الآية

بسم الله الرحمن الرحيم

﴿قَالُواْ سُبْحَانَكَ لاَ عِلْمَ لَنَا إِلاَّ مَا عَلَّمْتَنَا إِنَّكَ أَنتَ الْعَلِيمُ الْحَكِيمُ﴾

سورة البقرة (٣٢)

Dedication

I dedicated this thesis To:

The sake of Allah, my Creator and my Master,

School of Electronics Engineering Sudan University of Science and Technology,

My parents: words are just not expressive enough; they introduced me to the joy of reading from birth enabling such a sudy to take place today,

My brothers, sister and my best friends without them none of my success will be possible,

My beloved kid: Dania who I can't force myself to stop loving her,

All the people in my life who touch my heart,

I hope this will be a source inspiration and motivationfor all whom watching my presentation.

Acknowledgement

All praise and thanks to ALLAH, who provided me the ability to complete this work. I am thankful of my mother and my family who is always support and help me the whole years of study. I hope I can give that back to them.

I would like to express my special appreciation and thanks to my advisor Dr. SALAH EDAM being such an inspiring person to work with and great thanks forencouraging my research and allowing me to grow as a research scientist. Your advice on both research as well as on my career have been priceless. I would also thank my co-supervisor Dr.SaraAljak for her support.

Abstract

The Constrained Application Protocol (CoAP) is used with TinyOS which called TinyCoAP to give the same features of HTTP while keeping a simple design and low overhead. TinyOS already have another implementation of CoAP in its distribution called CoapBlip.However, it’s a library doesn't meet the requirements ofTinyOS.TinyCoAP and CoapBlip are evaluated using Avrora and TOSSIM simulations, as well as implementations based on HTTP. The evaluation is performed in terms of latency, memory occupation, and energy consumption. It shows that TinyCoAPhas the best performance in most parameters comparing it with other implementations.

TinyCoAP shows important development in performance compared with CoapBlip which is limited by the implementation of dynamic RAM memory allocation and the use of an external C library. HTTP/TCP has the worst performance than that obtained by TinyCoAP. The performance of TinyCoAP is same as HTTP/UDP but with high reliability.

مستخلص

بروتوكول التطبيقات المقيدة (CoAP)قد استخدم مع نظام التشغيل TinyOS ويسمى ببرتوكول TinyCoAPلاعطاء نفس خصائص HTTP مع الاحتفاظ مع ببساطةالتصميموقلة الحموله. TinyOS لديه تطبيق آخر لبرتوكول CoAP يدعى CoapBlip ومع ذلك، فان مكتبته لا تلبي احتياجات TinyOS. تم تقييمTinyCoAPوCoapBlipبالإضافة إلى تطبيقات HTTP باستخدام برامج المحاكاة AVRORA و TOSSIM. تمتقييم كل من التطبيقات من حيث زمن التاخير واستقلال الذاكرة واستهلاك القدرة. ولقد اظهر التقييم ان TinyCoAP لديه أفضل أداء في معظم المعاملات مقارنة مع التطبيقات الاخرى.

أظهر TinyCoAP تطورا هاما في الأداء مقارنة مع CoapBlip التي تحد من تنفيذه ديناميكية تخصيص الذاكرة RAM واستخدام مكتبة Cالخارجية.أماHTTP /TCP فلديه أسوأ أداء من التي حصل عليهاTinyCoAP.ان أداء TinyCoAP هو نفس أداء HTTP/ UDP تقريبا لكن بوثوقية عالية .

List of Contents

List of Tables

List of Figures

Abbreviations and acronyms

For the purposes of the present document, the following abbreviations and acronyms apply:

6LoWPAN	IPv6 over Low Power Networks
IoT	Internet of Things
CoAP	Constrained Application Protocol
CoAP ACK	CoAP acknowledgment
CoRE	Constrained RESTful Environments
CON	CoAP confirmable message
DTLS	Datagram Transport Layer Security
EBHTTP	Embedded Binary HTTP
ETSI	European Telecommunications Standards Institute
EXI	Efficient XML Interchange
HART	Highway Addressable Remote Transducer
HTTP	Hypertext Transfer Protocol
IEFT	Internet Engineering Task Force
IP	Internet Protocol
IPv4	Internet Protocol version 4.
IPv6	Internet Protocol version 6.
LOWPAN	Low Power Wireless Personal Area Network
LR-WPAN	Low-Rate Wireless Personal Area Network

LTP	Lean Transport Protocol
M2M	Machine-to-Machine
NON	CoAP non-confirmable message
P2P	Point to Point
QOS	Quality Of Service
REST	REpresentational State Transfer
RST	CoAP reset message
SOAP	Service Oriented Application Protocol
TCP	Transport Control Protocol
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
WG	Working Group
WoT	Web of Things
WSN	Wireless Sensor Network
XML	Extensible Markup Language

Chapter One: Introduction

1.1 Overview

The importance of industries and researchers in the application of the Internet communication model in constrained networks and devices has developed to become one of the most hopeful developments in the Internet of the future. Most of works are being achieved to enable the using of standard and well-known protocols for interaction with WSNs. In most applications using web services on the Internethas been ubiquitous, and depends on the fundamental Representational State Transfer (REST) architecture of the web [1].

In particular, a great deal of effort has been focused on the integration of Wireless Sensor Networks (WSNs) and the Internet. The main interest in making WSNs part of the Internet is to allow both to interact with each other using the existing Web technologies. From this point of view, WSNs would no longer be stand-alone networks but part of ubiquitous networks. That refers to this new approach as the Web of Things (WoT)[2].

There are hard works in the Internet of Things (IoT) to reuse Internet technologies to integrate WSNs into existing Internet infrastructure. Wireless Sensor Networks used to communicate via their own tools and technologies, often with proprietary protocols for separated applications. The necessary componentto enable efficient using of IP protocol in the constrained nodes and networks in the area of WSN isthe IPv6 over Low power Personal Area Networks (6LoWPANs), the adaptation layer[3] .

Using IP in WSNs is the first step towards the achievement of the WoT.it holds many opportunities for reasons of direct communication with WSNs. But the standard IP-Stack implementations cannot be directly adapted for the use in WSNs.In this sense, the 6LoWPANs protocol is a

IP-like but with a lighter weight which can be deployed in WSNs. 6LoWPAN enables the transmission of IPv6 packets in networks adopting the IEEE 802.15.4 standard [4]. The Internet Engineering Task Force (IETF)specific work group has detailed its definition in RFC 6282. 6LoWPAN has accelerated the integration of Wireless Sensor Networks (WSNs) and smart objects with the Internet.

In traditional Internet Web services using HTTP have demonstrated to be essential in enabling interoperable communications between computers. Although RESTful paradigm is suitable for low power embedded networks, the protocols and payload formats used to realize them are not completely usable (too much overhead of HTTP, TCP performance over lossy links, pull model inappropriate for sleeping nodes, complexity of XML)[3].

REST architectures allow IoT and Machine-to-Machine (M2M) applications to be developed on top ofshareable and reusable web services. The sensors become abstract resources identified by URIs,represented with arbitrary formats and manipulated with the same methods as HTTP[5]. As a consequence, RESTful WSNs drastically reduce the application development complexity. The functionalities and consequently of RESTful web service makes the integration of WSNs and smart objects with the Web is possible [3]. The use of Web services on top of IP based WSNs facilitates the software reusability and reduces the complexity of the application development [6].

The Internet Engineering Task Force (IETF) Constrained RESTful environment (CoRE) Working Group has done major standardization work for introducing the web service paradigm into networks of smart objects. The CoRE group has defined a REST based web transfer protocol called Constrained Application Protocol (CoAP). CoAP is designed to present simplicity, low overhead and M2M communication which is necessary to enable interaction with embedded objects in the

IoT[3].CoAP includes several HTTP functionalities re-designed for small embedded devices such as sensor nodes. In order to make the protocol suitable to IoTand M2M applications, various new functionalities have been added [7].

The CoAP protocol recent research mainly concentrate on evaluating the performance presents by the multiple features of protocol, comparing CoAP with HTTP and REST based approaches, discussing the network auto-configurationcapabilities offered by CoAP, analyzing the scalability and implementation possibilities in the Internet of Things concept. There are already number of CoAP implementations and applications for investigating and development purposes available. These are present on different platforms and languages (as CoAP standardization is still not complete, their development is also in progress): generic Libcoap for C language, TinyOSand Contiki OS implementations, jCoAP and Californium for Java, CoAPy for Python, Copper CoAP browser plug-in for Firefox or HTTP-CoAP Bridge and browser for Android[3].

1.2 Problem Statement:

The Internet protocol (IP) protocol is heavy for tiny devices, the communications between WSNs and the Internet became possible due to the standardization of IPv6 over Low-power Personal Area Networks (6LoWPANs). The definition of 6LoWPAN protocol has provided the necessary IP capabilities to WSN allowing interoperability with external IP networks. However, 6LoWPAN does not enable integration at upper layers. The commonly used HTTP fails to meet WSN requirements due to its high complexity and over-head, as a result. Therefore, this works focus in how the integration will be in upper layers.

1.3 Proposed Solution:

To solve compatibility problems, the CoAP application protocol is presented and evaluated in order to provide a solution for upper layers in WSNs, which takes into account the main characteristics of these networks. So, to make complete integration the 6LoWPAN protocol is used in lower layers and CoAP application protocol will be in upper layer. Then CoAP application protocol will be evaluated in terms of power consumption, latency, memory occupation and compared to HTTP.

1.4 Objectives

- Provide a comprehensive analysis of the functioning of CoAP including an evaluation of the reliability mechanism.
- Propose and develop a CoAP implementation for the TinyOS operating system .
- Test and evaluate CoAP and HTTP ,the evaluation is performed in terms of latency, memory occupation, response time and energy consumption

1.5 Methodology:

In the first phase of this research, describe the major functionalities of CoAP highlighting the differences with HTTP. In Second phase all features of CoAP application protocol for constrained devices has beendiscussed. It also shows how this protocol works.

In the last phase of this research, test and evaluate CoAP implementationon TinyOS operating system in term of latency, memory occupation and energy consumption and compare the result with HTTP.

1.6 Thesis outlines :

The rest of the research is organized as follows:

Chapter Two: Background and Literature review

This chapter analyses current state of art of fundamental concepts of WSNs, its application classes followedby an overview of related work**.**

Chapter Three: Constrained application protocols

This chapter gives the basic concepts and background for integrating WSN with other networks using web services .Web service integration WSNs nodes uses CoAP protocols in the application layer and UDP in the transport layer. In order to make them compatible, some kind of translation must be made**.**So this chapter presents the CoAP application protocol and explains how this protocol works and all its features. Then it shows the detailed operations and scenarios for both HTTP and CoAP.

Chapter Four: Results and Discussion

This chapter provides the implementation details of a CoAP protocol.It discusses the possible simulators to implement this technique and why the Avrora and TOSSIM simulators are selected. Then, it gives details of possible operating system to implement this protocol and why the TinyOS operating system is selected.

Chapter Five: Conclusion and Recommendation

This chapter summarizes the conclusions of the conducted research and presents directions for future work. In the appendix, details for installing TinyOS and its extensions are presented.

Chapter Two: Background and Literature Review

2.1 Background:

In some applications, using wired networks for controlling the environment is usually unpractical and costly. For this reason, it is interesting to create low-cost network architectures that give mobility to its terminals. In this sense, the deployment of Wireless Sensor Networks (WSNs) is a good solution.

A WSN consists of distributed autonomous sensors able to monitor physical or environmental conditions and to cooperatively send their data through the network to a main location .These networks are composed of hundreds of low-power and low-cost devices that are characterized by having constrained resources, limited operational capabilities and a short communication range [8]. These constraints are the critical aspects that influence the choice of a protocol stack. A widely-used protocol for the physical and link layers is IEEE 802.15.4. When a WSN uses this standard, it is called a Low Power Wireless Personal Area Network (LoWPAN).

