.2.3 Selection of the Operating System

As described in section 2, Contiki [34] and TinyOS [39] are two well-known operating systems developed for sensor nodes.

TinyOS is an open source project developed by the University of California in Berkeley which supports quite a few sensor node hardware platforms that runs on different microcontroller including the ATMEL AVR family of 8-bit microcontrollers, the Texas Instruments MSP430 family of 16-bit microcontrollers, different generations of ARM cores and etc.

The programming language of TinyOS is NesC, a “dialect” of the C programming language [39].

Table 3.2 shows some of contiki and TinyOS specifications.

Features	TinyOS	Contiki
Publication (Year)	ASPLOS (2000)	EmNets (2004)
Website	www.tinyos.net	www.sics.se/contiki
Static/Dynamic System	Static	Dynamic
Monolithic/Modular System	Monolithic	Modular
Networking Support	Active Message	uIP, uIPv6, Rime
Real-Time Guarantee	No	No
Language Support	nesC	C
Event Based Programming	Yes	Yes
Multi-Threading Support	Partial (through TinyThreads)	Yes (also supports Protothreads)
Wireless reprogramming	Yes	Yes
File System	Single level (ELF, Matchbox)	Coffee

Table 3.2: Comparison between TinyOS and Contiki OS

As presented before both operating systems fulfill the requirements we need for our project, although, Contiki is really especial as an OS in the communication protocol support. It offers IPv4 and IPv6, and a lightweight layered protocol stack called the rime stack which is explained in section 2.9.1.

By comparing of Contiki and TinyOS in terms of special features and also advantages of Contiki OS over the TinyOS that will be explained below, Contiki was our preferred Operating System for this project.

Concurrency: TinyOS only uses event-driven kernel in order to accomplish the concurrency requirements whilst Contiki uses different libraries to perform multithreading on top of event-driven kernel [34].

Flexibility: Both operating systems are flexible to handle different types of applications. Contiki can dynamically replace only the changed programs of the application, while an application using TinyOS has to be replaced completely, including the operating system. So Contiki would be the better option in case when the node software has to be updated often in a big WSN. [34]

Some of the issues we faced in regard to use Contiki OS are described as follow:

First, lack of good documentation or paper on Contiki OS yet, could take a very long time to fix the simple issues.

Second, Contiki does not include the most updated MSPGCC4 compiler needed to compile the z1 applications all, hence in order to run some codes on Z1 we have to program the drivers that is not so easy.

3.3 System architecture

Our system architecture assumes a WSN testbed that consists of three layers: One or more clients, a server, and a WSN testbed that illustrates the real world implementation of a RESTful WSN. So our WEB architecture is based on client-server model. In order to demonstrate the benefits of CoAP, we ran two simple experiments with the Contiki Operating System: the first one using CoAP over 6LoWPAN and the second one using HTTP over 6LoWPAN.

Figure 3.4 shows our testbed network. WSN client (HTTP or CoAP) queries the resources to WSN server (HTTP or CoAP) by issuing a Restful (GET/PUT/POST/DELETE) request through internet. In comparison with normal web services the resources asked by client would be different with respect to the data that could be measured by the server such as humidity, temperature, small packages, etc.

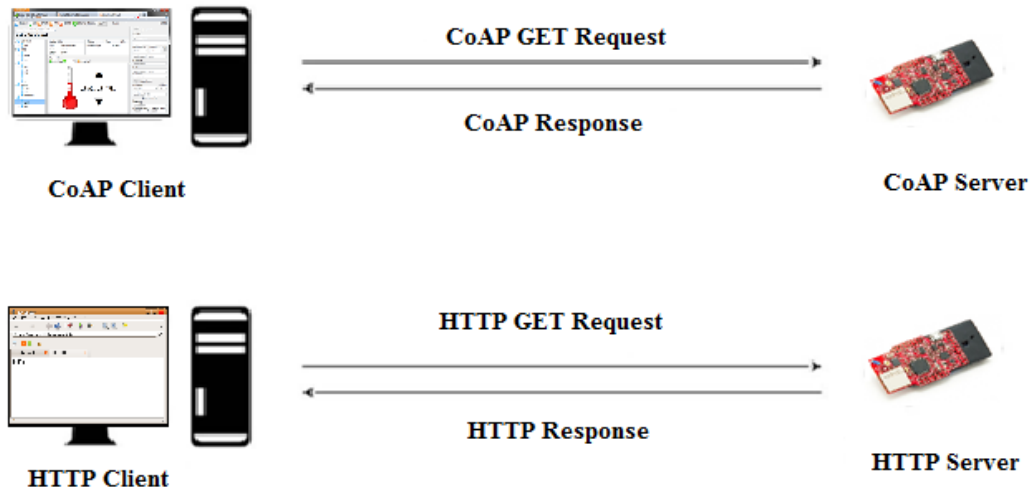


Figure 3.4: Testbed network. HTTP and CoAP

The HTTP and CoAP clients are located in a PC while the servers are embedded in a sensor.

The whole testbed services are usually implemented centrally as server-side software, and client-side software provides only the user interface to access those services. Sensor nodes acting as source and sink node.

Figure 3.5 shows our general system layout. WSN Client sends a query to the WSN Server nodes respectively to get their data using sensor node IPv6 address. A brief description of each module and an example of use is described below.

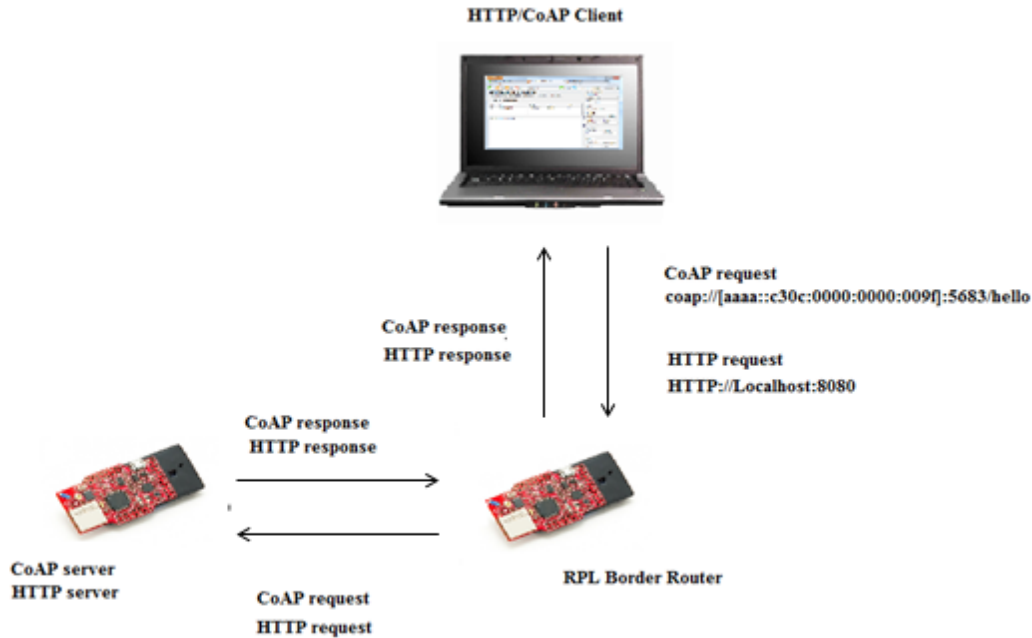


Figure 3.5: The complete architecture

3.3.1 RPL Border Router

Once the server got the request will answer back to the client through an additional element of the system called gateway (RPL-Border-Router), that acts as a bridge between sensor nodes and clients to forward the IPv6 packets received from a WSN Client to a WSN server by using Tunslip tool from Contiki to create a SLIP tunnel between a USB serial port and a virtual network interface. The received packet from USB port will be forwarded to the server node.

In this scenario RPL Border Router that is shown in Figure 3.5 is our Gateway that it periodically checks if more nodes joined or left the network to calculate the best route to forward the packet to its destination.

3.3.2 CoAP and HTTP Server

A CoAP or HTTP endpoint acts as a WSN server which a given resource resides or is to be created. Each resource defined by a name and has a handler function which is called by the REST (GET/PUT/POST/DELETE) query to serve the request. As an example "helloworld" in figure 3.5 is a resource that would be handled by "helloworld_handler" function which is called when a web service request is received for "helloworld" resource.

3.3.3 CoAP and HTTP Client

CoAP or HTTP clients are outside of the network and must be connected to the sensor nodes from outside of network through Internet to ask the server about the resources offered by it. To enable resource discovery an IP address and port number need to be standardized.

IANA has assigned the port number 5683 and the service name "CoAP", in accordance with [40] whilst HTTP uses port 8080.

The Client just creates and sends a request to the server via RPL Border Router and receives the response with the content and the response code.

An example of CoAP request is explained below.

CoAP Client sends a request to the RPL Border Router module that seems like:

```
coap://[aaaa::c30c:0000:0000:009f]:5683/hello
```

Where "[aaaa::c30c::009f]" is the IPv6 address of the CoAP server, 5683 the CoAP port number and "resource" the name of the resource as defined in the CoAP Server handler of that resource.

The RPL Border Router receives the request and forwards it to the CoAP server. Once the server receives the request, prepares the proper response involves the content and CoAP response code that would be "4.04 Not Found" in case it cannot find the resource and "2.05 Content" in case the resources exists. CoAP server forwards the response to the RPL border router which further forwards it to the CoAP client.

Chapter 4

4 Experimental environment

4.1 Software tools

4.1.1 Copper plug-in

The Copper (Cu) CoAP [44] user-agent for Firefox installs a handler for the 'coap' URI scheme and allows users to browse and interact with Internet of Things devices. Basically, Copper enables CoAP URIs to be simple entered into the address bar of the browser, thus extending the browser to behave like a CoAP Client. Using of Copper simplifies the debugging procedure where the work implementation can be controlled properly in a significant way.

The Figure 4.1 shows Copper plug-in user interface. In order to make a request for a resource located in the CoAP Server sensor node, the URI needs to be written as follow:

`”coap://[aaaa:c30c::0002]:5683/resource”`

Note that the `”aaaa:c30c::0002”` is the CoAP Server address, `”5683”` is the default configuration port for CoAP while `”/resource”` is the name of the requested resource.

