UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Departament D'Enginyeria Telemàtica

PhD Dissertation

New Architectures for Ubiquitous Networks: Use and Adaptation of Internet Protocols over Wireless Sensor Networks

---

Doctorando
Alessandro Ludovici

Director
Dr. Anna Calveras Auge

Index

List of tables

List of Figures

Acronyms

Carrier-Sense Multiple Access with Collision Avoidance (CSMA/CA)
Clear Channel Assessment (CCA)
Constrained Application Protocol (CoAP)
CoAP acknowledgment (CoAP ACK)
CoAP confirmable message (CON)
CoAP non-confirmable message (NON)
CoAP reset message (RST)
Congestion Window Size (CWND)
Controlled Mesh Under (CMU)
Enhanced Route Over (ERO)
Fully Function Devices (FFDs)
Internet Engineering Task Force (IETF)
Internet of Things (IoT)
Internet Protocol (IP)
IPv6 over Low Power Networks (6LoWPAN)
Link layer acknowledgment (MAC ACK)
Low Power Wireless Personal Area Network (LOWPAN)
Maximum Segment Size (MSS)
Mesh Under (MU)
Quality of Service (QoS)
Reduced Function Devices (RFDs)
Representational State Transfer State (REST)
Resource Directory (RD)
Retransmission Timeout (RTO)
Round Trip Time (RTT)
Route Over (RO)
Service Oriented Application Protocol (SOAP)
Signal to Noise Ratio (SNR)
Uniform Resource Identifier (URI)
Web of Things (WoT)
Wireless Sensor Network (WSN)

1. Introduction

In the past years, the idea to adopt the Internet protocol (IP) to connect everyday objects led to the definition of a new and fascinating vision, which is the so-called Internet of Things (IoT) [1]. The IoT is considered as the most promising development of the Internet of the future. In this vision, physical objects would be able to communicate between each other or with humans' through Internet. Recent estimations[1] predicted that more than 30 billions of devices will be connected wirelessly to Internet by the 2020.

The adoption of IP in Wireless Sensor Networks (WSNs) is playing a major role in the realization of the IoT vision. The possibilities opened by the IoT have virtually no limitation in the fields where they can be applied. Smart energy grid, building and home automation, e-health and intelligent transport systems are only few examples of applications domains that would benefit. In fact, the nature of wireless communications and the small size of sensor devices facilitate the development of WSNs in all kind of environments. The use of the IP allows WSNs to no longer be stand-alone networks but part of ubiquitous networks.

Thanks to its multiple applications, the IoT is attracting more and more interest from both the researcher and industrial communities. The lack of standardization, however, led to the definition and development of proprietary architectures and protocols. This fragmentation would require developing gateways and proxies to adapt the different architectures [2]. The interoperability and accessibility of IoT devices, therefore, would be limited using non-standard solutions. We believe that the IoT vision could only be realized using standard protocols and architectures [2]-[4]. In this sense, we consider that the standardization effort of the IETF [5, 6] will have a huge impact on the realization and growth of the IoT industry [6]. The IETF has standardized the use of the IPv6 protocols in WSNs. The resulting protocol stack is known as IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) [7]. 6LoWPAN enables the transmission of IPv6 datagrams over low-power networks based on the IEEE 802.15.4 standard [8]. Furthermore, organizations such as the IP for Smart Objects alliance (IPSO) [9] are promoting the use of IP in embedded devices.

The IoT vision should not be limited to provide network layer interoperability between WSNs and Internet. Enabling IP networking alone in WSNs does not allow the potential of the IoT to be fully realized. The use of standard Web services [10, 11], instead, would made possible the interaction between IoT devices and Web applications. IoT devices would be equivalent to any other Web resource and standard Web mechanisms could be used to access to them. We refer to this new approach as Web of Things (WoT).

The implementation of Web services in WSNs should be based on existing protocols and architectures. This would avoid the interoperability problems that could arise from building from scratch new ones. Traditional Web services are developed following two architectural styles: Representational State Transfer (REST) [12] and Simple Object Access Protocol (SOAP) [13]. In this thesis, we adopt REST as the reference architecture. It defines the design principles of the protocols and communication techniques that are the basis of our research.

The choice of the reference Web service architecture is consequent to the analysis of both REST and SOAP. This analysis requires a review and discussion of both architectures focused to understand which one

[1]ABI Research. https://www.abiresearch.com/press/more-than-30-billion-devices-will-wirelessly-conne

best fit with the characteristics and requirements of WSNs. From the results of this discussion derives the paper *Integration of Wireless Sensor Networks in IP-based networks through Web Services* that has been presented in the 4th Symposium of Ubiquitous Computing and Ambient Intelligence (UCAMI) [P6].

REST is an architectural style for distributed hypermedia systems originally defined to represent the model of the current Web architecture. It defines a set of architectural constraints that attempt to minimize latency and enforce security while maximizing the independence and scalability of components.

The first constraint adopted by REST is the Client-Server interaction. This allows improving the portability of the user interface and the scalability of the whole system by separating the concept of user interface and data storage. The client-server interaction must be stateless. The server does not store any context while the client manages the state of the session. This implies that any request sent from client to server contains all the information required to process it.

A further constraint defined by REST is the use of cache. This allows a client to store a response and reuse it over a period of time. The presence of cache, therefore, would reduce the number of interactions between client and server, which result in a reduction of energy consumption as well as the possibility for a node to enter in sleepy mode.

REST constrains the interface between components to uniform, which allows separating the component's implementation from the service it provides. WSNs and Web applications, therefore, would be able to communicate through REST interfaces using different protocols. REST defines four principles to obtain uniform interfaces, these are:

- Resource identification through representation. A resource is identified by a Uniform Resource Identifier (URI) [12]. A resource identifies the target of the client-server interaction.
- Manipulation of resources through representations. Resources are manipulated using a fixed set of operations. These allow creating or deleting resources, retrieving the current state of a resource and transfer a new state in a resource.
- Self-descriptive messages. Since resources are decoupled from their representations, their content can be accessed through various formats (e.g., HTML, XML, etc.).
- Hypermedia as the engine of application state. The application state is defined by the pending and active requests, the transfer and processing of representations and the topology of active components. As reported in [12], "The model application is an engine moving from one state to the next by examining and choosing from among the alternative state transitions in the current set of representations". This model corresponds to the user interface of a Web browser.

In order to clarify some aspects of REST it is important to explain the key concepts of this architecture, which are the representation and the resource. As defined in [12], a representation consists of data, metadata describing the data and optionally metadata describing metadata for verification of the message integrity. By definition, a resource is any information that can be named (e.g., document, image, service, etc) [12]. Considering a WSN, a resource is the information provided by a sensor (e.g., temperature, humidity, etc).

SOAP is a protocol specification intended for exchanging structured information in a decentralized, distributed environment [13]. SOAP has to be considered as a communication protocol that enables basic message exchanging but not as a protocol that provides service description and discovery. The information is structured in a message format defined using the Extensible Markup Language (XML) [14]. This XML element is called envelope and contains the SOAP header and the body of the message where the payload is contained. XML is a verbose data format, which is expensive in terms of overhead. REST, instead, is flexible in regards the data formatting language, which allows reducing the space required by data.

SOAP uses HTTP [15] as a transport protocol. It has therefore, a propensity to exploit the Web as a transport system. As a consequence, applications interact between each other hiding the resources they handle. REST uses HTTP as an application protocol and allow accessing and interacting to a resource only through a constrained set of HTTP methods, which are GET, POST, PUT, and DELETE. This allows the interface of a resource to be generic and with a well-known and shared semantic. SOAP, instead, focuses the Web service design around the definition of its interface by using the WSDL language [16].

In conclusion, Web services based on REST have a lightweight and simple implementation, which allow lowering the resource consumption respect to SOAP. Furthermore, the use of uniform interfaces and constrained methods to access the resources allows a less complex integration with Web applications and a lighter communication than that required by SOAP. REST, therefore, is a good choice to develop Web services tailored for WSNs. According to this vision, the IETF created a work group called CoRE [17], which seeks to standardize the use of REST Web services in constrained networks and devices [18]. To this end, its first attempt of standardization is the Constrained Application Protocol (CoAP) [19]-[21].

CoAP supports the same application transfer paradigm and basic features of HTTP. Furthermore, it adds new functionalities tailored to the constraints and characteristics of IoT devices. Typical devices are, in fact, battery-powered and equipped with few kilobytes of memory and CPUs with reduced processing power. The implementation of Web services, therefore, has to be focused to minimize their impact on these resources. The overhead introduced by Internet and Web protocols could be too demanding for the limited space of IEEE 802.15.4 frames. Therefore, the use of communication techniques able to transfer large application data could be required. To this end, this thesis focuses on the study of low-resource demanding protocols, communication techniques and software solutions to evaluate, optimise and implement the IoT and WoT paradigms. In the next section we detail and discuss the motivations that are behind this thesis.

1.1. Motivations

WSNs are data-centric networks where the information flow is mainly constituted by data collected by sensors, which have to be transferred elsewhere in the WSN or to external networks. The limited communication range of a sensor node could not allow transferring the data directly from the source to destination. Intermediate nodes might participate in the data transfer by forwarding the data received up to the final destination. We use the term multi-hop to refer to this particular WSN. We refer to WSNs where nodes are in direct communication as single-hop.

Both the transmission and the reception of a packet are predominant over the total energy used by a node to perform its tasks. Therefore, it is of paramount importance to design solutions that optimize the data transmission in multi-hop networks. Besides lowering the energy consumption, these solutions have to guarantee high performance in terms of end-to-end delay and reliability.

The performance and the energy consumption of multi-hop WSNs are highly influenced by the packet forwarding technique in use. Thereby, the optimization of packet forwarding is key to develop reliable and low-resource demanding WSNs. In this thesis we focus on WSNs adopting the 6LoWPAN protocol. This defines two forwarding techniques, which are called mesh under (MU) and route over (RO). In part of this thesis, we study the effects that MU and RO have on WSNs. Our aim is to design an original forwarding mechanism that is able to enhance the performance of MU and RO in communications that require packet fragmentation. In fact, as we explain in details in chapter 3 both MU and RO have drawbacks when dealing with packet fragmentation.

Fragmentation is not typically associated with WSNs applications. The data collected by sensor nodes in the most common applications is normally constituted by few bytes and fits with the space available in the IEEE 802.15.4. There are, however, important applications that require more space than that available. WSNs, in fact, can be used to monitor physical variables and phenomena that require a node to sample their value at high rates and continuously over a period of time. Each sensor node could store the samples in a data log that is later sent to a sink node. The size of the data log may not fit with the packet constraints of WSNs protocols. Example of these applications can be found in [22]-[24]. In [22] a WSN has been deployed to monitor railway vibrations and produces packets of 7 KB per node. In [23] the authors present a WSNs application for structure health monitoring where each node produces data logs of 512 KB. A WSN used to monitor a Volcano activity [24] produces data logs of 256 bytes per node. Furthermore, the rapid technological evolution of sensors would allow deploying WSNs for multimedia applications. The data produced by these applications could be significant and could require the application of fragmentation.

6LoWPAN uses fragmentation to send data that do not fit in a single IEEE 802.15.4 frame. The MTU defined by the IEEE 802.15.4 standard is, in fact, fixed to 127 bytes. The overhead of the 802.15.4 header and the presence of security mechanisms reduce the space to 87 bytes. The presence of 6LoWPAN, UDP and CoAP headers further reduce this space.

6LoWPAN fragmentation is resource demanding. The memory necessary to assemble a fragmented packet could exceed that available and buffer overflow can occur. Furthermore, the use of fragmentation causes an increase of energy consumption [25]. Fragmentation causes also a growth of packet error probability and forces a node to retransmit the packet. Repeated retransmissions led to a wasteful use of the bandwidth. Furthermore, 6LoWPAN fragmentation has drawbacks on its end-to-end reliability support. The lost of a fragment, in fact, causes the retransmission of the entire 6LoWPAN fragmented packet. 6LoWPAN does not define any end-to-end reliability mechanism to recover from losses of single fragments. The transmission of 6LoWPAN packets, in fact, only relies on the link layer acknowledgments (MAC ACK) as defined in [8]. These are used to acknowledge the reception of a single 802.15.4 data frame over a single hop. In multi-hop WSNs this mechanism does not guarantee the correct reception of the fragmented packet.

CoAP defines an alternative to 6LoWPAN fragmentation, which is called CoAP blockwise transfer [26]. The aim of CoAP blockwise transfer is to enhance reliability avoiding 6LoWPAN fragmentation. In this technique a packet is divided into blocks and the data transfer into multiple request/response transactions. The transmission of a single block corresponds to a single CoAP request/response transaction. The failure of a single block or of the corresponding request causes only the retransmission of the request.

Although the design of CoAP blockwise transfer improves reliability and reduces the number of retransmitted messages, its effect on end-to-end delay could be not as good. Furthermore, depending from the WSN configuration the packet error probability could be low enough to discourage the use of CoAP blockwise transfer. An analysis of 6LoWPAN fragmentation and CoAP blockwise transfer is therefore needed to understand which technique best applies to each particular configuration. In this sense, we design an analytical model to study the behaviour of these techniques in one-hop WSNs adopting a star topology. This model must consider the presence of the carrier-sense multiple access with collision avoidance (CSMA/CA) mechanism defined by the IEEE 802.15.4 standard. The inclusion of this mechanism is of paramount importance to evaluate the packet losses caused by collisions and channel access failures.

As mentioned, WSNs are data-centric networks characterized by having constrained resources. The data transfer model to adopt should, therefore, be compliant with these characteristics. Its design has to be as simple and as lighter as possible to minimize the resource consumption. In particular, the number of messages exchanged between nodes should be reduced to save bandwidth. In this context, the traditional request/response model is inefficient. Clients have to periodically poll the server to receive updates of the state of a monitored resource or event. As a consequence, the traffic will increase dramatically congesting and overloading the WSN. Furthermore, the resource consumption would grow. Instead, an asynchronous model would be more suitable for these characteristics. CoAP includes asynchronous data transfer through the observe option [27]. This allows a client node to register to a resource exposed by a server node and receives update of its states. In real applications, however, the nodes interested in receiving updates could be numerous and with different Quality of Service (QoS) requirements. CoAP offers QoS support for reliability but timeliness is not fully supported. In particular, on-time delivery of the updates is not implemented. However, critical applications such as e-health or industrial monitoring have strict requirements about timeliness. In this thesis we also focus on designing a solution to provide QoS support for timeliness in the observe option of CoAP.

The use of CoAP in WSNs has been studied in several works. However, none of them contain an extensive evaluation of CoAP. In particular, the reliability mechanism provided by the protocol has not been evaluated. Although a comparison between CoAP and HTTP is present in all these works, they do not consider the possibility of using HTTP with transport protocols different from TCP. Furthermore, we recognized that the CoAP implementations that have been used to evaluate CoAP are not optimized to be embedded in WSNs. Therefore, they do not show the real performance of CoAP. To overcome these limitations we design an optimized CoAP implementation for the TinyOS [28] operating system. We evaluate and compare the performance of our implementation with that of the officially distributed by TinyOS.

Furthermore, we include the evaluation of the reliability mechanism of CoAP and we consider the use of HTTP with different transport layer solutions.

While CoAP is a promising protocol to realize the WoT vision, we think that applications based on HTTP will still be used to access to CoAP networks. In this perspective, the presence of a proxy that is able to map both protocols is key for the diffusion of WoT applications based on CoAP. In this thesis we design a CoAP proxy that is able to connect Web applications to CoAP networks. As explained before, the use of the asynchronous data transfer provided by the observe option could allow to reduce resource consumption in WSNs. However, HTTP does not define communication models equivalent to that of the observe option. One of the challenges of our proxy design is to allow to Web applications to benefit from the asynchronous model of CoAP. This would reduce the traffic caused by Web applications that need to retrieve data continuously from WSNs.

Motivated by the exposed above, a study of new protocols for implementing the IoT and WoT visions is needed. In particular, in the current literature there is a lack of research works on the effects that large data transactions have on WNSs. In this sense, we decide to study the forwarding techniques of 6LoWPAN focusing on packet fragmentation and CoAP blockwise transfer. We also study CoAP focusing on the design of embedded software solutions as well as on its QoS provision. CoAP is still under standardization and its diffusion in real applications is limited. Our research work, therefore, could help positively its development and application in real scenarios.

1.2. State-of-the-art

The analysis of the state-of-the-art starts with the discussion of 6LoWPAN forwarding techniques. We present related works on MU and RO as well as the motivations behind their definition. Then, it follows the presentation of the existing proposals for Web services in WSNs and the state-of-the-art of the research works on application protocols for WSNs and CoAP. Finally, we present related works on analytical models of large data transaction in WSNs

1.2.1. Forwarding Techniques in 6LoWPAN

As mentioned, forwarding techniques have a significant impact over the performance and resource consumption of multi-hop WSNs. Several works have studied the problematic related to forwarding in multi-hop 6LoWPAN networks. In particular, a great deal of attention has been given to the definition of alternative RO techniques. In the rest of this section we present related works on forwarding techniques in 6LoWPAN. We start reviewing the key concepts behind the definition of MU and RO. Then, it follows the review of the state-of-the-art of this research topic.

In [29] the author details some of the fundamental assumptions made during the development of the 6LoWPAN protocol. Among these, he reviews the basic concepts behind the definition of MU and RO. The author claims that the original design of 6LoWPAN follows a MU philosophy. However, the use of IP

diagnostic tools or source routing would be no longer possible under this assumption. To overcome these problems, the author proposes the use of IP based routing and to separate the concepts of routing and forwarding. In this way, the routing protocol manages the IP routing tables while the network layer forwards the IP packets. The author claims that a major drawback of RO is that the IPv6 routing would require that each node in the network should be its own subnet. To overcome this problem, the author suggests using RO and MU together. In this perspective, a WSN could be split into many small networks based in MU but using RO to communicate with each other.

In [30] the authors discuss the 6LoWPAN key concepts including the definition of MU and RO. As in [29], the lack of tools used to form, maintain and diagnose IP networks are considered as a weakness of MU. The authors recall that issues of link versus network layer routing are present also in other networks technologies (e.g., ATM, Frame Relay). Taking as example the IP over ATM, the authors suggest that a possible solution for routing in 6LoWPAN should be inspired to the Multiprotocol Label Switching (MPLS) in which the routing engine is located at network layer but the forwarding is done at layer two.

In [31] the authors discuss the problems arising from the use of IP above link technologies designed for constrained networks. The authors claim that widely used link technologies (e.g., Ethernet or IEEE 802.11) emulate a single broadcast domain to perform link-layer forwarding or routing. In this way, the services provided by the network-layer to form and maintain a network are simplified. In a single broadcast domain, a node is able to send an IP packet without the need to understand the physical topology of the network. The authors discuss MU and RO focusing on the possibility to emulate or not a single broadcast domain also in networks that adopt the 802.15.4 standard. In these networks, the implementation of MU would require the emulation of the broadcast domain. However, with this solution the IP routing protocol would not have the necessary visibility of the underlying radio topology to perform basic tasks (e.g. determine if a node is reachable). Furthermore, the use of IPv6 multicast would be costly and the use of data aggregation from multiple sources would be no longer applicable. The authors recommend to avoid the emulation of a single broadcast domain in MU and to adopt a RO architecture where the broadcast domain is equal to the radio transmission range.

Guidelines for the design of 6LoWPAN routing protocols are presented in [32]. The authors illustrate the differences between MU and RO by describing the reference network models related to each approach. In this work the authors also report specific requirements for MU.

Part of our work on forwarding techniques focuses on the performance evaluation of MU and RO in a real WSN. The result of this evaluation would help us to design an improved forwarding technique for 6LoWPAN. In the literature, however, there are few works that focus on performance evaluation. In particular, only the work presented in [33] was published at the time we started our research on forwarding techniques. This work, however, presented an analytical evaluation of MU and RO. Our research, therefore, covered the lack of performance evaluation of MU and RO in a real WSNs implementation. In [33], in fact, both techniques are compared using a probabilistic model that seeks to evaluate the packet arrival probability, the total number of transmissions and the total delay between source and destination. This analysis is performed considering a multi-hop network with communications that require 6LoWPNAN packet

fragmentation. Results in [33] demonstrate that RO has a higher fragment arrival probability than MU. Furthermore, the results show that RO can experience buffer overflow when the network traffic is high and a node receives packets from different paths. The evaluation of the delay shows that it is higher in communications that use RO. Furthermore, the authors prove that RO allows reducing the number of retransmissions.

The work presented in [34] also focuses on the evaluation of MU and RO. However, it is successive to the conclusion of our research. In [34], the authors present an analytical analysis of MU and RO focused to evaluate their performance in terms of end-to-end delay and reliability. The application scenario is an Advanced Metering Infrastructure (AMI) with a variable number of hops. The analysis considers the presence of 6LoWPAN fragmentation as well as its absence. RO proves to be more reliable than MU when the communication requires packet fragmentation. MU instead has better latency and it is preferable when the communication does not use fragmentation.

In [35], the authors present a survey on routing and mobility solutions for IPv6 communications. In this survey, MU and RO are presented and the possible routing protocols applicable to each technique are discussed. Both techniques are discussed considering a multi-hop network and the presence of fragmented packets. In this perspective, the authors claim that the hop-by-hop fragmentation and reassembling of 6LoWPAN packets is the major drawback of RO. Furthermore, they consider that MU benefits from the fact that it can use multiple paths to forward fragments of the same packet. RO, instead, is constrained to use a single path. Considering the results presented in [33] Oliveira et al. recommends choosing MU or RO depending on the requirements of the application (e.g., reliability, use of fragmented packets).

In the literature, a great deal of attention has been focused to the design of techniques able to improve the performance of RO. In this thesis, instead, we focus on the design of a forwarding technique able to enhance the performance of MU. To the best our knowledge, only the work in [36] considered the design of a new forwarding scheme based on MU. However, this is successive to the conclusion of our work.

In [36] the authors present a novel forwarding techniques, which they refer to as chained MU. The purpose is to enhance the packet arrival rate of MU in multi-hop networks. The proposed solution defines the presence of temporary assembling nodes, which are intermediate nodes between the source and the destination. These nodes reconstruct the original IP packet from the received fragments before fragmenting it again and delivering it to the next node or to destination.

Alternative techniques to RO are presented in [37]-[39]. In [37], the authors presented a technique called modified RO. This proposal seeks to enhance the performance of RO when this is used to forward 6LoWPAN fragmented packets. The retransmission of the entire fragmented packet is avoided by allowing to an intermediate node to request the retransmission of the missing fragments. The authors compare MU, RO and modified RO in an e-health scenario. Modified RO shows better performance and proves to lower the latency and packet loss ratio.

RO can be implemented with methods able to create virtual reassembly buffers that remember only the IPv6 header contained in the first fragment [38]. The work presented in [39] proposes to create a state associated to the IPv6 source address and to the datagram tag of the fragmentation header. This solution

allows establishing a virtual circuit for the subsequent fragments. We refer to this proposal as enhanced route over (ERO). In chapter 3 we present in detail this proposal.

1.2.2. Web services for Wireless Sensor Networks

The interest towards the application of Web services in WSNs has grown rapidly over the past year. Both SOAP and REST architectures have been developed and tested in these networks.

The authors of [40] developed a Web service architecture based on SOAP and WSDL for 6LoWPAN. In this work, the 6LoWPAN and the Web Service architecture are considered as two independent entities that communicate through a Web server and a 6loWPAN border router. The main drawback of this implementation is that the server has to continuously poll the sensors to obtain the data. In [41] the authors advise against the use of polling to obtain data from sensors. They recommend the use of event driven communication in order to minimize the resource consumption.

The authors of [41] presented in [42] a research work that aims to eliminate the presence of gateways by embedding Web services directly in constrained devices. These embedded Web services are based on SOAP and use the Device Profile for Web Services (DPWS) [43]. The authors develop a prototype and test its performance in terms of memory footprint and round-trip-time. Results from [42] prove that solutions based on SOAP have high latency, which is mainly due to the use of TCP. However, the authors state that the memory used by DPWS is significantly lower than that used by the operative system to allocate the buffer for IPv6 (1280 Bytes) and SOAP (2000 Bytes). Finally, the authors consider crucial the reduction the size of the SOAP/XML message through compression of XML. A similar conclusion is found in [44]. Here, the authors argue that the main drawback of SOAP solutions is the overhead generated by the verbose format of XML. The authors suggest considering the possibility to develop only REST Web services and not using SOAP in WSNs.

An early work that considers the application of REST in WSNs is presented in [10]. However, this work does not focus on providing Web services to access and interact with sensors but only to discover devices. The problem of the integration of WSNs and Internet is considered in [11]. In this work, the authors develop an HTTP like protocol called TinyREST. This protocol is thought to establish a communication between a WSN and a gateway. The gateway is considered as the interface between the Internet application and the WSN. The work presented in [45], considers Web servers embedded in sensor nodes to communicate directly with a Web client through HTTP.

1.2.3. Application Protocols for Wireless Sensor Networks

As mentioned, the development of Web service tailored for constrained networks is key to integrate WSNs in classic Web applications and to leverage the interoperability of different network technologies and low-layer protocols. As previously discussed, the REST architecture is the reference style for Web service development. The HTTP/TCP stack is the standard solution used to develop classical REST Web services. However, its implementation in constrained networks would not be feasible. The HTTP protocol header is, in

fact, too chatty for the limited space available in 6LoWPAN frames. As reported in [18], HTTP has evolved into a complex protocol that involves optional headers and a number of features that complicate its implementation in constrained devices. Moreover, the TCP protocol would have a poor effect on the available bandwidth of WSNs. Instead, the interoperability of different networks would be possible adopting this solution. Therefore, the development of a lightweight binary protocol that could be easily mapped to HTTP and bound to UDP would meet both requirements.

As previously anticipated, an early proposal [11], which is called TinyREST, defines a protocol that enables sensors to interact with Web clients. The TinyREST commands are created combining the request method with the URI of the sensor's resource. These request methods are limited to POST, GET and SUBSCRIBE requests of HTTP. Instead of using IP, TinyREST uses the networking tools provided by the active message stack of TinyOS. The authors claim that using "full IP packets in WSNs would be devastating for the usually limited power resources of sensors/actuators". The messages are transmitted as plain ASCII and generate an overhead of 29 bytes. The use of TinyREST is intended only for communications inside the WSN. The interaction with external Web services is performed through a gateway responsible for translating HTTP messages into TinyREST ones.

A proposal for a binary version of HTTP, which is called Embedded Binary HTTP (EBHTTP), has been defined in [46]. An application of EBHTTP for building REST Web Services is considered in [47]. EBHTTP is a binary-formatted and stateless encoding of the standard HTTP protocol. Its use is intended for resource constrained WSNs. The design of this protocol focuses in reducing the overhead of HTTP while maintaining the same semantic and communication paradigm. EBHTTP uses the UDP protocol instead of TCP.

An early proposal from the CoRE work group, which is called Chopan [48], follows the EBHTTP idea of compressing HTTP messages into a binary format. As EBHTTP, Chopan uses UDP as transport protocol. The proposal also includes the presence of transparent caching and gateways for translating Chopan into HTTP. The weaknesses of Chopan and EBHTTP can be found in the lack of reliability and in the processing power required to encode the HTTP protocol. This would be less than that required by adopting HTTP but it would be still expensive for the limited resource of a sensor. Furthermore, since they are bound to UDP they do not provide any reliability mechanism.

As a result of these limitations, attention has shifted to the definition of a new application protocol designed to fit to the constraints of WSNs and to comply with the REST principles, which is CoAP. As mentioned, CoAP is one of the main themes of this thesis and a detailed definition of its functionalities will be given in chapter 2. In the rest of this section we present the latest research on CoAP.

1.2.4. Constrained Application Protocol (CoAP)

As mentioned, one of the main contributions of this thesis is the design and optimization of software solutions to implement the IoT paradigm in WSNs. In this sense, we design a CoAP implementation to be embedded in WSN nodes and a CoAP proxy to interconnect Web applications to WSNs. Both solutions are evaluated in a real WSN scenario. In particular, the performance achieved by our implementation has been

compared to that of existing implementations based on CoAP and HTTP. The evaluation of our implementation not only allows to validate its design but also to analyse the performance of CoAP. Next we review the state-of-the-art of CoAP implementations for WSNs, CoAP proxies and CoAP performance evaluations.

The authors of [49] present a survey of CoAP implementations and show the results of an interoperability meeting organized by the European Telecommunications Standards Institute (ETSI). This survey presents implementations that target both WSNs as well as other environments. For the purpose of this thesis, we are interested in CoAP implementation for WSNs. In this sense, CoAP has been already implemented in the most popular operating systems (OSs) for WSNs such as Contiki [50] and TinyOS [51, 52]

A CoAP implementation, which we refer to as CoapBlip, is presented in [51]. The authors review its design and evaluate their implementation by comparing it to a HTTP implementation. This performance evaluation considers the ROM footprint and the average response time of CoAP and HTTP. Preliminary results of an evaluation show that CoAP yields better performance than HTTP. CoapBlip is released with the latest distribution of TinyOS, which is the target OS of our implementation. We compare, therefore, the performance of CoapBlip to that achieved by our implementation. Details of CoapBlip are presented in chapter 5 along with the design of our CoAP implementation.

In [53], CoapBlip is used to evaluate the CoAP protocol in combination with other low layer protocols. In this sense, it is evaluated along with the Routing Protocol for Low-power and Lossy Networks (RPL) and the Low Power Listening (LPL) protocol.

A further CoAP implementation for TinyOS is presented in [52]. The authors carried out a performance evaluation 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 TinyOS components used in their implementation. Differently from CoapBlip, we choose not to include this implementation in our performance evaluation. It is, in fact, developed on top of an unsupported and limited 6LoWPAN implementation named 6lowpancli [54]. In particular, as pointed out in [55, 56], 6lowpancli provides only basic functionalities of 6LoWPAN. 6Lowpancli does not support any type of neighbor discovery mechanism, it is completely static and requires manual configuration. As reported in [55, 56] 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 [55] 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.

The authors of [50] present a CoAP implementation for Contiki. The aim of this implementation is to achieve high-energy efficiency by leveraging a radio duty cycling mechanism. The implementation is evaluated in a multi-hop network. The results show that energy consumption is lower when using a radio duty cycle but leads to in a worsening of the latency performance.