In low layers of WSNs, IEEE 802.15.4 is being a standard. However, many problems arise when interconnecting different WSNs or sensor nodes from different manufacturers. As a result of growing interest of these networks, many solutions that restricted the possibility to interconnect and integrate various WSNs are developed. An existing and well-known protocol such as IP is using as a solution to solve this problem. But because of the highly constrained requirements, it was considered impractical

New developments show that it is possible to use efficient IPv6 communications over IEEE 802.15.4 links presenting an adaptation

layer. The out coming protocol stack Known as IPv6 over Low Power Wireless Personal Area Networks (6LoWPAN) [9].

Enabling IP on constrained devices has several advantages; WSNs can be connected to external IP networks without needing intermediate gateways. Furthermore, all knowledge from IP-based networks can be reused, avoiding the conception of new tools for managing, configuring or diagnosing these networks. IP connectivity would also allow a crucial creation in the Internet field. In this perspective, Internet would be a network with embedded objects that would be able to dealing with information and interact with their environment. This new concept is referred as the so-called Internet of Things. A key aspect to completely integrate these networks is to extend the actual Web architecture to WSNs. Furthermore, sensor nodes can be treated as any other Web resource that would be accessed using standard Web mechanisms. This new approach is known as Web of Things [10] .

The implementation of Web services in these networks must be supported on an architectural style adapted to WSNs requirements. Also, the implementation should reuse and adapt existing protocols and avoid the innovation of new ones in order to avoid interoperability problems. An IETF work group called Constrained RESTful Environments (CoRE) has been established with the goal of participating to the development and standardization of RESTful Web services for constrained networks. With this sense, the work group defined a new Web transfer protocol called Constrained Application Protocol (CoAP) [11] .

CoAP try to apply the same application transfer paradigm and basic features of HTTP to constrained networks, while keeping a simple design and low overhead. Unlike HTTP, CoAP uses UDP as transport protocol. This choice would enable CoAP to have a low impact on the limited bandwidth of the 802.15.4 wireless links. However, since UDP is

an unreliable protocol, CoAP has to implement its own mechanisms in order to guarantee reliability to those applications that use it [6].

2.1.1 Wireless Sensor Network (WSN)

Wireless sensor networks (WSN) are concentrated wireless networks of small, low-cost sensor nodes, which gather and spread out environmental data. WSNs make possible monitoring and controlling of physical environments from remote locations with better precisionthan other known monitoring systems such as remote sensing. These tiny sensor nodes leverage the idea of sensor networks based on cooperative effort of a large number of nodes [6].

Sensor networks represent an important development over conventional sensors. As they have the ability to route data back by a multi-hop infrastructure-less architecture to the base station or sink, which is the entity where information is required.

The most important constraints on sensor networks is the low power consumption requirement. Sensor nodes carry limited, generally irreplaceable, power sources. Therefore, while traditional networks want to get high quality of service (QoS) requirements, sensor network protocols focus firstly on energy conservation [6].

In addition to energy-aware techniques, WSN design often employs some approaches such as, in-network processing, multi-hop communication, and density control techniques to increase the network lifetime. Moreover, WSNs should be flexible to failures due to different reasons such as physical devastationof nodes or energy depletion. Several challenges still need to be overcome to have ubiquitous deployment of sensor networks. These challenges include dynamic topology, devices, heterogeneity, lack of quality of service, application support, manufacturing quality and ecological issues. These design

challenges make sensor networks different from other wireless ad-hoc or mesh networks. Therefore, the protocols and algorithms have been proposed for traditional wireless ad hoc networks are not well suited for WSN [12].

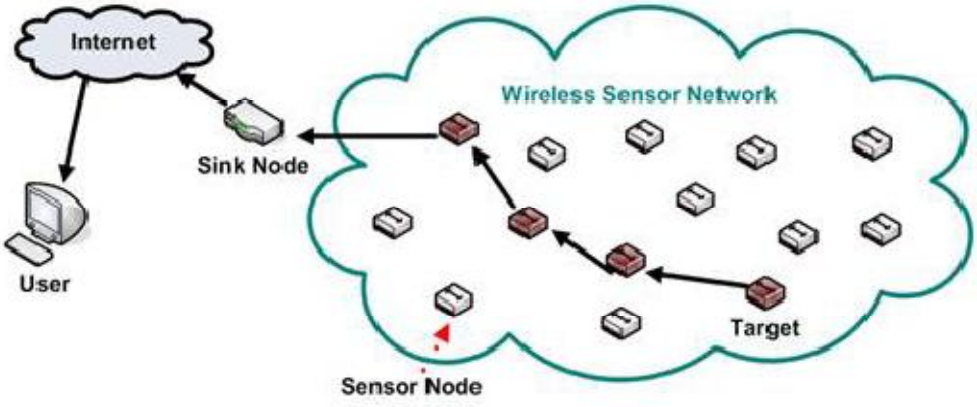

Figure 2.1: Example of a Wireless Sensor Network (WSN).

2.1.2 WSN Protocols:

The sensor nodes can communicate through the wireless medium but protocols and algorithms offered for traditional wireless ad hoc networks may not be well fitting for sensor networks. Sensor networks are application particular, and the sensor nodes work cooperatively together. In addition, the sensor nodes are energy constrained compared to traditional wireless ad hoc devices. Thus, the differences between sensor networks and ad hoc networks should be Knownto provide a general thought how the WSN protocols will be. The differences between both networks [13] can be summarized in the following main points:

•The number of sensor nodes in a sensor network can be several orders of magnitude higher than the nodes in an ad hoc network.

•Sensor nodes are densely deployed.

•Sensor nodes are prone to failures.

•The topology of a sensor network changes very frequently due to failure and duty cycles of nodes.

•sensor nodes mainly use a broadcast communication paradigm whereas most ad hoc networks are based on point-to-point communications.

•Sensor nodes are limited in power, computational capacities, and memory.

•Sensor nodes may not have global identification (ID) because of the large amount of overhead and large number of sensor nodes.

•Sensor networks are deployed with a specific sensing application in mind; ad hoc networks are mostly constructed for communication purposes.

A sensor network does not work in separation in functional deployment. For many considerable applications, however, it is necessary to integrate these sensor networks to the presented Internet Protocol (IP) networks. The protocol stack used by sensor nodes is given in Figure 2.2 This protocol stack combines power and routing realization, integrates data with networking protocols, connects power-efficiently using the wireless medium, and supports collaborative efforts of sensor nodes. The protocol stack consists of the application layer, transport layer, network layer, data link layer, physical layer, power management plane, mobility management plane, and task management plane [6].

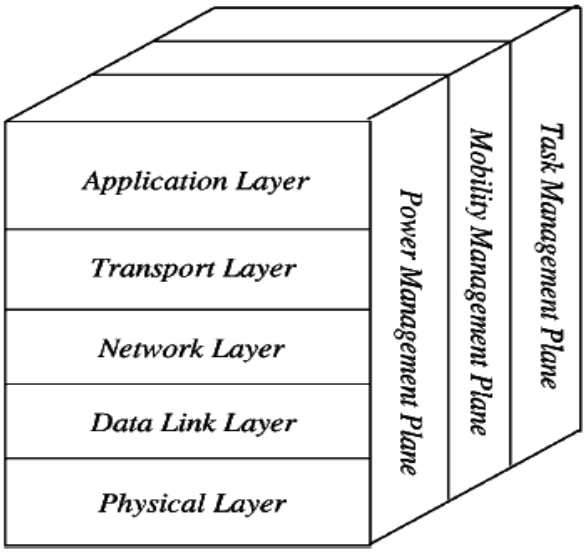

Figure 2.2: The wireless sensor network protocol stack

Application layer:

Various types of application software can be built and used on the application layer, depending on the sensing tasks. Sensor nodes can be used for continuous sensing, event detection, event identification and location sensing. The aim of micro-sensing and wireless communication of these nodes pledge many new application areas. This results in a wide range of application layer protocols.

Transport layer:

It helps to take care of the flow of data if the sensor networks application needs it. In common, the most important objectives of the transport layer are:

- To bridge application and network layers by application multiplexing and de-multiplexing.
- To provide data delivery service between the source and the sink with an error control mechanism.
- To regulate the amount of traffic injected into the network via flow and congestion control mechanisms.

On the other hand, the required transport layer functionalities to complete these objectives in the sensor networks are due to significant modifications in order to fitwith unique characteristics of the sensor network paradigm. For example, classic end-to-end, retransmission-based error control mechanisms adopted by transport control protocol (TCP) may not be practical for the sensor network area and for that reason may lead to loss of limited resources. On the other hand, the particularobjective of the sensor network also influences the design requirements of the transport layer protocols. For example, the sensor networks deployed for different applications may require different reliability levels as well as different congestion control approaches. As a result, improvement transport layer protocols is a challenge because the restrictions of the sensor nodes and the particular application requirements mostlydecide design principles of transport layer protocols [14].

Network layer:

The main function of this layer is routing the data given by the transport layer. Sensor nodes may be spreaddensely in an area to monitor a phenomenon. Therefore, they may be very close to each other. In such a situation, multi-hop communication may be a good selection for sensor networks with strict requirements on power consumption and transmission power levels. As the sensor nodes missing not much energy when transmitting a message because the distances between sensor nodes are shorter. As mentioned before, ad hoc routing techniques already suggested in the literature do not usually suitablerequirements of the sensor networks. Therefore, the network layer of the sensor networks is usually designed according to the following standards:

• Energy efficiency is always an important consideration.

• Sensor networks are mostly data centric.

• An ideal sensor network has attribute-based addressing and location awareness.

•Data aggregation is useful only when it does not hinder the collaborative effort of the sensor nodes.

•The routing protocol is easily integrated with other networks, e.g., Internet.

One of the design principles for the network layer is to allow easy integration with other networks such as the satellite network and the Internet. As shown in Figure 2.1, the sinks are the basis of a communication backbone that serves as a gateway to other networks. The users may query the sensor networks through the Internet or the satellite network, depending on the purpose of the query or the type of application the users are running.

Data link layer:

It is mainly responsible for multiplexing data streams, data frame detection, medium access, and error control; it make sure a reliable point-to-point and point-to-multipoint connections in a communication network. However, the collaborative and application-oriented nature of the sensor networks and the physical constraints of the sensor nodes, such as energy and processing limitations, decide the way in which these responsibilities are achieved[14].

Physical layer:

It is regularlyresponsible for modulation and demodulation of digital data; this work is executed by transceivers. In sensor networks, the challenge is to find modulation schemes and transceiver architectures that are easy, low cost, but still strong enough to introduce the required service.

2.2 Literature review:

A various studies have discussed the application of CoAP in WSNs. In this section we give details about these works.

Previous work by the authors of [14] offered a CoAP implementation for Contiki. The aim of this implementation was to obtain high-energy efficiency by leveraging a radio duty cycling mechanism. The implementation has been tested in a multi-hop network. The obtained results have shown that when using a radio duty cycle, energy consumption is lower but the latency performance is getting worse.

The authors [6] motivated the choice of the REST architecture and the taking up of the CoAP protocol. We also suggest modifications to an early format of the protocol and seek possible problems of its implementation.

In [5], the authors statements a simple comparison of CoAP and HTTP in terms of energy consumption. This work also presented the design of a gateway used to communicate a CoAP based WSN to an external IP network that uses HTTP. In [15], the authors of [5] compared the performance of CoAP to that of HTTP. The evaluation was done on the basis of energy consumption and response time. In particular, energy consumption was evaluated by means of simulation. The response time was measured in a real WSN. Both experiments were conducted considering a client querying an embedded server to obtain temperature and humidity values. The energy consumed was measured according to the variance of the inter-arrival packet time. The response time was calculated for the case where the server was at a distance of 1-hop and 2-hop from the client. The results show that CoAP returns a better performance in both the evaluation parameters considered by the authors.