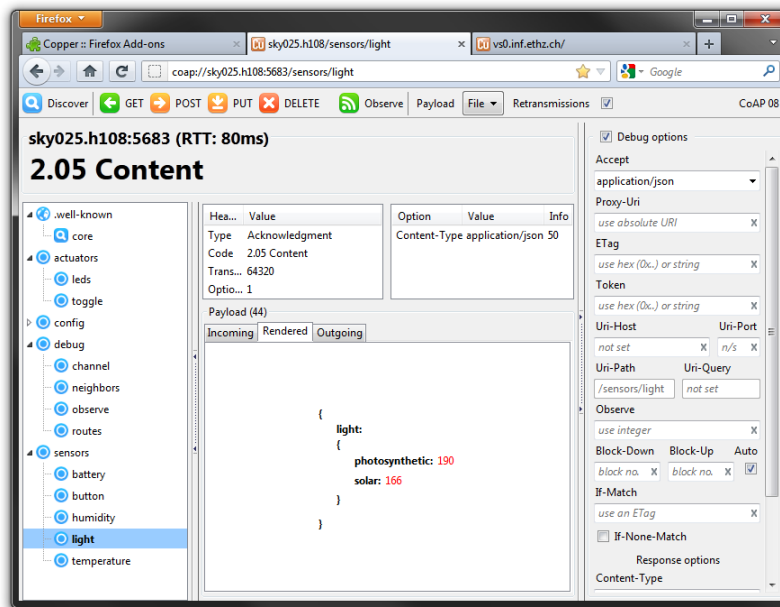






Figure 4.1: Copper Plug-in Interface

As it can be seen from figure 4.1 Client uses the buttons  GET,  POST,  PUT, and  DELETE to interact the resources hosted by a CoAP server. The resources are located on the left and will be displayed after issuing a discovery request to the sensor node. The

responses will be displayed in the browser. For example “Light/photosynthetic: 190” in the figure 4.1. Also, the copper plug-in shows the RTT (Round Trip Time) required between the request and the response (RTT=80 ms in this example). In addition, the type of the message can be configured (Confirmable or NON-Confirmable) as well as the block size in case in which the resource is a Blockwise resource.

Copper defined 'Behavior' menu to further customize the requests and their handling.

In case the response is larger than would fit in the buffer can repeat the request with a suitable value for the Block Option [44]. The size of each block could be set in the 'Behavior' menu in order to distribute the whole response to some blocks of size X. We can assign 16, 32, 64 or 128... 1024 Bytes to the X value.

4.1.2 Wireshark

Wireshark [47] is a network packet analyzer that tries to observe the messages exchanged between executing protocol entities which is an open source software project, and is released under the GNU General Public License (GPL).

We used Wireshark to analyze the network protocols such as HTTP, TCP, UDP, CoAP and IP. Figure 4.2 is an example of using Wireshark in our project which shows the layers in the TCP/IP network model of a packet.

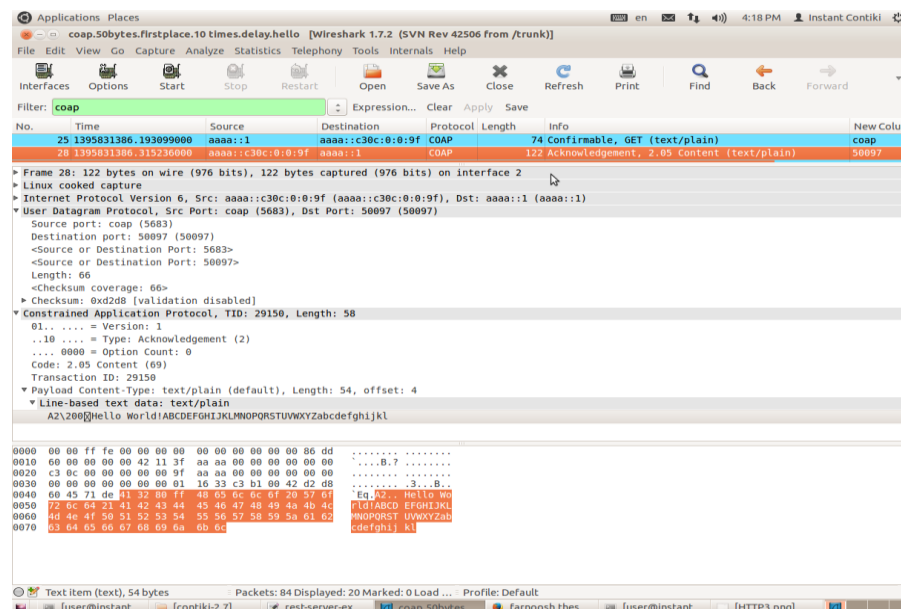


Figure 4.2: Wireshark tool

4.2 Scenarios

This chapter focuses on the implementation of experimental setup in an IEEE 802.15.4-based Wireless Sensor Network (WSN) to evaluate and analyze the performance of HTTP/TCP and CoAP/UDP network protocol stack in embedded devices. Contiki-2.7 OS was the operating system used for our implementations.

Different scenarios have been created in order to develop experiments about the used protocols. They are explained in the following subsections. For each value, the experiment was repeated 10 times and the results have been extracted from the average of these repetitions.

In order to compare the CoAP/UDP performance with HTTP/TCP performance in WSN two scenarios are categorized that each one explained in the following subsections

- CoAP/UDP scenario
- HTTP/TCP scenario

Figure 4.3 shows the real environment used for our tests. The architecture of laboratory and the different places with distances are shown in Figure 4.4. In this scenario the RPL Border Router (Gateway) was placed in the first place (Orange star) and the server was placed in three different places shown with yellow stars. The justification of these scenarios was to choose a first scenario without obstacles and with the client and server very near, and the second and third scenarios in which the client and server are in different rooms with a line of sight and with a non-line of sight and obstacles.

The distances between the Border Router and the Servers are shown in the figure 4.4 and are approximative. It is not so important the distance as the fact of having line of sight and non-line of sight conditions since the typical deployment scenarios are those ones in which the client and the RPL border router are near in distance (several meters) but with obstacles, e.g a balcony, a wall, a room with furniture, etc.



Figure 4.3: Real laboratory environment

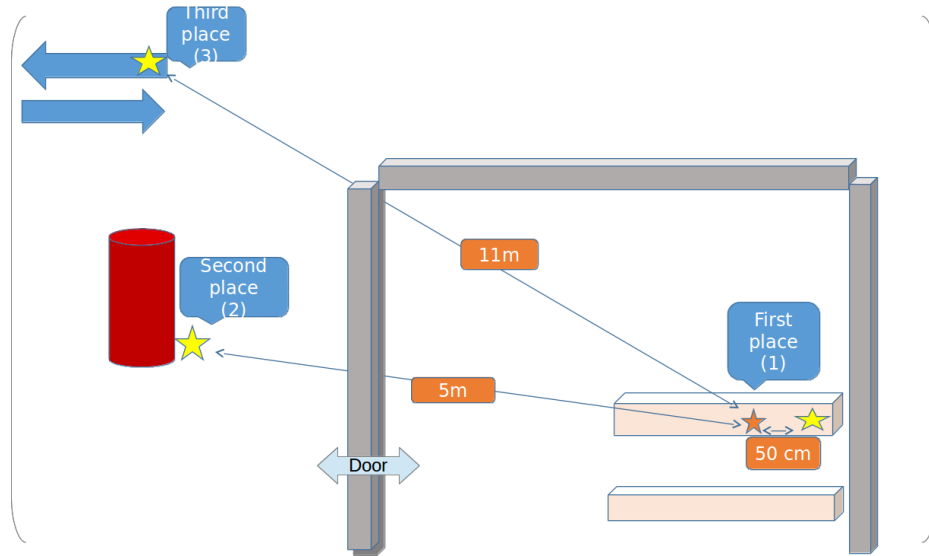


Figure 4.4: Architecture of laboratory

A general purpose implementation of COAP and HTTP was to see the transmission delay of these two protocols in delivering the responses in the system presented in figure 3.5.

4.2.1 CoAP scenario

The principal goal of this experiment is to deepen the understanding of the advantages of CoAP/UDP rather than HTTP/TCP in WSN.

In this scenario we analyze the CoAP performance by calculating the transmission delay and network layer overheads on the Wireless Sensor Network using different transaction sizes of 1, 50, 100 and 150 bytes.

4.2.1.1 CoAP implementation

Regarding to CoAP scenario, a COAP server supported in the Contiki OS has been installed in a Z1 node and a RPL border router also supported by Contiki has been installed in a second Z1 node.

A CoAP-client is running on a PC under Linux OS using Copper plug-in in order to retrieve the 'resources' hosted by a CoAP server by sending RESTFull requests with "GET" method to retrieve resources from WSN nodes.

CoAP uses the datagram-oriented UDP transport protocol to exchange messages. In our experiments CoAP use Piggy-backing [23] to carry the response in the acknowledgment message since the result was immediately available.

CoAP Client uses Copper plug-in in order to issue the request using server's IPv6 and send it to the RPL Border Router. The RPL Border Router will forward the IPv6 packets received from a CoAP Client to a CoAP server. A vice-versa procedure was then followed in order that the CoAP Client receives the CoAP response with the Hello-world on an ACK message. Figure 4.5 shows the response sent by server to CoAP client.

We repeated this experiment with different transactions. The resources are abstractions controlled by the server and identified by a Universal Resource Identifier (URI). Figure 4.5 shows how to send the get request using copper plug-in.

Coap URL would be like following:

coap://[aaaa::c30c:0000:0000:009f]:5683/hello

- "[aaaa::c30c:0000:0000:009f]" is our server IPv6 address.
- "5683" is the port number used by CoAP
- "hello" is the resource name.

As mentioned previously, different transaction sizes have been issued: 1, 50, 100, 150, 500 and 1000 bytes have been requested by the CoAP client.

As CoAP is designed to transfer small data like temperature, humidity, etc. In the experiments, there is no problem in using CoAP with transactions smaller than 64 bytes, due to the size of the payload. Remember that IEEE 802.15.4 has a maximum packet size of 128 bytes and uses payloads near 100 bytes. Taking into account the 6LoWPAN headers, the CoAP payload resulted in sizes lower than 64 bytes. On the other hand transactions of sizes larger than 64 bytes will need the Blockwise mechanism [45] explained in subsection 4.2.2. More details of Blokwise transfer can be seen in subsection 2.8.6.