As mentioned, along with the evaluation of our original implementation we provide a comprehensive analysis of the functioning of CoAP including an evaluation of the reliability mechanism. The performance of CoAP has been evaluated in several works [50–52, 57–59]. However, none of these contain an extensive

evaluation of the protocol. In particular, the CoAP reliability mechanism has not been evaluated. Although these works present a comparison between CoAP and HTTP, they do not consider the possibility of using HTTP with transport protocols different from TCP. In this thesis, we evaluate the performance of an HTTP server using UDP and persistent TCP connections. In fact, a fair comparison between CoAP and HTTP should at least include the above-mentioned possibilities in order to minimize the effects that TCP has on the HTTP performance, otherwise a comparison between CoAP and HTTP would result in an evaluation of UDP and TCP.

In [57], the authors report a simple comparison of CoAP and HTTP in terms of energy consumption. This work also describes the design of a gateway used to connect a CoAP based WSN to an external IP network that uses HTTP. In [58], the authors of [57] compare the performance of CoAP to that of HTTP. The evaluation is carried out on the basis of energy consumption and response time. In particular, energy consumption is evaluated by means of simulation. The response time, however, is measured in a real WSN. Both experiments consider a client querying an embedded server to obtain temperature and humidity values. The energy consumed is measured according to the variation of the interarrival packet time. The response time is calculated for the case where the server is at a distance of 1-hop and 2-hop from the client. The results obtained show that CoAP yields a better performance in both the evaluation parameters.

A study on network sensor deployment [59] used CoAP and HTTP as data transport protocol for sensor network reprogramming. Both protocols are evaluated over a duty cycled radio layer. Results are obtained through simulation and show that CoAP and HTTP provide similar results. In [60], the authors present a framework for machine-to-machine (M2M) communications using CoAP. They also present an improved publish/subscribe mechanism also based on CoAP. Both solutions are evaluated showing the advantage of using CoAP instead of HTTP.

As mentioned, the design and development of a CoAP proxy is one of the main contributions of this thesis. The presence of a CoAP proxy is of paramount importance to interconnect HTTP based networks with CoAP WSNs. It is expected, in fact, that most of the accesses to CoAP WSNs will come from traditional HTTP networks.

Our CoAP proxy is designed to provide support to applications that need to continuously retrieve data from the WSN. Traditionally, the HTTP long-polling technique has been used in these applications. However, it could result inefficient in this scenario. The use of HTTP long-polling, in fact, forces Web applications to query constantly the CoAP proxy to receive data from the WSN. This could cause an excessive communication overhead and a consequent increase of latency and network traffic.

Although HTTP long-polling shows limitations, none of the existing works on CoAP proxies consider alternative to it use. Furthermore, they do not present any performance evaluation. In this thesis, we consider alternative to HTTP long-polling and present a performance evaluation of the CoAP proxy. Our contribution is, therefore, significant to state-of-the-art of CoAP proxies. The existing works that focus on CoAP proxy design are reviewed next.

The authors of [61] describe the main building blocks of a simple CoAP gateway used to monitor remotely a WSN. The gateway performs as a cross-proxy to translate HTTP requests into CoAP ones. The use

of a 6LoWPAN gateway to monitor a CoAP WSN is also presented in [62]. The authors of this paper design an update system to achieve the interaction between the Web application and the gateway. In [63], the authors develop a CoAP gateway for home automation. A HTTP-CoAP proxy is used in [64] to expose the services provided by a framework for smart energy grid to Web applications. A study of lightweight protocols to minimize resource consumption in constrained gateways is presented in [65]. The authors compare the performance obtained by CoAP to that of MQTT [66]. CoAP proves to be the most efficient in terms of energy and bandwidth usage. CoAP is tested considering the classic request/response data transfer as well as the asynchronous one provided by the observe option.

The CoAP observe option is also one of the main topics of this thesis and its definition is given in details in chapter 2. As mentioned, our research on the observe option seeks to improve its Quality of Service (QoS) provision in terms of timeliness. Furthermore, our study aims to define a mechanism to let a client negotiates a particular QoS level. The negotiation of the QoS level is not common in publish/subscribe protocols. Although important, few of them have this characteristic.

The authors of [67] provide a mechanism for negotiating QoS in terms of timeliness. In this model, a subscriber can specify the delay constraint as an attribute of its subscription request. The broker will then select the best path to route the notification and, therefore, meet the delay requirement.

Reliability can be provided with best effort or persistence mode. In best effort the retransmission of lost updates is not provided. In persistence mode the publisher is able to retransmit updates. The solutions presented in [68]-[70] provide reliability only with best effort. Persistence is provided in [71, 72] and [66]. These systems, however, only provide reliability from the fault-tolerance point of view. They use mechanisms to permit to brokers to recover after a failure or, to route information towards active brokers. They do not negotiate the reliability with subscribers. In [73], the authors propose a framework to provide reliability and timeliness for publish/subscribe protocols in Wide Area Networks (WANs). Both parameters are provided using gossip protocols and network coding.

Regarding the protocols designed for WSNs, MQTT-s [74] allows to subscribers to define three QoS levels for reliability. The first level offers a best effort delivery service. The second allows the retransmission of an update until the receiver acknowledges it. Retransmitted messages may arrive duplicated at the destination. To overcome this problem, the third level ensures that the same message is received only once. A comparative performance evaluation of MQTT-s and observe is presented in [75]. The authors focus on the reliability support of both protocols. They propose to use a retransmission timer that adapts to network conditions instead of using the fixed timer defined by both. A performance evaluation done in different network topologies shows that the proposed approach increases the packet delivery ratio of observe and MQTT-s.

The authors of [76] define QoS support for real-time publish/subscribe in WSNs. A Subscriber can specify the maximum tolerated delay for receiving updates. A dispatcher is used to meet the delay requirement. Depending on the required delay, the notification can be buffered in a QoS or in a non-QoS queue. The dispatcher gives priority to updates with real-time constraints. In [77], the same authors propose a publish/subscribe middleware for WSNs that provides a mechanism for supporting fault-tolerance and real-

time requirements. This is achieved through a cluster-based organization of the WSN. The middleware receives QoS requirements from the cluster-head nodes. These requirements are about the services of the WSN and their default operation conditions in terms of delay, data rate and energy. Once the middleware has this information it is able to provide the required QoS.

In [78], the authors present a middleware to provide a publish/subscribe scheme for WSNs. The publisher provides QoS for reliability, priority and deadline. Reliability is achieved through retransmission while priority allows finding the short-path to destination based on the importance of the packet. Deadline allows a node to discard a packet if its deadline has expired. However, the QoS levels are chosen by the publisher according to the importance of the packet to send.

The QoS negotiation that we propose also permits a client to select the updates it wishes to receive. A similar proposal can be found in [79]. The authors present an extension to the observe protocol, which is called conditional observe [80]. This allows a client to filter the notification sent by a publisher by indicating a threshold value. Furthermore, the client can indicate if the value that should be contained in the desired updates is equal, above or under the threshold. In [60] the authors propose an extension of the observe protocol that allows a client to specify the period of time during which it wants to receive the updates. This extension is called duration.

1.2.5. Analytical model of large CoAP data transactions

In part of this thesis, we propose a novel analytical model to study CoAP blockwise transfer and 6LoWPAN fragmentation in one-hop WSNs adopting a star topology. To the best of our knowledge this is the first research that evaluates and compares analytically the performance of these communication techniques. In the literature there are no works that propose an analytical model for the same problematic or that compare 6LoWPAN fragmentation to CoAP blockwise transfer.

A great deal of attention has been given to study the IEEE 802.15.4 channel access mechanism, which is out of the scope of this thesis. To the best of our knowledge there are only two works that focus on CoAP blockwise transfer and 6LoWPAN fragmentation. The study in [25] evaluates the effects of 6LoWPAN fragmentation on the energy consumption. CoAP blockwise transfer is considered in [81]. The authors present a service management system that seeks to reduce the energy consumed by CoAP blockwise transfer. The authors reduced the overhead introduced by transport and application layer data by keeping the communication at network level between the most constrained nodes. The packet size is further reduced by the definition of a parameterized resource description and representation.

A thorough study of both communication techniques is therefore needed to understand their behaviour and optimize their use in WSNs

1.3. Thesis Methodology

In this section we describe the tasks that compose the realization of this thesis. For each one we describe its goals and the tasks that compose it.

1. 6LoWPAN forwarding techniques

The purpose of this task is the optimization of the 6loWPAN forwarding techniques. This phase is composed by the followings sub-tasks:

a. Study of the state-of-the-art.
b. Implementation of MU and RO in TinyOS.
c. Performance evaluation of MU and RO with non-fragmented packets.
d. Design of a new alternative for 6LoWPAN.
e. Performance evaluation of MU, RO and our new forwarding technique with fragmented packets.
f. Dissemination of results

2. Analysis of large packet transmission

The purpose of this task is to study the performance of a 6LoWPAN network when the communication involves data that do not fit in a single IEEE 802.15.4 frame. We develop an analytical model to study CoAP blockwise transfer of and 6LoWPAN fragmentation in one-hop WSNs with star topology. This task is composed by the followings sub-tasks:

a. Study of the state-of-the-art
b. Research the reference model for the IEEE 802.15.4 unslotted CSMA/CA mechanism
c. Design of the analytical model.
d. Evaluation of the model in topologies.
e. Dissemination of results

3. Web services architectures for constrained networks

This task includes the study of the current architectural styles for Web services. The aim is to find the architecture that best fits with the constraints of WSNs. This task is composed by the followings sub-tasks:

a. Study of the state-of-the-art.
b. Analysis of SOAP and REST architectures.
c. Evaluation of the architectures and application protocol.
d. Choice of the architecture and application protocol.
e. Dissemination of results

4. Application protocols for 6LoWPAN

The Web service architecture chosen in the previous task determines the application protocol to be used in our work. This task requires the study of the state-of-the-art of the application protocols for WSNs. The suitable protocol is implemented and evaluated in a real 6LoWPAN network. The results collected will be useful to optimize the design of CoAP and reduce the impact it would have on WSNs resources. CoAP is the protocol that we chose to implement and study. This task is composed by the followings sub-tasks:

a. Study of the state-of-the-art.
b. Analysis of the CoAP protocol.
c. Implementation and performance evaluation of CoAP.
d. Implementation and performance evaluation of a CoAP proxy
e. Optimization of CoAP based on the obtained results.
f. Proposal for QoS support for timeliness in the CoAP observe option
g. Dissemination of results

1.4. Thesis Outline

In Chapter 2 we present the protocols adopted in this thesis. We focus on the features of these protocols that are relevant for our work.

In Chapter 3 we discuss the work and results from our research on the 6LoWPAN forwarding techniques.

In Chapter 4 we present the analytical model and the performance evaluation of the CoAP blockwise transfer and 6LoWPAN fragmentation.

In Chapter 5 we present our implementation of CoAP and discuss its performance evaluation.

In Chapter 6 we describe the design and implementation of a CoAP proxy.

In Chapter 7 we present our contribution to the study of the CoAP observe option.

In Chapter 8 we conclude this thesis and give possible future developments.

2. Protocols

In this chapter, we review the protocols that constitute the basis of the research presented in thesis. The purpose is to familiarize the reader with the key concepts behind these protocols by presenting their main properties and characteristics.

We start reviewing the IEEE 802.15.4 protocol, which is the de facto standard for MAC and physical layers of WSNs. Networks adopting this standard are defined as Low Power Wireless Personal Area Network (LoWPAN). These networks are intended to be low-cost wireless networks with limited power and low throughput. Then, we present the 6LoWPAN protocol. As mentioned, 6LoWPAN enables the transmission of IPv6 packets in networks adopting the IEEE 802.15.4 standard. It constitutes, therefore, the main ground on top of which realize the IoT vision in WSNs. We describe the main properties of 6LoWPAN focusing on the RO and MU forwarding techniques and the fragmentation mechanism, which are relevant for this thesis. We conclude this chapter reviewing the CoAP protocol. As mentioned, CoAP seeks to implement the key features of HTTP while adding its own mechanisms to best adapt to WSN characteristics. Among them, CoAP defines CoAP blockwise transfer and the observe option, which are presented along with the main characteristics of CoAP.

2.1. IEEE 802.15.4

The IEEE 802.15.4 standard defines the protocol and interconnection of devices via radio communication in a personal area network (PAN) [8]. It uses the OSI reference model and defines the physical layer (PHY) and the medium access control (MAC) sub-layer of the data link layer.

LoWPANs are usually battery powered and have low data-rate. The definition of the 802.15.4, therefore, reflects these constraints. The main characteristics of a LoWPAN are summarized as follows:

- Data rates of 250 kb/s, 100 kb/s, 40 kb/s and 20 kb/s
- Allocation of IEEE 16-bit short or IEEE 64-bit extended addresses
- Use of CSMA/CA
- Fully acknowledged protocol for transfer reliability
- Low power consumption
- Energy detection
- Link quality indication
- 16 channels in the 2450 MHz band, 30 channel in the 915 MHz band and 3 in the 868 MHz band.

The devices used in LoWPANs are distinguished into reduced function devices (RFDs) and fully function devices (FFDs). An FFD can be used as PAN coordinator, coordinator or as device. The use of RFDs is recommended for simple applications where sending large amount of data is not necessary. An RFD can only communicate with a FFD and can be associated only to a single FFD at a time. FFDs do not have these restrictions.

Addresses are assigned in the association phase. It can be allocated up to 264 addresses if they are IEEE 64-bit extended addresses, while 216 if they are IEEE 16-bit short addresses. The communication between single nodes is limited to a personal operating space (POS) that, at the maximum power, is fixed to 10 meters. Each PAN has associated a unique identifier (PAN-ID) that allows to devices to communicate within its PAN using the short address format.

IEEE 802.15.4 networks are self-healing and self-organizing. The former mean that devices are able to detect and recover from errors appearing in either devices or in communication links. The latter facilitate a node to detect the presence of other nodes and to organize them into a structured PAN.

A LoWPAN can be formed following two different topologies, which are star or peer-to-peer. The choice depends from the network application field. A peer-to-peer topology is used to implement complex networks where devices can communicate between each other using multi-hop routing. Conversely, a star topology allows only one-hop communications between the PAN coordinator and the device. Figure 1 shows the star and peer-to-peer topologies.

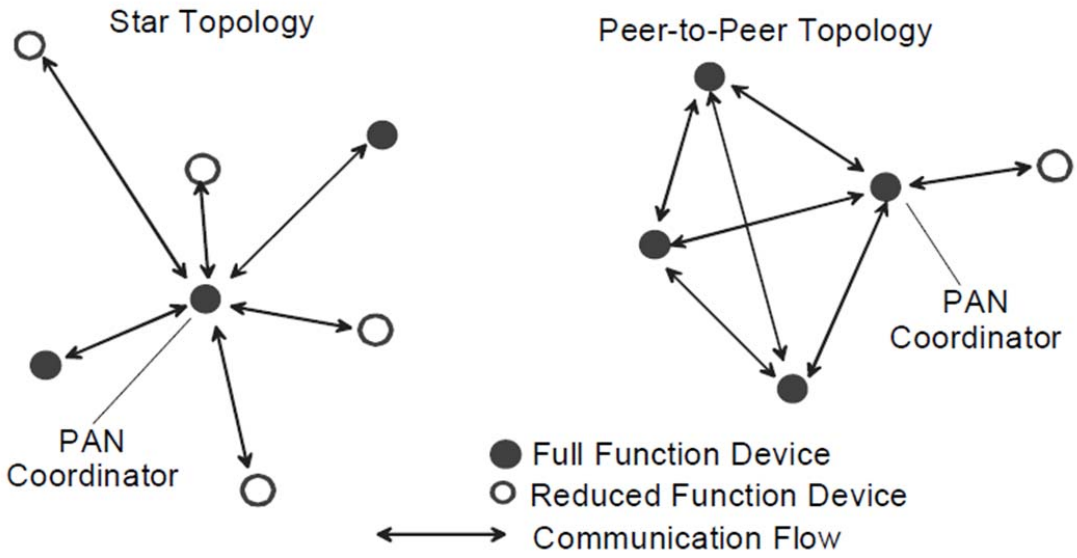


Figure 1 Topologies of 6LoWPAN

2.1.1. Frame Structure

The IEEE 802.15.4 standard defines four possible frames:

- A beacon frame, which is used by a coordinator to transmit beacons
- A data frame, which is used for all transfers of data
- An acknowledgment frame, which is used for confirming successful frame reception
- A MAC command frame, which is used for handling all MAC peer entity control transfers

The data frame contains the payload and headers generated by upper layers, which are referred to as MAC payload. This is prefixed with a MAC header (MHR) and appended by a MFR composed by a 16-bit frame check sequence (FCS) field. The result of this process is the MAC Protocol Data Unit (MPDU). Afterwards,

The MPDU is passed to the physical layer as physical service data unit (PSDU). Then, a synchronization header (SHR) and a physical header (PHR) are attached to the PSDU in order to form the physical protocol data unit (PPDU). This represents the data frame ready to be transmitted over the 802.15.4 link.

As mentioned, the standard fixes the maximum packet size to 127 bytes. The MAC and PHY header generate an overhead of 25 bytes leaving 102 bytes as maximum length of MAC payload. The presence of security mechanisms further reduces the available space to 87 bytes. Figure 2 shows the MAC command frame encapsulated in the PHY packet.

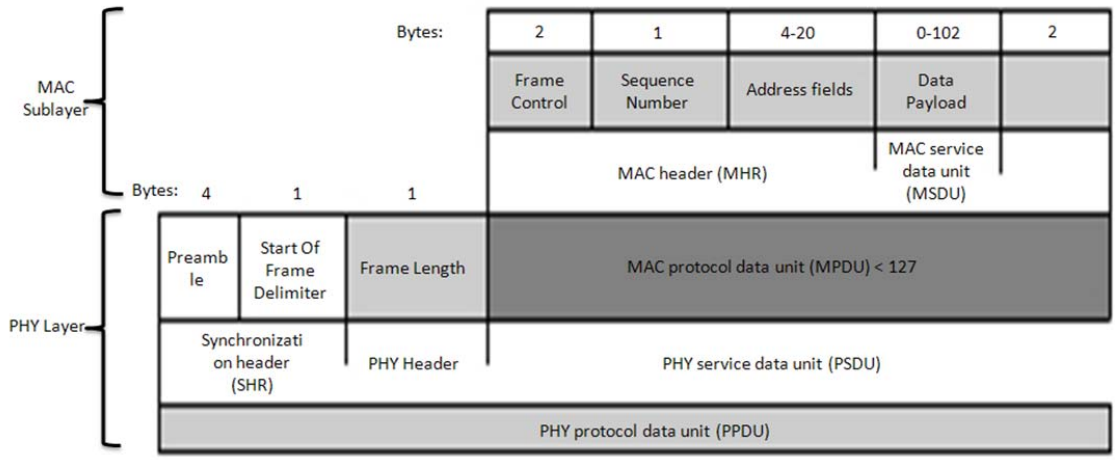


Figure 2 MAC command frame and PHY packet

2.1.2. Data Transmission and Channel Access Mechanism

Data transfer is defined in three modalities: coordinator to device, device to coordinator or between peer devices. Communications between peer devices is available only in peer-to-peer networks. Data transfer to and from the coordinator differs depending if the PAN is beacon-enabled or non-beacon-enabled. Beacons are used in networks requiring synchronization and support for low-latency devices.

Regarding the channel access mechanisms, the IEEE 802.15.4 standard defines the use of unslotted CSMA/CA in non beacon-enabled networks and slotted CSMA/CA in beacon-enabled networks. In this thesis, we focus on non beacon-enabled networks using unslotted CSMA/CA. This MAC modality is of major interest in the standardization of IETF protocols [7]. Next we review the unslotted CSMA/CA mechanism.

The unslotted CSMA/CA mechanism is located at MAC layer. Its process is regulated by three variables, which are the number of backoffs *NB*, the backoff exponent *BE,* and the retransmissions counter *RT*. When a node wishes to transmit a packet, *BE* is settled to *macMinBE* while *NB* and *RT* are initialized to zero. The channel access is divided in two steps, a backoff period and the Clear Channel Assessment (CCA). In the backoff period, the MAC layer delays for a random number of *aUnitBackoffPeriod* units in the range $[0, W_k] = [0\,, 2^{BE} - 1]$ where W_k is the boundary of the backoff window and k the index representing the backoff stage. At the end of this period, the node performs the CCA. This is used to sense if the channel is busy or idle. During CCA, the node is in listening mode. It takes *aTurnaroundTime* units to

switch to transmitting mode. Should the CCA fails, the values of NB and BE are incremented by one and up to a maximum value of *macMaxCSMABackoffs* and *macMaxBE*. Should *BE* reach *macMaxBE*, it remains at this value until it is resettled. Instead, if *NB* exceeds *macMaxCSMABackoffs* the transmission is aborted and packet is discarded due to channel access failure. Otherwise, the CSMA/CA algorithm generates a random number of backoff periods and repeats the process. Should CCA be idle, the node starts the transmission of the packet. After it is completed, the node waits for the MAC ACK. The reception of the MAC ACK is interpreted as a successful transmission. Should the node fail to receive the MAC ACK due to collision or MAC ACK timeout, the variable *RT* is increased by one up to *macMaxFrameRetries*. If *RT* is less than this values, the MAC layer initializes *BE* to its default value of macMinBE and repeats the CSMA/CA mechanism. The packet is discarded due to the retry limit when *RT* reaches its maximum value.

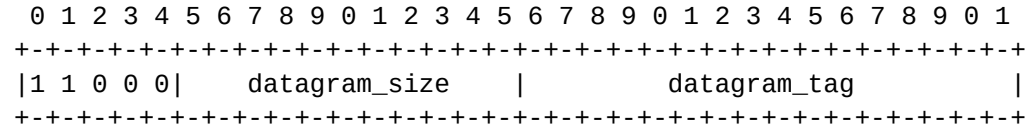
In the rest of the thesis, we denote by $m_0 = macMinBE$, $m_B = macMaxBE$, $m = macMaxCSMABackoffs$ and $n = macMaxFrameRetries$

2.2. 6LoWPAN

6LoWPAN introduces the adaptation layer between network and data link layers. This allows to IPv6 datagrams to meet the requirements of the IEEE 802.15.4. The IPv6 standard, in fact, defines an MTU fixed to 1280 bytes [82]. As mentioned, the MTU defined by IEEE 802.15.4 is equal to 127 bytes. The length of the IPv6 header (40-bytes) implies a huge overhead that, considering the presence of transport layer header (8 bytes for UDP), MAC header (25 bytes) and link-layer security (21 bytes) would leave only 33 bytes available for application layer payload.

The adaptation layer solves these problems enabling the compression of the IPv6 header and the fragmentation of packets that exceed the MTU of the MAC layer. In case of fragmentation, a fragmentation header is appended to each fragment. Two distinct headers are used to indicate whether it corresponds to the first fragment or is one of the followings. Figure 3 shows the fragmentation header for the first fragment and the subsequent fragments.

(a) First fragment.



(b) Subsequent fragment.

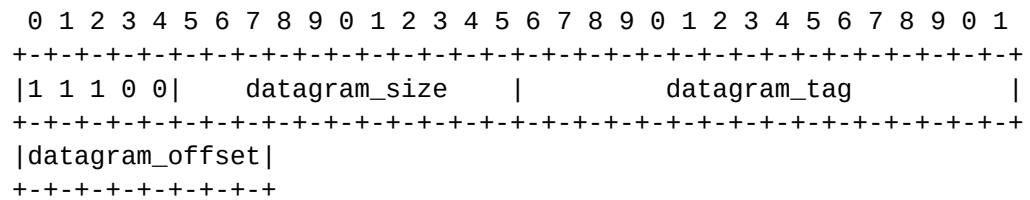


Figure 3 6LoWPAN Fragment headers. (a) First fragment; (b) Subsequent fragment

With reference to Figure3 (a) and (b), the first 4 bit of both headers indicate the dispatch values that the adaptation header checks to identify what kind of fragment it is dealing with. The datagram_size field uses 11 bits to encode the size of the entire IP packet before fragmentation. The value of this field must be the same for all the fragments composing the IP packet. The 16-bit length datagram_tag field identifies that a sequence of fragments is part of the same IP packet. The 8-bit field datagram_offset is defined only for subsequent fragments. It specifies the offset, in module of 8 bits, of the fragment from the beginning of the payload datagram.

As anticipated, to reduce overhead the adaptation layer encodes the IPv6 and UDP headers following the header compression technique specified in [83]. The unique local, global, and multicast IPv6 addresses are encoded through a state-full compression based on shared state within contexts. This compression technique also allows encoding the hop limit value of the IPv6 header. The resulting 6LoWPAN header can be 2 or 3 octets long and is followed by no encoded or partially encoded fields of the IPv6 header. Figure 4 shows the encoding format of this header.

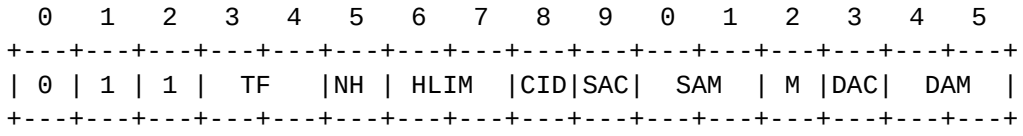


Figure 4 format of the IPv6 compressed header

Besides the fragmentation and the compression headers, the 6LoWPAN standard defines other two headers: the mesh and the broadcast headers. The presence of the mesh header indicates that the packet has to be routed using MU routing. The broadcast header is present to support multicast/broadcast routing. Regarding the mesh header, the first 2 bits, set to 1 and 0, respectively, specify the mesh header dispatch value; V and F bits indicate the length of the originator and final addresses. If they have the value of 0, the addresses are IEEE extended 64-bit addresses; if the value is 1, they are short 16-bit addresses. Originator and final addresses are the address of the node starting the communication and its destination, respectively. The remaining 4 bits of the first octet indicates the number of hops. It can be defined up to 14 hops. An extra octet can be added to define a number of hops greater than 14 by setting all the 4-bit of hop left to 1. Figure 5 shows the mesh header as defined in [7].

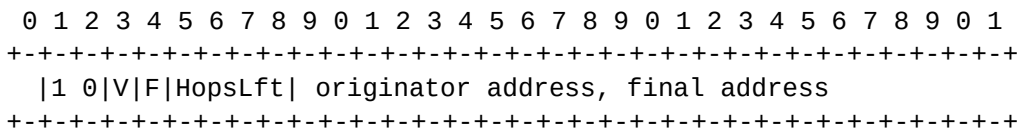


Figure 5 6LoWPAN mesh header format

When multiple headers are used simultaneously, the order in which they appear is the following: mesh header, broadcast header, fragmentation header and compression header. With the presence of the mesh header the adaptation layer can be involved in forwarding decisions instead of the network layer. As

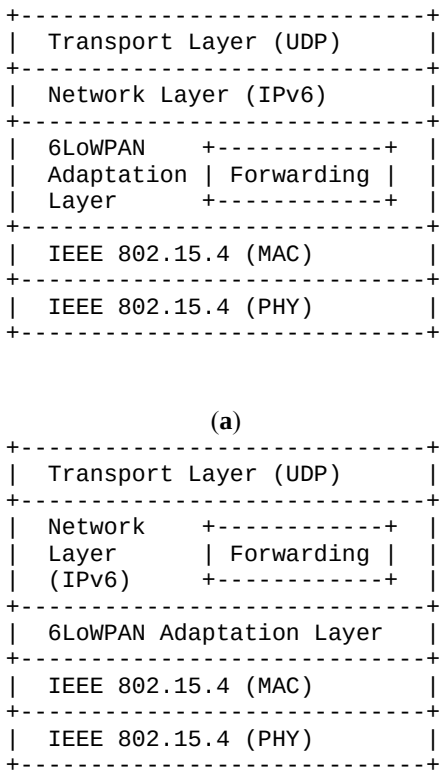
mentioned, the forwarding techniques of 6LoWPAN are a main part of this thesis. They are presented in detail in the next section.

2.2.1. Forwarding Techniques

As anticipated, 6LoWPAN divides forwarding techniques into MU and RO. As shown in Figure 6, the distinction is based on which layer of the 6LoWPAN protocol stack is in charge of forwarding decisions; in RO they are taken at the network layer, and in MU at the adaptation layer. The main difference between these two schemes depends on how packets or fragments are processed before being forwarded.

In this section, we focus our attention on these different approaches. Our purpose is to study these forwarding strategies when dealing with fragmented and non-fragmented packets.

Figure 6 6LoWPAN protocol stack. The network layer is responsible for forwarding decision in RO while for MU it is the adaptation layer. A) MU B) RO



(**a**)

(**b**)

2.2.1.1. Mesh Under

In MU, packet forwarding is transparent to fragmentation. The adaptation layer treats each incoming packet or fragment in the same way. There is no control of the 6LoWPAN fragmentation headers. To forward

a packet or a fragment, the adaptation layer combines the information contained in the mesh header (Figure 5) with the source and destination addressees carried in the IEEE 802.15.4 header. In this way, the IPv6 header does not need to be unpacked. As anticipated, when sending packets or fragments the adaptation layer adds a mesh header to the 6LoWPAN frame indicating that it should be handled with MU.

Should the received frame be recognized as mesh frame, the MU routine gets the information contained in the mesh header that is, the source and destination address and the hop limit. Should the received frame need to be forwarded, the mesh header information and the destination address contained in the IEEE 802.15.4 header are passed to the MU forwarding routine that, return to the MU routine the IEEE 802.15.4 address of the next hop. Once the MU updates the hop limit field, the frame is ready to be forwarded to the next hop. All the forwarding process is done without ever leaving the adaptation layer.

2.2.1.2. Route Over

Since the 6LoWPAN frames are forwarded at network layer, it is necessary that the adaptation layer processes the received frames in order to recreate the original packet. This operation occurs at each hop [39].

Should the received frames not be part of a fragmented IPv6 packet, they only need to be passed to network layer and then processed by the routine responsible for unpacking the compressed IPv6 header. Should the packet need to be forwarded, the routine responsible for RO forwarding looks at the routing table in order to choose the next-hop. The packet then goes back to the adaptation layer, which compresses the IP header again and sends it.

As previously anticipated, should the received frames be part of the same fragmented packet, the adaptation layer reassembles them in order to reconstruct the original packet. Hence, all the incoming fragments are stored in a proper buffer and the reconstruction process starts only when the last fragment arrives. Once reconstructed, the original IP packet is passed to network layer. If the packet has to be forwarded, the forwarding routine processes and sends it back to the adaptation layer. Finally, the IP packet is fragmented again and its fragments are sent to the next-hop. These operations are performed in each node the packet goes through before reaching its destination.