A study used CoAP and HTTP on network sensor deployment [16] as data transport protocol for sensor network reprogramming. The Results were gained from measuring both protocols over a duty cycled radio layer through simulation view that CoAP and HTTP present similar results. In [17], the authors present a framework for M2M communications using CoAP. They also present an improved publish/subscribe mechanism also based in CoAP. Both solutions are evaluated showing the advantage of using CoAP instead of HTTP.

The authors of [18] give an overview of the current CoAP implementations and present the results of compatibility meeting organized by the European Telecommunications Standards Institute (ETSI). In [19], the authors present a CoAP implementation for TinyOS and the implementation of a compression mechanism of the XML format. A performance evaluation was carried out considering the CoAP request success probability as a function of the request rate of the client node. Furthermore, the authors report results from an evaluation of the memory occupation of the TinyOS components used in their implementation. Finally, the authors proved the ability of the XML compression scheme by studying its processing-time. As previously commented, this CoAP implementation was developed on top of an unsupported and limited 6LoWPAN implementation named 6lowpancli [20]. In particular, as pointed out in [21],[22]6lowpancli provides only basic work of 6LoWPAN. 6Lowpancli doesn't support any type of neighbor discovery mechanism, it is completely static and requires manual configuration. As reported in [21],[22] the support for mesh network is not provided and when a packet with different destination address is received, it is just dropped. The results of a performance evaluation done in [21] show that 6lowpancli does not perform well in

terms of energy consumption and latency. Thereby, its limitation would affect any implementation build on top of it.

As previously mentioned, CoapBlip is currently included in the latest distribution of TinyOS. The authors of CoapBlip present its design in [23]. They evaluated their implementation and compared it to HTTP. The performance evaluation considers the ROM occupation and the average response time of CoAP and HTTP. The results of an evaluation show that CoAP shows better performance than HTTP. In [24], CoapBlip has been used to evaluate the CoAP protocol in integrations with other low layer protocols. In this sense, it has been evaluated along with the Routing Protocol for Low-power and Lossy Networks (RPL) and the Low Power Listening (LPL) protocol.

Chapter Three: Constrained Application protocol

3.1Constrained RESTful Environments (CoRE)

IETF formed this WG with the main objective of presenting the Constrained Application Protocol (CoAP), a RESTful protocol appropriate for constrained environments. The Representational State Transfer (REST) paradigm donates to designing APIs so that every data exchange can be done with the GET, POST, DELETE and UPDATE operations of the HTTP protocol [25].

The work of the CoRE WG has been chartered because of introducing a web-oriented binary protocol, unsophisticated enough to be handled by severely restricteddevices, yet simple to map onto HTTP. The reason behind this approach is driven by the Widespread of HTTP in the Web, allowing HTTP connection over constrained environments will further expand its applicability and become ubiquitous. The recently proposed protocol is attemptingtoobtain this objective defining a binary representation of REST, which contain the most important and useful features of HTTP [26].

Next-generation M2M environments are estimated to be the destroyer application for this protocol: for instance, a lot of consideration has been devoted to the design of publish/subscribe mechanisms, since this approach is considered to be key for connecting constrained devices and evading network congestion. As many spread content-generating networks, Smart Grids would experience different benefits from a web-like communication model: in fact, web services are well-known in the traditional Internet for their applicability to al-most every kind of application. Following this guideline, the WG is driving the Constrained

Application Protocol (CoAP) to be employed for M2M communication, resulting in a web-compatible standard for M2M applicability [19].
By designCoAP is directly mappable to the current HTTP realization: by forcing its intrinsic Compatibility, the SG system design can be heavily simplified, by directly allowing each network device to deal with standard Internet languages and, at the same time, keeping the energy and traffic load on the constrained environment low.

3.2 Application Protocols and Formats:

Constrained Application Protocol (CoAP) [10] is at this time being standard within the CoRE working group of the IETF, which is introducing a REST–based framework for resource–oriented applications optimized for constrained IP networks and devices. by enabling this protocol set, restricted packet sizes, low-energy devices and unreliable channels are simple to be manage[26].
CoAPis based on the REST architectural style participating the objectives and the intrinsic limitation listed above. It is designed for simple stateless mapping with HTTP, and for providing M2M interaction. HTTP compatibility is obtained by maintaining the same interaction model, using a subset of the HTTP methods. Nodes supporting CoAPoffering flexible services over any IP network using UDP, and they also a strong communication framework to communicate sensor nodes to the Internet. Any HTTP client or server can deal with CoAP-Ready endpoints by easy installing a translation proxy between the two devices. This will not be a load for the proxy, since these translation processes have been designed not to be time and computationallyrequirements. Also, CoAPmake as a message layer between the application protocol and UDP.

3.3 Constrained Application Protocol (CoAP)

Although HTTP is widely used with Web Services, it is by no means the only protocol for M2M communication. The Internet Engineering Task Force (IETF) Constrained RESTful Environments (CoRE) [27] working group published the first draft of a RESTful web transfer protocol called Constrained Application Protocol (CoAP) [28]. CoAP includes several HTTP functionalities which have been re-designed for M2M applications over constrained environments on the IoT, meaning it takes into account the low processing power and constraints of small embedded devices, such as sensors.

In addition, CoAP provide a number of characteristic that HTTP lacks, such as built-in resource discovery, IP multicast support, and asynchronous message exchange. There are many implementations of CoAP in various languages, such as libcoap1 (an open source C-implementation) and Sensinode'sNanoService.The summary of the main features addressed by CoAP are [28]:

• Constrained web protocol fulfilling M2M requirements.

• UDP binding with optional reliability, supporting unicast and multicast requests.

• Asynchronous message exchanges.

•Native push model

• Small header overhead and parsing complexity.

• URI and Content-type support.

• Simple proxy and caching capabilities.

• Ability to operate with cyclic sleeping nodes, asynchronous message exchanges [29].

• 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).

3.3.1 CoAP Structure Model

The interaction model of CoAP in figure 3.1is 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.

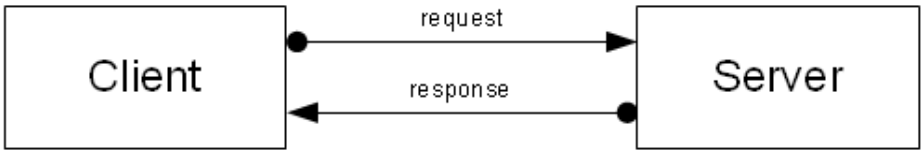

Figure 3.1: The CoAP Interaction model

A CoAP request is comparable to that of HTTP, and is sent by a client to request an action (using a method code) on a resource (known by a URI) on a server. The server then sends a response with a response code; this response may contain a resource representation.

Unlike HTTP, CoAP deals with these interchanges asynchronously over a datagram-oriented transport such as UDP.

This is done logically using a layer of messages that supports optional reliability (with exponential back-off). CoAP is organized in two layers as shown in figure 3.2 ,the transaction layer handles asynchronous nature of a single message exchange between two points and used to deal with UDP. The Request/Response layer is responsible for the requests/response transmission using Method and Response codes and for the resource manipulation. CoAP is however a single protocol, with messaging and request/response just features of the CoAPheader.

The dual layer approach allow CoAP to provide reliability mechanisms even without TCP as transport protocol [30],[7].CoAP defines four types of messages: Confirmable, Non-confirmable, Acknowledgement, Reset; method codes and response codes included in some of these messages

make them carry requests or responses[11]. The basic exchanges of the four types of messages are somewhat orthogonal to the request/response interactions; 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.

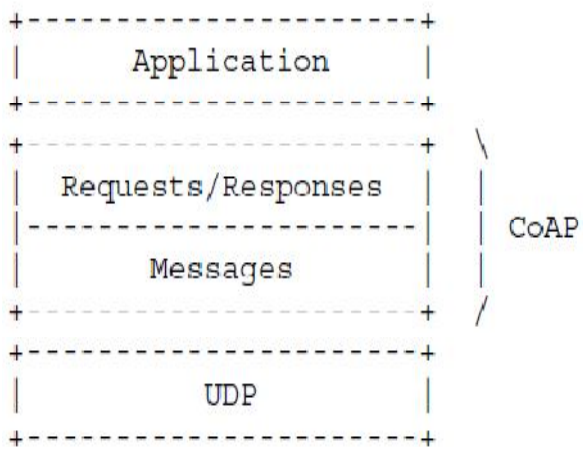

Figure 3.2: Abstract Layering of CoAP

3.3.1.1. REQUEST/RESPONSE LAYER MODEL

The CoAP client/server interaction model, depicted in Figure 3.1, assesses that CoAP requests are sent by clients in order to request an action on a resource of the server. After the request elaboration, the server sends back a CoAP response containing an appropriate response code and optionally a resource representation.

After receiving a request, a server responds with a CoAP response. There are three types of responses:

• **Piggy-backed**: The response is carried directly in the acknowledgment message. The response is returned in the acknowledgment message independently of whether the response indicates success or failure as in figure 3.3.

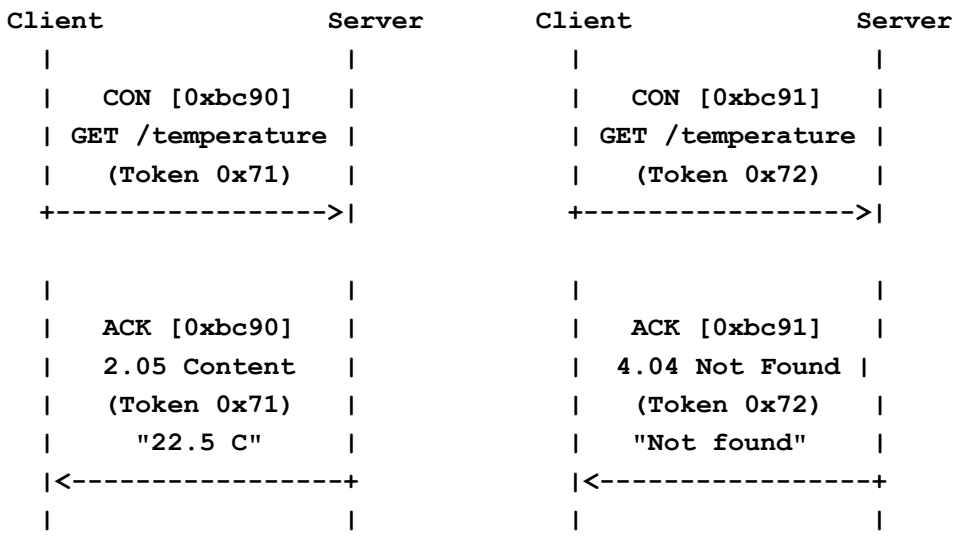

Figure 3.3: The successful and failure response results of GET method

• Separate: In some cases, it may not be possible to return a response immediately. In order to avoid packet retransmission, the server sends an ACK to promise the client it will process the request. When the server finally processes it, then a CON message is sent as in figure 3.4.

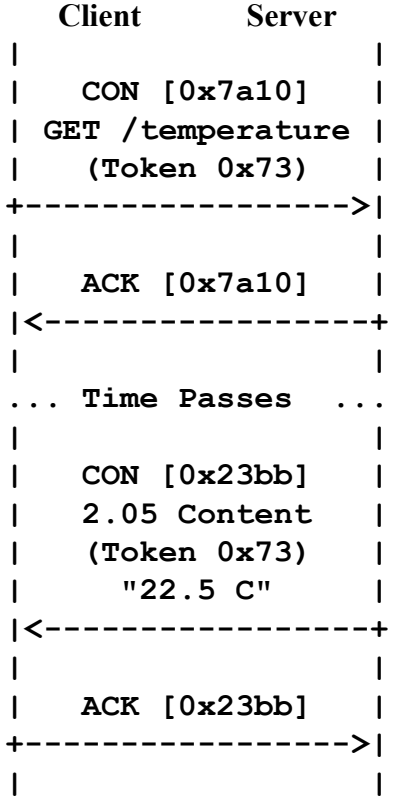

Figure 3.4: GET request with a separate response

• **Non-confirmable**: If the request is not confirmable, then the response is also not confirmable as shown in fig 3.5. A response is identified by the Code field in the CoAP message header. There are three code classes:

• Success (2.x). The request was successfully received, understood, and accepted

• Client Error (4.x). The request has bad syntax or cannot be fulfilled.