Figure 4.5: Copper plug-in URI, Server resources and CoAP response

In order to illustrate how a transaction works, Figure 4.6 shows an example of retrieving a 50 byte transaction captured by Wireshark. Different network layers of transferring data in CoAP are shown in this figure.

As it can be seen from the picture, CoAP client starts the session by sending a 74 Byte Confirmable (CON) messages to carry a get request to the CoAP server. The CoAP server received the “GET” at time “1395831386.1930” and answered back to the CoAP client. (Blue line in the picture)

CoAP client received the CoAP Acknowledgment (ACK) messages with a 2.05 Content type message (Orange line in the picture). We define the Transmission delay as the total time measured from the time the CoAP Client issued the request until the time it received the response. In this example the delay would be: Transmission delay= 386.315-386.193=122 ms.

We point out that the Z1 server sensor that issues the data has not enough resources in terms of memory to install a network analyzer such as wireshark or similar. Thus, the captures are done in

the client that requests these data. The client is installed in a PC that has the resources to install the wireshark packet analyzer.

The overheads of network layers captured by wireshark are shown in the middle of the picture. Since wireshark catches the packets from Border Router to CoAP client so we couldn't catch the 6lowpan header. The message content has been shown in the bottom of the figure 4.6.

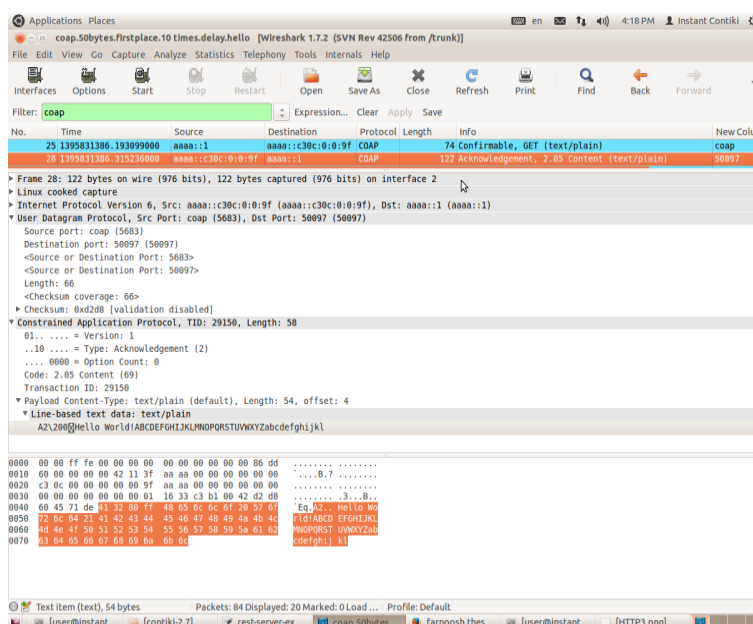


Figure 3.6: Network layers in CoAP

CoAP/UDP exchanging data is shown in Figure 4.7. CoAP server got the request at time 6.193s and CoAP client got the answer at time 6.315s.

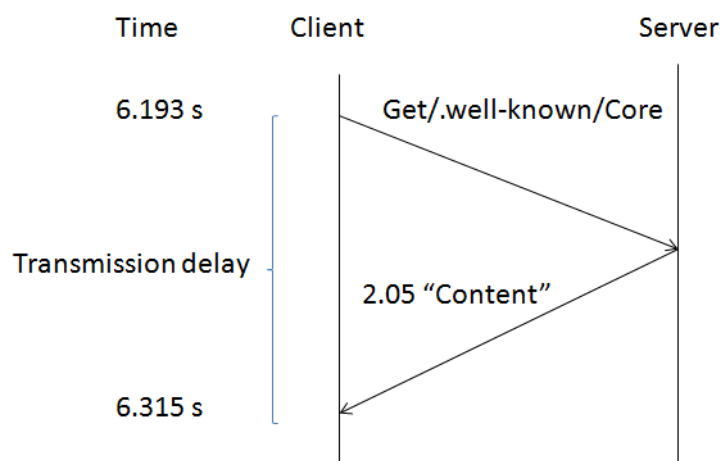


Figure 4.7: CoAP/UDP exchanging data

4.2.2 Coap Block-wise implementation

CoAP does not work very well with 6lowpan fragmentation in order to send larger data [48]. It is explained in more detail in section 5.1. CoAP uses Blockwise for transferring multiple blocks of information from a resource representation in multiple request-response pairs. [45]

CoAP is based on datagram transports, which limits the maximum size of resource representations (64 KB) that can be transferred. [23] Block-wise transfer is a mechanism defined by CoAP in order to split large data into blocks for sending and reassemble the blocks on the application layer upon receipt.

The scenario of sending the request and getting the response is the same of previous experiment. The only difference is that the client sends a GET request containing Block option, indicating block number and desired block size. In return, the server sends a response containing the requested block number and size. The transaction repeats until the client obtains the whole resource. If a response generated by a resource handler exceeds the client's requested block size, the server automatically divides the response and transfer it in several blocks.

In this experiment we used Chunks resource handle in order to transfer different Chuck Sizes which exceeds 64byte. Chuck Size is actually the size of the requested resource. We changed this parameter by editing the resource in the CoAP server. Additionally, in the copper plug-in we set the block size to 16 Bytes. That means the whole message would be divided into blocks of length 16 bytes. So the number of blocks needed to transfer a message of size 128 bytes, is equal to 8.

In order to illustrate how Blockwise works, an example of Blockwise transfer with chunk size equal to 100 bytes is presented in figure 4.8.

In this example the confirmable message to carry the GET request is 82 bytes. Blockwise mechanism sends a GET request per each block of message. So as a result it needs 14 packets (7 GET request packets and 7 Blocks of response) in order to transfer a message of size 100 with block size equal to 16 bytes. All packets captured by Wireshark are shown in figure 4.8.

Filter: coap		Expression... Clear Apply Save				
No.	Time	Source	Destination	Protocol	Length	Info
287	1395921759.172628000	aaaa::1	aaaa::c30c:0:0:9f	COAP	82	Confirmable, G
292	1395921759.201607000	aaaa::c30c:0:0:9f	aaaa::1	COAP	88	Acknowledgement
293	1395921759.320522000	aaaa::1	aaaa::c30c:0:0:9f	COAP	82	Confirmable, G
298	1395921759.349777000	aaaa::c30c:0:0:9f	aaaa::1	COAP	88	Acknowledgement
301	1395921759.477720000	aaaa::1	aaaa::c30c:0:0:9f	COAP	82	Confirmable, G
306	1395921759.506853000	aaaa::c30c:0:0:9f	aaaa::1	COAP	88	Acknowledgement
307	1395921759.624419000	aaaa::1	aaaa::c30c:0:0:9f	COAP	82	Confirmable, G
312	1395921759.653424000	aaaa::c30c:0:0:9f	aaaa::1	COAP	76	Acknowledgement

Frame 312: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface 3
 Linux cooked capture
 Internet Protocol Version 6. Src: aaaa::c30c:0:0:9f (aaaa::c30c:0:0:9f). Dst: aaaa::1 (aaaa::1)

Figure 4.8: Blockwise transfer, exchanging 100 bytes transaction using Chunks recourse

Figure 4.9 shows the CoAP packets exchanged using Blockwise transfer mechanism. The whole transaction is divided to some 2.05 content type messages and there is a Get message for each block that it seems there are some separated transactions to transfer. In this test the Transmission delay is defined as the difference time between the first GET request sent and the last 2.05 content type message received. In the example shown this one is: Transmission delay = 759.6534 - 759.1726 = 480 ms.

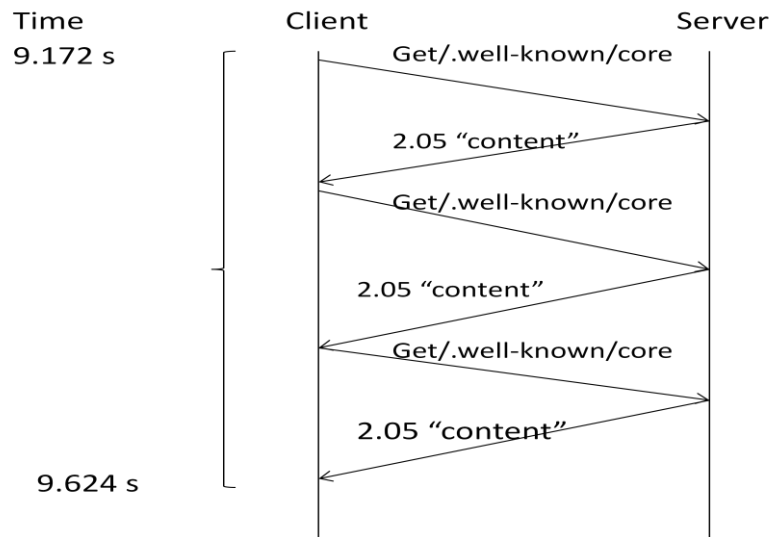


Figure 4.9: Blockwise transfer message exchange

4.2.3 HTTP scenario

The same scenario (motes and environment) as CoAP implementation was created in this experiment.

Figure 4.10 presented the experimental setup, consists of a testbed of two Z1 motes and a desktop computer running Ubuntu Linux. A HTTP server supported by the Contiki OS was installed in a Z1 sensor node and a RPL border router was installed in a second Z1 sensor node which is connected to the IP network. The Border Router sends and receives packets to and from the Linux PC by using Serial Line IP (SLIP). The communication between the sensors was over IEEE 802.15.4 protocol.

Regarding to this experiment, HTTP client and server was running under Contiki 2.7 operating system. HTTP client used curl (CurlHttpClient) command which is an object oriented wrapper of PHP cURL extension to send a GET request to the HTTP server through border router in order to retrieve the data.

The request would appear as follows:

```
curl -H "User-Agent: curl" aaaa::c30c:0:0:009f:8080/helloworld
```

“User Agent” field to fool the server into thinking you're one of those browsers. “aaaa::c30c:0:0:009f” is the server IP address. We specified the port number in the URL with a colon and a number immediately following the host name. Here doing HTTP to port 8080 and finally the resource name used in this experiment is specified after port number called helloworld.

Once the Border router receives this GET request from HTTP client, will forward the request to the HTTP server over IEEE 802.15.4 and the HTTP server sends back the response to the client .