2.3. CoAP

As mentioned, CoAP is a new application protocol for constrained networks and nodes. CoAP is designed to have low complexity and obtained performances adjusted to the limited capabilities of WSNs nodes. The definition of CoAP follows the requirements of the REST architecture.

CoAP uses a request/response model to exchange data between a client and a server node. However, the request/response interaction is conceptually separated from the asynchronous exchange of messages based on the UDP protocol. From a theoretical perspective, CoAP can be seen as a two-layer protocol whit a message layer used to deal with UDP and the other layer used for request/response interactions [19]. This allows that

the request/response interactions are transparent to the message exchanged between client and server. Figure 7 shows the conceptual layering of CoAP.

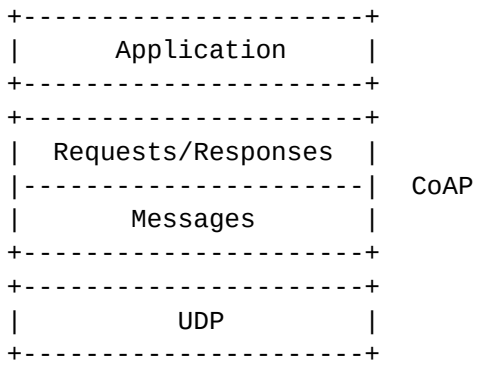


Figure 7 Layering of CoAP

CoAP requests and responses are carried in CoAP messages. Four types of messages are defined: confirmable (CON), non-confirmable (NON), acknowledgment (ACK) and reset (RST).

Reliability is provided using CON messages with stop-and-wait retransmissions of requests [19]. A server receiving a CON request has to acknowledge it to the client that initiated the communication. The server might send an empty ACK to indicate that the response would be deferred. In this case, the client will acknowledge the arrival of the response message. Should the response be immediate, the ACK sent by the server contains the response and the transaction ends with its reception. After sending a CON request message, the client starts a timeout with exponential backoff in order to retransmit periodically the request in case it has not been acknowledged. Finally, a server might send a RST response to indicate that it is not able to process the CON request. NON messages are used when reliability is not required.

A CoAP server can detect if a message is duplicated. This is possible since messages are labelled with a random identification number (message ID). A token value is used to match requests and responses. This is generated by the client and inserted in the CoAP header. The server includes in its response the same token.

The methods that can be used in a request are limited to GET, POST, PUT and DELETE methods of HTTP. Regarding the response codes, CoAP defines its own codes and uses also a small subset on those of HTTP. CoAP Responses could be cacheable. The use of a cache memory allows to reduce latency and bandwidth usage and to improve the interaction with sleepy nodes. Cache can be located in a CoAP proxy server. This is also in charge of mapping the CoAP header to the HTTP one and vice versa.

A header, options and payload compose the CoAP message. The header has a length fixed to four bytes while the options could have a variable length. The payload is prefixed by a payload marker, which indicates the end of the options and the start of the payload. Figure 8 shows the CoAP message format.

The fields in the header are defined as follows:

- Version (Ver): Indicates the CoAP version number.
- Type (T): Indicates if the message type is Confirmable (0), Non-Confirmable (1), Acknowledgement (2) or Reset (3).

- Token Length: Indicates the length of the token.
- Option Count (OC): Indicates the number of options after the header. If set to 0, there are no options and the payload immediately follows the header.
- Code: Indicates if the message carries a request or a response or is empty. 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.
- Token: This field follows the CoAP header and allows matching requests with responses.

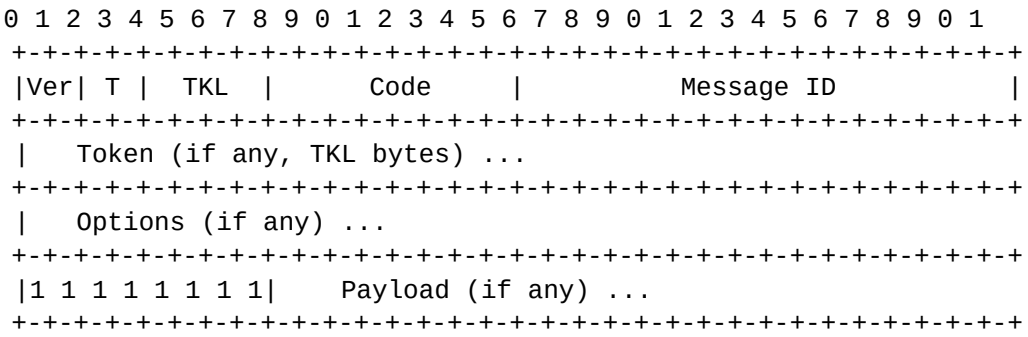


Figure 8 Message format

CoAP is particularly suitable for applications based on machine-to-machine communications. As an example, CoAP could be applied in smart energy applications to provide end-to-end connectivity to energy providers and the equipment of consumers. Figure 9 shows the architecture of a CoAP-based WSN. According to [19], a CoAP node can have the function of both client and server simultaneously. The term endpoint is used to refer to a CoAP node. However, in this thesis we will refer to it as a server or client. This allows us to give greater emphasis to the function that the CoAP node has in our experiments and thus avoid confusion. A CoAP proxy can be used to enable communication between a CoAP based WSN and a HTTP external network. This proxy can work also as a gateway for connecting to other WSNs. As mentioned, the implementations of the proxy as well as that of CoAP are two main topics of this thesis. We present in section 5 and 6 their design and the results of their performance evaluation.

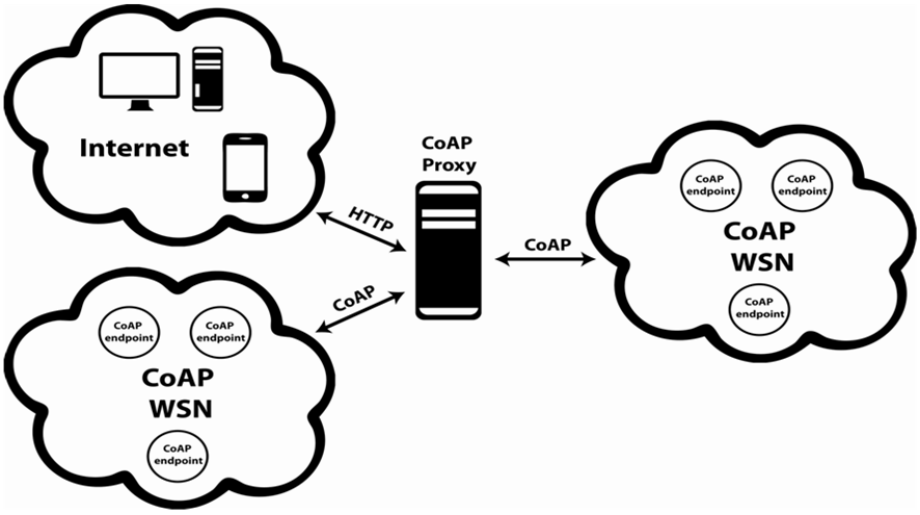


Figure 9 Architecture of a CoAP-based Wireless Sensor Network (WSN). The proxy enables integration between the WSN and external networks that use HTTP.

2.3.1. Observe Option

The observe option enables asynchronous data transfer of CoAP messages. Although its communication paradigm is similar to that of the publish/subscribe model, the observer implies a simpler architecture and a less complex data transaction. The observe transfer model is formed by two components: the observer and the subject. The observer is a client interested in being notified by changes of the state of a resource while the subject is the server that provides it. The conventional architecture of publish/subscribe systems for WSNs is based on the presence of a broker. This is in charge of coordinating and distributing updates and subscriptions requests that it receives from publishers and subscribers. Differently from publish/subscribe systems, the observe model avoids the use of a broker allowing the subject and the observer to communicate directly. Therefore, the resulting architecture is less complex and it is able to reduce latency. The presence of intermediaries, however, is expected to enhance scalability. An observer can register its interest to an intermediary node. This registers itself to the subject and then it forwards to the observer the updates that it will receive. All the process is transparent and it is particularly suitable for multi-hop or large-scale networks where the subject is located many hops away from the observer. Figure 10 shows the architecture of a CoAP based WSN adopting the observe protocol extension.

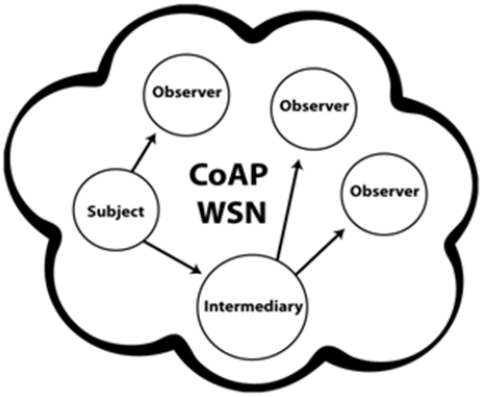


Figure 10 CoAP based WSN implementing the observer extension. An intermediary can be used for scalability purposes.

Registration process and updates

The registration procedure followed by the observer is kept as simple as possible. The observer shows its interest by sending an extended GET request message to the subject. As a consequence, the subject registers it as an observer and then notifies it when the state of the resource changes. To confirm the successful registration, the subject sends an extended response with the current state of the resource. The term extended means that the CoAP request or response contains the observer option as defined in [19]. The value of this option is always zero if it is contained in a request. In a response, instead, it is different from zero and it is used as a sequence number. The token contained in the CoAP header is used to match the updates received by the observer with its original registration. The same token value is used for all the subsequent updates. The subject, however, may decline the observer request by answering with a response not containing the observe option. This indicates to the observer that its request has been rejected.

An observer has the option to be removed from the list of observers of a resource. Should an observer respond to an update with a RST message, the subject removes it from this list. An observer is also cancelled if it sends a simple GET request message to the observed resource. Figure 11 shows an observer registering to a resource, then receiving an update and deleting its interest.

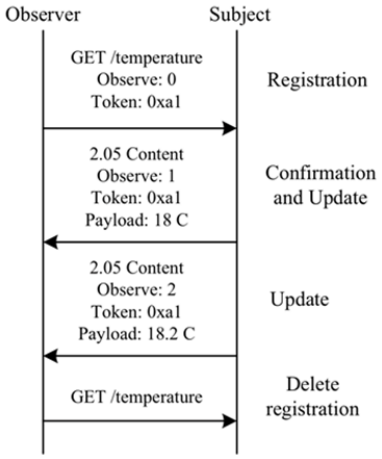


Figure 11 The observer delete its interest sending a GET request without the observe option

2.3.2. CoAP blockwise transfer

As mentioned, CoAP blockwise transfer enables the transmission of large CoAP packets in separated blocks. It is activated including the block option [26] to the CoAP header. CoAP defines two block options, the block_1 and block_2. Their use depends whether the payload is present in a response to a GET request (block_2) or in the POST or PUT request (block_1). In this thesis we focus on data transactions that use the block_1 option.

As illustrated in Figure 12, the block option contains three kind of information: the size of the block (SZX), a flag to indicate if more blocks are following (M) and the sequence number of the block (NUM). The SZX and M fields have fixed size. The NUM field can have three different sizes: 4, 12 or 24 bits.

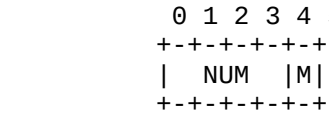


Figure 12 Encoding of the block option

The whole CoAP packet is transferred in multiples request/response transactions. The transmission of a block is consequent to the reception of the relative request. Each request must contain the block option. In this case, the NUM field indicates the number of the block that the client is expecting to receive. The M field is equal to zero while the SZX field is used to indicate the desired size of the block. The size of the blocks, in fact, can be negotiated between the client and server. This negotiation can take place at any point of the block transfer. Figure 13 shows a CoAP blockwise transfer of two blocks with negotiation of their size.

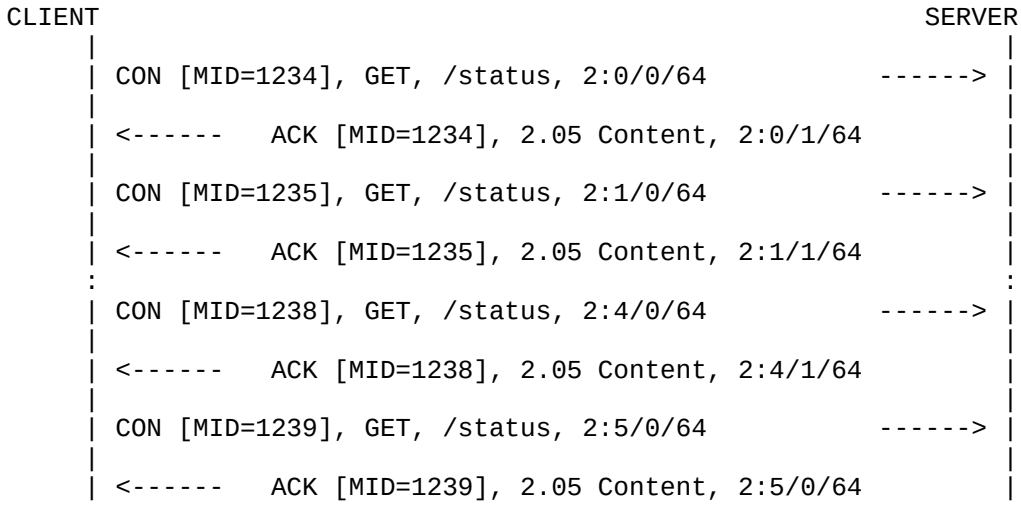


Figure 13 CoAP blockwise transfer with early negotiation of the block size

CoAP blockwise transfer is a reliable communication model. Each block transmission, in fact, relies on the CoAP end-to-end reliability mechanism. The failure of the block or request transmission forces the client to retransmit the request. A request for a subsequent block is transmitted only after that the previous has been

satisfied. Figure 14 shows CoAP blockwise transfer with the client retransmitting a request after the block transmission fails.

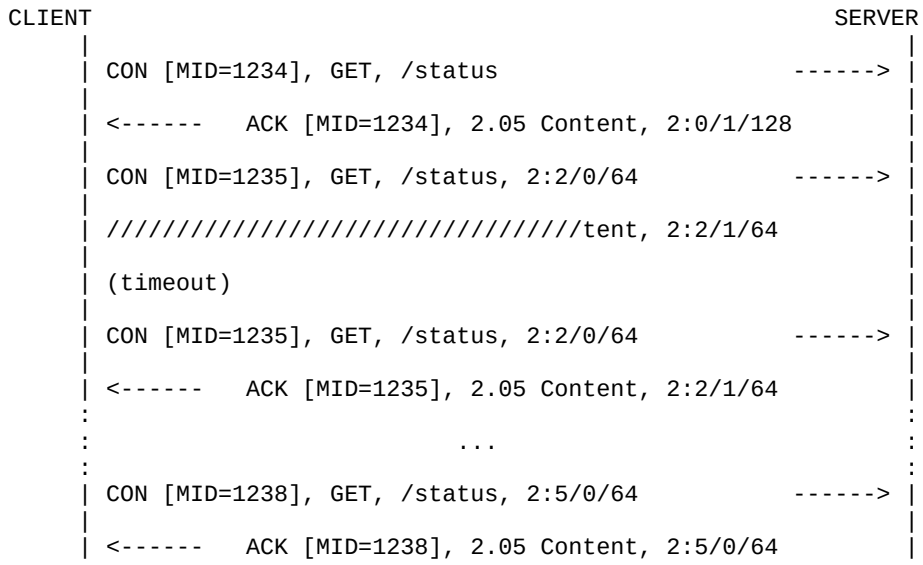


Figure 14 CoAP blockwise transfer GET with late negotiation and lost ACK

CoAP blockwise transfer can be used in combination with the observe option. In this case, the first block of an observe update is sent without the initial request of the client. Then, the server sends the subsequent blocks immediately after the client acknowledges the previous one. Both the initial observe registration request and the relative response can contain the block option. The client, in fact, can request the use of the CoAP blockwise transfer including the block option in the observe request. The server, however, could acknowledge an observe request not containing the block option with an ACK containing it.

The combination of CoAP blockwise transfer and the observe option is of particular interest for the purpose of this thesis. The analytical model that is presented in chapter 4, in fact, seeks to evaluate and compare the performance of CoAP blockwise transfer and 6LoWPAN fragmentation in data transactions that use the observe option.

3. Efficient 6LoWPAN Forwarding

In this chapter, we review and discuss the results of our research on 6LoWPAN forwarding. As mentioned, the optimization of 6LoWPAN forwarding techniques is of paramount importance to lower the energy consumption as well as to improve the reliability and the overall performance of multi-hop WSNs. In particular, communications that require 6LoWPAN fragmentation can have a significant enhancement of their performance with optimized packet forwarding.

We start the chapter analysing first the performance of MU and RO in 6LoWPAN communications not requiring packet fragmentation. MU and RO have, in fact, different operating principles and effects that vary with the presence or absence of fragmentation. Split their analysis allows gaining a better insight into their functioning in both kind of communication.

The reference network scenario for both analyses is a multi-hop WSN. The evaluation of communications with un-fragmented 6LoWPAN packets is done in terms of end-to-end delay according to different payload sizes and number of hops. In the analysis of 6LoWPAN fragmentation we also include round-trip-time (RTT), packet loss and energy consumption as performance metric. These metrics would not allow appreciating the different performance of MU and RO in un-fragmented packet forwarding. The evaluation of packet loss and energy consumption is significant only in presence of fragmentation.

Beside the analysis of MU and RO we propose a novel forwarding technique able to improve the performance of MU with fragmented packets. We observed, in fact, that MU is particularly affected by a high number of retransmissions and a consequent growth of the packet loss percentage. We found the main cause in the absence of control on the fragment forwarding process. Actually, MU is not able to distinguish if the frames to be forwarded are part of a 6LoWPAN fragmented packet or not. Consequently, if a fragment is dropped, then the subsequent fragments are forwarded, although it is not possible to reconstruct the packet. This results in a waste of bandwidth. To overcome this problem, we propose a new approach to MU that enables the forwarding process to be controlled by monitoring the fragmentation header (Figure 3). We refer to this approach as controlled mesh under (CMU).

In our analysis we also consider the ERO technique, which we introduced in section 1.3.1. As mentioned, this proposal seeks to avoid the hop-by-hop fragments reassembling by establishing a virtual circuit between the source and the destination nodes of the fragmented packet.

3.1. Test-bed Implementation

The analysis of fragmented and un-fragmented packets forwarding shares the same test-bed. In this section we present its implementation and give details over the end-to-end delay evaluation, which has the same definitions in both analyses. The review of the test-bed implementation includes the software and hardware platforms used in the performance evaluations. These also correspond to those used in the rest of this thesis.

The Crossbow's TelosB mote is the hardware platform we used four our experiments [84]. It is an open source, low-power wireless sensor module. TelosB motes have a 16-bit RISC MCU at 8 MHz and 16 registers. The platform offers 10 kB of RAM, 48kB of flash memory and 16 kB of EEPROM. Requiring at least 1.8 V, it draws 1.8 mA in the active mode and 5.1 μA in the sleep mode. The MCU has an internal voltage reference and a temperature sensor. Further sensors available on the platform are a visible light sensor (Hamamatsu S1087), a visible to IR light sensor (Hamamatsu S1087-01) and a combined humidity and temperature sensor (Sensirion SHT11). TelosB motes can be plugged via a USB port to a computer through which the motes can be programmed.

As a software solution, we use an open-source TinyOS based 6LoWPAN implementation developed by the University of California at Berkeley called Blip [85]. Blip implements a multi-path routing algorithm. Consequently, each node maintains multiple next-hop entries for any given path. Different fragments of the same packet may take different paths through the network. In this way, the routing algorithm may influence results. Since our aim is to evaluate the forwarding strategies only, we should prevent the collected results from being altered from the routing algorithm performances. We solve this problem by using static routes in which each node has two default next-hop entries selected, depending on the destination address of the mesh header or the IPv6 header. Moreover, it should be pointed out that only MU and RO would work with multi-path routing algorithms. Both CMU and ERO must use the same path to forward all the fragments. In fact, if a fragment could be forwarded to multiples next hops, then we could not create either any state associated with forwarding or ensure the in-order delivery of fragments. However, creating a state in the source node would allow the use of these alternative solutions, although it would not be possible to use any multi-path forwarding.

Blip implements the ERO routing scheme. Although Blip supports MU, only the functions to interact with the mesh header are implemented. We develop the appropriate code and modify some of the existing to use MU, CMU and RO in Blip.

Figure 15 shows the network topology for a two-hop scenario. The base station acts as a border router and bridge between the serial and radio link. It is plugged via a USB port to a computer running a Linux OS. Forwarding of packets is executed in the relay node. The output power of the nodes is fixed to −25 dBm, while distances between sensors are equal to 20 cm. The chosen values of power and distance prove to be sufficient to create a multi-hop network.

In the end-to-end delay tests, the base station is the destination of the fragments, while the source is located in the sensor node. We consider various scenarios in which these nodes are at a distance ranging from two to four hops. The topology of the network used as test-bed reflects possible applications of a WSN requiring a small number of nodes. Possible applications can be found in the healthcare domain, where sensors monitor the medical parameter of a patient and report the results to a base station. Another example can be found in the sports domain, where the WSN could be used to monitor the performance or the training of an athlete. Nevertheless, this limited topology is sufficient to test the forwarding strategies of the routing techniques discussed in this thesis.

The end-to-end delay time tests seek to emulate a possible application scenario where a sensor is in charge of monitoring a certain environment variable and periodically reports the collected values. End-to-delay results do not take into account the time the base station requires to process the incoming packets.

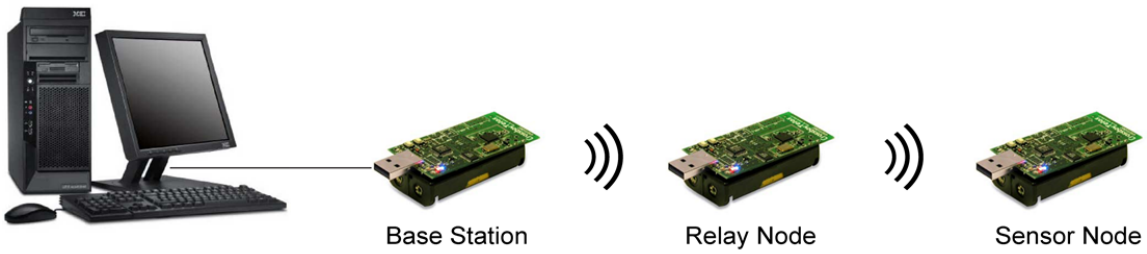


Figure 15 Topology for a two-hop network

3.2. Analysis of mesh under and route over with un-fragmented 6LoWPAN packets

In this section, we present the results of the performance evaluation of MU and RO for un-fragmented packet forwarding. We consider two different evaluations. First, we evaluate the end-to-end delay according to the payload size with a constant number of hops. Then, we evaluate the delay evolution according to the number of hops while keeping the payload to a constant value.

3.2.1. Results and Discussion

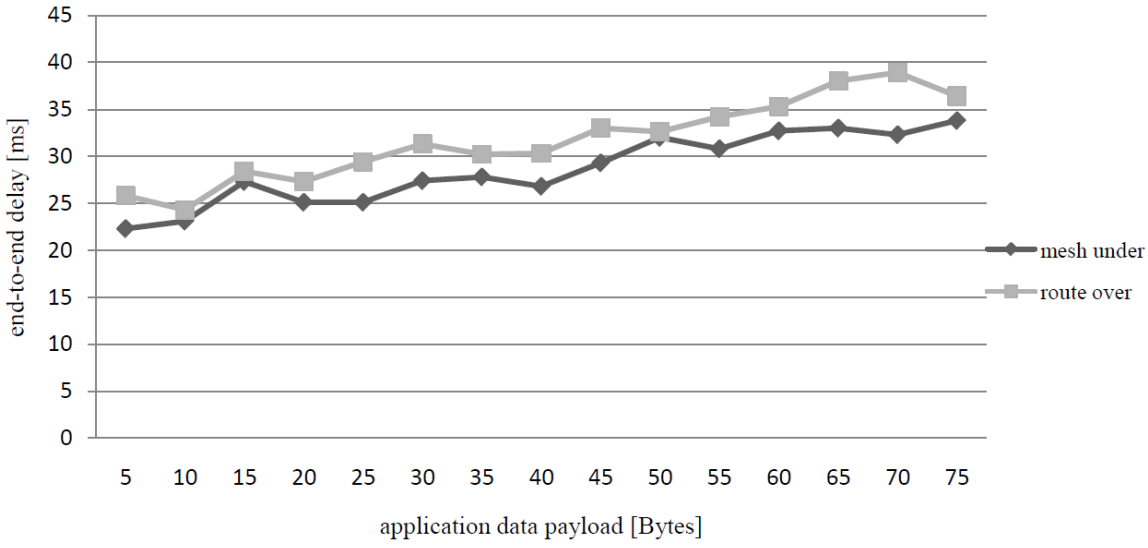


Figure 16 end-to-end delay variation according to application data payload

Figure 16 shows the end-to-end delay time for a 6LoWPAN communication in a WSN with 4 hops and a payload ranging from 5 to the maximum allowed to avoid fragmentation, which is 75 bytes. The end-to-end

delay is calculated as the sum of the nodes processing and propagation times. The resulting end-to-end delay is the mean value of the delay of 10 measures.

MU has a lower end-to-end delay. Respect to RO, MU has an average improvement of 3,1 ms with a peak of 6,6 ms for 70 bytes and a minimum of 0,6 ms for 50 bytes of application data payload. It has, in fact, a less expensive processing respect to that of RO. In this sense, MU avoids the hop-by-hop decompression and compression of the IPv6 header. As mentioned, RO is located at network layer and, therefore, it uses the information contained in the IPv6 header to forward the packet. The transmission time, however, has the same value in both forwarding techniques.

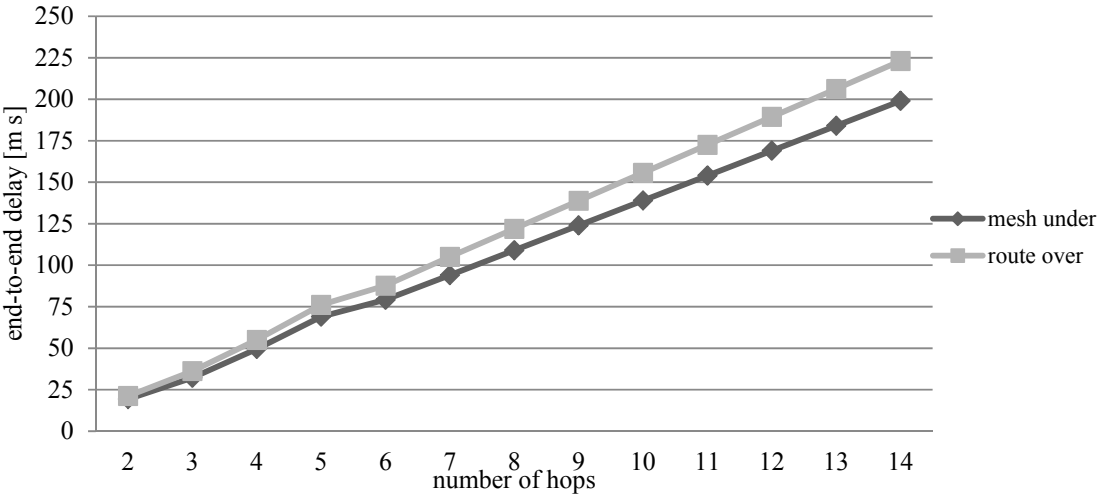


Figure 17 end-to-end variation according to the number of hops

Figure 17 shows the average end-to-end delay evolution according to the number of hops between source and destination. In this evaluation, the payload size is fixed to 75 bytes. The average values from 2 to 5 hops are obtained as in the case of variable payload size. Instead, the average end-to-end delay from 6 to 14 hops is obtained through a simulation of a 6LoWPAN communication. We observed, in fact, that the delay trends for both MU and RO follow a linear evolution. The transmission time, in fact, is constant at each hop while the node processing time is independent from the number of hops. Thereby, we can calculate the mean value of the processing time for each node and express the end-to-end delay as:

$$T = n \times (\mathrm{t_t} + \mathrm{t_p})$$

Where T is the end-to-end delay, n the number of hops, $\mathrm{t_t}$ the transmission time and $\mathrm{t_p}$ the node processing time.

MU outperforms RO also in this case. Furthermore, the difference becomes bigger as the number of hops increases. For a 2 hops networks, the delay of MU is lower of 1,9 ms respect to RO. Considering 14 hops the difference is 24,05 ms. We can estimate, therefore, that the introduction of a new hop augment of 1,84 ms the difference between end-to-end delay of MU and RO.

An interesting analysis focuses in the time spent by a node to process and forward a packet. As previously mentioned MU forwards packets from the adaptation layer and avoids the decompression and compression of the IPv6 packets. The resulting node processing mean time of MU is 11 ms with a standard deviation of 2,1 ms. Instead, in RO it is equal to 12,85 ms with a standard deviation of 2,3 ms. The difference between these times gives an estimation of the time spent by RO in the compression and decompression routines, which is equal to 1,85 ms.

3.3. Analysis of 6LoWPAN forwarding techniques with fragmented packets

In this section, we analyse 6LoWPAN forwarding focusing on communications requiring 6LoWPAN packet fragmentation. The analysis is conducted through a performance evaluation of MU and RO in terms of end-to-end delay, RTT, packet loss and current consumption. Moreover, we present a new forwarding technique based on MU, which seeks to improve the MU fragment processing by adding control on the fragment forwarding process. As mentioned, the analysis of 6LoWPAN fragment forwarding also considers ERO.

3.3.1. Controlled Mesh Under (CMU)

Since in the MU forwarding process there is no control on the frames to be forwarded, unnecessary fragments may be propagated. In fact, if any fragment gets lost before it reaches the destination then the rest of fragments will be forwarded unnecessarily. In this case, the whole packet cannot be reassembled. Even though no fragment is lost, they may still arrive at destination out-of-order. This would complicate the reconstruction process at the destination node. Adding control on the MU forwarding process would reduce the probability of out-of-order delivery of fragments, and the transmission of useless fragments would thereby be avoided. This would result in a better use of the bandwidth and in a simplification of the fragment reassembling, thus allowing low complex and less resource demanding code to be developed.