• Server Error (5.x). The server failed to fulfill an apparently valid request.

Response codes are designed to be extensible. If one of them is not recognized, then it must be treated as a being equivalent to the generic Response Code of that class.

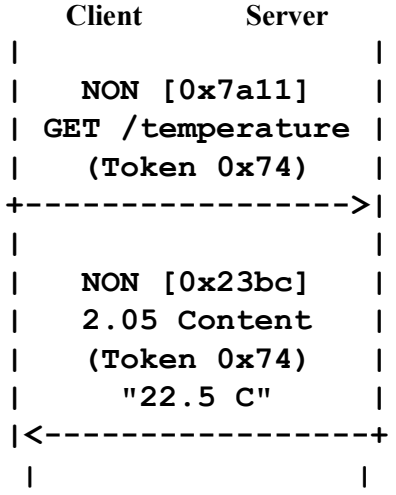

Figure 3.5: A Request and a Response Carried in Non-confirmableMessages

CoAP Methods:

The client request contains a method that specifies the action requested, an unique identifier of the server resource called Uniform Resource Identifier (URI) and optionally a payload containing meta-data about the request. The CoAP standard defines four different methods:

• **GET**: retrieves an information representation of the resource.

• **POST**: carries an information representation and asks the receiver to process it. The output depends on the target resource, usually involving resource creation or update.

• **PUT**: requests an update operation of the resource identified by the request URI with the carried information representation.

• **DELETE**: causes the deletion of the resource identified by the request URI.

Upon reception of the request, the server elaborates it and, if no errors occur, sends back to the client its response containing a response code that indicates the result of the request process. Response codes are divided into three classes 2.xx (Success),4.xx (Client Error) and 5.xx (Server Error) as it had mentioned before.

The fraction of the response code just denoted with xx does not have any categorization role: it gives instead additional details of the output of the request process. For example, the most common HTTP response code is the 404 or not found error, which indicates that the client request was correct but the server was not able to find the resource pointed by the URI field. The matching between requests and responses is achieved by means of a token, that is an unique identifier of any request/response couple between two specific endpoints. This field is included on every CoAP request as well as in every CoAP response.

3.3.1.2. MESSAGE LAYER MODEL

As CoAP is bound to the non-reliable protocol UDP, it implements a lightweight reliability mechanism trying to recreate TCP. The main characteristics are:

• Simple stop-and-wait retransmission reliability with exponential back-off.

• Duplicate message detection.

• Multicast support.

CoAP defines four types of messages: Confirmable, Non-Confirmable, Acknowledgement, Reset. The exchange of messages is orthogonal to the request/response interactions [31]. Requests can be carried both in Confirmable and Non-Confirmable messages. Responses can be carried equally in Confirmable and Non-Confirmable messages, but also piggy-backed in Acknowledgement messages, CoAP type messages are:

• **Confirmable (CON):** This message is sent when a reliable transmission is needed. The protocol guarantees that the message will not be lost within certain conditions. Because messages are transported over UDP, the reliability is accomplished with packet retransmission if a response is not received in a given time out[11]. It increases exponentially with every new retransmission and, thus, provides a simple congestion mechanism. The packet will be lost if the maximum number of retransmissions is reached.

• **NON-Confirmable (NON**): This message is sent if a reliable transmission is not needed. It is useful for requests that are sent regularly. This message may carry a response for a NON request.

• **Acknowledge (ACK):**This message carries a response to acknowledge a CON request. This type of messages may carry response data or not. In the first case, the response is called piggy- backed response and in the second case separate response. The second one is used when the server cannot process the request immediately but promises that it will be processed.

• **Reset (RST):** This message indicates that a CON messages has arrived but there is no context to process it.

CoAP message reliability

A reliable transmission is started marking a packet as confirmable.A recipient must acknowledge such message with an acknowledge message or reject it with a reset message. The sender transmits the CON message at exponential increasing intervals until receives an ACK, RESET or it runs out of attempts. For each time out expired, the time out is doubled, as shown in figure 3.6 [12] .

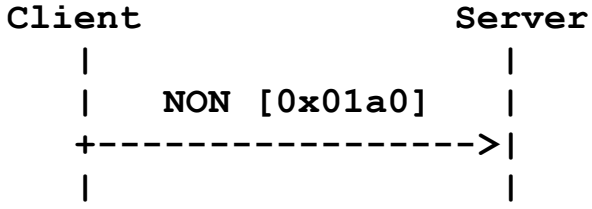

Figure 3.6: CoAP reliable message transmission

The recipient should acknowledge each duplicate copy of the CON message using the same ACK but it should process any request or response only once. It should ignore any duplicates and process the message only once.

Figure 3.7shows an example of unreliable CoAP transmission message. A message is not acknowledged or rejected. If recipient lacks the context to process the message, the message must be simply ignored. The recipient must be prepared to receive the same message multiple times.

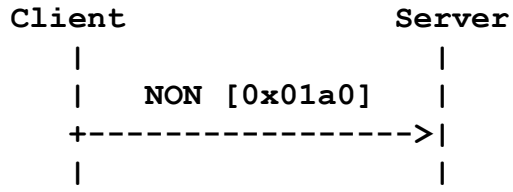

Figure 3.7: Unreliable message Transport

3.3.1.3. CoAP Message Format:

The figure 3.8 shows how a CoAP message. It has three different parts which are transported over an UDP packet:

• **CoAP header**: Provides basic information to recognize the CoAP version, the type of message, a message code and a message identifier. It also provides information to parse the message.

• **CoAP options**: Are used to provide parameters needed to fulfill requests.

• **CoAPpayload** : Contains the message body.

The CoAP header has the following fields:

• **Version (Ver):** Indicates the CoAP version number. Implementations of this specification MUST set this field to 1.

• **Type (T).** Indicates the message type: CON, NON, ACK or RST.

• **Option Count (OC):** Indicates the number of options after the header. If OC set to 0, there are no options and the payload (if any) immediately follows the header.

• **Code:** Indicates if the message carries a request (code values from 1 to 31) or a response (code values from 64 to 191), or is empty (0). (All other code values are reserved.) In case of a request, the Code field indicates the Request Method; in case of a response a Response Code.

• **Message ID**: Used for the detection of message duplication, and to match messages of type ACK/RST and messages of type CON.

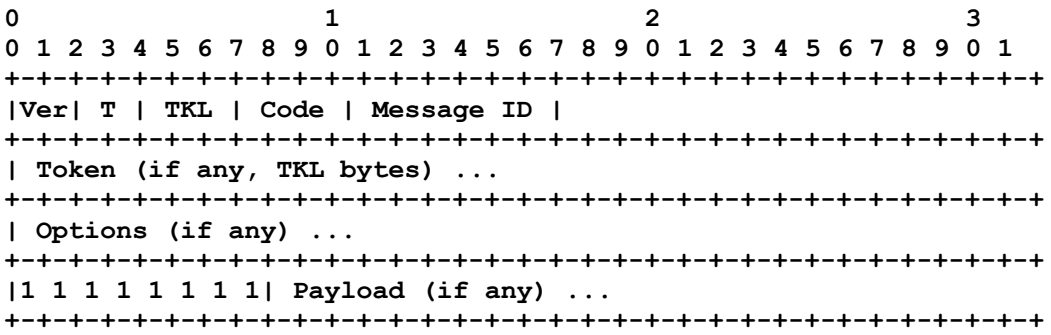

Figure 3.8:CoAP Message Format

3.3.1.4. Options:

Options are identified by an option number. Odd numbers indicate critical options and even numbers elective options. Figure 3.9 shows the option format. Options fields are:

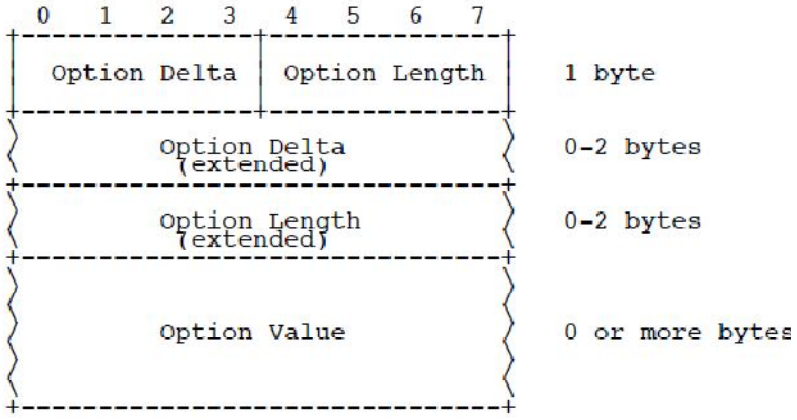

Figure 3.9: Option format fields in CoAP message format.

• **Option Delta**: 4-bit unsigned integer. It indicates the difference between the option Number of the current option and the option number of the previous option.

• **Length**: 4-bit unsigned integer. It indicates the length of the option Value. When this field is set to 15 an 8-bit unsigned integer is added allowing lengths ranging from 15 to 270 bytes. Options can be critical or elective. The difference is how an unrecognized option is handled in an end-point:

• **Elective**: Must ignore messages with unrecognized options.

• **Critical**: That occur in a CON message request must cause the return of 4.02 response code.

• **Critical**: That occur in a CON message response and in a NON message must silently ignore the message. There are several types of options:

• **Token**: It is used to match a response with a request. Every request has a client-generated token which the server must echo in any response.

• **Uri-Host**: It specifies the Internet host of the resource being requested. The default value is the IP literal representing the destination IP address.

• **Uri-Port**: It specifies the port number of the resource. The default value is the destination port.

• **Uri-Path**: It specifies one segment of the absolute path to the resource.

• **Uri-Query**: It specifies a query string.

• **Proxy-Uri**: It is used to make a request to a proxy. The proxy is requested to forward the request or service it from a valid cache and return the response.

• **Content-Type**: It indicates the representation format of the message payload given as a numeric value.

• **Accept**: It indicates when included one or more times in a request, one or more media types, each of which is an acceptable media type for the client, in the order of preference.

• **Max-Age**: The maximum time a response may be cached before it must be considered not fresh. When included in a request, it indicates the minimum value for the maximum age of cache response the client will accept.

• **E-Tag**: In a response, provides the current value of the entity-tag for the enclosed representation of the target resource. An entity-tag is intended for use as a resource-local identifier for differentiating between representations of the same resource that vary over time.

• **Location-Path and Location-Query**: It indicates the location of a resource as an absolute path URI. It can be included in a response to indicate the location of a new resource created with POST.

• **If-Match**: It may be used to make a request conditional on the current existence or value of an ETag for one or more representations of the target resource.

• **If-None-Match**: It may be used to make a request conditional on the non-existence of the target resource. If-None-Match is useful for resource creation requests, such as PUT requests, as a means for protecting against accidental overwrites when multiple clients are reacting in parallel on the same resource.

Table 3.1 :CoAP message Options

No.	C	U	N	R	Name	Format	Length	Default
1	x			x	If-Match	opaque	0-8	(none)
3	x	x	-		Uri-Host	string	1-255	(see below)
4				x	ETag	opaque	1-8	(none)
5	x				If-None-Match	empty	0	(none)
7	x	x	-		Uri-Port	uint	0-2	(see below)
8				x	Location-Path	string	0-255	(none)
11	x	x	-	x	Uri-Path	string	0-255	(none)
12					Content-Format	uint	0-2	(none)
14		x	-		Max-Age	uint	0-4	60
15	x	x	-	x	Uri-Query	string	0-255	(none)
16					Accept	uint	0-2	(none)
20				x	Location-Query	string	0-255	(none)
35	x	x	-		Proxy-Uri	string	1-1034	(none)
39	x	x	-		Proxy-Scheme	string	1-255	(none)

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

3.4 CoAP URIs Scheme

CoAP uses the “coap” and “coaps” URI schemes (compared to the “http” and “https” URI schemes respectively) for identifying CoAP resources and to provide a means of locating the resources. The URI support in a CoAP server is simplified as the client already parses the URI and splits it into host, port, path and query options (uri-host, uri-port, uri-path, uri-query), making use of default values for efficiency. The options encode the different components of the request URI in a way that no percent-

encoding is visible in the option values and that the full URI can be reconstructed in any involved endpoint [3].Here's an example of a CoAPURI:

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.