HTTP protocol is built upon TCP/IP. TCP uses three-way handshake to initialize and end a connection, that means, the client must send a SYN and receive an ACK for it from the server.[20]

The following figure 4.11 was obtained from a Wireshark shows the TCP/IP and HTTP packets of transmitting a 100 byte transaction. The normal TCP connection establishment and termination, detailing the different states through which the client and server pass could be seen from the Figure 4.12. The Client sent the SYN to establish the connection. Server got the SYN at time 8.118, and then sent back the SYN/ACK. Once the server got the ACK, client sent the GET request to the server to start the packet transmission.

The Transmission Delay is defined as the time the client initiates the 3WSH mechanism until the client closes the connection issuing a FIN/ACK segment. TCP employs *fast retransmit* in a large window size to enhance its performance and link utilization.

As illustrated in following captures shown, the server initiated the close at time 9.947 by sending FIN packet to the client and finally the session were closed at 10.001, so the delay in this experiment was calculated as follows: Transmission delay=10.001-8.118= 1.883 seconds.

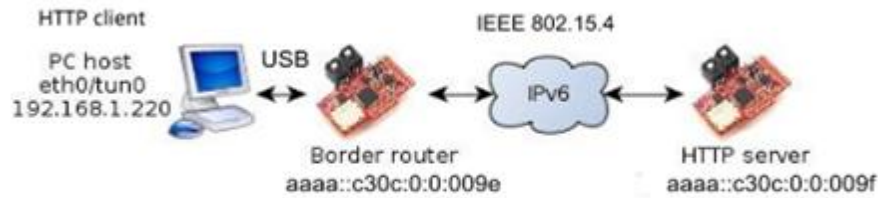


Figure 4.10: HTTP transferring data architecture in WSN

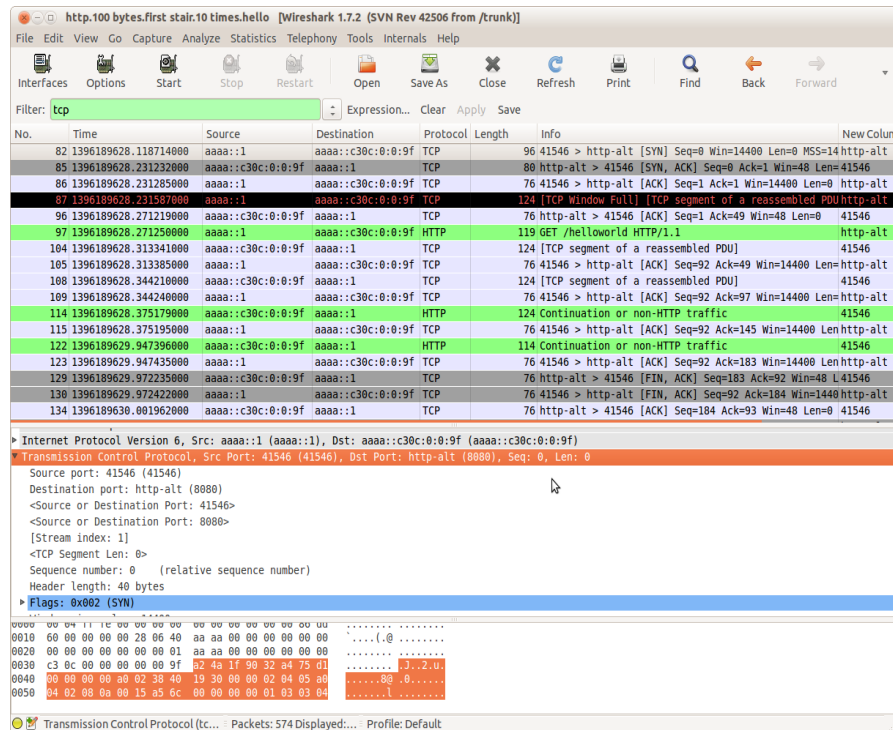


Figure 4.11: TCP connection with the timestamps

The initial SYN (packet 82) start at time 8.118. The first TCP header is 40 bytes long. The rest TCP header lengths are 20 bytes

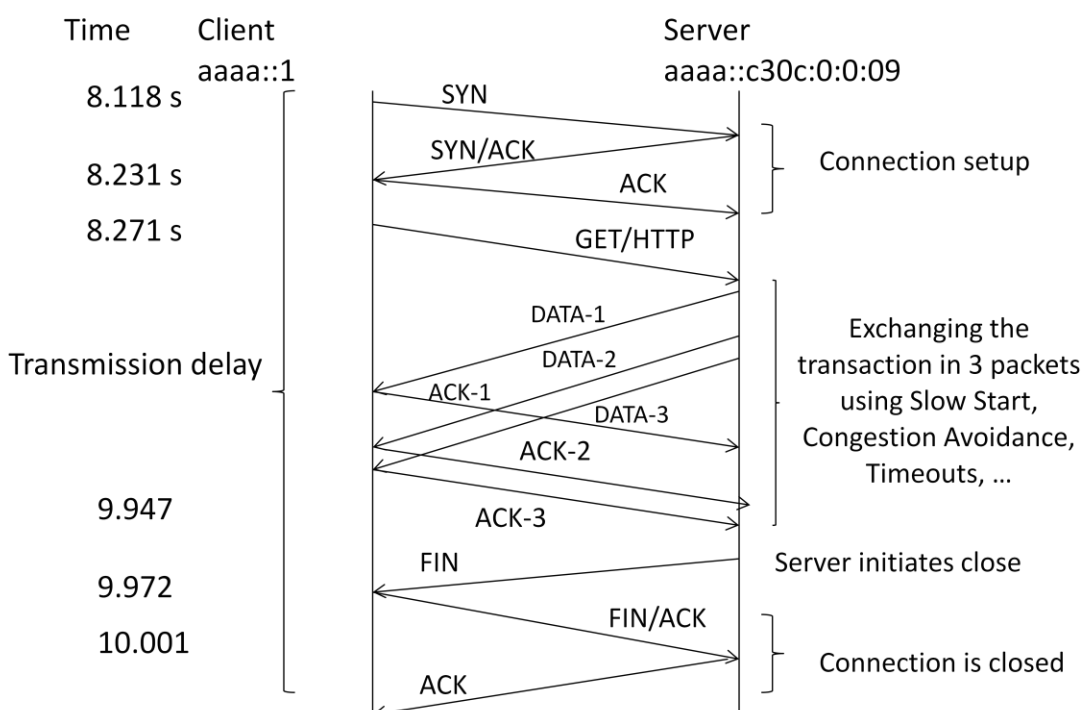
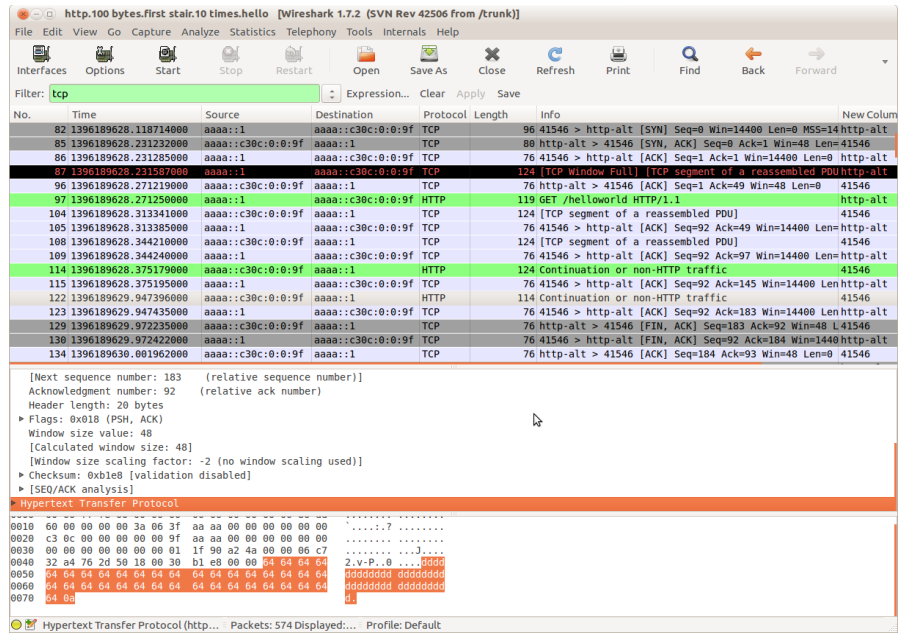


Figure 4.12: TCP 3-way handshake

Client starts to establish the connection at time 8.118, and sent the GET request at time 8.271. Server initiated close at time 9.947 and finally the session were closed at 10.001, so the delay is equal to 1.883 s

Due to the IEEE 802.15.4's standard packet size equal to 127 octets consisting of a 40 byte IPv6 header, 20 bytes TCP header, the available space for data payload is furthermore reduced and very few bytes are left for data. So the whole 100 bytes of transaction couldn't be sent in one packet and it distributed in 2 packets (HTTP/TCP packets in Figure 4.13). Last part of transaction is highlighted in the picture with orange color as Hyper Text Transfer Protocol. In this figure 37 bytes of the message is transferred in packet number 112 and totally 17 packets needed to transfer a 100 bytes data over TCP/IP.



Figur 4.13: Hyper Text Transfer Protocol

Chapter 5

5 Experimental results

5.1 CoAP scenario

In the following, experiments on packet reception delay for the different scenarios and for several packet sizes are implemented using CoAP application layer based on IEEE 802.15.4 communication protocol.

Table 5.1 shows a few results taken to evaluate the transmission delay by increasing the distance between server mote and border router with different transaction sizes on the CoAP server. The total number of bytes transmitted shows the size of the transaction during a retrieval of a resource. In this table, scenarios 1, 2 and 3 are those ones shown in the Figure 4.4. Figure 5.1 represents the diagram results.

places	Transmission delay for 1 byte transaction (sec)	Transmission delay for 50 bytes transaction (sec)	Transmission delay for 100 bytes transaction (sec)	Transmission delay for 150 bytes transaction (sec)	Transmission delay for 500 bytes transaction (sec)	Transmission delay for 1000 bytes transaction (sec)
1	0.169	0.176	1.508	1.694	11.796	19.484
2	0.167	0.311	1.763	3.277	Not received	NOT received
3	0.377	0.296	1.186	8.794	Not received	NOT received

Table 5.1: Communication timing requirements for transferring data over CoAP/UDP based on 802.15.4

As an example the transmission delay of the smallest transaction of 1 Byte in the first scenario that is the closest distance to the border router, was around 0.169 sec. Note that this is the total time the requests/responses needed to travel along the whole system (RPL Border Router and CoAP Server). By increasing the size of transaction to 50 bytes it can be observed that the delay was almost the same equal to 0.176 sec that means the delay caused by distance is not too much for small packets that it could be seen in figure 5.1 (Red and Green line).