Our CMU proposal seeks to solve these problems. We propose adding a control to MU forwarding process that allows the fragmentation header of the incoming fragments to be monitored.

The control starts each time a node receives a frame containing both the mesh and the first fragment headers. It begins by storing the information contained in these headers. In more detail, this information is relative to the tag and the size fields of the first fragment header (Figure 3) and the originator address of the mesh header (Figure 5). This information allows us to determine if the subsequent packets are part of the same 6LoWPAN fragmented packet. Should the subsequent fragments belong to the same packet, the CMU verifies if the reception is in-order. This is established by checking the offset field of the fragmentation header. Should the fragment be the one expected, the forwarding routine starts the MU forwarding process. If the received fragment does not match the one expected, then the previous node is asked to retransmit the expected fragment. When the correct fragment has been received, the forwarding process can be resumed.

Should the fragment not be received, the forwarding process is cleared and subsequent fragments will not be forwarded, thereby avoiding bandwidth waste.

3.3.2. Enhanced Route Over (ERO)

As mentioned, the major drawback of RO is the hop-by-hop fragment reassembly. This characteristic can significantly increase latency and the energy required by a node to forward a packet. However, different solutions can be applied to RO in order to solve this problem. As mentioned in section 1.3.1, RO could be implemented with methods able to create virtual reassembly buffers that remember just the IPv6 header contained in the first fragment [38]. Furthermore, in [39] the creation of a state associated to the IPv6 source address and to the datagram tag (Figure 3) is proposed. This information is contained respectively in the IPv6 and fragmentation headers carried by the first fragment. This solution allows a virtual circuit to be established for the subsequent fragments.

In nodes implementing ERO, the adaptation layer checks the fragmentation header of each incoming frame. Should the fragment be recognized as the first, it is sent to the IP layer to unpack the IPv6 header. Should the fragment need to be forwarded, the node gather the information required to create the state associated to forwarding and establishes the virtual circuit. When each subsequent fragment reaches the node, this is forwarded through the virtual circuit without the need to check any routing table. The virtual circuit is deactivated and the state erased from memory after the last fragment has been forwarded.

3.3.3. Results and Discussion

In this section, we compare the different forwarding strategies of MU, RO, CMU and ERO focusing on the obtained performance in terms of end-to-end delay, RTT, packet loss and current consumption.

For RTT tests we consider a two-hop network formed by a base station sending ping requests to a node located at a distance of two-hop. We consider this distance sufficient to give a correct RTT performance evaluation and to appreciate the different effects each forwarding strategy has on it. The same network is used for the packet loss evaluation. In this case we evaluate the packet loss as a function of the payload size. We then evaluate the average end-to-end delay time obtained by transmitting UDP packets according to different payload sizes and network topologies.

As for the current consumption, it corresponds to the current drawn by a node forwarding ping requests and replies in a two-hop network. We are interested in measuring the current drawn by the relay node in receiving, processing and sending fragments. For each different payload size, we run tests sampling the current consumption each 0.02 ms. The values obtained refer to the average current consumed by the relay node from the time a fragment is received up to its transmission to the next-hop. The reported average values have a confidence level of 95%. Current drawn in inactivity states is not taken into account. The device used for these measures is the Agilent Technologies DC power Analyzer N67705A.

3.3.3.1. Round-Trip-Time Evaluation and Packet Loss

Figure 18 shows the RTT performance for RO, MU, CMU and ERO. Each point in the graph represents the average value of 100 ping responses that successfully reached the destination. Payload size ranges from 100 to 1,100 bytes with increments of 50 bytes. The number of fragments goes from 2 for a 100 bytes payload, up to 12 for 1,100 bytes. The reported average values have a confidence level of 95%. Each ping request is sent as soon as the preceding ping reply is received. Results of RTT are strongly influenced by the time the base station spends elaborating and passing the ping response to the OS. We estimate this time to be in the order of 178 ms. Further delay is introduced by the OS to generate the ping request and pass it to the base station.

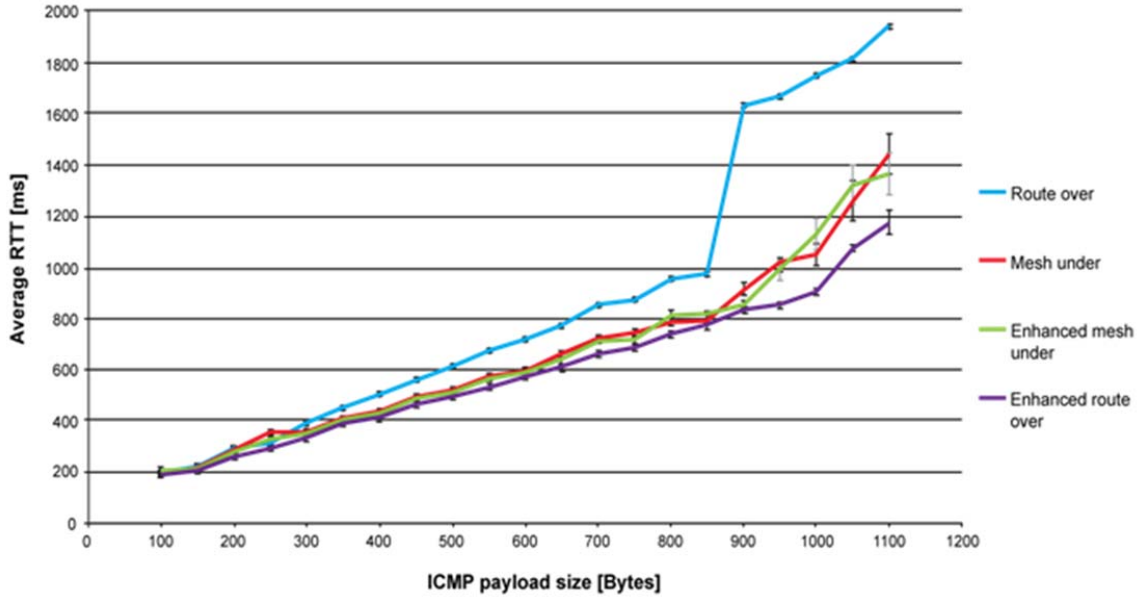


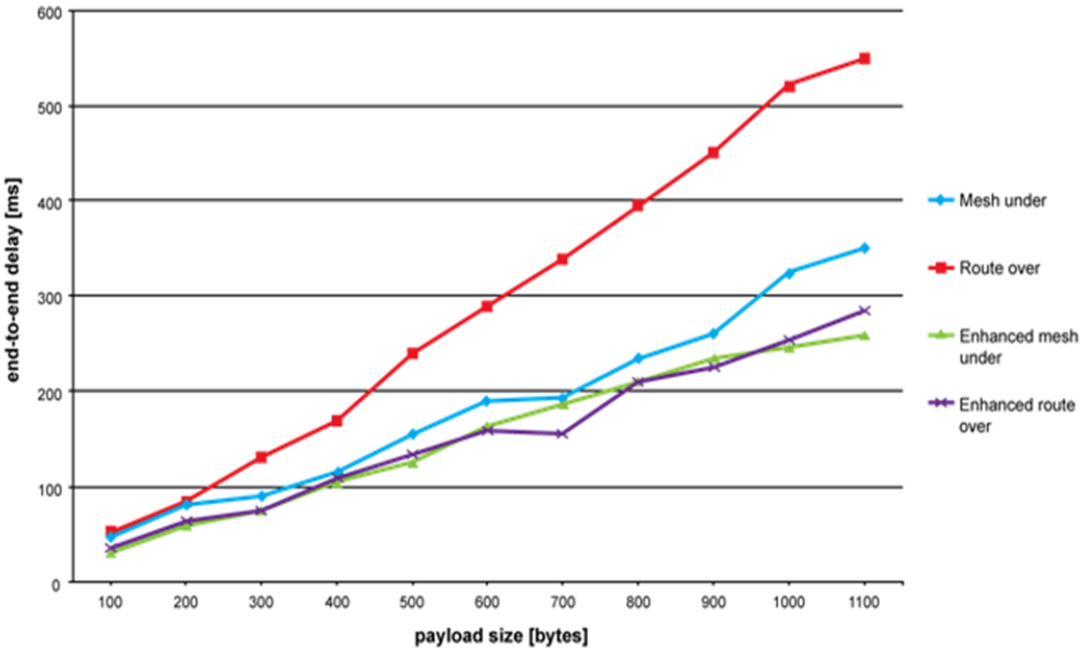
Figure 18 RTT evolution according to ICMP payload size. Buffer congestion affects RO when reaching a payload size of 900 bytes, causing the big jump in the average round-trip delay time.

As expected, RO has the worst performance. Reassembling and fragmenting packets at each hop slows down the communication, especially when the payload size is high and more fragments are involved in the communication. However, this feature of RO reduces the standard deviation, which results in the routing scheme having the lowest deviation. Because of the high number of retransmissions affecting MU, and to a lesser extent CMU and ERO, the standard deviation for both mesh techniques is quite high, while it remains low for ERO.

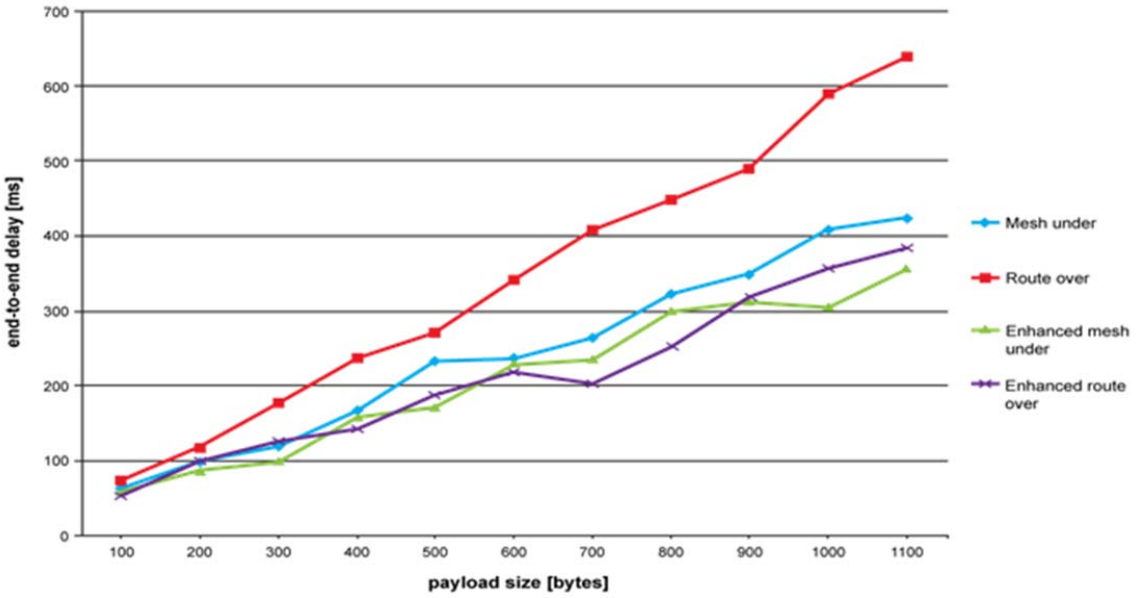
In Figure 18 one may observe that the trend of RTT has almost a linear evolution for each considered solution. However, when approaching the maximum packet size this trend changes quickly. In particular, the RTT performance of RO rapidly gets worse, between 800 and 900 bytes. This is explained by the fact that the buffer capacity is reaching its maximum, which causes memory congestion. Moreover, by increasing the RAM usage as the packet size augments [55], Blip leaves a very limited space to perform the packet processing required by RO. We find this behaviour to be a major cause of memory congestion. Furthermore,

we find that memory congestion occurs when the relay node was subjected to an uninterrupted packet flows, such as the one generated by ping requests and response. In fact, spacing the packet transmission, as is done for end-to-end delay evaluation (Figure 19), solved this problem. Regarding MU and CMU, the worsening of RTT is explained by observing that the number of retransmitted fragments is high in comparison with the other techniques. ERO has the best RTT performance. However, for payload size lower than 900 bytes, the performance is very similar with that obtained in CMU, while improving it for higher payload size.

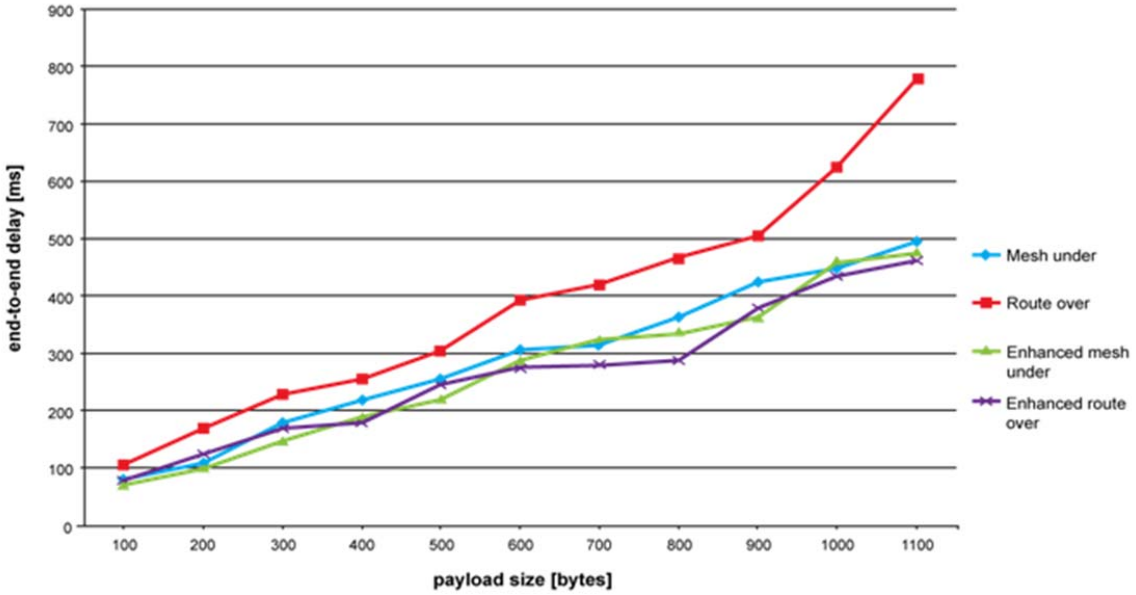
Figure 19 End-to-end delay time evolution. The number of retransmissions is lower in CMU than in MU, resulting in a better end-to-end delay time trend. (a) End-to-end delay time for a two hops network. (b) End-to-end delay time for a three hops network. (c) End-to-end delay time for a four hops network.



(**a**)



(**b**)



(**c**)

The buffer problems affecting RO also influence the packet loss percentage results. As shown in Table 1, from a payload size of 900 bytes to 1,100 bytes, the packet loss percentage obtained in RO becomes of the order of magnitude of the other routing techniques. For lower payload sizes RO proves to be more robust to packet loss, as expected. The performance in packet loss of ERO compared with that obtained by RO shows a worsening of the packet loss. Respect to ERO, CMU has a better packet loss up to 500 bytes, while for higher payload size this loss becomes similar.

Table 1 Packet loss percentage. RO proves to be more robust to packet loss than the other techniques. However, starting from a payload size of 900 bytes, buffer congestion causes a rapid worsening of RO packet loss. Link retransmissions due to collisions are the main cause of packet loss for MU, CMU and ERO.

Payload size [bytes]	RO	MU	CMU	ERO
100	0%	0%	0%	0%
200	0%	0%	0%	1%
300	0%	4%	2%	5%
400	3%	15%	4%	6%
500	3%	21%	10%	13%
600	2%	27%	20%	16%
700	3%	32%	24%	23%
800	3%	37%	28%	29%
900	33%	35%	33%	34%
1,000	49%	42%	35%	31%
1,100	58%	48%	41%	41%

The control on fragment forwarding provided by CMU improves the packet loss performance with respect to MU, which has the worst packet loss percentage. In fact, CMU avoids the propagation of unnecessary fragments and, therefore lower its channel occupancy. In this way, it is subjected to less retransmissions caused by collisions and consequently to a lower packet loss. The major cause of packet loss in MU, CMU and ERO is found in retransmissions caused by collisions. Collisions occur because the relay node is continuously receiving fragments and it has to forward them instantly. In this scenario, the relay node may not detect the reception of a fragment if it is forwarding another one. As a consequence, the node that transmitted the dropped fragment will retransmit it to the relay node. It should be pointed out that the retransmission policy used in Blip drops fragments after a maximum of five retransmissions. On the other hand, collisions do not affect RO, since a node using this technique has to wait until the reception of the last fragment to start reconstructing the packet and begin the forwarding process.

Returning to RTT performance, a main cause of the worst performances of RO is found in the time elapsed between the reception and forwarding of a fragment. Actually, a node implementing RO is forced to wait until the reception of the last fragment before forwarding the first. We estimate that for a payload size of 1,000 bytes, the time elapsed between the reception and the forwarding of the first fragments is in the order of 125 ms. This value corresponds to the time spent by the previous node to set up and send all the fragments composing the original packet. In mesh techniques and in ERO, the forwarding is immediate to the reception of the fragment. It only takes the time to process the fragment. In the analysis of un-fragmented packet forwarding, we estimated this time to be 11 ms with a standard deviation of 2.1 ms. A further delay is due to the compression/decompression of the IPv6 header. However, the order of magnitude of this delay is not comparable with that introduced by packet reconstruction or fragmentation.

MU and CMU performance look very similar. The control process that CMU executes for fragment forwarding does not lessen its performance, or produce any significant enhancement with respect to MU. On

the other hand, ERO shows considerable improvement with respect to RO. As expected, latency decreases significantly, avoiding hop-by-hop fragments reassembling.

3.3.3.2. End-to-End delay Evaluation

Results of end-to-end delay time evaluation are shown in Figures 19(a–c). As explained above, our aim is to emulate an application scenario where a node is sensing a certain environmental variable and periodically reports its value to the base station. This period has been fixed to 5 seconds.

As can be seen in Figure 19, RO confirms its negative trend. CMU, MU and ERO have the lowest end-to-end delay time, with the former having the best performance. Once again, the high number of retransmissions that occurs in MU makes the difference when compared with CMU. Respect to RO, ERO significantly improves the end-to-end delay performance by avoiding hop-by-hop fragment reassembling.

By augmenting the number of hops, we observe that the end-to-delay performance for MU and CMU become similar. This can be appreciated for higher payload sizes. Contrary to our expectations, increasing the number of hops does not augment the difference between RO and MU. In fact, the more hops we have the more retransmissions are required to propagate fragments with a MU technique. This makes the RO trend approach MU, CMU and ERO trends. However, differences become more significant for higher payload sizes. As experienced previously, RO gets worse when the number of fragments composing the packet becomes high and the buffer is approaching its limit.

3.3.3.3. Current Consumption

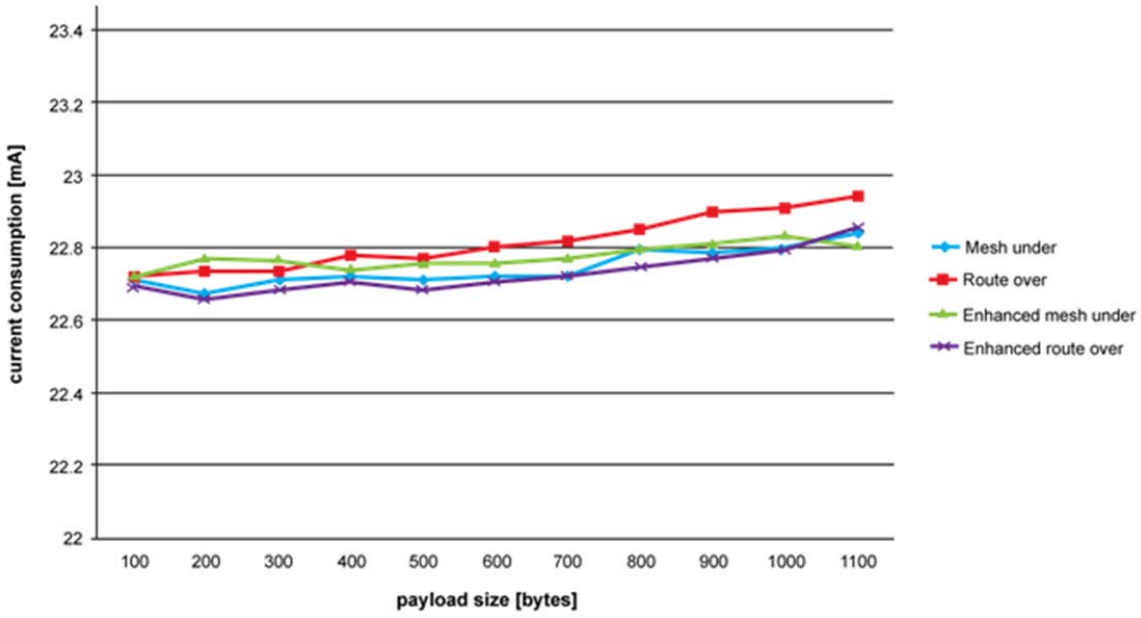


Figure 20 Current consumption evolution according to ICMP payload size. Hop-by-hop fragment reassembling performed by RO proves to be energy demanding. The control on packet forwarding introduced in CMU, slightly increases current consumption compared with MU.

Figure 20 shows the results obtained for current consumption. As expected, a node adopting a RO technique consumes more energy than others. Once again, hop-by-hop fragment reassembling proves to be costly in constrained network environment. This can be appreciated by comparing the RO performance with that of ERO. In fact, ERO simplifies the fragment forwarding and reduces significantly the current drawn by the node. ERO gives the best performance in terms of current consumption. It should be pointed out that since ERO is the default routing technique of Blip; it is optimized in order to work with it. Consequently, ERO is inclined to work better on Blip.

MU has a performance very similar to that of ERO. Although affected by a large number of retransmissions, by avoiding complex fragment processing MU maintains low current consumption. However, in comparison with the less complex MU, the augmented process complexity of CMU does not significantly increase current consumption. Here, the augmented complexity is aimed at the control associated to fragment forwarding. Nevertheless, considering the overall current consumption and taking as an example a network composed of many nodes, better management of the controlled mesh bandwidth would result in energy-saving. In fact, in CMU, forwarding of unnecessary fragments would be avoided and the nodes would be subjected to a lower workload.

Finally, we observed that standard deviation is very low for all the forwarding schemes, and consequently deemed not relevant for the energy consumption results.

3.4. Conclusions and Contributions

The application domain where 6LoWPAN is deployed plays a substantial role in choosing which forwarding solution adopt. The high packet loss experienced in MU does not make it recommendable for use in applications requiring a high degree of reliability. RO is suggested for these critical applications. First, CMU or, secondly, ERO can both ensure reliability, but for a smaller payload range than that of RO. CMU lowers the packet loss with respect to MU by providing a better bandwidth management. Actually, packet loss percentage is quite high for all the considered forwarding solutions when the payload size is large. For these large payloads, we observe a rapid worsening of the 6LoWPAN performance, regardless of the forwarding technique used. In particular, RO experiences memory congestion problems for payload greater than 800 bytes.

Applications with strict latency requirements should implement a CMU or ERO solution. MU has an acceptable latency performance, but lower as compared with CMU in the case of end-to-end delay time. However, applications generating low traffic and small packet size could also implement MU.

MU is preferable to RO in 6LoWPAN communications not requiring fragmentation. It is able to reduce node processing time by forwarding from the adaptation layer. RO, instead, forwards packets at network layer and needs, therefore, the information contained in the compressed IPv6 header. Thereby, it has to decompress the header at each hop and compress it again before forwarding the packet.

Energy consumption is crucial in sensor networks. Sensors, in fact, are usually constrained in power supply, since they are battery powered. Our study demonstrates that forwarding solutions subjected to a

higher workload have a poor behavior in terms of consumed energy. In this sense, although subjected to a lower packet loss and to less fragment retransmission, RO consumes more current than the other routing solutions. Its alternative solution, that is, ERO, lowers current consumption by avoiding hop-by-hop fragment reassembling.

Fragment retransmission is another crucial aspect in energy consumption. It is well known that the peak in energy consumption is located in the transmission and reception states. As a consequence, forwarding solutions characterized by a high retransmission rate spend more time in both states and are inclined to waste more energy than other techniques. MU turned out to be the technique subjected to the highest number of retransmissions. MU compensates energetically for the cost of retransmissions with its energetically efficient fragment processing. As a result, MU shows good performance in terms of energy consumption. As regards CMU, the control added to monitor fragment forwarding requires a slight increase of energy compared with MU. Furthermore, the better usage of the communication channel allowed by CMU lowers the overall network current consumption by avoiding the propagation of useless fragments.

In conclusion, RO proves to be more robust to packet loss, but less energy saving than the other routing schemes. Weaknesses of RO are also found in the high latency experienced in packet transmission. ERO proves to be capable of solving these limitations in latency and energy performance, but it is unable to maintain the packet loss to the same degree as RO. Both mesh techniques show a good performance in terms of latency and energy consumption, with CMU yielding a better result in the end-to-end delay performance. While increasing the complexity of fragment forwarding, CMU does not result in a significant growth of the consumed current. The high packet loss of MU decreases in CMU thanks to better management of the channel bandwidth.

From the results of the research presented in this chapter derives the paper *Forwarding Techniques for IP Fragmented Packets in a Real 6LoWPAN Network* that is published in the journal "Sensors" [P1] and the paper *Implementation and evaluation of Multi-hop routing in 6LoWPAN* which has been presented in the 9th conference of telematics engineering (JITEL) [P5].

4. Analytical model of CoAP large data transactions

In this chapter we present a novel analytical model to study fragmentation methods in WSNs adopting CoAP and the IEEE 802.15.4 standard. The model includes the effects of fragmentation on the contention level at MAC layer and the results are validated through Monte Carlo simulations. CoAP blockwise transfer and 6LoWPAN fragmentation are included in the analysis. Both techniques are compared in terms of reliability and delay. A major contribution is the possibility to understand the behavior of both techniques with different network conditions.

As mentioned, our model considers the use of the unslotted CSMA/CA mechanism defined by the IEEE 802.15.4 standard. This allows evaluating the packet losses caused by collisions and channel access failures. The model presented in [105] is used as reference for the CSMA/CA mechanism. Its application to our study presents several challenges. Although the application layer might generate packets following a Poisson distribution, the subsequent fragmentation or block division implies a bursty transmission at MAC layer. We refer to this traffic condition as 'mixed' traffic. At MAC layer, in fact, only the arrival of the first fragment or block follows a Poisson distribution. The arrival of the remaining fragments or blocks is characterized by saturated traffic condition. Therefore, contrary to the assumption made in [105], the busy channel and collision probabilities vary over the time depending on the backoff and transmission stages. The analysis of this behaviour requires adapting the original CSMA/CA model to the new traffic conditions.

Existing works on the IEEE 802.15.4 MAC protocol have considered both saturated traffic (i.e., when node queues are always non-empty) and unsaturated traffic conditions. Bursty ON-OFF traffic conditions have only been considered in [106]. None of the existing works focus on the presence of mixed traffic conditions. Besides the analytical study of 6LoWPAN fragmentation and CoAP blockwise transfer, a further original contribution is the analysis of the IEEE 802.15.4 MAC protocol under mixed traffic conditions.

Next we discuss the model for traffic generation used in our analysis.

4.1. Traffic generation model

A major contribution of our model, with respect to [105] is the analysis of the bursty traffic conditions along with 6LoWPAN packet fragmentation and CoAP blockwise transfer, as we detail as follows.

We assume that the CoAP layer generates observe updates following a Poisson distribution with rate λ. Each update is divided into F fragments or B blocks. Each fragment or block is included into a MAC frame of length L. The frame containing the CoAP ACK has length L_{ACK}. Both frames are transmitted using the unslotted CSMA/CA mechanism of the IEEE 802.15.4 standard. We consider low traffic generation, i.e., traffic generation rate at node l, $\lambda_l \ll 1/(L * F)$, which is consistent with the minimum RTO of 1 second recommended by CoAP [19]. With higher rates the retransmission mechanism of CoAP could not be used. The RTO, therefore, represents an upper bound to the generation rate.

At MAC layer, the traffic arrival is characterized by a Poisson distribution of parameter λ_l for the first fragment and by bursty traffic for the following F-1 fragments or B-1 blocks. The probability of generating the first fragment or block of an update at node l in a unit time S_b is derived as

$$q_l = 1 - e^{(\lambda_l / S_b)} \quad (1)$$

In the rest of the chapter we consider S_b = aUnitBackoffPeriod as the basic unit time as in [105]. We recall that it corresponds to the transmission time of 20 symbols [8].

The probability of generating a new fragment or block after the previous one has been acknowledged or discarded is 1.

4.2. Analytical model of the CSMA/CA mechanism

In this section, we develop a generalized model of the IEEE 802.15.4 MAC considering the presence of 6LoWPAN fragmentation and CoAP blockwise transfer. The analysis aims at deriving the reliability as the probability of successful frame reception and the delay for successfully received frames. Both are relative to the MAC layer and will be included in performance indicator expressions for the CoAP layer, which are presented in the next section.

The analysis is based on the Markov chain model presented in [105] that accounts for the presence of heterogeneous traffic with different node packet generation rates and hidden terminals.

We first determine the CCA probability τ_l, namely the probability that node *l* performs the carrier sensing procedure in a randomly chosen time unit. For each generated fragment, the CCA probability accounts for the number of times the CCA procedure is repeated due to busy channel and retransmissions, i.e.,

$$\tau_l = \sum_{i=0}^{m} \prod_{j=0}^{i} (\alpha_{l,j}) \sum_{k=0}^{n} \left(\left(1 - \prod_{j=0}^{m} \alpha_{l,j} \right) P_{coll,l} \right)^k b_{0,0,0}^{(l)} \quad (2)$$

where $\alpha_{l,j}$ is the busy channel probability of node *l* during the *j-th* backoff stage, $b_{0,0,0}^{(l)}$ is the idle probability, and $P_{coll,l}$ is the collision probability that we derive next.

For unsaturated traffic conditions, the idle probability is the reciprocal of the frame generation probability at MAC layer, i.e.,

$$b_{0,0,0}{}^{(l)} = 1/(q_l * F) \quad (3)$$

When traffic gets saturated, the idle probability is calculated by applying the normalization condition of the corresponding Markov chain, as detailed in Proposition 4.1 in [105].