3.5 Caching

Nodes can cache their responses in order to reduce the response time and network bandwidth consumption on future. Unlike HTTP [21], caching of the CoAP responses does not depend on the request method, but on the particular response code [31] equivalent requests. The goal of caching is to reuse a prior response message to satisfy a current request. Table 3.1 shows which response codes can be cached and the relation between caching and the response codes. A node must not use a stored response unless:

• The request method and the one used to obtain the stored response must match.

• All options match between those in the presented request and those of the request used to obtain the stored response.

• The stored response is either fresh or successfully validated. There are two ways to decide if a cache can be used to satisfy a request:

•**Freshness model:** The mechanism for determining freshness is for an origin server to provide an explicit time in the future using Max-Age. In this way, If an origin server wants to prevent caching it must explicitly include a Max-Age option with a value of zero seconds. If the client has

certain influence in the freshness calculation it can include a Max-Age option in a request [32].

• **Validation model:** When an end-point has one or more stored responses for a GET request but it cannot use any of them, it can use the E-tag option in the GET request to give the origin server an opportunity to both select a stored response to be used and to update its freshness. Each stored response has an entity-tag that should be sent to the server via an E-tag option. The server response 2.03 (Valid) indicates that the stored response identified by its E-tag option can be reused. For any other response, it should be used to satisfy the request.

Table 3.2: *Relation between CoAP response codes and caching.*

Code	Caching	
2.01	No	Mark any stored response for the created resource (location options) as not fresh.
2.02	No	Mark any stored response for the deleted resource as not fresh.
2.03	/	Update the stored response with the value of the Max-Age Option.
2.04	No	Mark any stored response for the changed resource as not fresh.
2.05	Yes	Use Max-Age for freshness model and ETag for validation model.

3.6 Implementation:

TinyOS is an OS for WSNs designed to meet the requirements of constrained networks and devices. It is composed by a set of reusable components that can be used to build specific applications. TinyOS is implemented in the NesC language [31]. NesC is a C dialect designed to improve code efficiency and robustness in embedded software applications [33]. Through its simplicity, NesC is able to reduce RAM occupation, code size, and prevents low-level bugs. The programming model of TinyOSis also based on this language. Besides NesC, TinyOS allows using more complex languages such as Java, Python or C. In particular, C code can be embedded in NesC programs or can be used to build libraries for TinyOS. As we will explain later in this section, a

TinyOS based WSN can achieve better performance and be more reliable when using exclusively NesC. The design philosophy of TinyCoAP follows the principles of the TinyOS programming model. The code is structured in TinyOS components and the use of external libraries is avoided. TinyCoAPis completely written in NesC. The rest of this section focuses on the memory allocation system, library and the data structures of TinyCoAP.

3.6.1 Structure of the Library

TinyCoAPis designed behind the aim that better performance and reliable run-time execution are both executed integrating it with the OS core libraries. It presents a CoAP library native for TinyOS. Using these design characteristics, the core functionalities of CoAPare offered as TinyOS components. These components are improved as part of the TinyOS network library.Not like TinyCoAP, CoapBlipis thought as an adaptation of a C library for generic embedded systems. A TinyOS component is employed as an adapter between this library and the TinyOS application. TinyCoAP bases completely on code developed in the NesC language and evade using external C libraries. This allows obtaining a high code optimization and having less effect on the WSN node memory. These benefits derive mainly from the different organization and functioning of C and NesC programs. Typical C programs are composed by functions that are specified in separated files. These are compiled separately and then linked together by matching global name of functions. The interaction between them is achieved dynamically during run-time by using function pointers. Pointers are stored in the RAM memory and therefore cause a growth of its occupancy. In contrast with C, TinyOS programs are considered as a set of components connected together to carry out a specific task. These

interact between each other using the interfaces that they offer. Applications occur at compile-time which components they use and then, they apparently wire the interfaces they will use at run-time. Thanks to this static wiring, TinyOS programs keeps away from using function pointers and therefore they are capable to decrease the RAM memory footprint. The TinyCoAP library is consistsof five components. Its design follows the CoAP principals layering. The message layer is implemented by three components. CoapPDU, where PDU set for Protocol Data Unit, is the important component of this sub-layer. It introduces the interface used to create, read and write CoAP packets. The interface require to create or delete options is offered by the CoapOption component. The creation, use and organization of the linked lists areachieved by the interface offered by the CoapList component. Linked lists are useful for repeating the packets that are in the memory pool waiting for being processed. CoapList is also used to laying up and repeat the options that contain a packet and to control retransmissions. CoapPDUis wired to CoapList and CoapOption. This enables CoapPDU to deal with the options composed in a CoAP packet. additionally, each element of the message layer is wired to the TinyOSPoolC component. This is used to assign the memory required to complete their operation. PoolC specifies memory corresponding to the data structure that is identified by each component. The wiring of the message layer components. The request/response matching layer of CoAPis carried out by the CoapServer and CoapClient components. CoapClient introduces the interface used to send CoAP requests. The interface introduced by CoapServer enables initializing and connecting the server to a particular UDP port. The retransmission technique and the CoAP packet development are also achieved by these components. CoapServer executed the discovery of CoAP resources [34] and the observe option of

CoAP. The management of the resources presented by the server is performed in anindividual interface. The resources are generated through a parameterized interface. This is calledCoapResource and gives commands and events to keep resources and the separate response mechanism of CoAP.

3.6.2 RAM Memory Allocation

The most significant concept to take into account when embedding software applications in WSN nodes is managing the allocation of RAM memory. The management of memory allocation has to handle with the limited size of RAM memory and the short of hardware memory protection that describe constrained nodes. From that point of view, managing the RAM memory dynamically could raise the chanceof having failure nodes or could consume the existing memory. In fact, the shortage of hardware memory protection does not avoid the risks of containing a collision between the heap and stack or a memory leak [35]. Moreover, the size of the allocated RAM memory would be complicated to mange with this allocation system. TinyCoAP evades these threats by allocating RAM memory statically. The size of the allocated memory is known at execute time and the possibility of memory exhaustion is therefore evaded. in addition, static allocation would reduced the risks of failures consequentto collision of the heap and the stack. for that reason it would improve the network reliability. A more enhancement is obtained enabling TinyCoAP to make CoAP responses without allocating new memory. TinyCoAP creates responses using the memory already specified to store the related CoAP requests. Furthermore, the decreasing of the RAM memory footprint this allowing a lighter packet processing with less influence on the CPU. As a result, the decreasing of the CPU use would minimize the energy consumption. As reported in

[35], the CPU expend 4.6 mA when active and 2.4 mA when idle while the radio consume 3.9 mA when receiving. Therefore, the TinyCoAP management of buffers would save CPU cycles and improve the battery life of nodes. The static allocation of memory made by TinyCoAP is compliant with the RAM memory management specified in NesC. Actually, NesC does not maintain dynamic memory allocation. This propertiesenables avoiding memory fragmentation and run-time allocation failures [36]. on the other hand, a position may arise in which applications might require dynamic allocation. To avoiding this problem, TinyOS introduces a component calledPoolC that reproduce the dynamic memory allocation. Should PoolC be enabled, the most pool memory size would be specified statically at compile time. Through the execution time, the applications will get the amount of RAM memory they require from that presented in the pool. An concluding memory leak would make the pool to empty, but the heap and stack would not collide. As mentioned above, TinyCoAP uses PoolC to assign the buffers required to keep the CoAP packets and the linked lists. In a different way from TinyCoAP, CoapBlip implements a dynamic memory allocation management. It uses the malloc memory management library to allocate memory for buffers and linked lists.

3.6.3 Data Structure

As declared above, TinyCoAP components are structured following the conceptual layering of CoAP. The message layer is being on top of Blip. CoapBlip also implements this 6LoWPAN stack. Should Blip receive a UDP packet, it verifies the existence of the CoAP header. If it is exist, the interface introduced by CoapPDU keeps it in a CoAP PDU. This PDU is stored in the memory already allocated through PoolC. The use of PoolC enables TinyCoAP to begin at compile time the maximum size

a packet can get and the maximum number of packets it can handle. The maximum length of options and the maximum number of packets that can be queued by a node can also be specified. These characteristics make TinyCoAP powerful against possible memory leaks and always present it with room in the memory for the received packets. Moreover, TinyCoAP is easily flexible to different applications. The TinyCoAP PDU data structure is designed to be used with PoolC. It prevents the use of pointers for reaching to the various components of the PDU. Table 4.1explain the CoAP PDU used in CoapBlip and TinyCoAP. In TinyCoAP, the received CoAP message is firstlykept in the UDP buffer as an invalid element. This element is then transformed into a coap_pdu_t structure and stored in the memory pool. Once the PDU structure has been made, the UDP buffer is ready to accept a new incoming packet. In TinyCoAP the maximum payload allowed for requests and responses can be defined at compile time. Thus, the memory usage can be accommodated to the application requirements and to the features of the sensor. CoapBlip uses pointers to reach to various parts of the PDU. Should a CoAP packet be received, CoapBlip stores it in a buffer allocated through malloc and initializes the pointers defined in coap_pdu_t. This buffer is placed at UDP level and its size is always equivalent to the maximum packet size enabled by CoapBlip. Therefore, although CoapBlip uses malloc, the memory is always allocated with the same size.

Table 3.3: CoAP PDU structures

CoapBlip	TinyCoAP
typedef struct {	typedef struct {
coap_hdr_t *hdr;	uint8_t timestamp;
unsigned short length;	coap_hdr_t hdr;
coap_list_t *options;	struct sockaddr_in6 addr;
unsigned char *data;	uint8_t payload [MAX];
} coap_pdu_t;	uint16_t payload_len;
	coap_list_t opt_list;
	} coap_pdu_t;

CoapBlip stores the PDU in the UDP buffer and uses a pointer to provide access. TinyCoAP saves it in the memory allocated with PoolC.

3.6.4 Tools:

A software solution to integrate RESTful Web services in WSNs based on the CoAP protocol is presented. This software is a library for the TinyOS operating system that has been developed in order to easily create new applications that can use and other Web-based services using the CoAP protocol. The Figure below is architecture of a CoAP-based Wireless Sensor Network (WSN).

TinyOS is an "operating system" designed for low-power wireless embedded systems. Fundamentally, it is a work scheduler and a collection of drivers for microcontrollers and other ICs commonly used in wireless embedded platforms. TinyOS[33] is an embedded OS for WSNs designed to meet the requirements of constrained networks and devices. It is composed by a set of reusable components that can be used to build specific applications. TinyOS is implemented in the NesC language [31]. NesC is a C dialect designed to improve code efficiency and robustness in embedded software applications [33].

Through its simplicity, NesC is able to reduce RAM occupation, code size, and prevents low-level bugs. The programming model of TinyOSis

also based on this language. Besides NesC, TinyOS allows using more complex languages such as Java, Python or C. In particular, C code can be embedded in NesC programs or it can be used to build libraries that TinyOS components can use. TinyOS based WSN can achieve better performance and be more reliable when using exclusively NesC.

In this work TOSSIM and Avrora simulations will be used. TOSSIM is the TinyOS mote simulator which has been developed, to ease the development of sensor network applications. TOSSIM scales to thousands of nodes, and compiles directly from TinyOS code; developers can test not only their algorithms, but also their implementations. TOSSIM simulates the TinyOS network stack at the bit level, allowing experimentation with low-level protocols in addition to top-level application systems. Users can connect to TOSSIM and interact with it using the same tools as one would for a real-world networking, making the transition between the two easy. TOSSIM also has a GUI tool, TinyViz, which can visualize and interact with running simulations. Using an simple plug-in model, users can develop new visualizations and interfaces for TinyViz.