It can be understood by the Figure 5.1 and Table 5.1 that the different scenarios does not have too much effect in the delay for small packet sizes. The reason is that CoAP protocol uses only one packet to transfer the transaction up to 64 bytes since the packet size of IEEE 802.15.4 is only 128 bytes, part of the packet size is utilized for layers overhead, so it needs only one packet for the GET request and one packet for the response. Without losses, therefore, there is no significant difference in the time required to transfer the transactions lower than 50 Bytes.

By increasing the transaction size the response generated by a resource handler exceeds the client's requested block size and causes the response to be divided into some blocks using simple stop-and-wait mechanism called "Blockwise transfers" [45]. Thus for transfers larger than 50 bytes the Block-wise mechanism comes into action.

The results show that the penalty of Blockwise transfers in terms of transmission delay is quite high in comparison with lower transfer sizes. As it is explained in previous chapter, using Blockwise transfer needs additional GET request packets (one packet per each block of data) to transfer whole transaction, so the amount of delay straightly depends on the number of transmitted frames. As it can be seen from the picture there is significant difference between the transmission delay of transactions with size 100 bytes and 150 bytes. The total number of blocks needed for transferring a transaction of 100 and 150 bytes with block size equal to 16 bytes without any noise and retransmission packets are 14 and 20 frames respectively. It means that in case of transaction of 100 bytes there are 7 GET requests of size 81 and 7 response messages which are carried in the resulting Acknowledgement (ACK) messages of size 88 bytes that is called a piggy-backed [23] response.

Another point that should be taking into account is the significant delay time of sending a 150 byte transaction from third place with more environmental obstacles. Interference and channel noise that occurs during transmission causes more delay in large data packets than smaller packets and forces CoAP retransmissions. Furthermore we did not have any problem in order to received the large data using blockwise transaction in the first scenario since CoAP Blockwise mechanism divided the data to some small blocks and transfers each part as a single transaction, waits to receive the ACK and sends the next GET message to request the next part of the message whiles in the second and third scenarios the client couldn't received the requested resource due to the retransmission loss caused by environmental obstacles and channel noises.

CoAP Blockwise is used to increase reliability using CoAP confirmable (CON) and acknowledge messages in large data transaction in the application layer instead of using 6LoWPAN fragmentation. In case of failure of a single packet causes only the retransmission of the relative request in Blockwise transfer while the fragmented packets of 6LoWPAN would be sent in a unique CoAP request/response transaction, therefore the ACK would be sent when all the fragments have been received. Since the number of retransmitted packets its effect on

reliability and delay transmission, we used Blockwise transfer for carrying the large data since it has less retransmitted packet. The only issue we faced was higher traffic generated in the network by Blockwise transfer due to the transmission of a CoAP ACK for each block.

Our experiments confirm that CoAP request/response cycles are most efficient when each message fits into a single 802.15.4 frame and CoAP is not suitable for sending large sums of data.

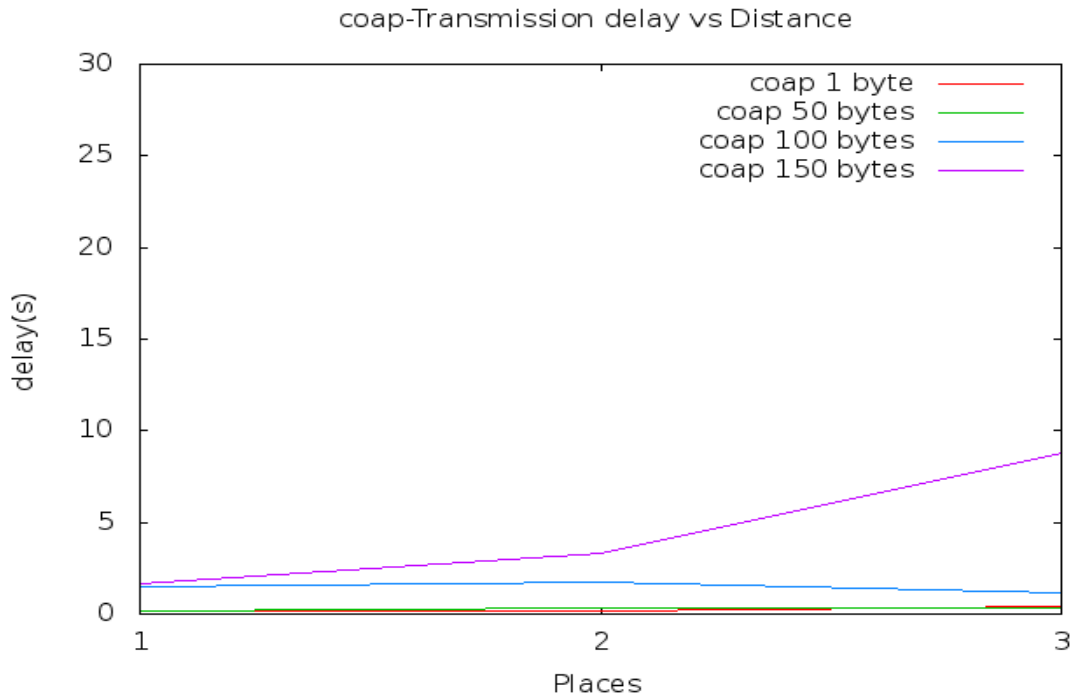


Figure 5.1: CoAP transmission delay vs. distance with different transaction sizes

5.2 HTTP scenario

Like CoAP scenario, experiments on packet reception delay against distances with different transaction sizes are made over HTTP/TCP application layer based on 802.15.4 communication protocol. We evaluated and measured the transmission delay by increasing the distance between border router that was located in the first place in Figure 4.4 and HTTP server by putting it in places 1, 2 and 3 for our experiments. Table 5.2 and Figure 5.2 present the results taken to evaluate the transmission delay

# places	Transaction Delay (sec) for 1 byte	Transaction Delay (sec) for 50 bytes	Transaction Delay (sec) for 100 bytes	Transaction Delay (sec) for 150 bytes	Transaction more than 150 bytes
1	0.378	0.366	0.777	0.802	NOT received
2	0.490	0.700	0.643	0.907	
3	1.808	1.551	8.165	21.021	

Table 5. 2: Communication timing requirements for data transferring over HTTP/TCP based on 802.15.4

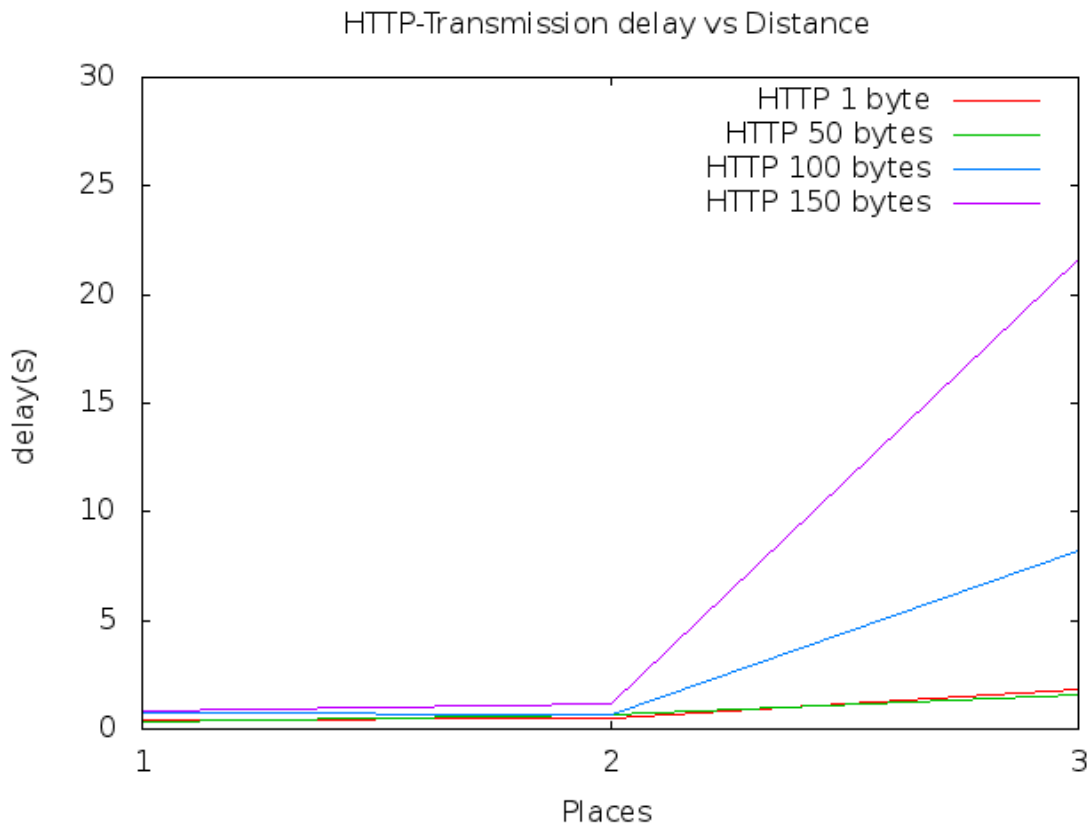


Figure 5.2: HTTP transmission delay in different locations

As it can be seen from the experimental results, there was not a significant difference in transmission delay obtained from transaction of size 1 and 50 bytes since they don't need more than 1 single HTTP packet to exchange through the network. The number of HTTP packets needs to exchange 50 and 100 bytes are shown in Figure 5.3 and 5.4 that are 2 and 4 HTTP packets respectively.

The most important result found out from the Table 5.2 is the delay time for transmission 150 bytes transaction in the third scenario. We faced some retransmissions in this experiment due to the decreasing in Wireless signal strength by increasing the distance between devices, furthermore there were additional interference from environmental sources such as stairs, elevator, columns, etc. that block radio waves. It is described in more detail in the next section.

In order to illustrate how HTTP/TCP based on 802.15.4 protocol divides the transaction to some small packets, an example of transferring 150 bytes transaction gathered by Wireshark represented in Figure 5.3. The 3 packets of an HTTP message are shown in the picture.