The busy channel probability due to packet transmission for the first fragment or block is the probability that no other node accessed the channel and found it idle in the previous *L* time units. After the previous fragment or block has been acknowledged at MAC layer, the node generates a random backoff in the window

$[0 - W_0]$ before sensing the carrier. The busy channel probability for the following fragment, block, or CoAP ACK is given by the probability that no other node accesses the channel during $(W_0 + 1)/2$ time units. In average terms, the channel will be busy if no other nodes accessed and found it idle in the previous L_{eq} time units:

$$L_{eq} = (L + F * (W_0 + 1)/2)/(F + 1) \ for\ 6LoWPAN\ Fragmentation \quad (4)$$

$$L_{eq} = (L + (2\ B - 1) * (W_0 + 1)/2)/(2\ B) \ for\ Blockwise\ transfer \quad (5)$$

We recall that the MAC ACK is transmitted right after the reception of a MAC frame. Its transmission does not undergo the backoff procedure. Instead, this procedure applies for the CoAP ACK. For 6LoWPAN Fragmentation there are F packets and 1 potential CoAP ACK, while for CoAP blockwise transfer there are B blocks and B potential CoAP ACKs.

Should the channel be busy during the first backoff, there is a higher probability that the channel will be still busy after the backoff in the window$[0 - W_1]$. This behavior is due to the bursty traffic generation. Under this condition, the channel will be idle after the second backoff with a probability

$$r_1 = ((W_1 + 1)/2)/(\bar{L} + (W_1 + 1)/2) \quad (6)$$

where

$$\begin{aligned} \bar{L} &= (F * L + L_{ACK})/(F + 1) \quad for\ 6LoWPAN\ fragmentation \\ \bar{L} &= (L + L_{ACK})/2 \qquad for\ blockwise\ transfer \end{aligned} \quad (7)$$

The busy channel probability is then approximated by $\left(1 - r_j\right)$ for $W_j < L * F$ or $W_j < L * B$.

For $W_j > L * F$ or $W_j > L * B$ the busy channel probability $\alpha_{l,j}$ can be calculated in asynchronous fashion, as in the original model in [105].

In conclusion, the busy channel probability is written as

$$\alpha_{l,j} = \begin{cases} L_{eq} \sum_{i=1}^{N-1} \sum_{q=1}^{C_{l,i}} \prod_{k=1}^{i} \tau_{k_q} \left(1 - \prod_{k=1}^{i} \bar{\alpha}_{k_q}\right) \prod_{h=1+1}^{N-1} \left(1 - \tau_{h_q}\right) \ for\ W_j > F\lambda, B\lambda \\ 1 - \frac{(W_j + 1)}{2} \left(\bar{L} + \frac{(W_j + 1)}{2}\right)^{-1} \ otherwise \end{cases} \quad (8)$$

where $C_{l,i} = \binom{l}{i}$ and $\bar{\alpha}_l = \frac{\sum_{j=0}^{m} \alpha_{l,j}}{m+1}$ is the average busy channel probability for all backoff stages.

The collision probability is the probability that a contending node performs the CCA in the same time unit, i.e.,

$$P_{coll,l} = \bar{\alpha}_l / L_{eq} \quad (9)$$

The expressions of the CCA probability, the busy channel probability and the collision probability form a system of non-linear equations that can be solved through numerical methods as specified in [105].

The IEEE 802.15.4 protocol does not distinguish between higher layer packets. Therefore, the probability that the transmission of a block, fragment or CoAP ACK for node *l* fails has the same expression for each of them. We refer to it as $P_{frame,l}$:

$$P_{frame,l} = P_{cf,l} + P_{cr,l} \quad (10)$$

where $P_{cf,l}$ corresponds to the probability for node *l* that the frame is discarded due to channel access failure and $P_{cr,l}$ to the probability for node *l* of a packet to be discarded due to retry limit. Therefore, we have

$$P_{cf,l} = \prod_{j=0}^{m} \alpha_{l,j} \sum_{k=0}^{n} P_{coll,l} \left(1 - \prod_{j=0}^{m} \alpha_{l,j}\right) \quad (11)$$

$$P_{cr,l} = \left(P_{coll,l} \left(1 - \prod_{j=0}^{m} \alpha_{l,j}\right)\right)^{n+1} \quad (12)$$

Once the CCA probability, the busy channel probability, and the collision probability are derived, the delay for successfully received frames $D_{frame,l}$ and CoAP ACKs $D_{ACK,l}$ are obtained by using the procedure presented in [105].

4.3. Analytical model of CoAP large data transactions

In this section we present the analytical model of CoAP data transactions using 6LoWPAN fragmentation or CoAP blockwise transfer. We also derive the expression of the transition probabilities that characterize the model. The reference WSN topology is a star network. In this scenario, a client is in direct communication (single-hop) with the servers that are in the network. The analytical model is present only in the client and server nodes. As explained next, we model these nodes with two different Markov chains.

Analytical model

The analytical model of the client is shown in Figure 21. It is composed by two states, which are the same for CoAP blockwise transfer and 6LoWPAN fragmentation. The only differences are the transition probabilities between the states.

The first state is the IDLE, which means that the client is waiting for the reception of a block or the reassembled update sent with 6LoWPAN fragmentation. The client visits the acknowledgment transmission state (CoAP ACK_TX) after it receives successfully a block or a reassembled fragmented update. In both cases it sends the relative CoAP ACK.

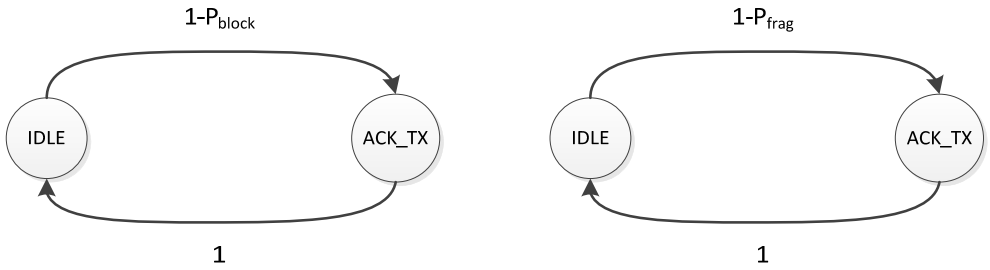


Figure 21 Markov chain for the client. The chain at left models the client when receiving updates using the CoAP blockwise transfer. The model at right represents 6LoWPAN fragmentation.

The transition probabilities of each chain are equivalent to the probability of receiving correctly a block or all the fragments of an update. The probability $P_{block,l}$ that a single block fails at node *l* is therefore equal to:

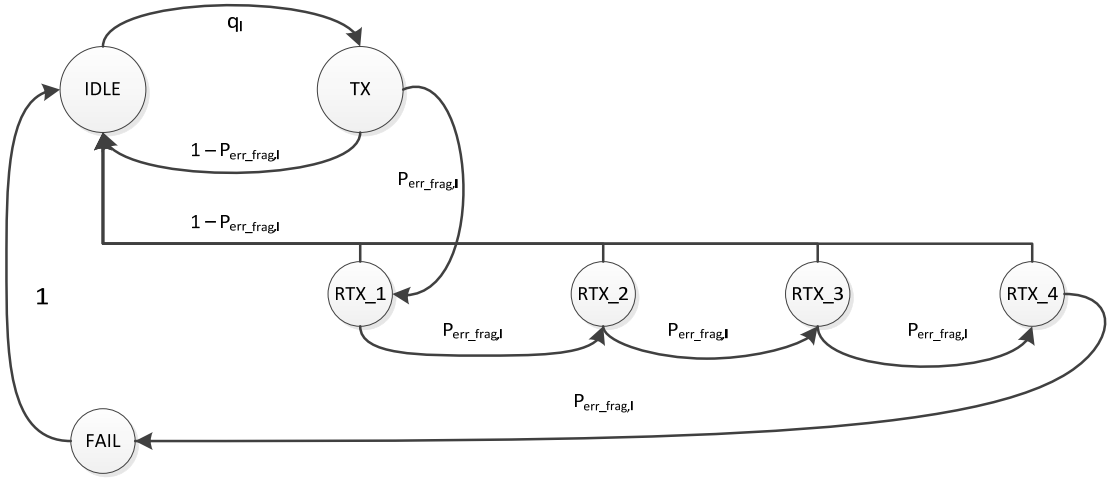
$$P_{block,l} = P_{frame,l} \tag{13}$$

where $P_{frame,l}$ is derived in Equation (10).

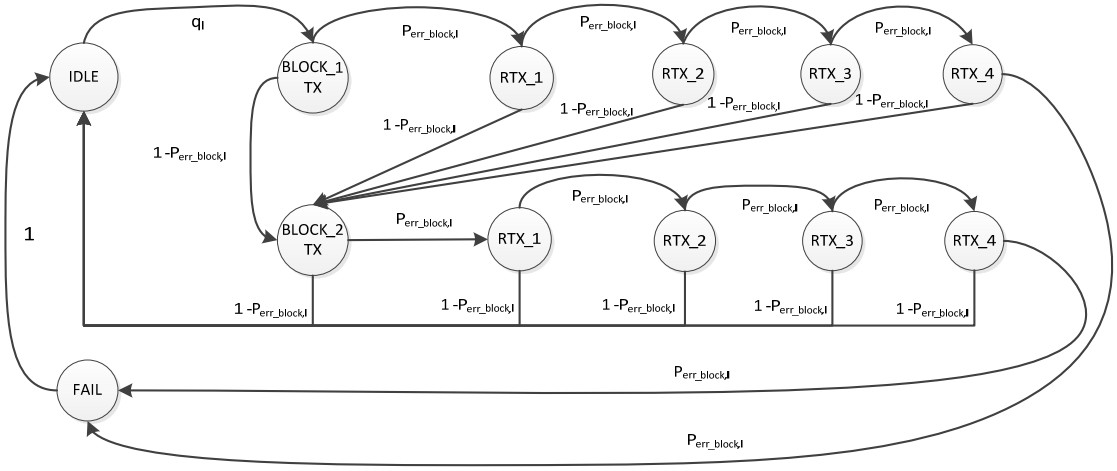
The probability $P_{frag,l}$ that the transmission of a fragmented update fails is equal to:

$$P_{frag,l} = 1 - (1 - P_{frame,l})^F \tag{14}$$

Figure 22 (a-b) shows the server's Markov chains for CoAP blockwise transfer and 6LoWPAN fragmentation. Figure 22 (b) considers the transmission of an observe update composed by two blocks. Both chains have four retransmission states, which correspond to the maximum number of retransmissions defined by CoAP. The model for CoAP blockwise transfer has a transmission and a retransmission stage for each block that composes the update. The CoAP layer, in fact, manages the transmission of each block and sends the subsequent only after the previous one has been acknowledged by the client. The CoAP layer instead, has a single transmission and retransmission stage when the update is transmitted with 6LoWPAN fragmentation.



a) The server retransmits all the fragments if the transmission of any of them fails or it does not receive the CoAP ACK



b) The server retransmits a single block if its transmission fails or it does not receive the relative CoAP ACK. The Markov chain represents the transmission of an observe update composed by two blocks.

Figure 22 Markov chains for the server. a) 6LoWPAN Fragmentation case b) CoAP blockwise transfer case

The server is in the IDLE state when it is waiting for the generation of the next update. The transition probability from the IDLE to the transmission state of the fragmented update (TX) or the first block (BLOCK_1 TX) is the probability q_l that we derived in equation (1). The server goes back to IDLE state after it receives the CoAP ACK relative to the last block or to the fragmented update. The failure of the update transmission also causes the server to go back to the IDLE state.

The retransmission states are visited after the failure of the transmission of a block, fragment or CoAP ACK. The transition probability between the transmission state and the first retransmission state of the CoAP blockwise transfer and 6LoWPAN fragmentation models are expressed as $P_{err_{block},l}$ and $P_{err_{frag},l}$ respectively. These probabilities are also valid for the transition between the retransmission states.

The probabilities at node l $P_{err_{block},l}$ and $P_{err_{frag},l}$ that a block or a fragmented update is retransmitted are defined as follows:

$$P_{err_{block},l} = P_{block,l} + P_{ack_{block},l} \quad (15)$$

$$P_{err_{frag},l} = P_{frag,l} + P_{ack_{frag},l} \quad (16)$$

Where $P_{ack_{block},l}$ and $P_{ack_{frag},l}$ are the probabilities at node l that the transmission of the CoAP ACK relative to the block or to the fragmented update fails, respectively. These are equal to:

$$P_{ack_{block},l} = P_{frame,l} * (1 - P_{block,l}) \quad (17)$$

$$P_{ack_{frag},l} = P_{frame,l} * (1 - P_{frag,l}) \quad (18)$$

The unsuccessful retransmission for c consecutive times of a block or of the fragmented update causes the update transmission to fail and the server to visit the FAIL state.

The probabilities $P_{fail_{blok},l}$ and $P_{fail_{frag},l}$ that an update transmission fails is equal to the probability that the transmission as well as the retransmissions of the update fail. These are equal to:

$$P_{fail_{blok},l} = \sum_{i=1}^{B} {P_{err_{block},l}}^{c+1} * \left(1 - {P_{err_{block},l}}^{c+1}\right)^{i-1} \quad (19)$$

$$P_{fail_{frag},l} = {P_{err_{frag},l}}^{c+1} \quad (20)$$

In the rest of this section we present the expressions for the performance metrics that we use to evaluate CoAP blockwise transfer and 6LoWPAN fragmentation.

Reliability

WSN applications that monitor a critical environment or a critical physical variable require that the data collected by sensor nodes must be delivered reliably to destination. However, wireless links are error prone and ensuring end-to-end reliable data transfer is one of the major challenges in WSNs. In the proposed model we define reliability as the probability that the update sent by a CoAP server arrives correctly at destination.

In the previous section we derived the probability that the transmission of an update fails. Next, we derive the expression of the end-to-end reliability for the CoAP blockwise transfer $R_{block,l}$ and the 6LoWPAN fragmentation $R_{frag,l}$.

$$R_{block,l} = 1 - P_{fail_{block},l} \quad (21)$$

$$R_{frag,l} = 1 - P_{fail_{frag},l} \quad (22)$$

Latency

The latency that can be tolerated by an application is of paramount importance to choose the appropriate data transfer technique. WSN applications could have strict deadline requirements on the validity of the data collected by a device. Scenarios such as e-Health or industrial monitoring are an example of those applications. In this paper, we define latency as the time required to complete a data transaction between server and client. The reception of the CoAP ACK relative to the fragmented update or to the last block determines the end of the transaction.

The latency of a CoAP transaction has to consider the delay caused by an unsuccessful frame transmission. The value of the RTO includes this delay. The expression of the delay of a frame transmission is derived in [105].

Next we present the expressions for latency.

Latency for 6LoWPAN fragmentation

The latency of an update transmission that uses 6LoWPAN fragmentation is equal to the sum of the transmission delays of the CoAP ACK and of each fragment. We define the latency for 6LoWPAN fragmentation $D_{frag,l}$ as:

$$D_{frag,l} = \sum_{j=0}^{c} \Pr(\mathcal{F}_j|\mathcal{F})\ D_j \qquad (23)$$

where $\Pr(\mathcal{F}_j|\mathcal{F})$ is the probability of successful update transmission at the *(j+1)th* attempt given a successful update transmission within *(c+1)* attempts.

$$\Pr(\mathcal{F}_j|\mathcal{F}) = \frac{(1 - P_{err_{frag}}){P_{err_{frag}}}^{j}}{R_{frag}} \qquad (24)$$

D_j is the delay of an update that is successfully transmitted at the *(j+1)th* attempt, i.e.,

$$D_j = D_{ACK,l} + f * D_{frame,l} + j * (RTO + D_{ACK,l} + fD_{frame,l}) \qquad (25)$$

Where $D_{\text{CoAP ACK,l}}$ is the delay of the CoAP ACK transmission, $D_{\text{frame,l}}$ is the delay of a fragment at node *l* and RTO is the value of the CoAP retransmission timeout.

Latency for CoAP blockwise transfer

The latency of an update transmission using CoAP blockwise transfer is defined as follows:

$$D_{block,l} = B \sum_{j=0}^{c} \Pr(\mathcal{L}_j|\mathcal{L})\ D_j \qquad (26)$$

where $\Pr(\mathcal{L}_j|\mathcal{L})$ is the probability of successful block transmission at the *(j+1)th* attempt given a successful block transmission within *(c+1)* attempts.

$$\Pr(\mathcal{L}_j|\mathcal{L}) = \frac{(1 - P_{err_{block}}){P_{err_{block}}}^{j}}{R_{block}} \qquad (27)$$

D_j is the delay of a block that is successfully transmitted at the *(j+1)th* attempt, i.e.,

$$D_j = D_{ACK,l} + D_{frame,l} + j * (RTO + D_{ACK,l} + D_{frame,l}) \qquad (28)$$

4.4. Performance evaluation

In this section we validate the model by Monte Carlo simulations and present the results of the performance evaluation. We base the simulation parameters on the specification of the IEEE 802.15.4 [8] and CoAP [7] protocols. We evaluate our models with different values of the traffic pattern and for CoAP updates composed by a variable number of fragments and blocks. The MAC parameters are selected as $m_0=3$, $m_B=5$, $m=4$, $n=0$ in accordance with the 802.15.4 standard [2]. The MAC frames have size $L= L_{ACK}=127$ *bytes* and

the MAC ACK frame $L_{MAC\ ACK}$=*11 byte*s. We study various traffic and fragmentation scenarios by considering *N= [10, 15, 20]* nodes with update generation rates λ= *[0.1… 1]* pkt/s ad updates divided into *B= [1, 3, 5, 7]* blocks (in CoAP blockwise transfer) or *F= [1, 3, 5, 7]* fragments (in 6LoWPAN fragmentation). As previously mentioned, the generation rate λ is constrained by the value of the minimum RTO recommended by CoAP [19], which is equal to *1s* with and with a random backoff of 0.5s. We set the number of CoAP retransmissions to *c=1*.

A. Reliability

Figures 23, 25 and 27 show the average reliability of CoAP blockwise transfer computed over all the links for a star topology network with mixed traffic conditions. Figures 24, 26 and 28 show the average reliability of 6LoWPAN fragmentation for the same scenario. A good agreement between simulations and analytical results of the model is observed.

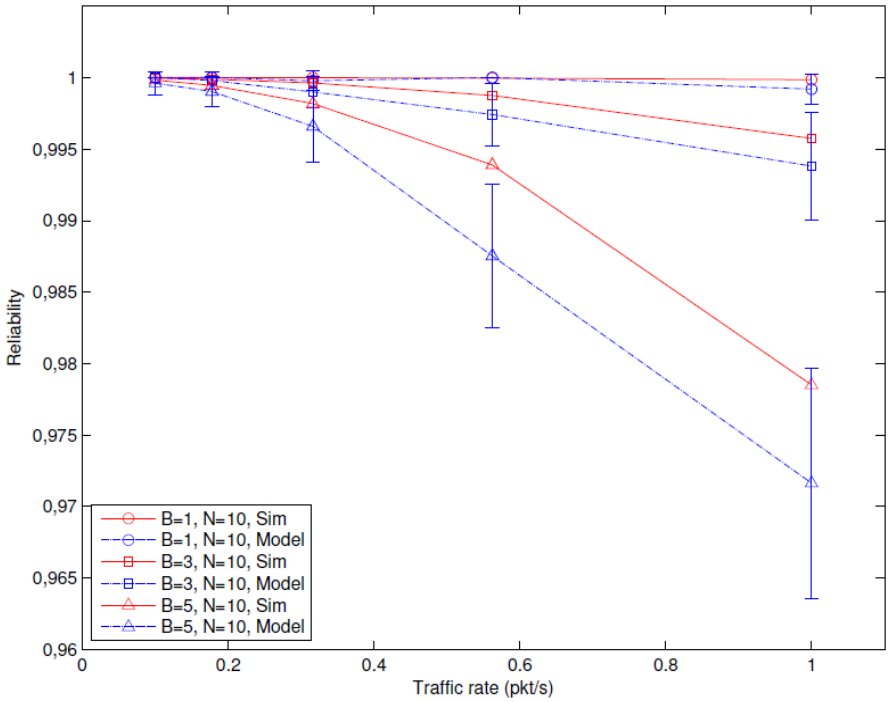


Figure 23 CoAP blockwise transfer reliability versus traffic rate for a star topology network composed by 10 nodes.

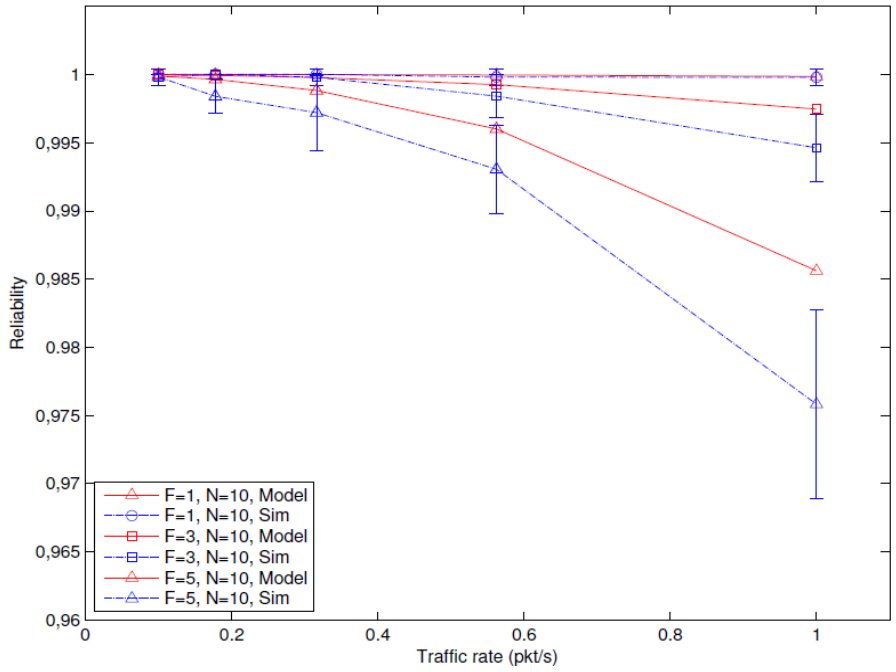


Figure 24 6LoWPAN fragmentation reliability versus traffic rate for a star topology network composed by 10 nodes.

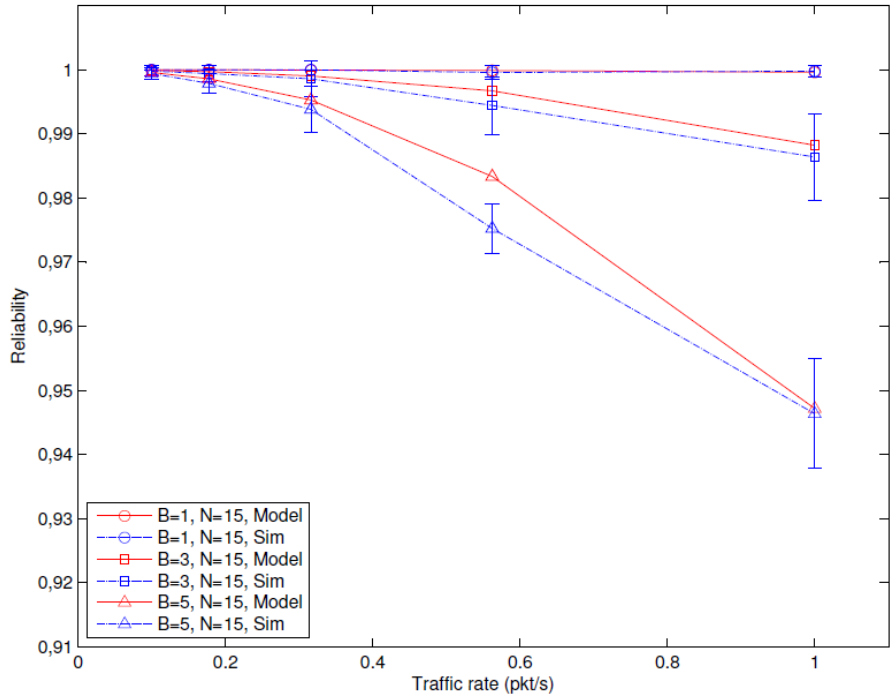


Figure 25 CoAP blockwise transfer reliability versus traffic rate for a star topology network composed by 15 nodes.

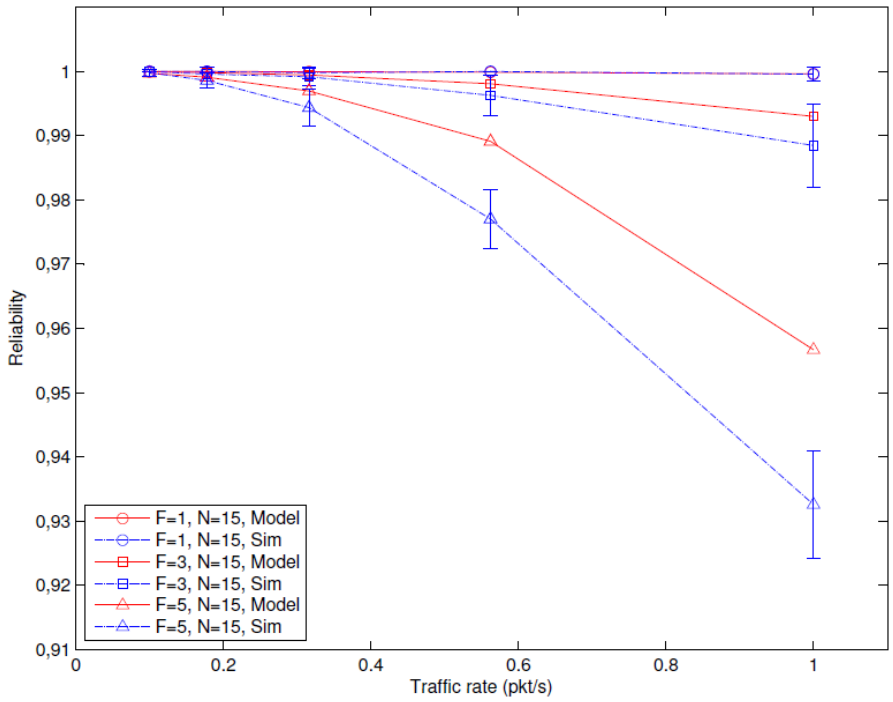


Figure 26 6LoWPAN fragmentation reliability versus traffic rate for a star topology network composed by 15 nodes.

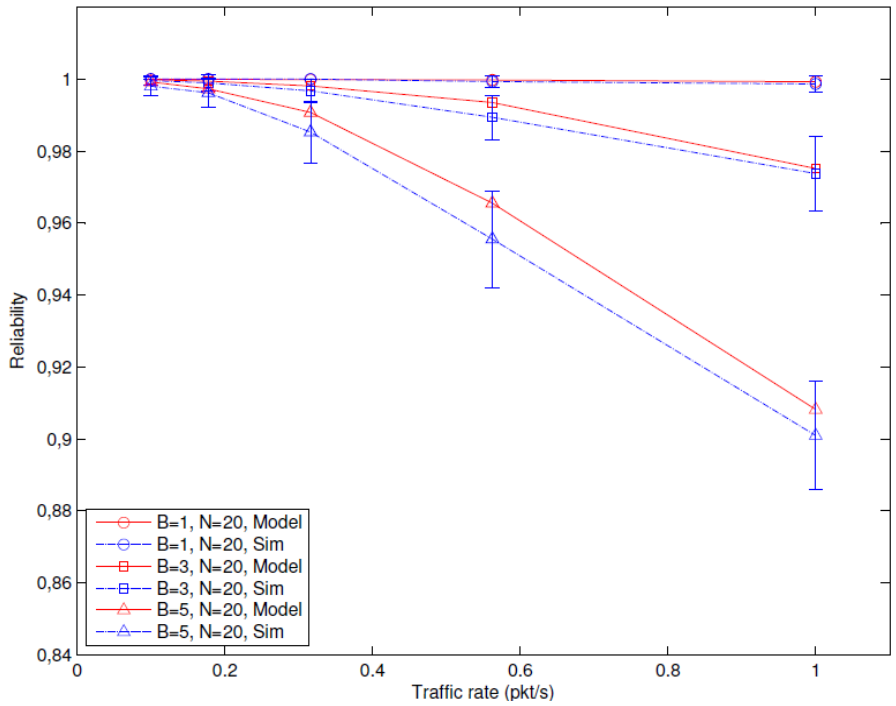


Figure 27 CoAP blockwise transfer reliability versus traffic rate for a star topology network composed by 20 nodes.

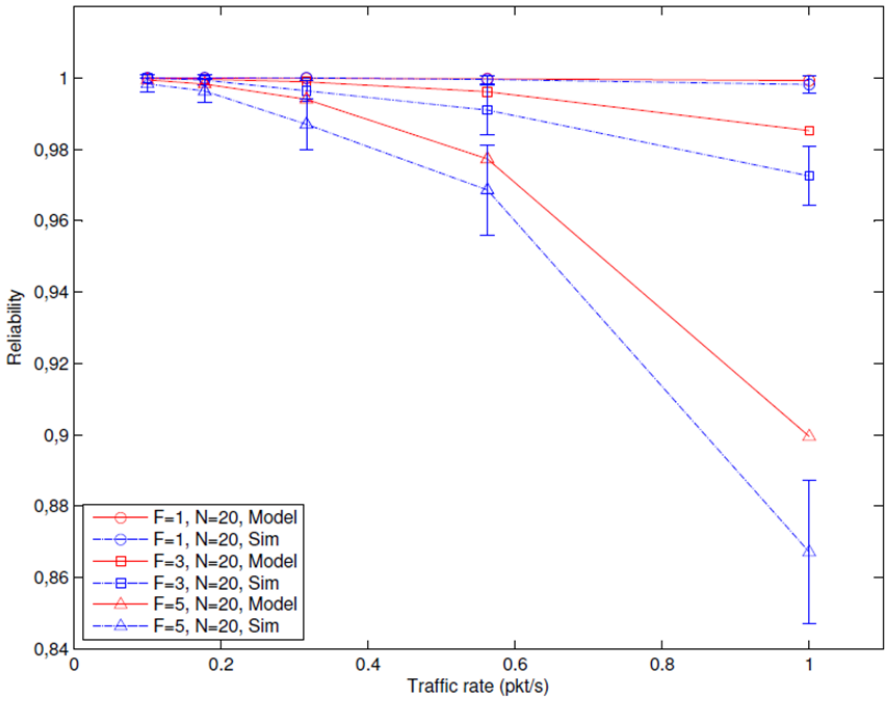


Figure 28 6LoWPAN fragmentation reliability versus traffic rate for a star topology network composed by 20 nodes.