Avrora, a research project of the UCLA Compilers Group, is a set of simulation and analysis tools for programs written for the AVR microcontroller produced byAtmel and the Mica2 sensor nodes. Avrora contains a flexible framework for simulating and analyzing assembly programs, providing a clean Java API and infrastructure for experimentation, profiling, and analysis.

Avrora Simulation is an important step in the development cycle of embedded systems, allowing more detailed inspection of the dynamic execution of microcontroller programs and diagnosis of software problems before the software is deployed onto the target hardware. Avrora is a clean and open implementation motivated by this need. It

also provides a framework for program analysis, allowing static checking of embedded software and an infrastructure for future program analysis research. Avrora is flexible, providing a Java API for developing analyses and removes the need to build a large support structure to investigate program analysis.

3.6.5 Test bed:

In this work, the performance of TinyCoAP, CoapBlip and HTTP, including different implementations for the transport layer used by HTTP, HTTP/TCP and HTTP/UDP is compared anddiscussed.

The implementation includes client/server transactions. The server get back information when the client sends requests to it. All the requests are sent using the GET method. The server receives a request with test as URI and the CoAP or HTTP server replies with a payload consisted by sequence of bits of fixed size. In this way, the node does not make sensing operation that might affect in the results. Therefore, the experimentsmake only for the performance of each technique in processing and replying to the received messages. The network can be simple in this work because a single client/server transactions is evaluated and deploying complex architectures can be evaded as shown in Figure 3.1.

Chapter Four: Results and Discussion

This chapter shows the implementation of CoAP in TinyOS, it refer as TinyCoAP. TinyOS has already included an implementation of CoAP called CoapBlip. However, this is based on a library not originally designed to meet the requirements of TinyOS. Thereby, it does not allow to CoAP to realize its full potential and minimize resource consumption. Better performance and minimal resource consumption can be achieved by using native library. A comprehensive performance evaluation is made to prove the effectiveness of this approach. In particularTinyCoAP and CoapBlip are tested and evaluated using avrora simulation, as well as solutions based on HTTP. The evaluation is performed in terms of latency, memory occupation, and energy consumption.

4.1 Results

The results of a performance evaluation for all the considered solution has been discussed in this section.The evaluationinvolves various parameters. First, the amount of RAM and ROM memory used by each solution has been measured; then evaluate thelatency of request/response transactions is evaluated; after that, the energy consumed by each different solution to processing and reply to a request is measured .

4.1.1 Memory occupation

The amount of RAM and ROM memory allocated at compile time for each considered implementation is shown in Table 4.2 the values for HTTP/TCP uses the TCP buffers.

Table 4.1: ROM and RAM memory Occupation

Solution	ROM/Bytes	RAM /Bytes
TinyCoAP	39040	8319
CoapBlip	43540	6800
HTTP/UDP	40430	6696
HTTP/TCP	45035	7089

TinyCoAP occupies more RAM memory than the other implementation because it specifies all the memory needed for buffering the CoAP packets at compile time. The ROM memory occupation specifythe complexity and weight of the code of each implementation. In fact, the compiled code is stored in the ROM memory. CoapBlip has the highest ROM memory occupation of optimization of the code. CoapBlip is an adaptation of a C library. This library is installed in the node along with the TinyOS component used to adapt it to the OS. The use of C libraries is usually toocomplex for the memory constraints of a mote and implies a growth of the memory occupation. Also HTTP solutions using TCP rely on a C library, so the ROM occupation increases also for these implementations.

TinyCoAPis written in NesC therefore it lowers the ROM and is optimized for TinyOS. The HTTP/UDP implementation has the lowest memory occupation. It has no reliability mechanism or request/response matching and it has a very low complexity. Therefore it can reduce the code size and memory occupation. RAM memory occupation is very low, since it does not implement any HTTP buffer. It just use UDP buffer provided by Blip.

4.1.2 Latency

one of the most significant parameters used to evaluate the goodness of the protocol design is Low latency values .The latency is defined as the time elapsed from the moment the sender sends a request until the moment it receives the response. Low latency values can significantly enhance user experience and benefit those applications that work in real-time.

The latency for each implementation has been tested comparing to show the differences between TinyCoAP and the other implementations. Payload size ranges from 1 to 30packet with increments of 1 packet ,each packet 33 bytes. The client sends a new request after receiving a response to the request previously sent.This is shown in table 4.2 and the simulation of the result is shown in figure 4.1.

Table 4.2: The latency of TinyCoAP andHTTP/TCP

No.packets	Latency(s)	
	TinyCoAP	HTTP/TCP
1	3	10
3	16	40.6
6	26	55.06
9	43	75
12	59	90
15	68	105
18	89	122
21	105	136
24	128	158
27	138	175
30	158	200

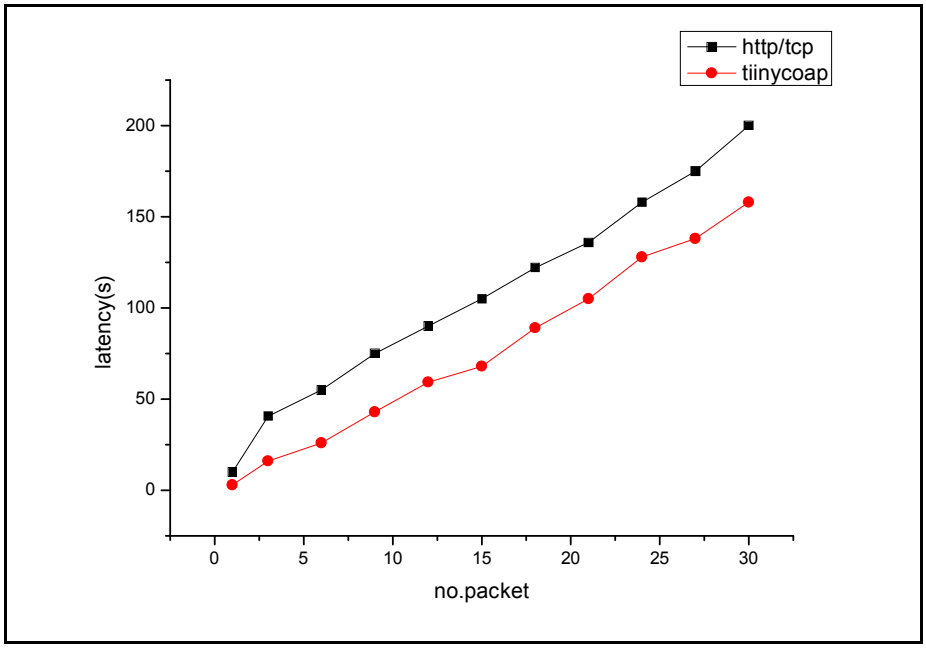

Figure 4.1: The implementation of HTTP/TCP and TinyCoAP

The lowest latency is obtained by the HTTP/UDP implementation, as shown table 4.3 and the simulation of the result is shown in figure 4.2. HTTP/UDP does not implement any reliability mechanism or HTTP logic. Therefore, it should be considered as a lower bound for latency.

Table 4.3 : The latency of TinyCoAP and HTTP/UDP

No.packets	Latency(s)	
	TinyCoAP	HTTP/UDP
1	3.9	3
3	16	14
6	26	25
9	45	43
12	59.2	53.9
15	68	65
18	89	85
21	105	102
24	128	125
27	138	137
30	158	159

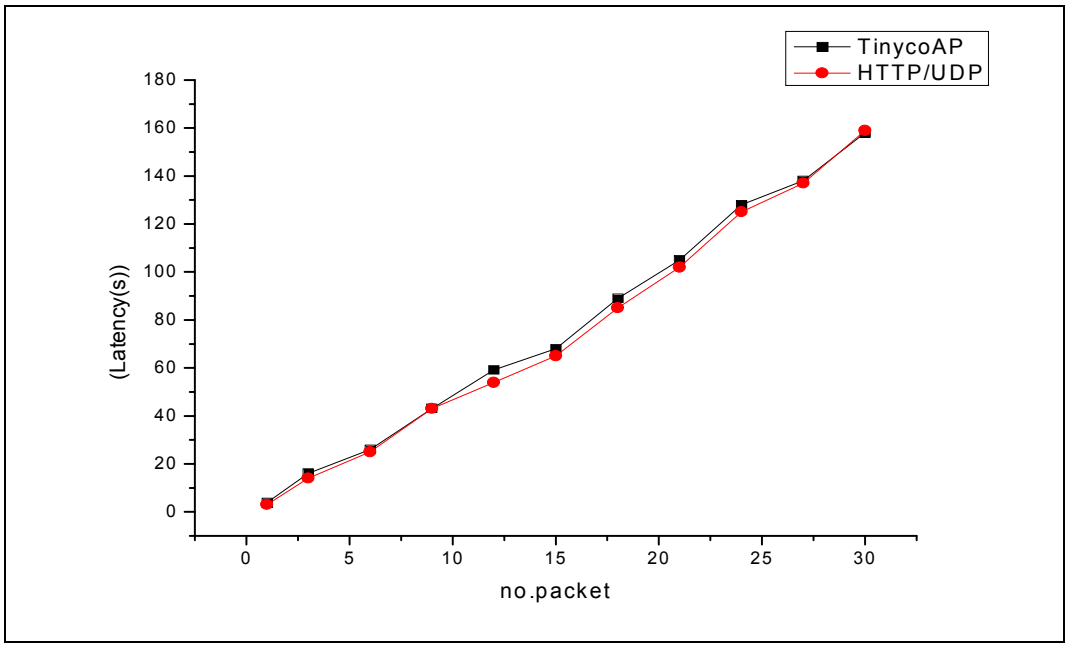

Figure 4.2: The implementation of latency for HTTP/UDP and TinyCoAP

In figure 4.3 and table 4.4show thatTinyCoAPis better than CoapBlip in terms of latency. TinyCoAP improved RAM memory management implemented and the memory allocation used by CoapBlip increases packet processing time and it can send 650 bytes the maximum payload size. Therefore Applications that work with aggregation or high payloadsizes cannot be used in CoapBlip or with HTTP implementations using TCP.

Table 4.4: The latency of TinyCoAPand CoapBlip

No.packets	Latency(s)	
	TinyCoAP	CoapBlip
1	3.9	4
3	16	20.5
6	26	34.9
9	43	58.32
12	59.2	61.5
15	68	76
18	89	118
21	105	135

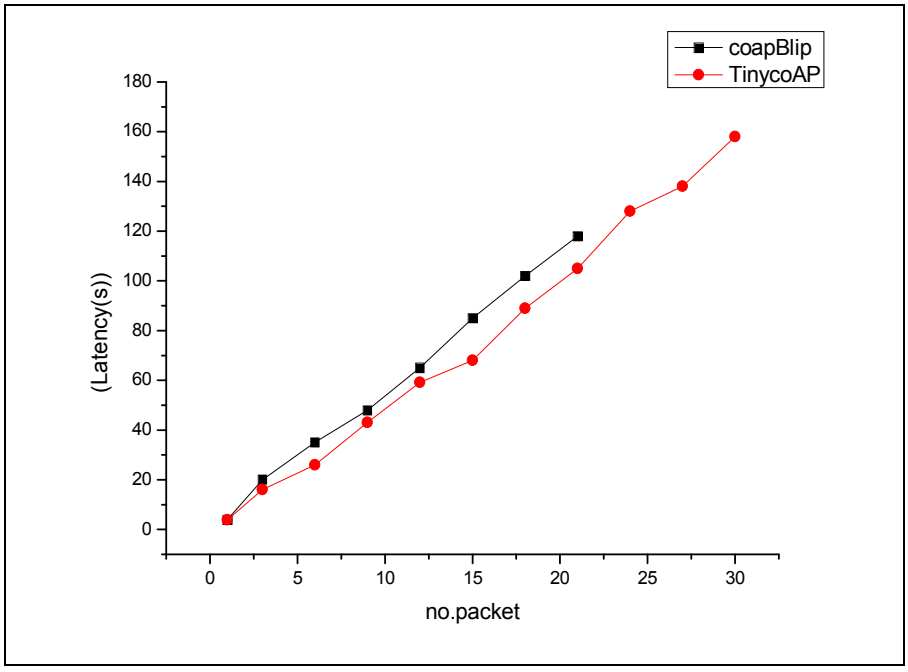

Figure 4.3:The implementation of latency forCoapBlip and TinyCoAP

The result of latency for all implementation is shown in figure 4.4.