No.	Time	Source	Destination	Protocol	Length	Info
44	1396189620.326851000	aaaa::1	aaaa::c30c:0:0:9f	HTTP	119	GET /helloworld HTTP/1.
61	1396189620.432140000	aaaa::c30c:0:0:9f	aaaa::1	HTTP	124	Continuation or non-HT
67	1396189620.462190000	aaaa::c30c:0:0:9f	aaaa::1	HTTP	114	Continuation or non-HT

Filter: http		Expression...	Clear	Apply	Save
Frame 67: 114 bytes on wire (912 bits), 114 bytes captured (912 bits) on interface 3 Linux cooked capture Internet Protocol Version 6, Src: aaaa::c30c:0:0:9f (aaaa::c30c:0:0:9f), Dst: aaaa::1 (aaaa::1) Transmission Control Protocol, Src Port: http-alt (8080), Dst Port: 41545 (41545), Seq: 145, Ack: 92, Len: 38 Hypertext Transfer Protocol Data (38 bytes) Data: 64... [Length: 38]					

1000	00 00 ff fe 00 00 00 00	00 00 00 00 00 00 86 dd
1010	60 00 00 00 00 00 3a 06 3f	aa aa 00 00 00 00 00 00?
1020	c3 0c 00 00 00 00 00 9f	aa aa 00 00 00 00 00 00I....
1030	00 00 00 00 00 00 00 01	1f 90 a2 49 00 00 06 a5P..0 ./..ddd
1040	7f 11 12 9c 50 18 00 30	c9 2f 00 00 64 64 64 64P..0 ./..ddd
1050	64 64 64 64 64 64 64 64	64 64 64 64 64 64 64 64P..0 ./..ddd
1060	64 64 64 64 64 64 64 64	64 64 64 64 64 64 64 64P..0 ./..ddd
1070	64 0a	P..0 ./..ddd

Figure 5.3: HTTP packets of exchanging 50bytes transaction

5.3 Network layers overhead

Table 5.3 shows the total number of bytes transmitted at each layer. TCP based HTTP has the highest number of bytes transmitted at each layer since it sends several other TCP messages. Also Delayed ACK optimization implemented in most of the TCP/IP stacks has a negative effect for our domain due to small request and response size.

Network layer	HTTP/TCP (Bytes)	CoAP/UDP (Bytes)
Application layer	81	4
Link Layer, IPv6	40	40
Network layer,	20	8
Transport layer, 802.15.4	25	25

Table 5.3: Network layer overhead. HTTP/TCP vs. CoAP/UDP

5.4 CoAP VS. HTTP

After comparing tables 5.1 and table 5.2 we figure out some results regards to the CoAP and HTTP application layer in term of delay time.

The transmission delays for transaction of size 1 and 50 bytes in different distances from border router implemented over HTTP obviously show the highest number of retrieval time and bytes transmitted, since Each HTTP/TCP uses Three-Way Handshake included additional packets such as TCP-SYN, TCP-SYN-ACK and TCP-ACK messages that could be seen from figure 4.13. In contrast CoAP over UDP based protocols uses Piggy-Backed with only 2 messages to retrieve data (GET and ACK) (figure 4.7).

The most interesting result is referred to the transferring a transaction of 100 bytes. According to these experiment CoAP/UDP used *Blockwise* transfer and HTTP/TCP used TCP *fast retransmit* in order to send the large transaction. In order to illustrate the CoAP/UDP and HTTP/TCP data exchange mechanism Figure 5.4 shows an example of both protocols captured by Wireshark tool. In the CoAP example, 100 bytes of data divided to the 7 ACK blocks with a GET request packet per each block, so in this case we needed 14 packets in order to send our transaction (according to the block size of 16 bytes and 802.15.4 MTU). Whereas in HTTP scenario we have some additional TCP messages such as SYN, ACK, FIN, etc. plus fragmented transaction that in this example there are 3 fragmented packets (according to the 802.15.4 MTU and TCP header sizes). In total it needs 17 packets to transfer the data. Regards to the small block size in CoAP, more packets needed to transfer the big data over CoAP . Therefore the time needed to transfer a 100 byte transaction over CoAP/UDP is higher than HTTP/TCP in the first and second places. As a result we can reduce the number of packets and the transmission delay time needed to transfer a transaction by setting a bigger block size. However the 802.15.4 MTU must be taken into account.

We have to notice that the average packet losses tested using the ping command was of zero in the network. So the fading and shadowing effects have no significant influence on network performance in those cases.

Filter: coap Expression... Clear Apply Save						
No.	Time	Source	Destination	Protocol	Length	Info
269	1395921757.931593000	aaaa::1	aaaa::c30c:0:0:9f	COAP	80	Confirmable, GET (text/plain)
274	1395921758.929472000	aaaa::c30c:0:0:9f	aaaa::1	COAP	136	Acknowledgement, 2.05 Content
275	1395921758.987373000	aaaa::1	aaaa::c30c:0:0:9f	COAP	82	Confirmable, GET (text/plain)
280	1395921759.017547000	aaaa::c30c:0:0:9f	aaaa::1	COAP	88	Acknowledgement, 2.05 Content
281	1395921759.074864000	aaaa::1	aaaa::c30c:0:0:9f	COAP	82	Confirmable, GET (text/plain)
286	1395921759.103856000	aaaa::c30c:0:0:9f	aaaa::1	COAP	88	Acknowledgement, 2.05 Content
287	1395921759.172628000	aaaa::1	aaaa::c30c:0:0:9f	COAP	82	Confirmable, GET (text/plain)
292	1395921759.201607000	aaaa::c30c:0:0:9f	aaaa::1	COAP	88	Acknowledgement, 2.05 Content
293	1395921759.320522000	aaaa::1	aaaa::c30c:0:0:9f	COAP	82	Confirmable, GET (text/plain)
298	1395921759.349777000	aaaa::c30c:0:0:9f	aaaa::1	COAP	88	Acknowledgement, 2.05 Content
301	1395921759.477720000	aaaa::1	aaaa::c30c:0:0:9f	COAP	82	Confirmable, GET (text/plain)
306	1395921759.506853000	aaaa::c30c:0:0:9f	aaaa::1	COAP	88	Acknowledgement, 2.05 Content
307	1395921759.624419000	aaaa::1	aaaa::c30c:0:0:9f	COAP	82	Confirmable, GET (text/plain)
312	1395921759.653424000	aaaa::c30c:0:0:9f	aaaa::1	COAP	76	Acknowledgement, 2.05 Content
Frame 312: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface 3						
Linux cooked capture						
Internet Protocol Version 6, Src: aaaa::c30c:0:0:9f (aaaa::c30c:0:0:9f), Dst: aaaa::1 (aaaa::1)						
User Datagram Protocol, Src Port: coap (5683), Dst Port: 38000 (38000)						
Constrained Application Protocol, TID: 23247, Length: 12						

CoAP /UDP Packets

Filter: tcp Expression... Clear Apply Save						
Time	Source	Destination	Protocol	Length	Info	
25	1396189231.655109000	aaaa::1	aaaa::c30c:0:0:9f	TCP	96 41525 > http-alt [SYN] Seq=0 Win=14400	
28	1396189232.418155000	aaaa::c30c:0:0:9f	aaaa::1	TCP	80 http-alt > 41525 [SYN, ACK] Seq=0 Ack=	
29	1396189232.418248000	aaaa::1	aaaa::c30c:0:0:9f	TCP	76 41525 > http-alt [ACK] Seq=1 Ack=1 Win=	
30	1396189232.418532000	aaaa::1	aaaa::c30c:0:0:9f	TCP	124 [TCP Window Full] [TCP segment of a re	
39	1396189232.459129000	aaaa::c30c:0:0:9f	aaaa::1	TCP	76 http-alt > 41525 [ACK] Seq=1 Ack=49 Win	
40	1396189232.459162000	aaaa::1	aaaa::c30c:0:0:9f	HTTP	119 GET /helloworld HTTP/1.1	
49	1396189232.501553000	aaaa::c30c:0:0:9f	aaaa::1	TCP	124 [TCP segment of a reassembled PDU]	
50	1396189232.501597000	aaaa::1	aaaa::c30c:0:0:9f	TCP	76 41525 > http-alt [ACK] Seq=92 Ack=49 Wi	
51	1396189232.532079000	aaaa::c30c:0:0:9f	aaaa::1	TCP	124 [TCP segment of a reassembled PDU]	
52	1396189232.532096000	aaaa::1	aaaa::c30c:0:0:9f	TCP	76 41525 > http-alt [ACK] Seq=92 Ack=97 Wi	
57	1396189232.563017000	aaaa::c30c:0:0:9f	aaaa::1	HTTP	124 Continuation or non-HTTP traffic	
58	1396189232.563034000	aaaa::1	aaaa::c30c:0:0:9f	TCP	76 41525 > http-alt [ACK] Seq=92 Ack=145 W	
63	1396189232.592053000	aaaa::c30c:0:0:9f	aaaa::1	HTTP	114 Continuation or non-HTTP traffic	
64	1396189232.592067000	aaaa::1	aaaa::c30c:0:0:9f	TCP	76 41525 > http-alt [ACK] Seq=92 Ack=183 W	
71	1396189232.618756000	aaaa::c30c:0:0:9f	aaaa::1	TCP	76 http-alt > 41525 [FIN, ACK] Seq=183 ACK	
72	1396189232.619578000	aaaa::1	aaaa::c30c:0:0:9f	TCP	76 41525 > http-alt [FIN, ACK] Seq=92 ACK=	
77	1396189232.647712000	aaaa::c30c:0:0:9f	aaaa::1	TCP	76 http-alt > 41525 [ACK] Seq=184 Ack=93 W	
Frame 77: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface 3						
Linux cooked capture						
Internet Protocol Version 6, Src: aaaa::c30c:0:0:9f (aaaa::c30c:0:0:9f), Dst: aaaa::1 (aaaa::1)						
Transmission Control Protocol, Src Port: http-alt (8080), Dst Port: 41525 (41525), Seq: 10, Ack: 93, Len: 0						

HTTP/TCP Packets

Figure 5.4: CoAP and HTTP large data transferring

One of the interesting results is regarded to the transmission of large data by increasing the distance between server and border router. In this test the server was in a place with average packet loss equal to 7.6 % due to the environmental obstacles. In this test a 100 byte transaction took around 8 and 21 seconds to receive to the client respectively over HTTP while it took around 1 and 8 seconds over CoAP (Table 5.4). By looking to the Wireshark result in Figure 5.5 it has been observed the impact of fading and shadowing on network performance and get some retransmissions packets that can reduce the effective throughput.