The reliability performance of 6LoWPAN fragmentation and CoAP blockwise transfer is very similar in the considered scenarios. The difference between the reliability values is lower than 2 %. In particular, CoAP blockwise transfer has a slightly better reliability when the traffic conditions congest the WSN, which is the case for N=20, B=5 and λ greater than 0.8 pkt/s. In these conditions CoAP blockwise transfer improves reliability by the 0.96% respect to 6LoWPAN fragmentation. However, 6LoWPAN fragmentation is slightly more reliable than CoAP blockwise transfer when the traffic conditions do not congest the WSN. However, for N=10 the trends of both solutions are very close and differ by the 0.2% for λ=1 pkt/s and F=3. This difference grows up to the 0.7% for F=5 with the same traffic rate. For N=15, 6LoWPAN fragmentation has a maximum improvement of the 1% over CoAP blockwise transfer, which is obtained for λ=1 pkt/s and F=5. A similar difference is observed for N=20, F=3 and the same traffic rate.

The same behavior can be observed in Figure 29, which shows the average reliability of CoAP blockwise transfer and 6LoWPAN fragmentation according to the variation of the number of blocks or fragments that compose an update. The average reliability is computed over all the links for a star topology network of N=15 nodes and λ=1 pkt/s. 6LoWPAN fragmentation improves slightly reliability for a number of fragments lower than five. The reliability trend of 6LoWPAN fragmentation undergoes a pronounced drop when the number of fragments grows and the WSN becomes more congested. In this situation, CoAP blockwise transfer has a less pronounced drop, which allows to outperform 6LoWPAN fragmentation. In particular, for B=7 CoAP blockwise transfer improves reliability by the 10.7% respect to fragmentation. In congested WSNs, in fact, the probability that a fragment or block is retransmitted is high. Therefore, consecutive failures of blocks belonging to the same update do not cause the failure of the update transmission as it would happen in

6LoWPAN. CoAP blockwise transfer, therefore, is able to reduce the number of lost updates establishing a reliable transfer for each single block.

In CoAP blockwise transfer, the transmission of a CoAP ACK for each block causes an increase of the channel occupancy. This has a counter-effect on the reliability that is evident when the network is not congested. In this situation, CoAP blockwise transfer increases the average network traffic augmenting the collision probability. 6LoWPAN fragmentation requires the transmission of fewer messages for a single update. It is able, therefore, to reduce the network traffic and the retransmission probability of an update. A node using 6LoWPAN fragmentation is able to reduce significantly the occupancy of the channel. Thereby, the probability that a concurrent node finds the channel busy when attempting the transmission is lower respect to CoAP blockwise transfer. Besides the higher probability of finding the channel idle, a fragment has less chance to collide with the transmission of another one or with a CoAP ACK. 6LoWPAN fragmentation is able, therefore, to improve reliability under these traffic conditions.

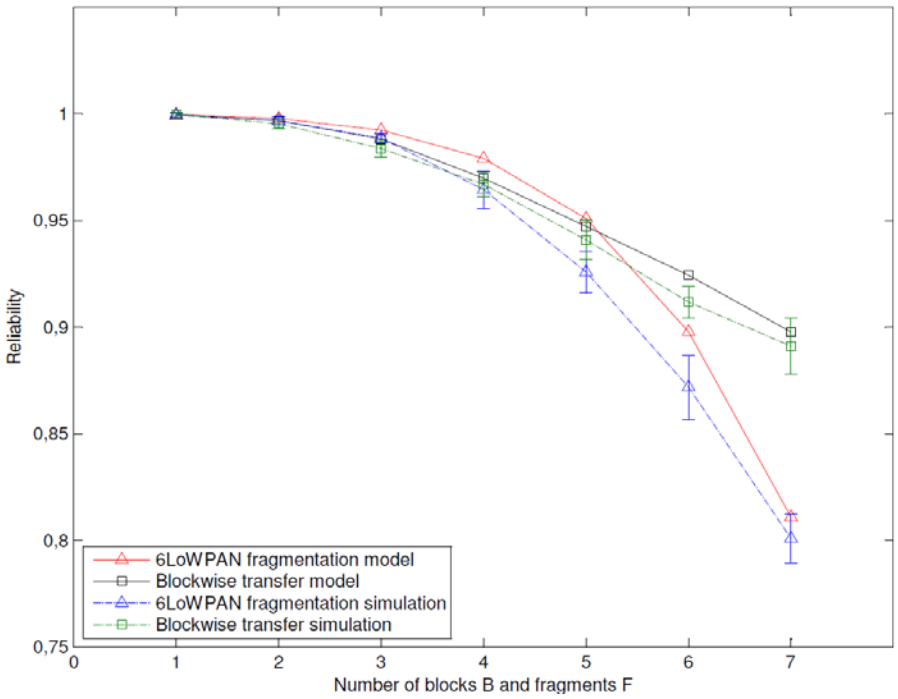


Figure 29 CoAP blockwise transfer and 6LoWPAN reliability versus the number of blocks or fragments that compose an update. 15 nodes compose the network and the traffic rate is fixed to 1 pkt/s

B. Latency

Figures 30, 32 and 34 show the average latency of CoAP blockwise transfer computed over all the links for a star topology network with mixed traffic conditions. Figures 31, 33 and 35 show the average latency for 6LoWPAN fragmentation. A good agreement between simulations and analytical results of the model is observed.

According to our performance evaluation, 6LoWPAN fragmentation outperforms CoAP blockwise transfer in terms of latency independently from the update generation rate and the number of nodes. The

difference between both techniques becomes higher with the growth of the update generation rate and the number of fragments or blocks involved in the communication. As mentioned, 6loWPAN fragmentation requires the interchange of fewer messages than CoAP blockwise transfer. Consequently, the latency is significantly lower than that experienced in CoAP blockwise transfer. The performance of CoAP blockwise transfer, however, could be improved by considering block sizes that allow sending a single block in more than one frame. This would reduce the number of CoAP ACKs and consequently the latency performance would improve. However, its performance would be always lower than that of 6LoWPAN fragmentation, which represents an upper bound to the performance of CoAP blockwise transfer. 6LoWPAN fragmentation could be considered as a particular case of CoAP blockwise transfer with a block size that allows sending a single CoAP ACK.

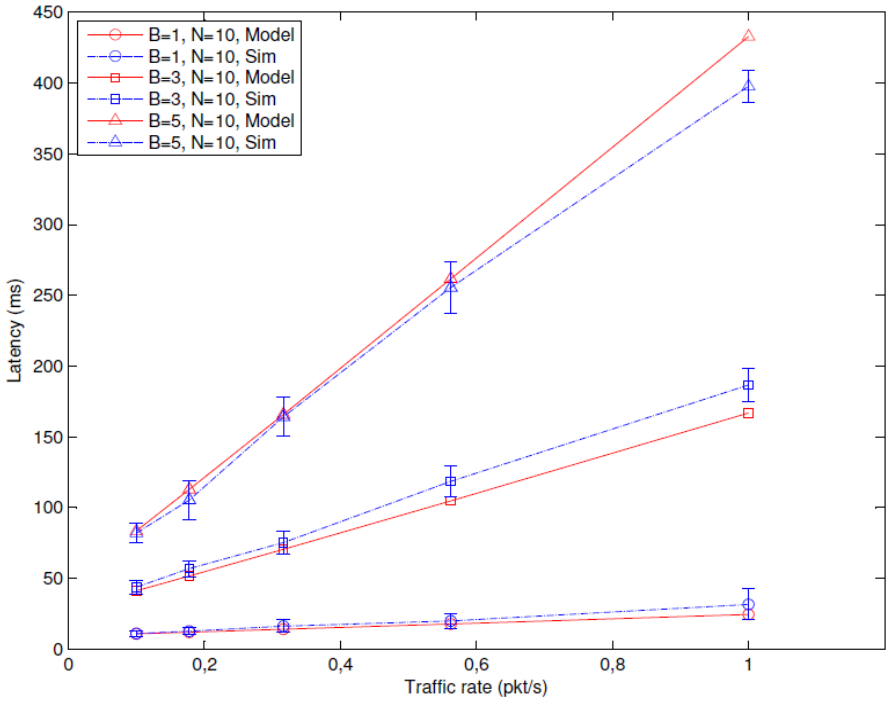


Figure 30 CoAP blockwise transfer Latency versus traffic rate for a star topology network composed by 10 nodes.

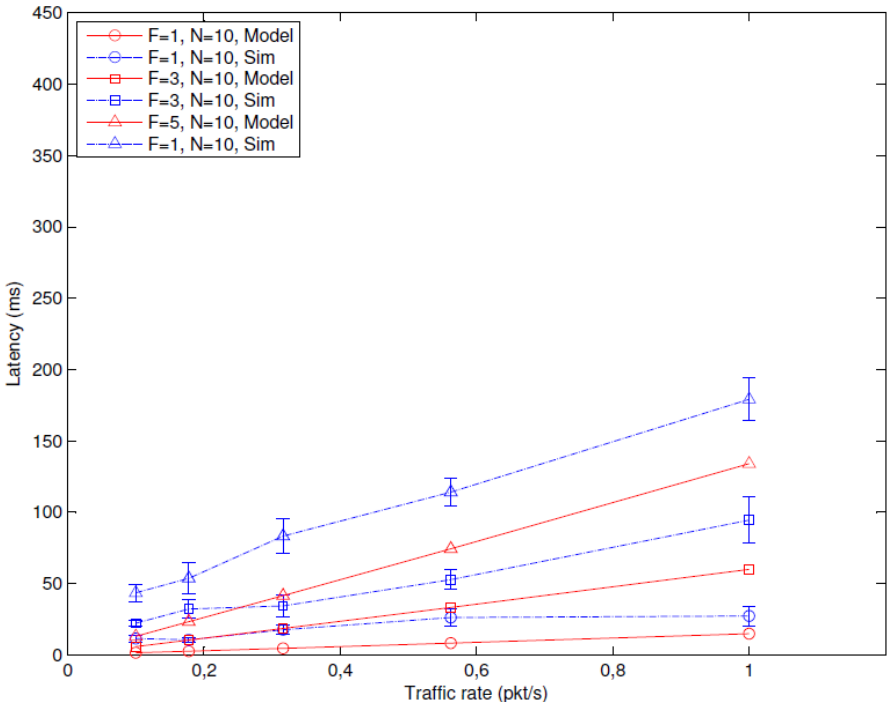


Figure 31 6LoWPAN Fragmentation Latency versus traffic rate for a star topology network composed by 10 nodes.

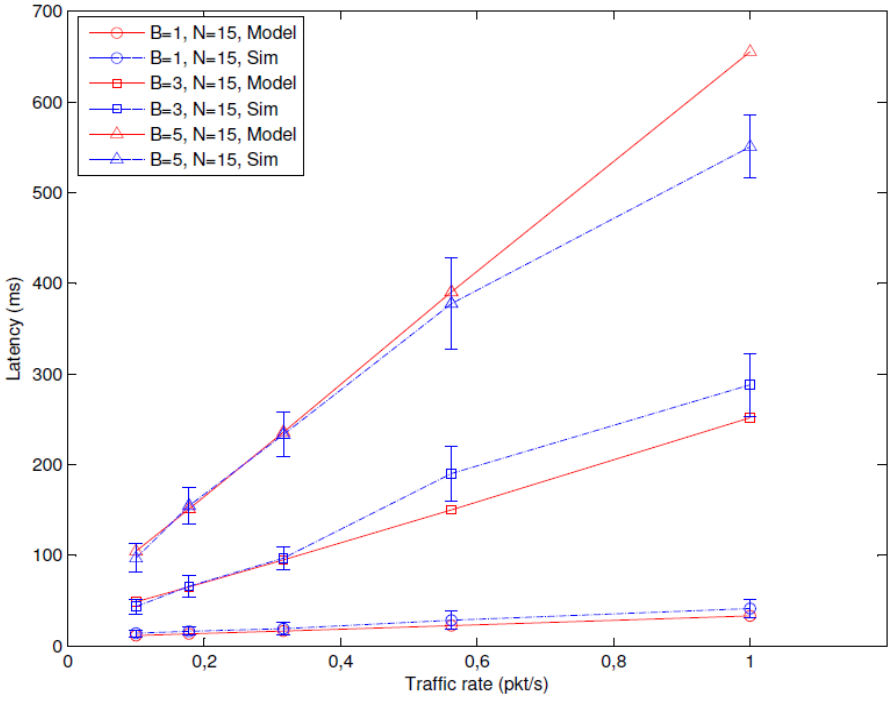


Figure 32 CoAP blockwise transfer Latency versus traffic rate for a star topology network composed by 15 nodes.

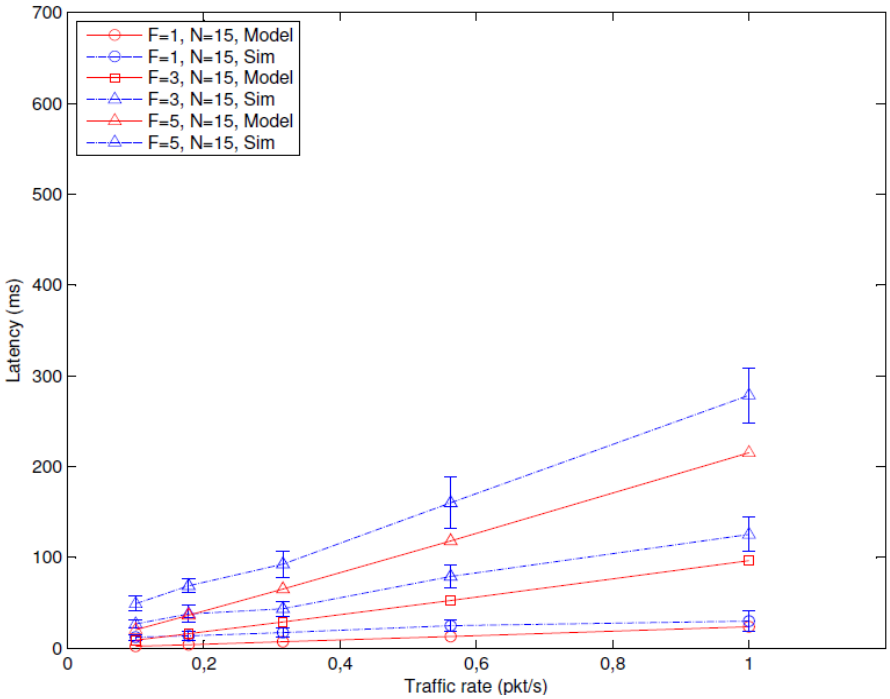


Figure 33 6LoWPAN Fragmentation Latency versus traffic rate for a star topology network composed by 15 nodes

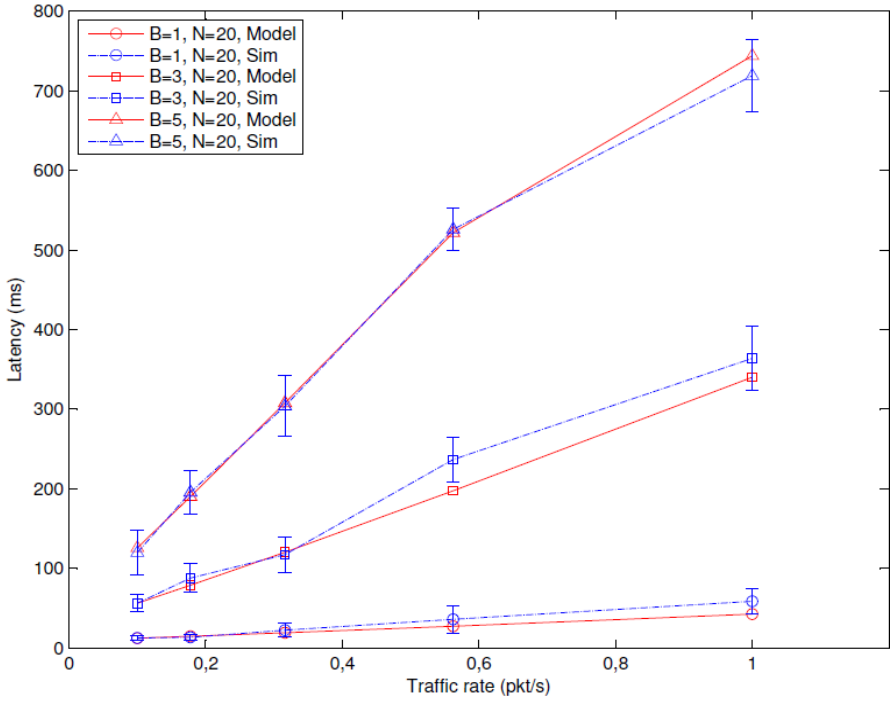


Figure 34 CoAP blockwise transfer Latency versus traffic rate for a star topology network composed by 20 nodes

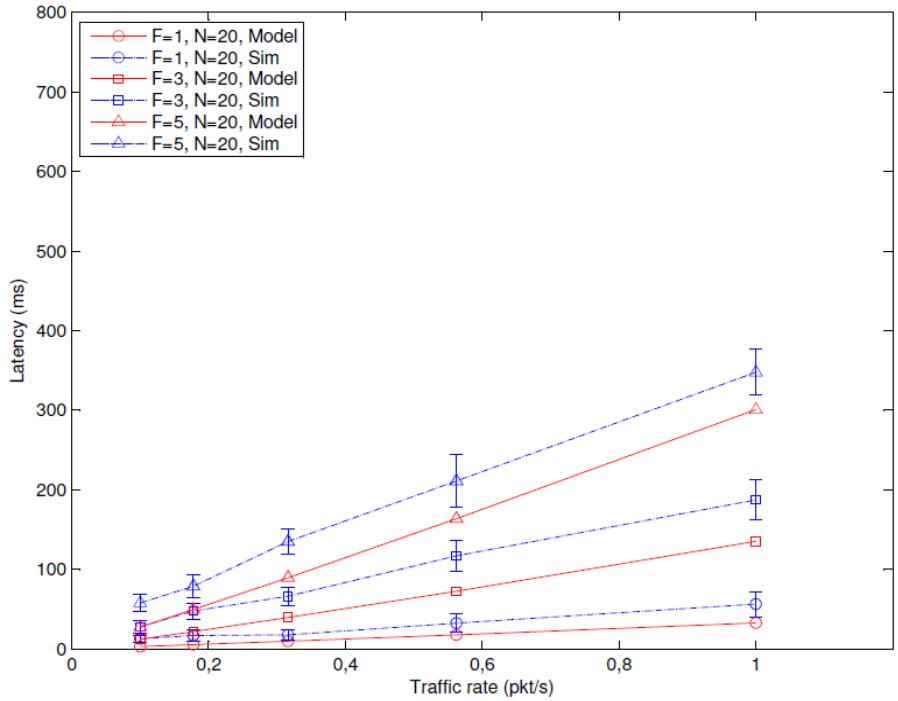


Figure 35 6LoWPAN Fragmentation Latency versus traffic rate for a star topology network composed by 20 nodes

The latency curve of CoAP blockwise transfer has a sharp rise for increasing values of the traffic rate, which further augments the difference with that of 6LoWPAN fragmentation. The same behavior can be observed in Figure 36, which shows the latency trends according to the number of blocks or fragments of an update in a WSN composed by 15 nodes and traffic rate of 1 pkt/s. The growth of the traffic rate as well as that of the number of fragments or blocks congest the WSN and augment the retransmission probability of an update. Although in case of congestion CoAP blockwise transfer shows a slightly better reliability, the cost in terms of latency of block-to-block retransmission does not allow improving its performance. The retransmission of the entire update in 6LoWPAN fragmentation has less effect on the average latency.

This behavior can be explained analyzing the Probability Density Function (PDF) of the latency, which is shown in Figure 37. It is evaluated in a star topology WSN composed by 15 nodes with a traffic rate of 1 pkt/s and updates composed by 5 fragments or blocks. The distribution of both solutions presents a long tail, which is due to the effect of retransmissions. The presence of block-to-block retransmissions causes the tail of CoAP blockwise transfer to be the longest one. This further worsens its average latency and causes the rapid growth of its curve. Since in CoAP blockwise transfer each block of an update could be retransmitted, the overall latency would be higher than that of retransmitting the entire update as done in 6LoWPAN fragmentation.

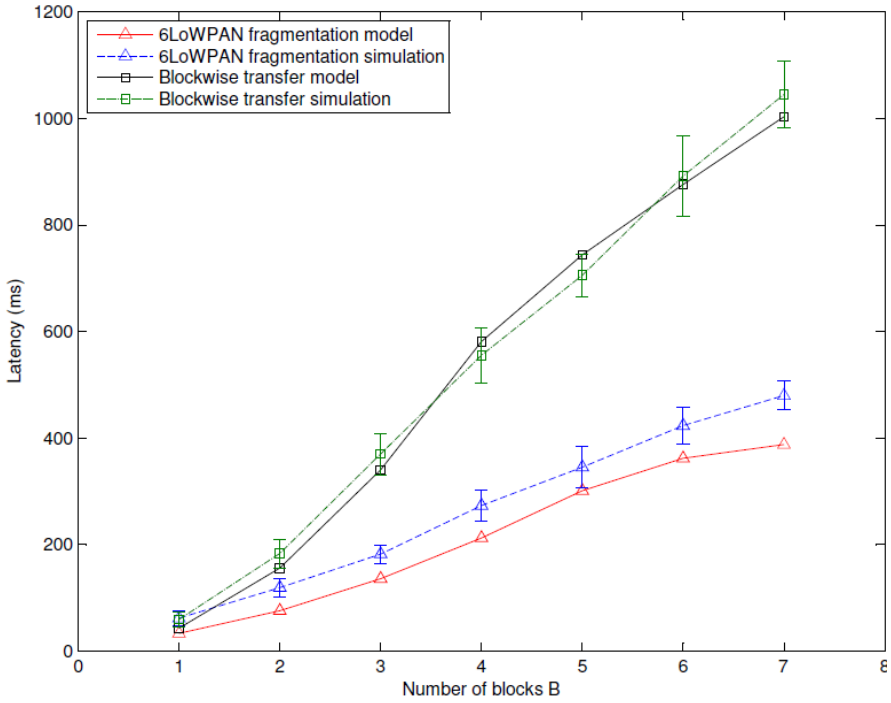


Figure 36 CoAP blockwise transfer and 6LoWPAN latency versus the number of blocks or fragments for a star topology network composed by 20 nodes and a traffic rate of 1 pkt/s

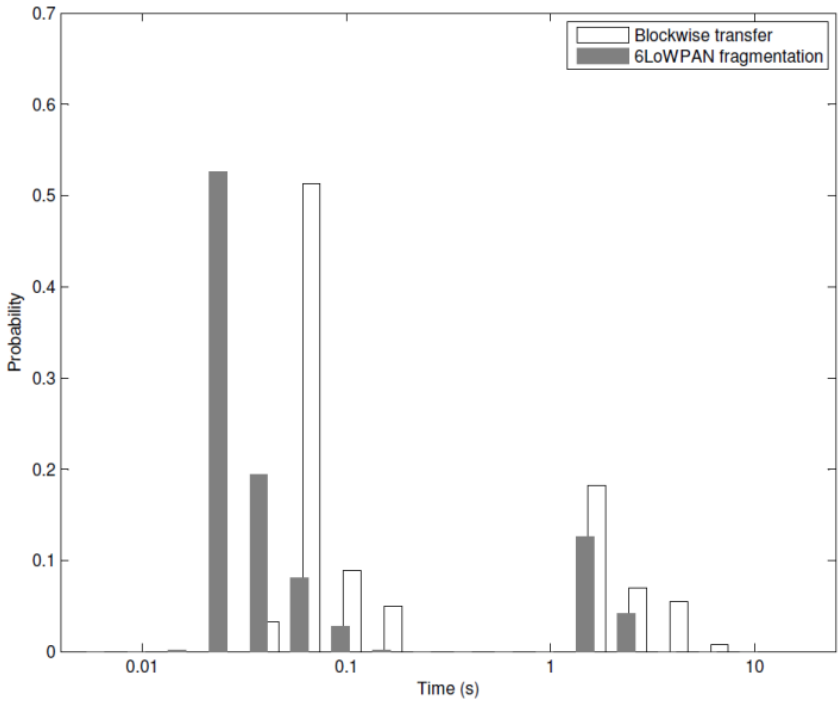


Figure 37 PDF of the latency for a star topology network with 15 nodes and a traffic rate of 1 pkt/s. 6LoWPAN. 5 fragments or blocks compose each update. For the sake of clarity, the x-axis is shown in logarithmic scale

Model limitations

Here, we discuss the fundamental limitations of the analytical model developed and analyzed in the previous sections.

First, we remark that the Markov chain model require the solution of systems of non-linear equations to derive MAC indicators such as the CCA probability, the busy channel probability and the collision probability in Section 3.3. For the use of such a model for online computation in real sensors, the complexity is a critical factor since the typical micro-controller does not support well a complex computing. In heterogeneous network conditions, a Markov chain has to be solved for each link, and the complexity increases with the number of links. The use of approximated model equations is advocated in [105], when the number of nodes exceeds 15, to guarantee bounded computation times in the order of seconds for typical sensor platforms.

The model includes the effects of bursty traffic on the busy channel probability in different backoff stages. However, we assume an average CCA probability τ_l in each time unit. As we see from the simulation results, this is a fair approximation when the number of blocks (or fragments) is limited. When the traffic in the network becomes saturated, the performance of the MAC layer is influenced also by higher order statistics of τ_l.

A practical limitation with high traffic conditions is also given by the retransmission mechanism of CoAP that defines a minimum RTO in the order of 1 second. This is specified by the standard to guarantee support for multi-hop communications. However, that limits the derivation of the offered traffic in the network, since the packet service time increase quickly to the update generation time, especially for block-wise transfer, where the retransmission is performed on a block-level, as we saw in Figure 36.

4.5. Conclusions and contribution

In this chapter, we have analyzed CoAP data transactions with large payloads in WSN with star topology. We have proposed a novel analytical model to study the performance of 6LoWPAN fragmentation and CoAP blockwise transfer. We have adopted reliability and latency as performance indicators. We have used Monte Carlo simulation to validate our model. The results demonstrate accuracy to estimate the performance of CoAP blockwise transfer and 6LoWPAN fragmentation.

As for reliability, we have observed a good performance of both techniques with small difference between them. However, depending on the traffic conditions a technique could be preferred to the other. In particular, CoAP blockwise transfer is a more reliable solution when traffic conditions lead to a congestion of the WSN, which is the case of applications with high traffic rates or that produces updates composed by many blocks. 6LoWPAN fragmentation is preferable when the WSN links are less congested.

A clear disadvantage of CoAP blockwise transfer is the latency required to transmit an update. The latency introduced by acknowledging each single block does not allow to CoAP blockwise transfer to have a trend closer to that of 6LoWPAN fragmentation. According to our result 6LoWPAN fragmentation outperforms CoAP blockwise transfer in terms of latency also in congested WSN.

In conclusion, applications that have strict requirements in terms of latency, i.e. real-time applications, should adopt 6LoWPAN fragmentation. With more relaxed constraints on latency, i.e. applications that uses data logging, CoAP blockwise transfer should be adopted when the traffic conditions are close to saturation. A good trade-off could be reached using an algorithm able to choose dynamically which technique use depending from the traffic conditions.

From the results of the research presented in this chapter derives the paper *Analytical model of large data transactions in CoAP networks* [P4].

5. TinyCoAP

In this chapter we present the design, implementation and evaluation of CoAP for TinyOS. We refer to its implementation as TinyCoAP. Typical WSNs nodes are battery-powered and often deployed in unattended environment. Embedded software applications for WSNs, therefore, should be designed and optimized concerning lowest energy consumption and reliable execution. The reduction of energy consumption is mainly achieved using radio duty cycling protocols. However, further reduction can be reached through effective memory management. This allows saving CPU cycles and reducing the code complexity. Both aspects are critical for lowering the energy consumed by the CPU processing and for reducing the risks of failures during execution. The optimization process of TinyCoAP is focused in developing an efficient and safe use of memory. In particular, we focus on severely constrained WSNs nodes featuring few kilobytes of memory and CPUs with reduced computational capabilities. Although some technology offers more powerful nodes, working with constrained hardware help us to fix a lower bound to the resources that could be used by TinyCoAP. In that way, we ensure that our design choices can deliver a highly optimized implementation suitable for any application domain.

As mentioned in section 1.2.4, TinyOS already includes 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. We argue that better performance and minimal resource consumption can be achieved developing a native library. We demonstrate the effectiveness of our approach by a comprehensive performance evaluation. In particular, we test and evaluate TinyCoAP and CoapBlip in a real scenario, as well as solutions based on HTTP. The evaluation is performed in terms of latency, memory occupation, and energy consumption. Furthermore, we evaluate the reliability of each solution by measuring the goodput obtained in a channel affected by Rayleigh fading. We also include a study on the effects that high workloads has on a server.

5.1. Implementation

One critical WSN design challenge involves providing reliable and performing solutions while coping with constrained resources. Memory, energy and bandwidth represent the resource constraints to meet. Thereby, the design process at any level has to be focused on optimizing their use. The goal of our implementation is, therefore, to minimize the resource consumption by developing a lightweight and efficient code optimized for the OS in use. Because of its popularity and diffusion, TinyOS has been chosen as the reference OS.

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 [86]. NesC is a C dialect designed to improve code efficiency and robustness in embedded software applications [28]. Through its simplicity, NesC is able to reduce RAM

occupation, code size, and prevents low-level bugs. The programming model of TinyOS is 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. TinyCoAP is completely written in NesC. The rest of this section focuses on the memory allocation system, library and the data structures of TinyCoAP.

5.1.1. Structure of the Library

TinyCoAP provides a CoAP library native for TinyOS. It is designed behind the idea that better performance and reliable run-time execution are both achieved integrating it with the OS core libraries. Following these design principles, the core functionalities of CoAP are provided as TinyOS components. These components are developed as part of the TinyOS network library. Differently from TinyCoAP, CoapBlip is thought as an adaptation of a C library for generic embedded systems. A TinyOS component is used as an adapter between this library and the TinyOS application.

TinyCoAP avoids using external C libraries and relies completely on code developed in the NesC language. This allows reaching a high code optimization and having less impact 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 conceived as a set of components connected together to perform a specific task. These interact between each other using the interfaces that they provide. Applications declare at compile-time which components they use and then, they explicitly wire the interfaces they will use at run-time. Thanks to this static wiring, TinyOS programs avoid using function pointers and therefore they are able to reduce the RAM memory footprint.