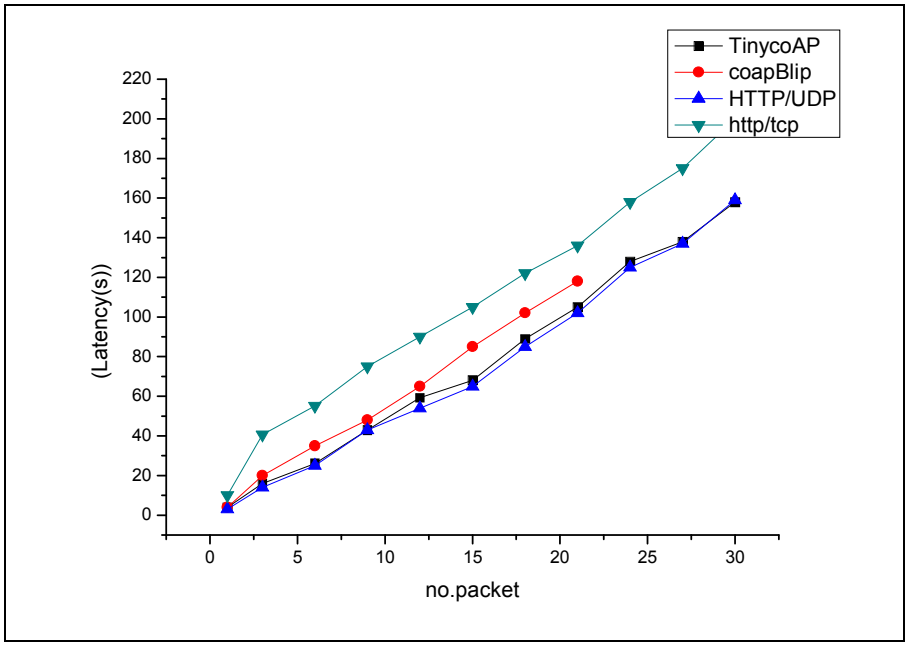

Figure 4.4 : The Latency for all implementation

4.1.3 Energy consumption

The energy consumption testshave been made for all implementation. The test measures the energy consumed by a node when replying to consecutive requests. It does not take into account the energy lost by the radio chip for listening the channel because it has the same effect in each implementation. The evaluation does not need to consider power- saving protocols for radio duty cycling. It measured the energy consumed for receiving, processing and sending a packet, The difference between the performances of each implementation is only due to the effects that the packet processing has on consumption. For each different payload size, the energy consumption has been tested and the number of node increased by 100 from 1 to 1000 nodes using the Avrora simulation.

HTTP implementations using TCP consume more energy than others. The reason is the message overhead caused by TCP lost more energy,so it is not Compatible with constrained networks. The performance is much worse than that obtained by TinyCoAP as shown in figure 4.5 and simulation result in table 4.5.The management of TCP connections

requires a high degree of complexity and the maintenance in memory of theconnection state. Consequently, there is a growth in the energy drawn by the RAM memory for keeping thesestates and the ratio between TinyCoAP and HTTP/TCP is about 1:3.5

Table 4.5: The energy of TinyCoAPand HTTP/UDP

num/nodes	**Energy/joule**	
	TinyCoAP	**HTTP/TCP**
10	1.625	5.211
50	1.641	5.431
100	1.651	5.821
150	1.698	5.931
200	1.731	5.991
250	1.761	6.008
300	1.812	6.0212
350	1.847	6.077
400	1.888	6.1043
450	1.903	6.139
500	1.9121	6.187
550	1.949	6.209
600	1.974	6.304
650	1.998	6.269
700	2.063	6.304
750	2.088	6.364
800	2.113	6.399
850	2.138	6.415
900	2.165	6.459
950	2.182	6.488
1000	2.207	6.541

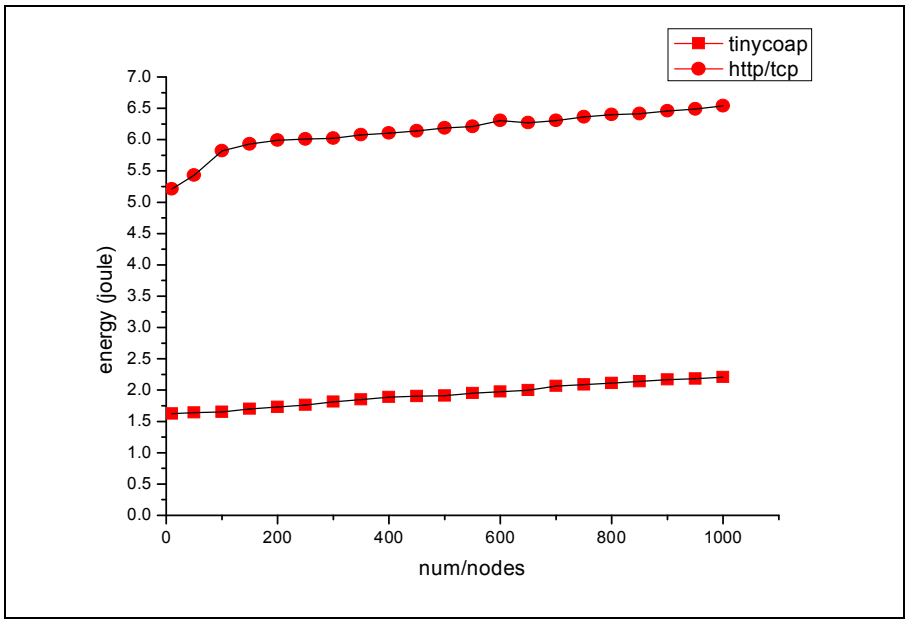

Figure 4.5: The energy consumption of HTTP/TCP and TinyCoAP

The mechanism implemented by CoapBlip to allocate and manage RAM memory shows that CoapBlip is to be unsuitable for constrained devices. The CoapBlip is again has worse performance with that of TinyCoAP as shown in figure 4.6 and simulation result in table 4.6 . However, the increase in packet size causes more consumption of CoapBlip energy.On another hand, TinyCoAP benefits from its different memory allocation mechanism. From the graph the ratio between two protocols is about 1:2.25

Table 4.6: The energy consumption of TinyCoAPand CoapBlip

num/nodes	Energy /joule	
	TinyCoAP	CoapBlip
10	1.625	3.662
50	1.641	3.712
100	1.651	3.781
150	1.698	3.887
200	1.731	3.931
250	1.761	4.001
300	1.812	4.071
350	1.847	4.106
400	1.888	4.192
450	1.903	4.192
500	1.9121	4.325
550	1.949	4.684
600	1.974	4.871
650	1.998	4.996
700	2.063	5.062
750	2.088	5.094
800	2.113	5.119
850	2.138	5.154
900	2.165	5.179
950	2.182	5.224
1000	2.207	5.294

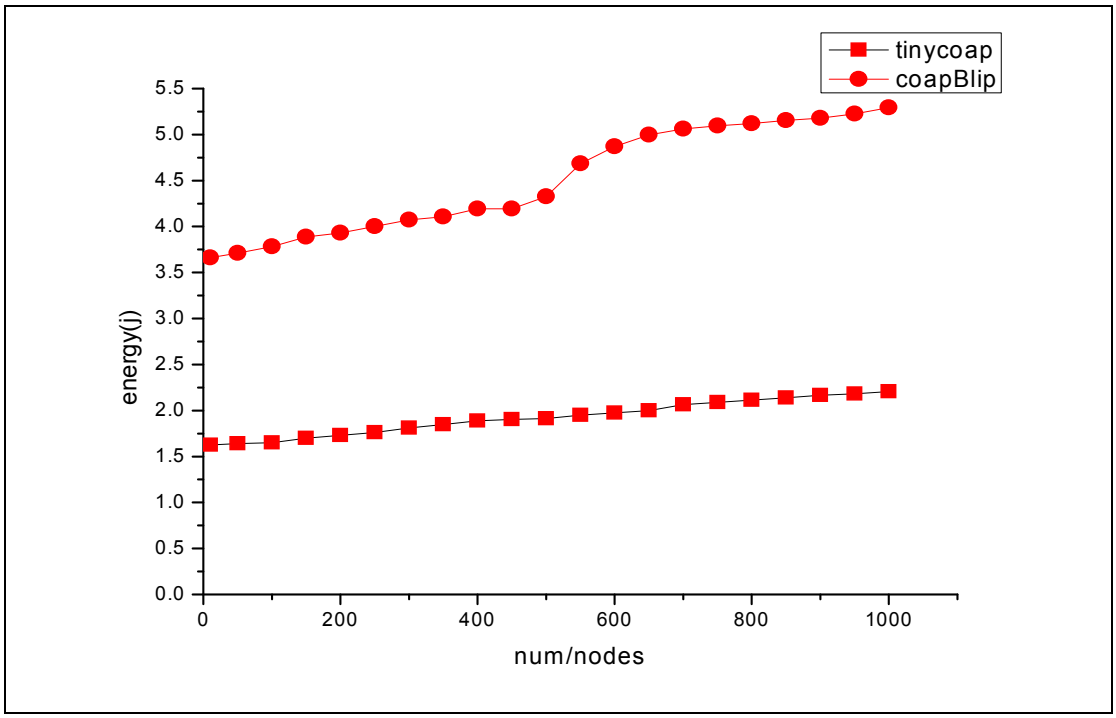

Figure 4.6: The implementation of energy consumption for CoapBlip and TinyCoAP

TinyCoAP has a performance that is highly similar to that of HTTP/UDP, the ratio between them approximately 1:1. This shows thatTinyCoAP is able to minimize the consumption of resources, as shown in table 4.7 figure 4.7.

Table 4.7: Comparison of energy between TinyCoAP and HTTP/UDP

num/nodes	Energy /joule	
	TinyCoAP	**HTTP/UDP**
10	1.625	1.6233
50	1.641	1.643
100	1.651	1.649
150	1.698	1.689
200	1.731	1.71
250	1.761	1.768
300	1.812	1.81
350	1.847	1.837
400	1.888	1.853
450	1.903	1.906
500	1.9121	1.911
550	1.949	1.936
600	1.974	1.971
650	1.998	1.996
700	2.063	2.021
750	2.088	2.098
800	2.113	2.123
850	2.138	2.148
900	2.165	2.183
950	2.182	2.208
1000	2.207	2.233

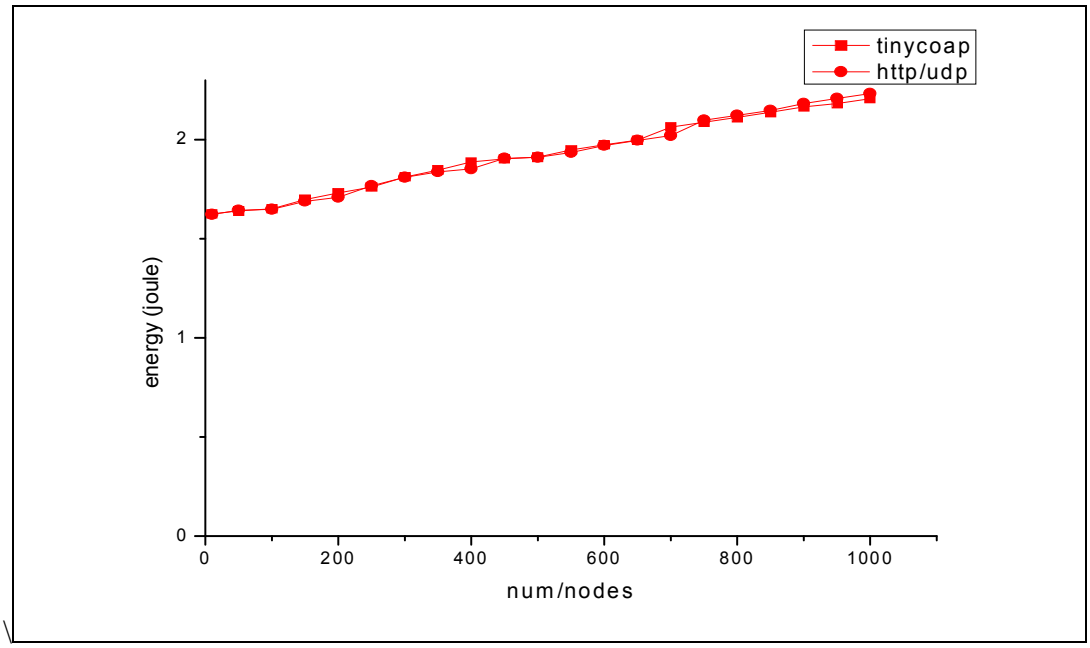

Figure 4.7: The energy consumption for HTTP/UDPand TinyCoAP

The energy consumption for all implementation is shown in figure 4.8.