Losses in HTTP/TCP required more time for retransmitting the entire transaction rather than CoAP/UDP, as the TCP retransmission mechanism (slow start, exponential back-off, timeouts multiple of 500 ms, etc) produce larger transaction delays than the simple CoAP retransmission mechanism; see the results of Table 5.2.

Another important result was that HTTP client did not receive the transaction bigger than 150 bytes. In the first scenario we did not have any environmental effects on the system as both Server and Border router were closed to each other, but the problem that we faced during this experiment was related to the low-power resource-constrained sensor nodes that are normally equipped with kilobytes of RAM which might be not enough for storing large data for processing. As the HTTP/TCP mechanism uses the *fast retransmit* mechanism to transfer the large data and sends the packets continuously the low-power microcontroller is not capable to

handle and process the large amount of data.

A: CoAP retransmission free

No.	Time	Source	Destination	Protocol	Length	Info
269	1395921757.931593000	aaaa::1	aaaa::c30c:0:0:9f	COAP	80	Confirmable, GET (text/p
274	1395921758.929472000	aaaa::c30c:0:0:9f	aaaa::1	COAP	136	Acknowledgement, 2.05 Co
275	1395921758.987373000	aaaa::1	aaaa::c30c:0:0:9f	COAP	82	Confirmable, GET (text/p
280	1395921759.017547000	aaaa::c30c:0:0:9f	aaaa::1	COAP	88	Acknowledgement, 2.05 Co
281	1395921759.074864000	aaaa::1	aaaa::c30c:0:0:9f	COAP	82	Confirmable, GET (text/p
286	1395921759.103856000	aaaa::c30c:0:0:9f	aaaa::1	COAP	88	Acknowledgement, 2.05 Co
287	1395921759.172628000	aaaa::1	aaaa::c30c:0:0:9f	COAP	82	Confirmable, GET (text/p
292	1395921759.201607000	aaaa::c30c:0:0:9f	aaaa::1	COAP	88	Acknowledgement, 2.05 Co
293	1395921759.320522000	aaaa::1	aaaa::c30c:0:0:9f	COAP	82	Confirmable, GET (text/p
298	1395921759.349777000	aaaa::c30c:0:0:9f	aaaa::1	COAP	88	Acknowledgement, 2.05 Co
301	1395921759.477720000	aaaa::1	aaaa::c30c:0:0:9f	COAP	82	Confirmable, GET (text/p
306	1395921759.506853000	aaaa::c30c:0:0:9f	aaaa::1	COAP	88	Acknowledgement, 2.05 Co
307	1395921759.624419000	aaaa::1	aaaa::c30c:0:0:9f	COAP	82	Confirmable, GET (text/p
312	1395921759.653424000	aaaa::c30c:0:0:9f	aaaa::1	COAP	76	Acknowledgement, 2.05 Co

B: HTTP retransmission

No.	Time	Source	Destination	Protocol	Length	Info
25	1396189610.63103	aaaa::1	aaaa::c30c:0:0:9f	TCP	96	41545 > http-alt [SYN] Seq=0 Win=14400 Len=http-alt
32	1396189610.91926	aaaa::c30c:0:0:9f	aaaa::1	TCP	80	http-alt > 41545 [SYN, ACK] Seq=0 Ack=1 Win=14400
33	1396189610.91931	aaaa::1	aaaa::c30c:0:0:9f	TCP	76	41545 > http-alt [ACK] Seq=1 Ack=1 Win=14400
34	1396189610.91959	aaaa::1	aaaa::c30c:0:0:9f	TCP	124	[TCP Window Full] [TCP segment of a reassehttp-alt
37	1396189610.91982	aaaa::1	aaaa::c30c:0:0:9f	TCP	124	[TCP Window Full] [TCP Retransmission] 415-http-alt
40	1396189610.98015	aaaa::1	aaaa::c30c:0:0:9f	TCP	124	[TCP Window Full] [TCP Retransmission] 415-http-alt
43	1396189620.32624	aaaa::c30c:0:0:9f	aaaa::1	TCP	76	http-alt > 41545 [ACK] Seq=1 Ack=49 Win=48 41545
44	1396189620.32685	aaaa::1	aaaa::c30c:0:0:9f	HTTP	119	GET /helloworld HTTP/1.1 http-alt
51	1396189620.36918	aaaa::c30c:0:0:9f	aaaa::1	TCP	124	[TCP segment of a reassembled PDU] 41545
52	1396189620.36978	aaaa::1	aaaa::c30c:0:0:9f	TCP	76	41545 > http-alt [ACK] Seq=92 Ack=49 Win=1 http-alt
57	1396189620.40148	aaaa::c30c:0:0:9f	aaaa::1	TCP	124	[TCP segment of a reassembled PDU] 41545
58	1396189620.40142	aaaa::1	aaaa::c30c:0:0:9f	TCP	76	41545 > http-alt [ACK] Seq=92 Ack=97 Win=1 http-alt
61	1396189620.43214	aaaa::c30c:0:0:9f	aaaa::1	HTTP	124	Continuation or non-HTTP traffic 41545
62	1396189620.43215	aaaa::1	aaaa::c30c:0:0:9f	TCP	76	41545 > http-alt [ACK] Seq=92 Ack=145 Win=1 http-alt
67	1396189620.46219	aaaa::c30c:0:0:9f	aaaa::1	HTTP	114	Continuation or non-HTTP traffic 41545
68	1396189620.46221	aaaa::1	aaaa::c30c:0:0:9f	TCP	76	41545 > http-alt [ACK] Seq=92 Ack=183 Win=1 http-alt
73	1396189620.48724	aaaa::c30c:0:0:9f	aaaa::1	TCP	76	http-alt > 41545 [FIN, ACK] Seq=183 Ack=92 41545
74	1396189620.48742	aaaa::1	aaaa::c30c:0:0:9f	TCP	76	41545 > http-alt [FIN, ACK] Seq=92 Ack=184 http-alt
81	1396189620.51844	aaaa::c30c:0:0:9f	aaaa::1	TCP	76	http-alt > 41545 [ACK] Seq=184 Ack=93 Win=1 41545
82	1396189620.11871	aaaa::1	aaaa::c30c:0:0:9f	TCP	96	41546 > http-alt [SYN] Seq=0 Win=14400 Len=http-alt
85	1396189620.23123	aaaa::c30c:0:0:9f	aaaa::1	TCP	80	http-alt > 41546 [SYN, ACK] Seq=0 Ack=1 Win=14400
86	1396189620.23128	aaaa::1	aaaa::c30c:0:0:9f	TCP	76	41546 > http-alt [ACK] Seq=1 Ack=1 Win=14400 http-alt

A: CoAP retransmission free

B: HTTP retransmission

Figure 5.5: Fading effect on HTTP and CoAP

Transaction size	Delay time over CoAP/UDP	Delay time over HTTP/TCP
1	0.377	1.808
50	0.296	1.551
100	1.186	8.165

Table 5. 4: Comparing CoAP and HTTP transmission delay in term of transaction size

Places	Delay time over CoAP of 50 bytes transaction	Delay time over HTTP of 50 bytes transaction
1	0.176	0.366
2	0.311	0.700
3	0.296	1.551

Table 5. 5: Comparing CoAP and HTTP transmission delay in term of distance

5.5 Qualitative Results

One of the objectives of the master thesis has been the integration of a Hw platform like Z1 Zolertia nodes with an operating system like Contiki implementing a developing protocol stack like CoAP. Here, we outline the difficulties encountered in the integration of such platform.

- a) We point out first the difficulty in working with a platform under development. The CoAP/UDP protocol stack with 6LowPAN over IEEE 802.15.4 used in the experiments was not fully tested at the beginning of the Master Thesis. Part of the master thesis consisted in making that this platform worked.
- b) Due to the lack of some Z1 drivers in Contiki, some results could not be achieved. In addition Contiki does not include the most updated MSPGCC4 compiler needed to compile the z1 applications all, hence in order to run some codes on Z1 we should have to program the drivers that is not so easy.
- c) Z1 memory constrained and processor's capacity could be evaluated as an important issue towards to the running of applications that we couldn't implement some applications on the mote due to the insufficient storage. Therefore we had to disable or configure some functionality of some applications such as Radio Duty Cycle to solve this issue.
- d) Another most important issue was the limitation of msp430-gcc 4.4.5 compiler in the current 12.04 LTS Instant Contiki for compiling the programs over the 64kB. As my project data exceeded of 64kB and the compiler in Contiki does not support data regions above the 64kB, I have limited the resources from Contiki on the Z1 mote in order to avoid the risk of stack overflows.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis we present how to embed RESTful Web services into WSNs. In particular, the purpose of this project is to evaluate and analyze the network protocols in IoT provided by CoAP/UDP and HTTP/TCP based on IEEE 802.15.4 protocol. All the solutions considered in this project are discussed and evaluated in a real Contiki based 6LoWPAN network using some Z1 sensor nodes.

We measured the latency experienced by a client when retrieving information from a server where HTTP/TCP has been compared with CoAP/UDP in order to evaluate the performance of a significant protocol in terms of efficient energy consumption and transmission delay.

Regarding to CoAP scenario we run `er-example-server.c` found in `contiki/examples/er-rest-example` as a CoAP server and `rpl-border-router.c` found in `contiki/examples/ipv6/rpl-border-router` as a border router on two Z1 sensors. CoAP-client is running on a PC under Linux OS using Copper plug-in in order to retrieve the 'resources' hosted by a CoAP server by sending RESTFull requests with "GET" method to retrieve resources from WSN nodes. In our experiments CoAP use Piggy-backed [23] to carry the response in the acknowledgment message since the result was immediately available.

Regarding to HTTP experiment, HTTP client and server was running under Contiki 2.7 operating system. HTTP client used `curl` (`CurlHttpClient`) command which is an object oriented wrapper of PHP cURL extension to send a GET request to the HTTP server through border router in order to retrieve the data. Unlike HTTP based protocols, CoAP operates over UDP and employs a simple retransmission mechanism instead of using complex congestion control as used in standard TCP.