The TinyCoAP library is composed by five components. Its design follows the CoAP conceptual layering. The message layer is implemented by three components. CoapPDU, where PDU stands for Protocol Data Unit, is the main component of this sub-layer. It provides the interface used to create, read and write CoAP packets. The interface needed to create or delete options is provided by the CoapOption component. The creation, use and managing of the linked lists is performed by the interface provided by the CoapList component. Linked lists are useful for iterating the packets that are in the memory pool waiting for being processed. CoapList is also used to store and iterate the options that compose a packet and to manage retransmissions. CoapPDU is wired to CoapList and CoapOption. This allows CoapPDU to work with the

options contained in a CoAP packet. Moreover, each component of the message layer is wired to the TinyOS PoolC component. This is used to allocate the memory needed to perform their operation. PoolC allocates memory according to the data structure that is specified by each component. The wiring of the message layer components is shown in Figure 38.

The request/response matching layer of CoAP is implemented by the CoapServer and CoapClient components. CoapClient provides the interface used to send CoAP requests. The interface provided by CoapServer allows initializing and binding the server to a specific UDP port. The retransmission mechanism and the CoAP packet processing are also implemented by these components. CoapServer implements the discovery of CoAP resources [87] and the observe option of CoAP. The management of the resources provided by the server is implemented in a distinct interface. The resources are created through a parameterized interface. This is called CoapResource and provides commands and events to handle resources and the separate response mechanism of CoAP.

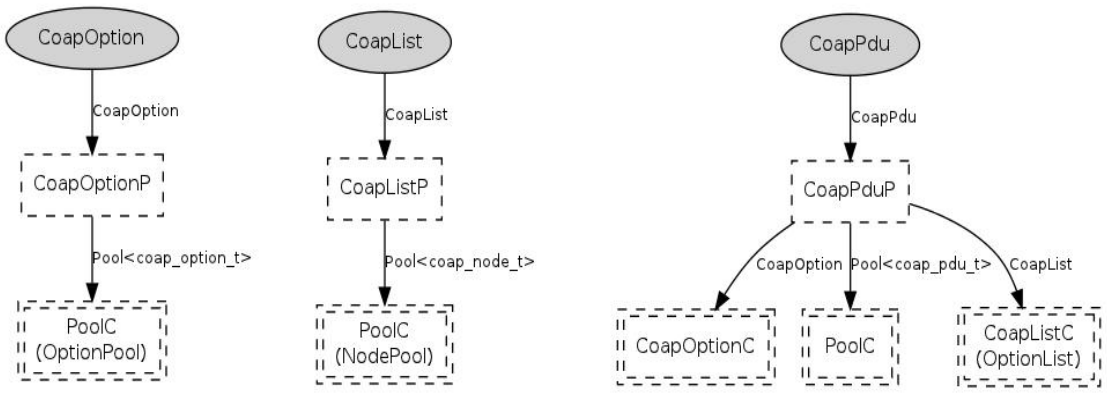


Figure 38 Wiring of the TinyCoAP interface for the CoAP message layer. PoolC is used to provide the memory needed by the components.

5.1.2. RAM Memory Allocation

Managing the allocation of RAM memory is one of the most critical aspects to consider when embedding software applications in WSN nodes. The management of memory allocation has to cope with the limited size of RAM memory and the lack of hardware memory protection that characterize constrained nodes. In this perspective, managing the RAM memory dynamically could increase the probability of having failure nodes or could exhaust the available memory. In fact, the lack of hardware memory protection does not prevent the risks of having a collision between the heap and stack or a memory leak [88]. Furthermore, the size of the allocated RAM memory would be difficult to control with this allocation system.

TinyCoAP avoids these risks by allocating RAM memory statically. The size of the allocated memory is known at compile time and the possibility of memory exhaustion is therefore avoided. Furthermore, static allocation would eliminate the risks of failures due to collision of the heap and the stack. Therefore it would enhance the network reliability. A further optimization is obtained allowing TinyCoAP to create CoAP responses without allocating new memory. TinyCoAP creates responses using the memory already allocated

to store the relative CoAP requests. Besides the reduction of the RAM memory footprint this enables a lighter packet processing with less impact on the CPU. As a consequence, the reduction of the CPU use would lower the energy consumption. As reported in [88], the CPU consumes 4.6 mA when active and 2.4 mA when idle while the radio uses 3.9 mA when receiving. Therefore, the TinyCoAP management of buffers would save CPU cycles and enhance the battery life of nodes.

The static allocation of memory done by TinyCoAP is compliant with the RAM memory management defined in NesC. In fact, NesC does not support dynamic memory allocation. This characteristic allows preventing memory fragmentation and run-time allocation failures [28]. However, a situation may arise in which applications might need dynamic allocation. To overcome this problem, TinyOS provides a component called PoolC that simulates the dynamic memory allocation. Should PoolC be used, the maximum pool memory size would be allocated statically at compile time. During the execution time, the applications will take the amount of RAM memory they need from that available in the pool. An eventual memory leak would cause the pool to empty, but the heap and stack would not collide. As mentioned above, TinyCoAP uses PoolC to allocate the buffers needed to store the CoAP packets and the linked lists. Differently from TinyCoAP, CoapBlip adopts a dynamic memory allocation management. It uses the malloc memory management library to allocate memory for buffers and linked lists.

5.1.3. Data Structure

As mentioned above, TinyCoAP components are organized following the conceptual layering of CoAP. The message layer is build on top of Blip. CoapBlip also uses this 6LoWPAN stack. Should Blip receive a UDP packet, it checks the presence of the CoAP header. If it is present, the interface provided by CoapPDU saves it in a CoAP PDU. This PDU is stored in the memory previously allocated through PoolC. The use of PoolC allows TinyCoAP to establish at compile time the maximum size a packet can have 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 features make TinyCoAP robust against possible memory leaks and always provide it with room in the memory for the incoming packets. Furthermore, TinyCoAP is easily adaptable to different applications. The TinyCoAP PDU data structure is designed to be used with PoolC. It avoids the use of pointers for accessing to the different parts of the PDU. Table 2 shows the CoAP PDU defined in CoapBlip and TinyCoAP.

In TinyCoAP, the received CoAP message is initially stored in the UDP buffer as a void element. This element is then converted into a coap_pdu_t structure and stored in the memory pool. Once the PDU structure has been created, the UDP buffer is ready to receive 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 adjusted to the application requirements and to the characteristic of the sensor.

CoapBlip uses pointers to access to different 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 equal to the maximum packet size allowed by CoapBlip. Thus, although CoapBlip uses malloc, the memory is always allocated with the same size.

Table 2 CoAP PDU structures. CoapBlip stores the PDU in the UDP buffer and uses a pointer to provide access. TinyCoAP saves it in the memory allocated with PoolC.

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;

5.2. Test-bed

As previously mentioned, we compare and discuss the performance obtained in a real 6LoWPAN network by TinyCoAP, CoapBlip and HTTP. Our experiments involve different solutions for the transport layer used by HTTP. We consider HTTP/TCP, HTTP/UDP and HTTP persistent. The third solution refers to the use of a persistent TCP connection.

Both HTTP and TCP protocols are included in Blip. The HTTP version it implements is the 1.0. The TCP version used by the client is one the most widely adopted, which is TCP Reno. The congestion window size (CWND) of client and server is fixed to 1. A larger CWND could overload the wireless link with too many transmissions, and thereby increases the probability of collision. The need to have more than one TCP packet in the network could be justified in a multi-hop scenario. In a one-hop network there is not such a need, and client and server would avoid competing for the radio channel. The Maximum Segment Size (MSS) of both sides always corresponds to the length of the TCP payload sent in each experiment. Thus, each HTTP packet is sent in one TCP segment. This allows maximizing the throughput by reducing the interchange of control messages and avoiding fragmentation.

The tests involve client/server transactions where a client sends requests to a server in order to retrieve information. All the requests are sent using the GET method. When receiving a request with test as URI, the CoAP or HTTP server replies with a payload composed by sequence of bits of fixed size. In this way, the node does not perform sensing operation that might influence the results. Therefore, the experiments account only for the performance of each technique in processing and replying to the received messages. All the tests are performed in a real WSN implementation.

Since our experiments are focused to evaluate single client/server transactions, we can keep our test-bed network simple and avoid deploying complex architectures. The test-bed network used in all the experiments is shown in Figure 39. The CoAP and HTTP clients as well as the proxy server are located in a PC. Each

request is sent through a 6LoWPAN base station attached to the USB port of the PC. The server is embedded in a TelosB mote located one-hop away from the 6LoWPAN base station.

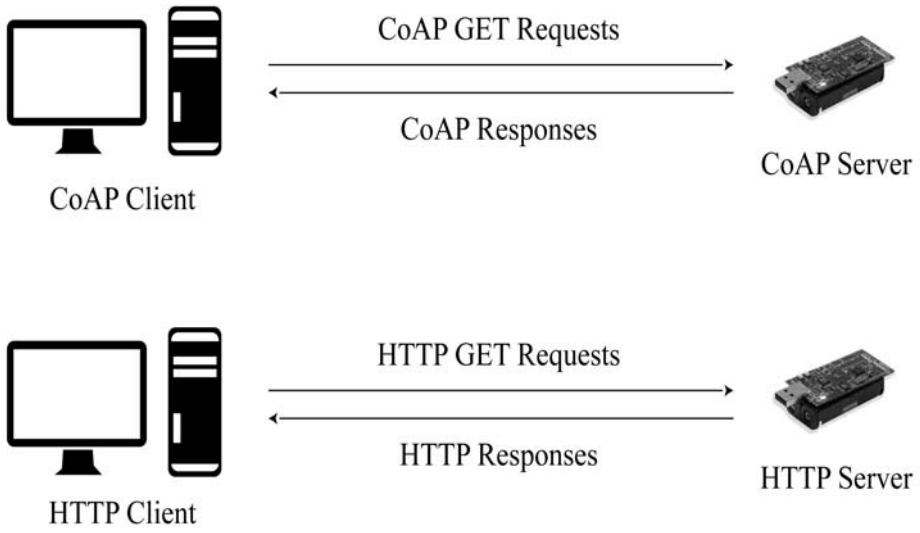


Figure 39 Test-bed network. The HTTP or CoAP clients are located in a PC while the servers are embedded in a sensor.

The CoAP CON request messages sent by the client have a total length of 14 bytes. The HTTP GET requests carry the same information of that sent using CoAP. However, the verbose format of HTTP implies a growth in size up to 37 bytes. Table 3 shows the composition of the GET requests for CoAP and HTTP.

Table 3 Composition of the HTTP and CoAP Confirmable (CON) requests. In both cases the requests are sent using the GET method.

CoAP	HTTP
CoAP HEADER: 5 bytes	*GET /test HTTP/1.0\r\n*
COAP URI_PATH: 6 bytes	*Host: fec0::2\r\n\r\n*
COAP TOKEN: 3 bytes	
PAYLOAD: 0 bytes	

5.3. Results and Discussion

In this section, we report the results of a performance evaluation for all the considered solutions. Our study evaluates various parameters. First, we measure the amount of RAM and ROM memory used by each solution; we then evaluate the latency of request/response transactions; after that, we measure the energy consumed by each different solution to processing and reply to a request. Moreover, we evaluate the client goodput obtained in 802.15.4 links affected by Rayleigh fading. This test allows us to evaluate the

performance of the CoAP reliability mechanism. Finally, we evaluate the effect that high workloads have on the server performance. In this case, we analyze the rate of the requests per second that it can serve as a function of the client request rate.

The results concerning latency, energy and reliability are reported according to the payload size with which a server replies to a client request. The maximum payload size varies depending to the implementation used. We found the different methods used to allocate RAM memory as the cause of this variation. The implementations using dynamic memory allocation achieve the lowest payload size. In this sense, CoapBlip is able to reach 650 bytes as maximum payload size while HTTP/TCP and HTTP persistent reach 800 bytes. TinyCoAP benefits from the use of static allocation and it is able to reach 1,200 bytes. The same size is achieved by HTTP/UDP. In this case the low complexity of the implementation and the absence of TCP buffers allow to HTTP to work with high payload sizes.

5.3.1. Memory Footprint

Table 4 shows the amount of RAM and ROM memory allocated at compile time for each considered implementation. The values correspond to a maximum payload size of 500 bytes. The values for HTTP/TCP and HTTP persistent are the same and are reported as HTTP. In fact, both solutions use the same TCP buffers and the use of a persistent connection does not vary their allocation.

Table 4 RAM and ROM memory occupation. TinyCoAP reserves all the memory required at compile time.

	TinyCoAP	**CoapBlip**	**HTTP**	**HTTP/UDP**
RAM	8,458 kB	7,102 kB	7,85 kB	3,922 kB
ROM	31,812 kB	42,576 kB	39,484 kB	27,802 kB

TinyCoAP allocates all the memory needed for buffering the CoAP packets at compile time. Therefore, it occupies more RAM memory than the other solutions. However, the occupation of RAM would remain at the same level at run-time while that of the other implementations would increase.

The ROM memory footprint gives an idea of the complexity and weight of the code of each implementation. In fact, the compiled code is stored in the ROM memory. A code with a small impact on ROM memory would allow adding further resources or enrich CoAP with more capabilities.

CoapBlip has the highest ROM memory footprint. We found the main cause in the lack of optimization of the code. As mentioned, CoapBlip is an adaptation of a C library. This library is therefore installed in the node along with the TinyOS component used to adapt it to the OS. The use of C libraries is usually too complex for the memory constraints of a mote and implies a growth of the memory footprint. Also HTTP solutions using TCP rely on a C library, thereby the ROM footprint increases also for these implementations. TinyCoAP lowers the ROM footprint by avoiding the use of C libraries. As mentioned it is written in nesC and therefore it is optimized for TinyOS.

The HTTP/UDP implementation has the lowest memory footprint. This solution has a very low complexity and provides no reliability mechanism or request/response matching. The code size can therefore

be minimized and ROM memory occupation can be reduced. Moreover, it is able to reach the minimal RAM memory occupation since it does not implement any HTTP buffer. This implementation uses exclusively the UDP buffer provided by Blip.

5.3.2. Latency

The latency experienced by a client while retrieving information from a server is one of the most important parameter used to evaluate the goodness of the server implementation. Low latency values can significantly enhance user experience and benefit those applications that work in real-time. We define latency as the time elapsed from the moment the client sends a request until the moment it receives the response.

Figure 40 shows the latency for each tested solution. Each point on the graph represents the average latency of 100 successful request/response transactions. Payload size ranges from 10 to 1,200 bytes with increments of 50 bytes. The client sends a new request after receiving a response to the request previously sent. Figure 41 shows only a portion of the latency trend, excluding that obtained by HTTP/TCP. In this way, the differences between the other implementations can be better appreciated.

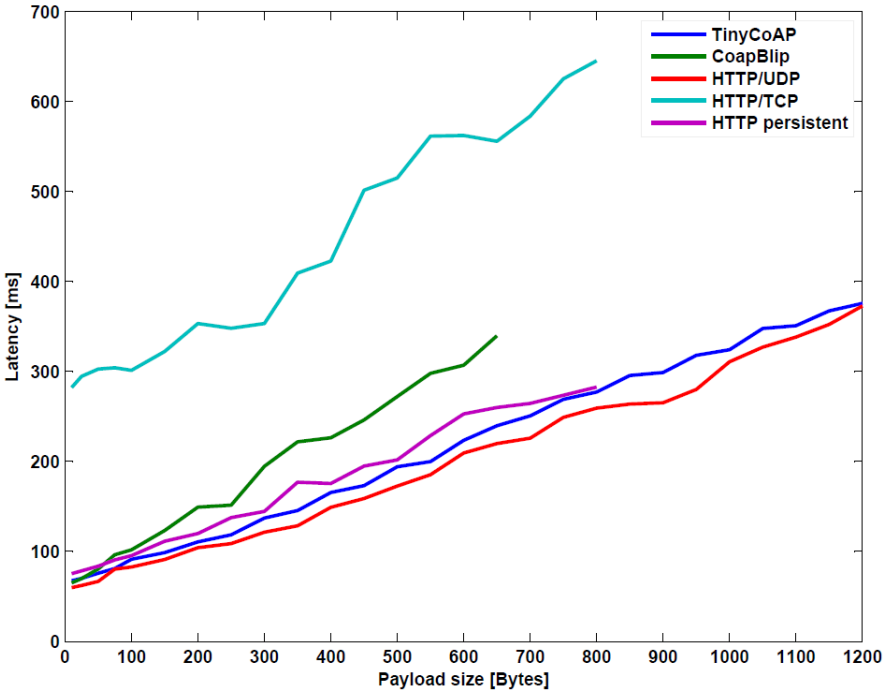


Figure 40 Latency evolution according to the payload size. The performance of HTTP increases significantly if a TCP persistent connection is used.

As expected, HTTP/TCP has the worst performance, the main reason is found in the latency introduced by the three-way handshake used by TCP to establish and close the connection. However, the negative impact of the handshake can be reduced using the same TCP connection for various HTTP transactions. The performance obtained by HTTP persistent confirms this aspect. As may be seen in Figure 40, a server that uses HTTP persistent is able to lower the latency and to make its trend closer to those of the fastest solutions.

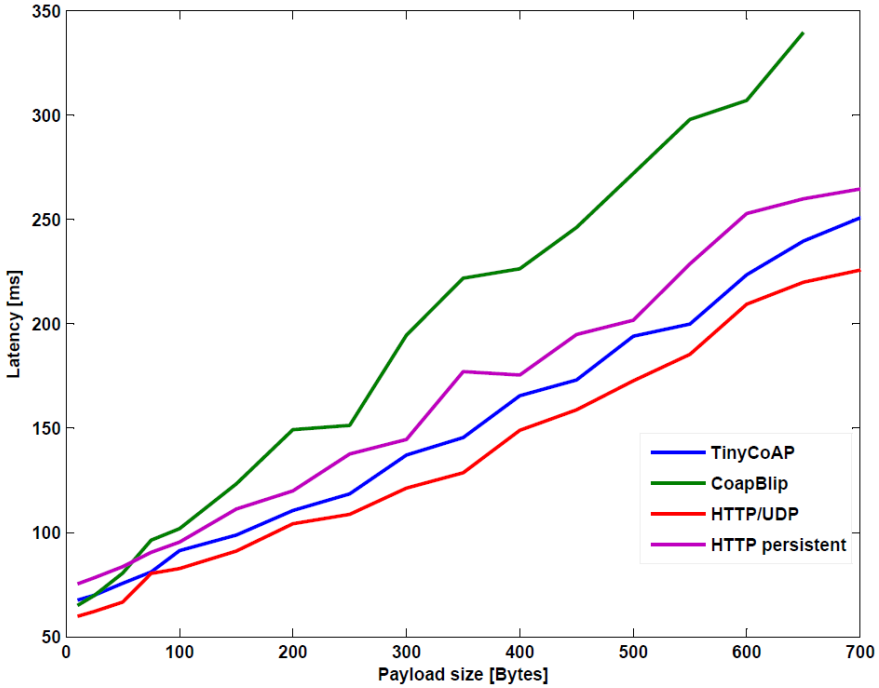


Figure 41 Details of the latency results. The better memory management of TinyCoAP allows the performance of CoapBlip to be improved.

The lowest latency trend is obtained by the HTTP/UDP solution. This solution implements a bare HTTP server able only to reply to GET requests. HTTP/UDP does not implement any reliability mechanism or HTTP logic. Therefore, it should be considered as a lower bound for latency.

TinyCoAP has a latency trend very close to that of HTTP/UDP, so the substantial growth of complexity of TinyCoAP is not reflected in a particular worsening of latency. TinyCoAP outperforms CoapBlip in terms of latency. The main reason is found in the enhanced RAM memory management implemented by TinyCoAP. The memory allocation used by CoapBlip causes a growth of the packet processing time and limits to 650 bytes the maximum payload size that it is able to send. Applications that use data aggregation or work with high payload sizes cannot be used in CoapBlip or with HTTP solutions using TCP. Concerning HTTP, the maximum allowed payload size is limited by the use of the TCP buffers that exhaust the RAM memory.

5.3.3. Energy Consumption

Figure 42 shows the results obtained in the energy consumption tests. The results are obtained measuring the energy consumed by a node when replying to ten consecutive requests. In fact, a fair comparison that accounts also for persistent TCP connections used in HTTP persistent requires to measure the energy consumed in more than one transaction.

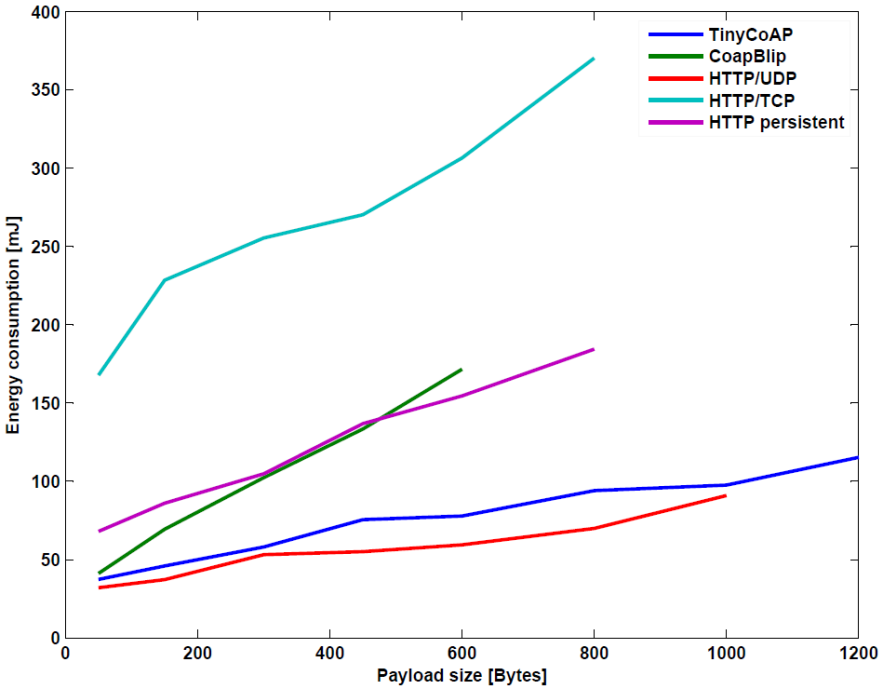


Figure 42 Energy consumption. TinyCoAP has a lightweight packet processing that allows the energy consumption approaching the trend of HTTP/UDP to be lowered.

The results of this test do not take into account the energy wasted by the radio chip for listening the channel. Consequently, our evaluation does not need to consider power-saving protocols for radio duty-cycling. We only measured the energy consumed for receiving, processing and sending a packet. As mentioned in section 5.1.2 the energy consumed by the CPU for packet processing is considerable. However, the energy consumed by the radio chip for receiving and sending a packet is still predominant over that used in packet processing. In this test, the consumption due to the radio chip has the same impact on the results of each implementation. 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, we ran tests sampling the energy consumption each 0.02 ms. The device used for these measures is the Agilent Technologies DC power Analyzer N67705A.

As expected, HTTP solutions based on TCP consume more energy than others. Once again, the message overhead caused by TCP proves to be costly for constrained networks. As seen in the latency tests, HTTP persistent improves the performance of HTTP/TCP. This is due to the use of a persistent TCP connection. However, its performance is still much worse than that obtained by TinyCoAP or HTTP/UDP. The management of TCP connections requires a high degree of complexity and the maintenance in memory of the connection state. Consequently, there is a growth in the energy drawn by the RAM memory for keeping these states.

CoapBlip has a performance comparable to that of HTTP persistent. The onset of the CoapBlip trend is lower than that of HTTP persistent and comparable with that of TinyCoAP or HTTP/UDP. However, the increase in packet size causes a sharp rise in the CoapBlip energy trend, which exceeds that of HTTP

persistent at 500 bytes. Once more, the mechanism implemented by CoapBlip to allocate and manage RAM memory proves to be unsuitable for constrained devices. Instead, TinyCoAP benefits from its different memory allocation mechanism. Furthermore, its different design has less impact on energy consumption. Similarly to that seen in the latency tests, TinyCoAP has a performance that is highly comparable to that of HTTP/UDP. This proves once again that TinyCoAP is able to minimize the consumption of resources.

5.3.4. Workload

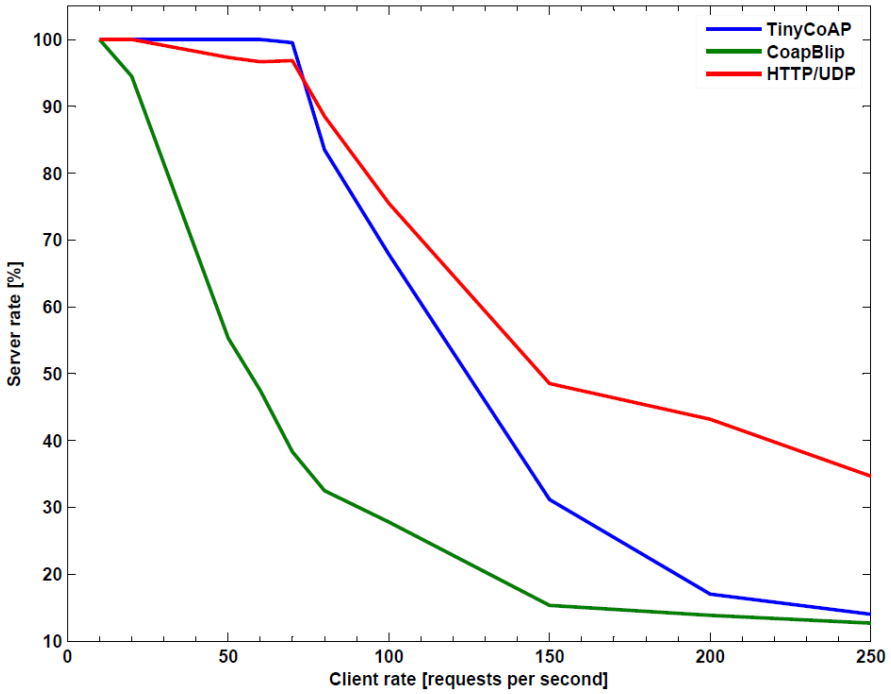


Figure 43 Number of requests handled by the server as a function of the client rate. The enhanced buffers of TinyCoAP allows to that to overcome the CoapBlip performance**.**

Comprehensive benchmarking requires a study of the effects that high workloads have on a server. In this study, we evaluate the percentage of requests that a server can handle as a function of the request rate of the client. We refer to the percentage of the handled requests with the term server rate. The test considers a client sending GET requests at a fixed rate. For each rate we run 100 tests. The resulting server rate is the average percentage of requests at which the server replies correctly in each test. The payload of the server response fills an IEEE 802.15.4 frame.

In this test, we evaluate only unreliable solutions. The presence of retransmissions does not allow maintaining constant the client rate. A client using CoAP reliability is forced to stop sending requests and start retransmitting. The server therefore would not be subjected to a constant workload. We avoid this problem using CoAP NON messages. The only reliability mechanism in use is that of the MAC layer.

We do not consider the use of HTTP/TCP. At present, the implementation of TCP does not allow accepting more than one connection at time. As a consequence, the rate with which a client sends requests is extremely limited. The average latency for HTTP/TCP considering a response sent in a full 802.15.4 frame is equal to 302 ms. Therefore, the maximum rate a client can have is estimated to 3 requests per second. We consider this rate too low for being compared with other solutions. Also the HTTP persistent has not been taken into account. In this solution, the rate at which a client sends requests cannot be controlled. It is the TCP congestion control algorithm that manages this rate. Figure 43 shows the average success rate of each considered solution.

The server rate performance is affected by the buffer size of the nodes and the mechanism used to access to the channel. A request or response could find the sending buffer full and, therefore, it would be discarded. This is more likely to happen when a client generates requests at a rate greater than that at which it can send them. Besides the processing time, the propagation time and the CSMA-CA mechanism are the main limitations of this rate. The propagation time accounts for the time needed to send the requests and receive the MAC acknowledgment. A client has to wait for the server to acknowledge the receipt of the frame before sending another. Therefore, if the packet generation time were lower than that taken for the CSMA-CA plus the propagation time, then the client would fill its buffer faster than it can empty it. According to [89] this time is estimated to 6.09 ms, which is comparable to the packet generation time relative to a client rate of 150 requests per second. This time, however, is only estimation. We expect that collisions are frequent in a congested wireless link. These imply MAC retransmissions and, consequently, a growth of the time needed to complete a request/response transaction. Packet collisions are also the main cause of the server rate degradation. These could happen when a node senses the channel idle during the turnaround time of its pair. Therefore, a request and a response could collide. This phenomenon is more likely to occur when the client rate is high. Furthermore, the channel would become quickly congested and therefore the probability that a node senses it busy would be high. The node gives up the transmission of a packet if it senses the channel busy more than the maximum allowed.

The explanations given above are valid for all the tested solutions. The cause of the different performance has to be found in the packet processing and memory allocation of each one. Regarding HTTP/UDP, its light and fast processing allows to this solution to achieve the best performance. The server is able to process fast the received requests and therefore augments the server rate. The HTTP/UDP processing also helps improving the client performance. The packets stay for less time in the buffer and, therefore, a new generated packet has more possibility to find space respect to other solutions. This is also the cause for the less pronounced drop that HTTP/UDP has respect to the CoAP based solutions.

In Figure 43 one may observe that when the request rate is higher than 80, the success rate of TinyCoAP and HTTP/UDP undergoes a pronounced drop. In fact, starting form this point the latency of a request/response transaction is comparable to the time taken by the client to generate and send a request. The probability of having a collision between a request and a response is therefore high. Starting from 150 requests per second, the drop is less sharp. The nodes are subjected to a workload close to its limits. The client rate can be augmented and the success rate will not lower as much as one may expect. As we explained

above, these client rates saturate quickly the buffer. The sending process of many requests fails and therefore they never leave the node. The number of requests that are sent correctly became stable. As a consequence the server rate shows small variations.

CoapBlip has the worst performance. The inefficient use of the RAM memory and the complex packet processing are the main limitations of its server rate. Moreover, the higher latency experienced by CoapBlip implies lower the point at which this become comparable to the client rate. As a consequence, packet collisions affect the server rate earlier than in the other solutions.