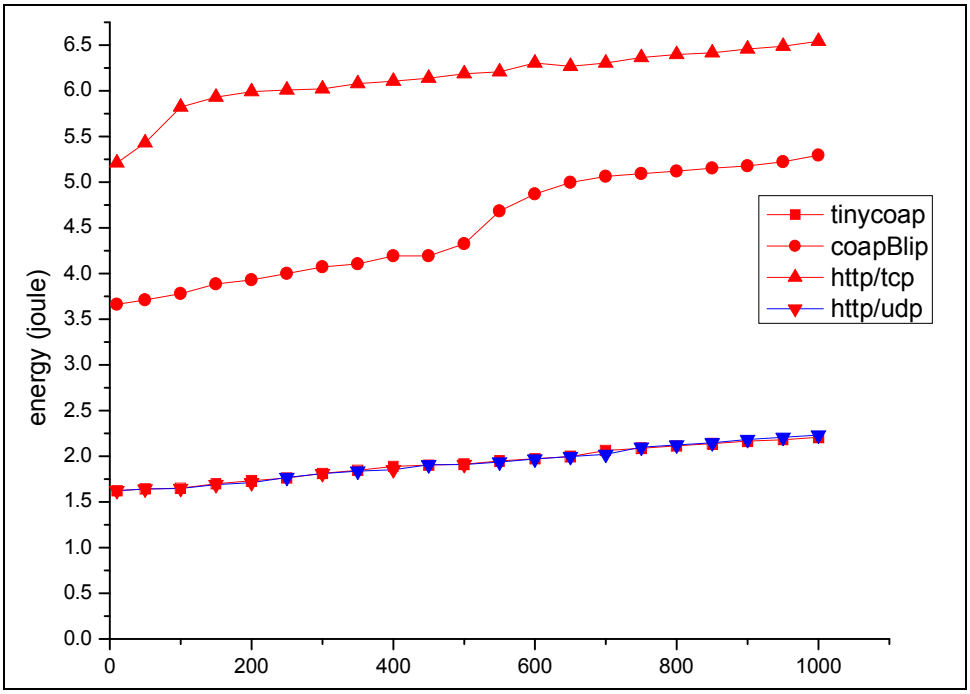

Figure 4.8: The energy consumption for all implementation

Chapter Five: Conclusion and Recommendation

5.1 Conclusion

This thesis has been committed to theimplementation and experimentation of a full-feature of TinyCoAP, which is original library for TinyOS .in addition to comparing it with the CoAP implementation distributed with TinyOS, called CoapBlip**.**

Along the dissertation we have evaluated all the solutions considered. This experience has allowed us to measure the amount of memory occupied at compile time, the latency practiced by a client when retrieving information from a server, and the energy consumed when replying to the client. HTTP is used with different solutions for the transport layer. Constant attention has been devoted to UDP and TCP connections. We denote to each of these solutions as HTTP/TCP and HTTP/UDP.

The best performance is introduced by TinyCoAP in the most of the considered parameters. In particular, TinyCoAP offer an important enhancement in performance compared with CoapBlip. The performance of CoapBlipis restricted by using the dynamic RAM memory allocation and the use of an external C library. TinyCoAP uses static allocation, so it is able to reach a high code optimization and to reduce the effect over the memory of WSN nodes

Results show that using the HTTP in WSNs produces high latency incomparison with using the CoAP protocol. The main cause is that HTTP uses the TCP protocol that uses several messages to establish a TCP connection.

In conclusion, TinyCoAP approve that it is a complete and flexible CoAP-based solution for integrating the Web communication paradigm

in TinyOS based WSNs. TinyCoAP fixes the problems founded in CoapBlip, and can improve performance considerably and to minimize the power consumption.

5.2 Recommendation

A further evaluation of the TinyCoAP implementation must be done in a real environment and simulation in order to investigate its performance in networks with higher number of nodes and parameter.Furthermore a CoAP-HTTP proxy must be design and implement to support applications that need to interact with WSN nodes. This could cause an unnecessary communication overhead and a resultant increase of latency and network traffic.

Reference

1. Davis, E.G., A. Calveras, and I. Demirkol, *Improving packet delivery performance of publish/subscribe protocols in wireless sensor networks.* Sensors, 2013. **13**(1): p. 648-680.
2. Buschmann, C., *CONET Newsletter.*
3. Kozák, J. and M. VACULÍK, *Application Protocol for constrained nodes in the Internet Of Things.* Journal of Information, Control and Management Systems, 2012. **10**(2).
4. Berners-Lee, T., R. Fielding, and L. Masinter, *Uniform resource identifiers (URI): generic syntax*. 1998, RFC 2396, August.
5. Colitti, W., K. Steenhaut, and N. De Caro, *Integrating wireless sensor networks with the web.* Extending the Internet to Low power and Lossy Networks (IP+ SN 2011), 2011.
6. Ludovici, A., P. Moreno, and A. Calveras, *TinyCoAP: a novel constrained application protocol (CoAP) implementation for embedding RESTful web services in wireless sensor networks based on TinyOS.* Journal of Sensor and Actuator Networks, 2013. **2**(2): p. 288-315.
7. Shelby, Z., *Constrained RESTful Environments (CoRE) Link Format.* 2012.
8. Bokare, M. and M.A. Ralegaonkar, *Wireless Sensor Network: A Promising Approach for Distributed Sensing Tasks.* Excel Journal of Engineering Technology and Management Science, 2012. **1**: p. 1-9.
9. Montenegro, G., et al., *Transmission of IPv6 packets over IEEE 802.15. 4 networks.* Internet proposed standard RFC, 2007. **4944**.
10. Moreno Yeste, P., *RESTful Web services in Wireless Sensor Networks.* 2011.

11. Alghamdi, T.A., A. Lasebae, and M. Aiash. *Security analysis of the constrained application protocol in the Internet of Things*. in*Future Generation Communication Technology (FGCT), 2013 Second International Conference on*. 2013: IEEE.
12. Akyildiz, I.F., et al., *Wireless sensor networks: a survey*. Computer networks, 2002. **38**(4): p. 393-422.
13. Ludovici, A. and A. Calveras. *Integration of Wireless Sensor Networks in IP-based networks through Web Services*. in*Proceedings of 4th Symposium of Ubiquitous Computing and Ambient Intelligence, Valencia, Spain*. 2010.
14. Kovatsch, M., S. Duquennoy, and A. Dunkels. *A low-power CoAP for Contiki*. in*Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on*. 2011: IEEE.
15. Colitti, W., et al. *Evaluation of constrained application protocol for wireless sensor networks*. in*Local & Metropolitan Area Networks (LANMAN), 2011 18th IEEE Workshop on*. 2011: IEEE.
16. Duquennoy, S., et al. *Leveraging IP for Sensor Network Deployment*. in *Proceedings of the workshop on Extending the Internet to Low power and Lossy Networks (IP+ SN 2011), Chicago, IL, USA*. 2011: Citeseer.
17. Chander, R.V., et al. *A REST based design for Web of Things in smart environments*. in*Parallel Distributed and Grid Computing (PDGC), 2012 2nd IEEE International Conference on*. 2012: IEEE.
18. Lerche, C., K. Hartke, and M. Kovatsch. *Industry adoption of the internet of things: a constrained application protocol survey*. in*Emerging Technologies & Factory Automation (ETFA), 2012 IEEE 17th Conference on*. 2012: IEEE.

19. Castellani, A.P., et al. *Web Services for the Internet of Things through CoAP and EXI*. in*Communications Workshops (ICC), 2011 IEEE International Conference on*. 2011: IEEE.
20. Harvan, M. and J. Schönwälder, *TinyOS Motes on the Internet: IPv6 over 802.15. 4 (6lowpan).* PIK-Praxis der Informationsverarbeitung und Kommunikation, 2008. **31**(4): p. 244-251.
21. Silva, R., J.S. Silva, and F. Boavida, *Evaluating 6lowPAN implementations in WSNs.* Proceedings of 9th Conferncia sobre Redes de Computadores Oeiras, Portugal, 2009. **21**.
22. Yibo, C., et al. *6LoWPAN stacks: a survey*. in*Wireless Communications, Networking and Mobile Computing (WiCOM), 2011 7th International Conference on*. 2011: IEEE.
23. Kuladinithi, K., et al., *Implementation of coap and its application in transport logistics.* Proc. IP+ SN, Chicago, IL, USA, 2011.
24. Potsch, T., et al. *Performance Evaluation of CoAP using RPL and LPL in TinyOS*. in*New Technologies, Mobility and Security (NTMS), 2012 5th International Conference on*. 2012: IEEE.
25. Kovatsch, F.M., *Scalable Web Technology*

for the Internet of Things, in *(Dr. sc. ETH Zurich*. 2015, Alexander-Universitat Erlangen: Germany.

26. Castellani, A.P., *Design, implementation and experimentation of a protocol stack for the Internet of Things.* 2012.
27. Ilyas, M. and I. Mahgoub, *Handbook of sensor networks: compact wireless and wired sensing systems*. 2004: CRC press.
28. Shelby, Z., K. Hartke, and C. Bormann, *The Constrained Application Protocol (CoAP).* 2014.
29. Bui, N., *Internet of things architecture (IoT-A), project deliverable D1. 1-SOTA report on existing integration*

frameworks/architectures for WSN, RFID and other emerging IOT related technology. 2014, Tech. Rep. 257521 [Online]. Available: http://www. iot-a. eu/public/public-documents/d1. 1/view, accessed on Jan. 21.

30. Mukhopadhyay, S.C. and N. Suryadevara, *Internet of Things: Challenges and Opportunities*. 2014: Springer.
31. Gay, D., et al. *The nesC language: A holistic approach to networked embedded systems*. in*Acm Sigplan Notices*. 2003: ACM.
32. Malnati, G., F. Mattern, and S. Ceri, *Web-Integrated Smart City Infrastructure.*
33. Levis, P., et al., *TinyOS: An operating system for sensor networks*, in *Ambient intelligence*. 2005, Springer. p. 115-148.
34. Shelby, Z., *CoRE Link Format, draft-ietf-core-link-format-11*. 2012, Internet draft, IETF 2012 (in progress).
35. Lauwens, B., B. Scheers, and A. Van de Capelle, *Performance analysis of unslotted CSMA/CA in wireless networks.* Telecommunication Systems, 2010. **44**(1-2): p. 109-123.
36. Zheng, M., et al., *Towards a model checker for nesc and wireless sensor networks*, in *Formal Methods and Software Engineering*. 2011, Springer. p. 372-387.