The results of the analysis suggest that:

CoAP protocol works better in transferring small transaction as it just needs 2 packets to transfer the data whereas HTTP needs 14 packets in the same case because of using TCP three way handshaking. In addition in case of sending large data, both protocols divide the message to some blocks and add one extra packet per each segment. So the number of packets using by CoAP/UDP and HTTP/TCP increases significantly. CoAP uses Blockwise transfer [23] while HTTP uses the TCP *three-way-handshake* messages and just divided the transaction to some smaller packets, so they are not suitable to transfer large data as it consume more energy and generate more traffic.

In case of exchanging large data from long distance the CoAP provides the better performance. We faced some retransmissions in this experiment due to the decreasing in Wireless signal strength by increasing the distance between devices. Losses in HTTP/TCP required more time for retransmitting the entire transaction rather than CoAP/UDP, as the TCP retransmission mechanism (slow start, exponential back-off, timeouts multiple of 500 ms, etc) produce larger transaction delays than the simple CoAP retransmission mechanism.

As a consequence the lower communication overhead of the CoAP protocol due to using lower number of messages when retrieving resources and the simpler hardware requirements caused CoAP/UDP based protocols perform better for constrained networks compared to HTTP based resource retrievals.

6.2 Summary conclusion

As a summary conclusion we can state that CoAP/UDP works well with IEEE802.15.4 since the MTU is small, something that penalizes the HTTP/TCP transactions. However, in a MAC with larger MTU at high rates, like WiFi IEEE802.11 family, with a MTU of 1500 Bytes, a HTTP transaction can be done with one packet at larger distances. Thus, it could provide a good solution for using the HTTP/TCP stack. The development of the LowPower WiFi seems to point in this direction: a MAC optimized to consume low energy but compatible with the HTTP/TCP stack instead of the current IEEE 802.15.4 low power MAC for sensors that has to optimize the application layer (CoAP instead of HTTP).

6.3 Future Work

One of the most important advantages of working on this project was, I could join to Zolertia Company located in Barcelona. I could carry on the project in both theoretical and practical framework where all tests have been done in a real world. IN addition I could work into a company where I could develop myself in terms of new knowledge and skills related to my study plan. Furthermore I have been familiar with a modern full-stack operating system designed for ARM Cortex-M-based MCUs called mBED [46] which is an open-source OS. The idea is to create an mBED_enabled platform of our own that is supported within the mbed platforms database and tools. We can implement our project on free mbed Online Compiler and mbed Developer Platform that provides us apply new ideas due the change of new technology.

BIBLIOGRAPHY AND REFERENCES

- [1] Dr. Ovidiu Vermesan SINTEF, Norway Dr. Peter Friess EU, Belgium. Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems. 2013
- [2] G. Montenegro, N. Kushalnagar, J. Hui, D. Culler. RFC 4944: Transmission of IPv6 Packets over IEEE 802.15.4 Networks. 2007
- [3] J. Yick, B. Mukherjee, and D. Ghosal. Wireless sensor network survey. Journal of Computer Networks, Volume 52(Number 12):22922330, 2008
- [4] RAFAEL BASSO. Wireless Sensor Networks in a Vehicle Environment. 2009
- [5] T. Le Dinh, W. Hu, P. Sikka, P. Corke, L. Overs, and S. Brosnan. Design and deployment of a remote robust sensor network: Experiences from an outdoor water quality monitoring network. In Local Computer Networks, 2007. LCN 2007. 32nd IEEE Conference on, pages 799806. IEEE, 2007
- [6] Bengi Aygün, Vehbi Cagri Gungor. Wireless sensor networks for structure health monitoring: recent advances and future research directions. Journal: Sensor Review Volume: 3 Number: 3 Year: 2011
- [7] Adam Dunkels, Björn Grönvall, and Thiemo Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)*, Tampa, Florida, USA, November 2004.
- [8] Wikipedia, “Wireless sensor node”, http://en.wikipedia.org/wiki/Sensor_node, accessed on May 2012.
- [9] List of wireless sensor nodes:
http://en.wikipedia.org/wiki/List_of_wireless_sensor_nodes#List_of_gateway_sensor_nodes.
- [10] Contiki operating system official website: <http://www.contiki-os.org/>.
- [11] TinyOS official website: <http://www.tinyos.net/>.
- [12] MSP430F2617 datasheet: <http://www.ti.com/lit/ds/symlink/msp430f2617.pdf>

- [13] CC2420 datasheet: <http://www-mtl.mit.edu/Courses/6.111/labkit/datasheets/CC2420.pdf>
- [14] ADXL345 datasheet:
http://www.analog.com/static/imported-files/data_sheets/ADXL345.pdf
- [15] TMP102 datasheet: <http://www.ti.com/lit/ds/symlink/tmp102.pdf>.
- [16] Z1 datasheet, http://zolertia.sourceforge.net/wiki/images/e/e8/Z1_RevC_Datasheet.pdf
- [17] IEEE Standard 802.15.4-2003: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs), May 2003.
- [18] S. Hara, D. Zhao, K. Yanagihara, J. Taketsugu, K. Fukui, S. Fukunaga, and K. Kitayama, "Propagation Characteristics of IEEE 802.15.4 Radio Signal and Their Application for Location Estimation," *Proc. IEEE Vehicular Technology Conf.*, pp. 97-101, June 2005.
- [19] Kushalnagaret, N. al. IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals, IETF RFC 4919, 2007.
- [20] Nam, C.S.; Jeong, H.J.; Shin, D.R. Extended Hierarchical Routing over 6LoWPAN, 4th International Conference on Networked Computing and Advanced Information Management, 2008, pp. 403 - 405.
- [21] S. Sheng, F. Arias ICANN, F. Obispo ISC, N. Kong CNNIC, A RESTful Web Service for Domain Name Registration Data, March 12, 2012.
- [22] R. Fielding UC Irvine, J. Gettys Compaq/W3C, J. Mogul Compaq, H. Frystyk W3C/MIT, L. Masinter Xerox, P. Leach Microsoft, T. Berners-Lee W3C/MIT, Hypertext Transfer Protocol - HTTP, June 1999.
- [23] Z. Shelby ARM, K. Hartke, C. Bormann Universitaet Bremen TZI, The Constrained Application Protocol (CoAP), June 2014
- [24] D. Pauli and D. I. Obersteg. Californium. Lab Project, D-INFK, ETH Zurich, Dec. 2011.
- [25] <http://www.vs.inf.ethz.ch/publ/papers/mkovatsc-2011-iotech-coap.pdf>
- [26] C. Lerche, K. Hartke, and M. Kovatsch. Industry Adoption of the Internet of Things: A

Constrained Application Protocol Survey. In Proceedings of the 7th International Workshop on Service Oriented Architectures in Converging Networked Environments (SOCNE 2012), Krakow, Poland, Sept. 2012.

[27] Fatma Bouabdallah, Nizar Bouabdallah, and Raouf Boutaba , “On Balancing Energy Consumption in Wireless Sensor Networks”,-July 2009 IEEE Transactions On Vehicular Technology, Volume. 58, Issue 6, Page(s). 117-123.

[28] Paulo Rogério Pereira, António Grilo, Francisco Rocha, Mário Serafim Nunes, Augusto Casaca, Claude Chaudet, Peter Almström and Mikael Johansson, “End-To-End Reliability In Wireless Sensor Networks:Survey And Research Challenges” International Conference on Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007, Page(s).2771 – 2774.

[29] Debnath Bhattacharyya, Tai-hoon Kim, Subhajit Pal. A Comparative Study of Wireless Sensor Networks and Their Routing Protocols. 24 November 2010

[30] René Hummen, Jens Hiller, Hanno Wirtz, Martin Henze, Hossein Shafagh, Klaus Wehrle. 6LoWPAN Fragmentation Attacks and Mitigation Mechanisms. 2013.

[31] J. Vasseur and A. Dunkels. Interconnecting Smart Objects with IP - The Next Internet. Morgan Kaufmann, 2010.

[32] J. Postel, ISI, User Datagram Protocol, 28 August 1980.

[33] Koojana Kuladinithi,Olaf Bergmann, Thomas Pötsch, Markus Becker, Carmelita Görg, Implementation of CoAP and its Application in Transport Logistics, (April 2011)

[34] Betreuer: Christoph Söllner, Tobias Reusing. Comparison of Operating Systems TinyOS and Contiki, 2012

[35]] A. Dunkels, B. Grönvall, T. Voigt Contiki – a Lightweight and Flexible Operating System for Tiny Networked Sensors In Proceedings of the First IEEE Workshop on Embedded Networked Sensors, Tampa, Florida, USA, 2004

[36] David E. Culler, Arch Rock Corp.TinyOS: Operating System Design for Wireless Sensor Networks, May 1, 2006

[37] Waspnote website - documentation," <http://www.libelium.com/development/waspnote/>

documentation.

[38] A. Dunkels, L. Mottola, N. Tsiftes, F. Osterlind, J. Eriksson, and N. Finne. The Announcement Layer: Beacon Coordination for the SensorNet Stack. In Proc. EWSN, Bonn, Germany, 2011.

[39] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, K. S. J. Pister System architecture directions for networked sensors In SIGPLAN Not. 35 (11), p.93–104, ACM, 2000

[40] M. Cotton, L. Eggert, J. Touch, M. Westerlund, S. Cheshire, Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry, August 2011

[41] Gascón, David; Alberto Bielsa; Félix Genicio; Marcos Yarza (9 May 2011). "[Over the Air Programming with 802.15.4 and ZigBee - OTA](#)". *Libelium*. Libelium. Retrieved 28 May 2012

[42] ThingsLinus Wallgren, Shahid Raza, and Thiemo Voigt1 Routing Attacks and Countermeasures in the RPL-Based Internet of, 2013.

[43] The page of Contiki: <https://github.com/contiki-os/contiki/wiki/Radio-duty-cycling>

[44] The main web page of CoAP: <http://people.inf.ethz.ch/mkovatsc/>

[45] C. Bormann, Z. Shelby, Ed ,Blockwise transfers in CoAP, February 15, 2012.

[46] mBED main page. <https://mbed.org/technology/os/>

[47] Wireshark main web page. <https://www.wireshark.org/>

[48] Alessandro Ludovici, Piergiuseppe Di Marco, Anna Calveras and Karl H. Johansson Analytical Model of Large Data Transactions in CoAP Networks. 2014