5.3.5. Reliability

The reliability of IP based Web communications is traditionally provided by the TCP protocol. However, application protocols that are bound to UDP should provide reliability by themselves. Providing reliable communications is of paramount importance, especially when using wireless links that are known to be prone to packet loss. As we already know, the IEEE 802.15.4 standard offers only hop-by-hop reliability by implementing packet retransmission at data-link layer. The lack of end-to-end reliability must therefore be compensated by implementing it at higher layers.

As anticipated, CoAP provides end-to-end reliability using CON messages through a simple stop-and-wait retransmission mechanism with exponential backoff. The value for the initial timeout is fixed to 2 seconds, while the maximum number of retransmissions is fixed to 4. Moreover, the CoAP definition allows this value to be changed according to the average RTT. Although not suggested in the standard, we recommend fixing a lower bound for the initial timeout. This timeout should not expire before the MAC layer reaches its maximum number of retransmissions. In [8], this number is fixed to 3, while the value of the timeout is not specified. TinyOS fixes this timeout to 512 symbols, which is equivalent to 8.192 ms.

In order to ensure a fair comparison, we should use the same values of initial timeout for all the solutions and avoid any randomness in the calculation of the subsequent values. We adopt the initial timeout value specified in the RFC 6298 [90]. This value is fixed to 1 second and is big enough to ensure that the CoAP first retransmission is sent after the last one at MAC layer. The maximum number of retransmissions is fixed to 4, while the maximum value for the TCP retransmission timer is fixed to 16 seconds. This value corresponds to the maximum retransmission timeout reached in CoAP communications.

The evaluation of the mechanisms that each solution offers to provide reliability is performed by considering data transfers in an IEEE 802.15.4 link under a Rayleigh fading model. The basis for the calculations required by this model to determine the packet loss is reported in [91]. In this work, the authors studied the impact of the signal to noise ratio (SNR) on the PHY-level packet loss rate of an 802.15.4 link. As reported in [91], the packet loss rate P is given by:

$$P = 1 - (1 - S)^{2m} \quad (1)$$

where S is the symbol error rate and m the length in bytes of the packet. The corresponding packet length in symbol is 2 m. The symbol error rate is strictly related to the bit error rate B. The latter contains the relationship with the SNR:

$$B = \frac{1}{2} - \left(1 - \sqrt{\frac{SNR}{1+SNR}}\right) \tag{2}$$

In this way, given an SNR, we can easily calculate the related packet error rate. However, this error rate does not correspond to that of the application layer, but to that of the PHY layer. The calculation of the error rate seen by the application layer requires one more step. The probability C of having an erroneous packet at application layer should take into account the retransmissions done at the MAC layer, and is given by:

$$C = P^{(r+1)} \tag{3}$$

where r is the maximum number of retransmissions allowed by the MAC layer.

However, the formula in Equation (4) gives the application layer packet error rate only for non-fragmented packets. Should a packet be fragmented, this error rate changes according to the number of fragments. Thus, if f fragments compose a packet, the CoAP packet error rate is given by:

$$C_f = \sum_{j=1}^{f} C \times (1-P)^{j-1} \tag{4}$$

The maximum number of fragments that Blip allows is fixed to 12. It should be pointed out that all the formulas are valid only for single or fragmented packet that occupeis the entire space available in 802.15.4 frames. As regards the presence of the PHY header, the frame can reach a maximum size of 133 bytes. According to Equation (5), the error rate for a non-fragmented packet is obtained for f equal to one.

The tests done for reliability cover four different SNR: 1 db, 1.5 db, 2 db and 2.5 db. In Table 5 we report the corresponding application layer packet error rates.

According to the number of fragments and the relative error rate, we calculate the average goodput of a data transfer. This value is calculated according to the goodput obtained in 100 consecutive transactions. Unlike the previous tests, the reliability evaluation does not take into account the HTTP/UDP solution. In fact, none of these protocols provide any reliability mechanism, the definition of which is beyond the scope of this thesis.

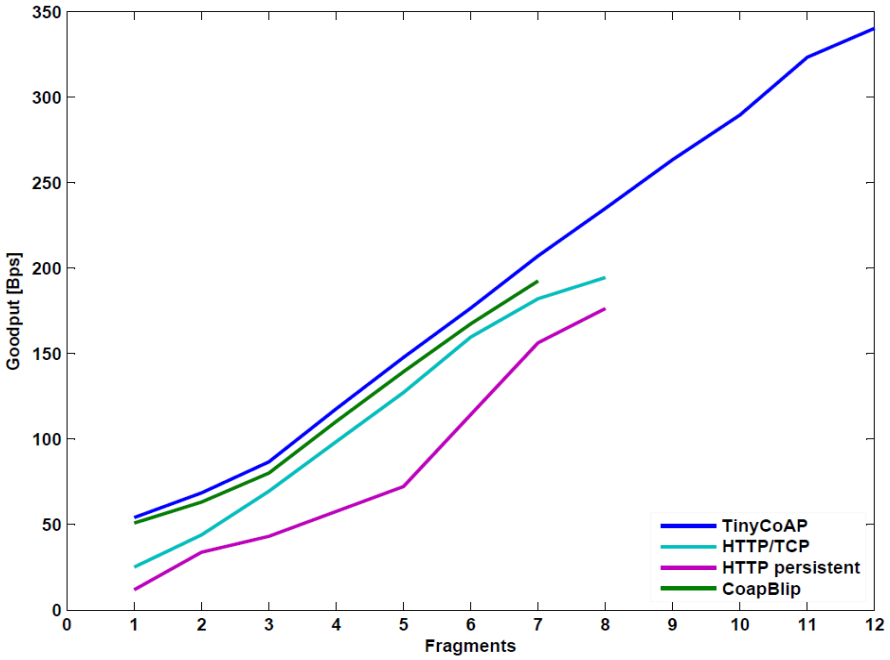
Errors are introduced by comparing the application layer error rates with a sequence of random numbers. Should the random number be less than or equal to the error rate, the CoAP or HTTP message received by the server is discarded at UDP layer. In this way, the application layer will consider that an error has occurred in the communication and will retransmit the packet. The same sequence of random numbers is used for each solution tested in order to ensure that in each experiment there are the same numbers of errors. Moreover, the errors will occur in the same order.

Table 5 Application layer packet error rate. These values refer to a single or fragmented packet that occupies the entire space of a 802.15.4 frame. The value for f = 1 refers to a single non-fragmented packet. A packet can consist of a maximum of 12 fragments.

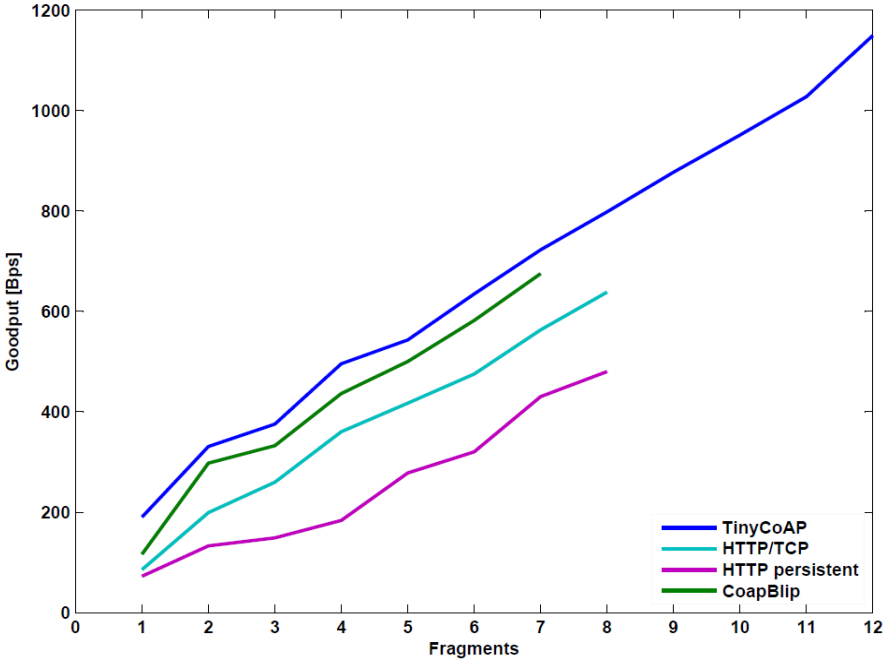
f	2.5 db	2 db	1.5 db	1 db
1	1.23%	5.20%	16.78%	39.64%
2	2.05%	7.91%	22.82%	47.82%
3	2.59%	9.33%	24.99%	49.51%
4	2.96%	10.08	25.77%	49.86%
5	3.20%	10.46%	26.06%	49.94%
6	3.37%	10.67%	26.16%	49.95%
7	3.47%	10.77%	26.19%	49.95%
8	3.55%	10.83%	26.21%	49.95%
9	3.59%	10.86%	26.21%	49.95%
10	3.63%	10.87%	26.21%	49.95%
11	3.65%	10.88%	26.21%	49.95%
12	3.66%	10.88%	26.21%	49.95%

Figure 44 shows the goodput in bytes per second obtained for each SNR level. TinyCoAP surpasses the other solutions in all cases. CoapBlip has a goodput trend that is lower but close to that of TinyCoAP. In this case, the differences between the implementations of the retransmission mechanisms of TinyCoAP and CoapBlip are minimal and negligible. The different performances are therefore due exclusively to the differences in packet processing, as previously explained.

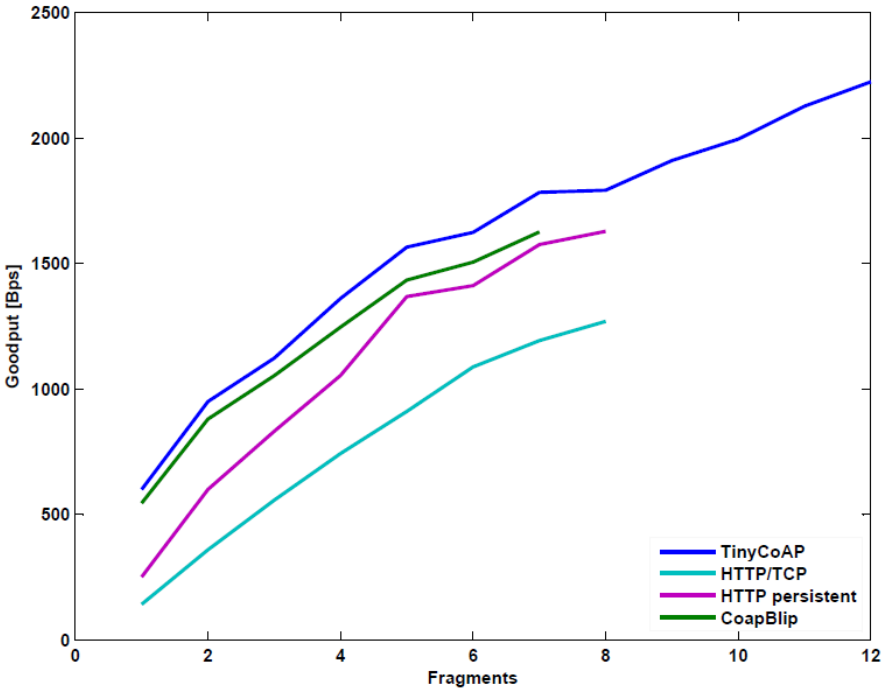
Figure 44 Goodput evolution in a channel under the Rayleigh fading model. (a) Goodput with SNR of 1 db; (b) Goodput with SNR of 1.5 db; (c) Goodput with SNR of 1.5 db; (d) Goodput with SNR of 2.5 db. The reliability mechanism implemented in CoAP yields a good performance. In TCP, the initial retransmission timeout is resettled after closing each connection, providing a better performance in channels with low SNR.



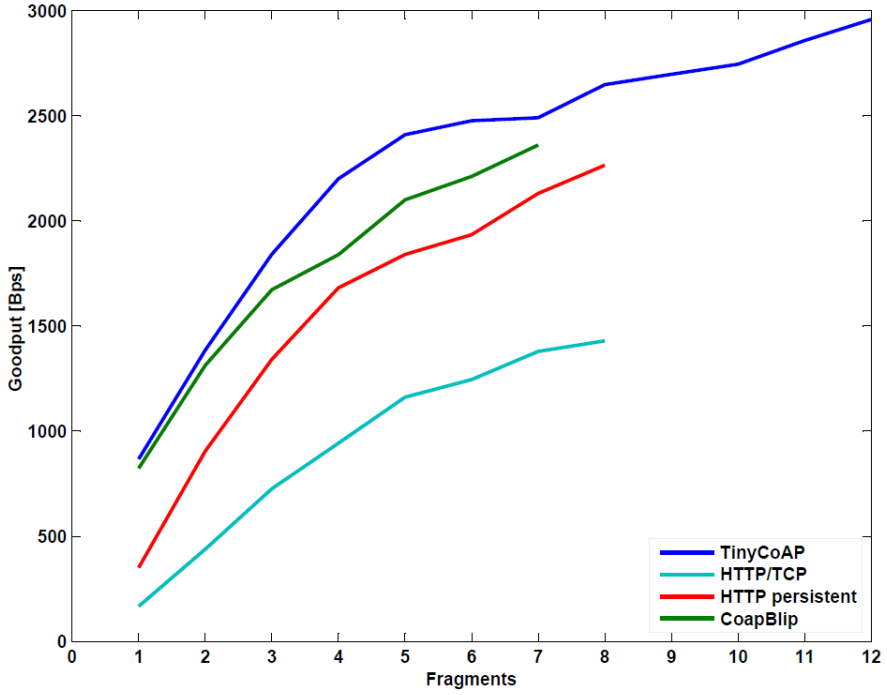
(**a**)



(**b**).



(**c**)



(**d**)

Once again, the solutions for which HTTP is adopted yield the worst performance. HTTP persistent has a goodput trend higher than that of HTTP/TCP in channels having a high SNR. However, when the SNR is

lower and the consequent error rate increases, the HTTP/TCP trend exceeds that of HTTP persistent. This can be explained by considering the behavior of the algorithm for the retransmission timeout (RTO) calculation specified in [90]. In fact, HTTP persistent and HTTP/TCP cannot measure the RTT of communications affected by retransmissions. The RTT is used by the algorithm to calculate the RTO and should be relative only to communications unaffected by retransmissions. However, the RTT of a retransmitted communication that has been acknowledged can only be measured by using the timestamp option of TCP. In our experiments, we employ a TCP version that does not use this option. Consequently, the behavior of HTTP/TCP and HTTP persistent could differ significantly when the channel presents a low SNR and retransmissions are more frequent. Should two successive communications in HTTP persistent be affected by retransmissions, the RTO would not be recalculated but rather augmented exponentially until a valid RTT measure can be taken. Alternatively, HTTP/TCP would be able to measure the RTT and calculate the RTO each time it establishes a new connection. Thus, when there are successive communications affected by retransmissions, the HTTP persistent would undergo a latency greater than that of HTTP/TCP.

In conclusion, although CoAP is bound to the unreliable UDP protocol, the reliability mechanism it provides shows very good behavior and performance, at least in channels affected by Rayleigh fading.

5.4. Conclusions and Contributions

In this chapter we have presented our original library for TinyOS, which we have called TinyCoAP. We have illustrated its design principles, described its implementation and presented its evaluation. Furthermore, we have compared it to the CoAP implementation distributed with TinyOS, called CoapBlip. The differences between the design of TinyCoAP and CoapBlip have been explained.

All the solutions have been discussed and evaluated in a real TinyOS based 6LoWPAN network. We have measured the amount of memory occupied at compile time, the latency experienced by a client when retrieving information from a server, and the energy consumed when replying to the client. We have also evaluated the performance of a server under high workloads. Finally, we have evaluated the average goodput obtained in an 802.15.4 link under a Rayleigh fading model. In particular, the purpose of this test was to evaluate the reliability mechanism provided by CoAP. In our tests, we have uses HTTP with three different solutions for the transport layer. We have considered the use of UDP, TCP and persistent TCP connections. We have referred to each of these solutions as HTTP/TCP, HTTP persistent and HTTP/UDP.

HTTP/UDP is only able to reply to a simple HTTP message without implementing any logic behind it. The purpose of this implementation was to show a lower bound of the performance and to demonstrate how that of TinyCoAP and other solutions are close to it. HTTP/UDP had the best performance in terms of latency and energy. Although TinyCoAP is a more complex implementation, its performance is very similar to that of HTTP/UDP. This proves that its design is able to minimize the impact on the constrained resource of WSNs nodes while achieving good performance.

TinyCoAP has provided the best performance in the rest of the considered parameters. In particular, TinyCoAP has shown a significant improvement in performance compared with CoapBlip. The performance of CoapBlip was limited by the adoption of dynamic RAM memory allocation and the use of an external C library. Thanks to a design compliant to the TinyOS programming model and to the static allocation of memory, TinyCoAP have solved the problems encountered in CoapBlip, thereby allowing CoAP to realize its full potential. TinyCoAP allows applications to work with a higher payload size than that achieved by CoapBlip or HTTP. This permits TinyCoAP to work with data aggregation, software update of nodes or video and audio applications that generate a high amount of data. Regarding HTTP, the performance obtained by HTTP persistent is an improvement on that provided by the traditional use of HTTP/TCP, which in any case is worse than that obtained by TinyCoAP.

In conclusion, TinyCoAP offers a lightweight, complete and flexible CoAP-based solution for implementing the Web communication paradigm in TinyOS based WSNs. TinyCoAP solves the problems experienced in CoapBlip, and is able to enhance performance significantly and to minimize the resource consumption.

From the results of the research presented in this chapter derives the paper *TinyCoAP: A novel Constrained Application Protocol (CoAP) Implementation for Embedding RESTful Web Services in Wireless Sensor Networks Based on TinyOS* that is published in the Journal of Sensors and Actuator Networks [P2].

The implementation of TinyCoAP is distributed as open-source library at http://sourceforge.net/projects/tinycoap/.

6. CoAP Proxy

6LoWPAN and CoAP allow the IoT and Web worlds to be closer than ever. However, they are still too different to be easily integrated and interconnected. IoT and the Web, in fact, have different physical characteristics and are based over similar but different standards. In this sense, a Web application would still use HTTP and IPv6 to access an IoT device instead of using CoAP and 6LoWPAN. 6LoWPAN, in fact, is a standard thought to add IP capabilities to WSN and exploit IPv6 characteristics such as neighbor discovery and the large address space. However, the difference between the network links used in IPv6 and 6LoWPAN architectures requires the presence of a gateway to establish end-to-end communication between two endpoints using these protocols. Furthermore, CoAP is still under standardization and its diffusion, therefore, is very limited. For the same reason, Web applications with a dual HTTP-CoAP stack are not diffused. Therefore, it is of paramount importance the presence of systems able to interconnect Web applications with IoT devices.

Motivated by the exposed above, in this thesis we design a CoAP proxy. It permits Web applications to transparently access the resources hosted in IoT devices based on CoAP, which we refer to as CoAP devices. Its main function is to adapt the different protocol stacks used by Web applications and CoAP devices (Figure 45). The CoAP proxy is designed to be located at the border of the 6LoWPAN WSN containing the CoAP devices, which enable the proxy to work also as 6LoWPAN edge router of the WSN. Moreover the proxy performs the function of CoAP gateway to interconnect disjointed CoAP networks. A graphical representation of the network architecture in which the CoAP proxy can be used is shown in Figure 46.

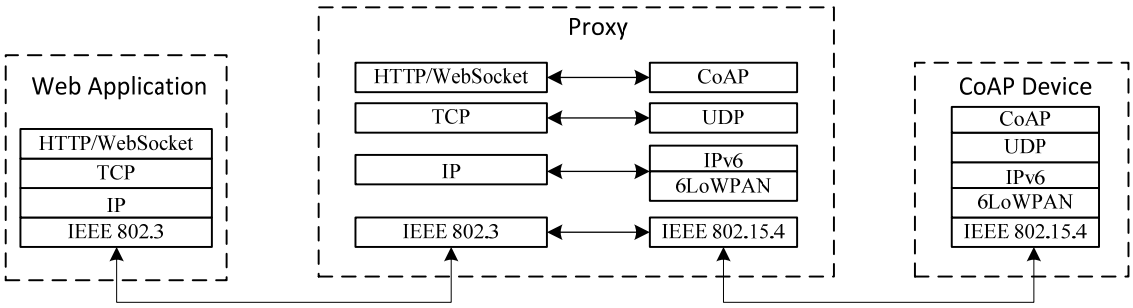


Figure 45 Protocol Stack. The CoAP proxy allows adapting the protocol stacks of Web applications and CoAP devices

The CoAP proxy is designed to provide support to applications that need to continuously retrieve data from the WSN. Traditionally, the HTTP long-polling technique has been used in these applications. However, it could result inefficient in this scenario. The use of HTTP long-polling, in fact, forces Web applications to query constantly the CoAP proxy to receive data from the WSN. This could cause an excessive communication overhead and a consequent increase of latency and network traffic [92]. To overcome these problems, we include the WebSocket protocol [93] in the CoAP proxy design. WebSocket aims at providing a bidirectional communication channel using a single TCP connection, which allows the CoAP proxy to efficiently support long-lived communications. To ensure compatibility with the largest number of Web

applications, the CoAP proxy also support HTTP long-polling. Furthermore, the availability of a dual HTTP long-polling/WebSocket stack allows the CoAP proxy to adapt to different application requirements.

We demonstrate the effectiveness of our design by a performance evaluation in a real WSN. This evaluation is performed in terms of latency and memory consumption. The CoAP proxy is evaluated considering long and short lived communications established between the Web application and a CoAP device. In short-lived communications the Web application requests to the CoAP proxy a resource hosted in the CoAP device. The proxy replies immediately with the representation of the resource. In this case, the communication ends after the Web application receives the response. Long-lived communications are used when the Web application needs to be notified about the changes of a resource over the time. In this case, the transmission is continuous and ends only when one side of the communication explicitly close it. In both situations, the performance of the CoAP proxy is evaluated according to the protocol used by the Web application to access the WSN.

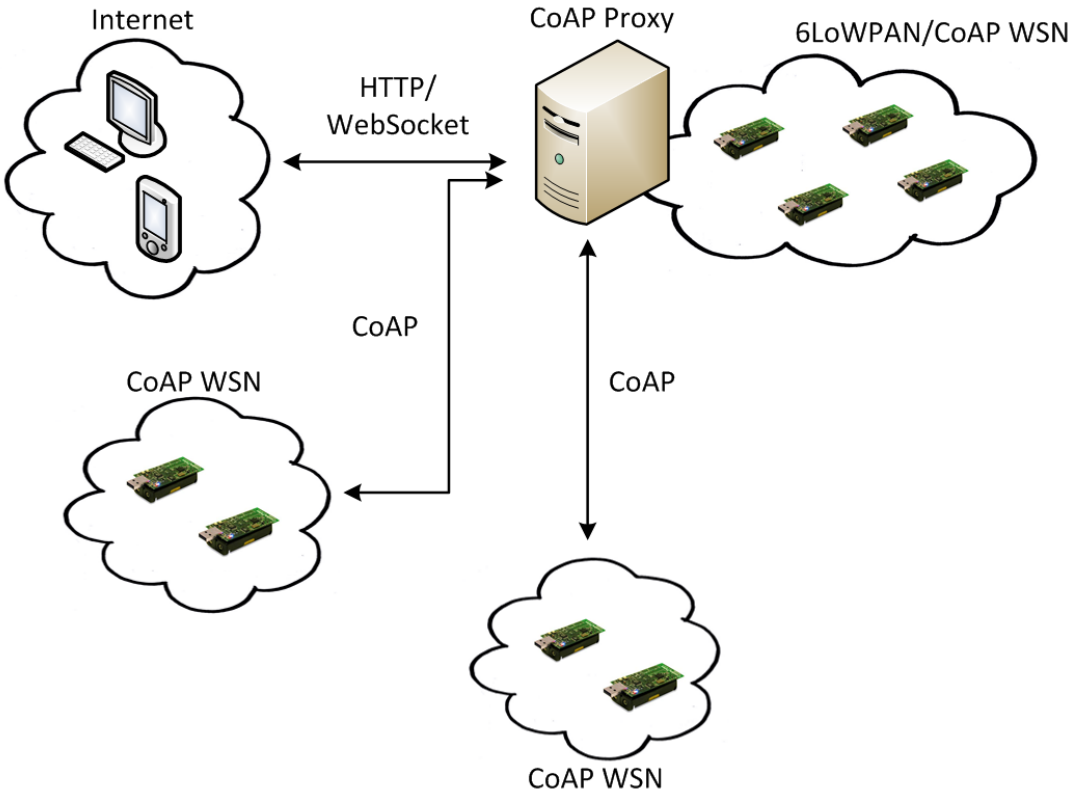


Figure 46 Network architecture. The CoAP proxy also has the functions of 6LoWPAN edge router and gateway to interconnect disjointed CoAP networks.

6.2. Design Considerations

. Should the proxy communicate with a Web application, the communication pattern has to consider the possibility that the data exchange could be long-lived. In this case, traditional communication patterns such as the HTTP long-polling could be inefficient. Instead, the interaction between the CoAP proxy and a CoAP

device has to consider the limited bandwidth that characterizes the wireless link. In this case, a request/response model could lead to a wasteful use of bandwidth. The rest of this section focuses on the design considerations for the communication patterns used by the CoAP proxy to communicate with the CoAP device and the Web application. Furthermore, we discuss the translation process followed by the CoAP proxy to map between the URIs [94] used by Web applications and CoAP.

6.2.1. Communication pattern between the CoAP proxy and the CoAP device

The CoAP proxy uses the observe option to receive updates from the CoAP device. In our design, the CoAP proxy is the only observer registered to the CoAP device. Web applications establish the observe relationship only with the proxy. Consequently, the WSN network traffic is significantly reduced allowing minimizing the bandwidth usage of the wireless link. Moreover, the CoAP device is subject to less work and it is able to minimize the resource consumption. The observe relationship between the CoAP proxy and the CoAP device is established when the first Web application requests it.

6.2.2. Communication pattern between the Web application and the CoAP proxy

The CoAP proxy is designed to support long-lived transmission of data. This permits to use the proxy in WSN applications such as industrial monitoring or e-health. HTTP has not been originally designed to work with long-lived applications. However, techniques that simulate this behavior can be applied. In this sense, the HTTP polling has been largely used to support long-lived communications over the Web. In this technique the Web application sends periodical requests to a server to obtain the data. If this is not available the server would send an empty response. Polling is only suitable when the message delivery interval is constant and known. Furthermore it should be long enough to ensure that the overhead would not increase latency or network traffic. A more efficient solution is provided by HTTP long-polling. In HTTP long-polling the server holds the client request until new data is available or the TCP timeout expires. This solution reduces significantly the number of useless messages that are interchanged in HTTP polling. Although HTTP long-polling solves some of the problems that affect HTTP polling, the overhead inherent to sending periodical HTTP requests still remain unsolved.

An effective solution for long-lived communications is provided by the WebSocket protocol, which establishes full-duplex and bidirectional communications over a single TCP socket. With WebSocket a Web application can receive data from the CoAP proxy avoiding establishing multiples HTTP connections. When new messages are available the CoAP proxy sends them over the existing connection.

A WebSocket communication has three phases: the opening and closing handshakes and the data transfer. A WebSocket connection can be initiated only after a TCP connection has been established. A client sends a WebSocket handshake request to initiate the opening handshake. This is equivalent to an HTTP upgrade request as specified in the Upgrade and Connection header fields of the handshake request. The request message contains the version of the protocol and the hostname of the server. The handshake request contains also the HTTP method and the URI of the resource as shown in the following example:

```
GET /temperature HTTP/1.1
Host: proxy.com
Upgrade: Websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Version: 13
```

To establish a WebSocket connection, the server has to prove to the client that it received the handshake request. The server concatenates the content of the Sec-WebSocket-Key header field with a Globally Unique Identifier (GUID) [95] to prove that. The server returns the hash of this concatenation in string format in its handshake response. This is contained in the Sec-WebSocket-Accept header field as shown in the following example:

```
HTTP/1.1 101 Switching Protocols
Upgrade: Websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
```

The server response includes also an HTTP status line. The WebSocket connection is only established when the status code is 101. The Upgrade and Connection fields are used to complete the HTTP upgrade.

Should the handshake phase be successful, the bidirectional channel is established and the data transfer phase can start. Data is sent in data frames defined by the WebSocket standard. The base data frame is formed by an option code, a payload length and data. The option code is used to interpret the data. A data frame can be sent either by the client or the server. The WebSocket connection is closed after a closing handshake. This can be initiated by any of the endpoints by sending a WebSocket close frame. The other endpoint also replies with a close frame. After the closing handshake, the endpoints close the TCP connection. Figure 47 shows the messages interchanged between a client and a server to establish a communication using the WebSocket protocol.

In short-lived communications, HTTP long-polling may have better performance than WebSocket. In HTTP long-polling, in fact, the interaction requires only the interchange of a HTTP request and a response message. Instead, in the WebSocket case the overhead introduced by the handshake phases could increase latency.

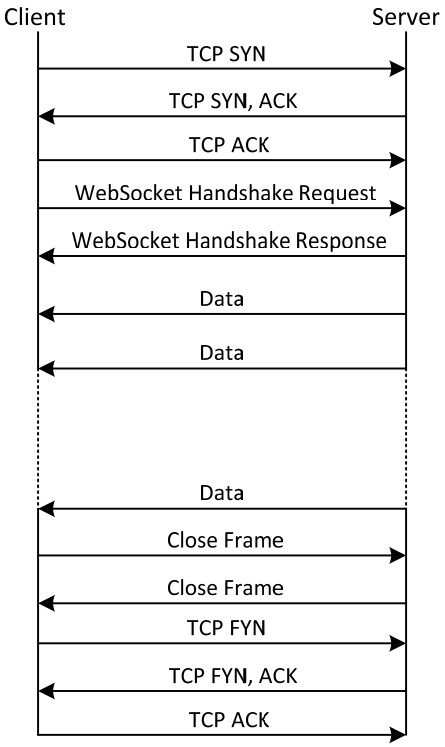


Figure 47 WebSocket protocol. The WebSocket communication consists of an opening handshake, a data transfer and a closing handshake.

6.2.3. Protocol Translation

Interconnecting Web applications and CoAP devices require providing a solution to translate the protocols involved in the communication. The methods, response codes and content-type supported by HTTP, WebSocket and CoAP are equivalent and allow a straightforward and transparent mapping. They all indentify resources using the syntax defined by the URI standard. The path and authority parts are equivalent while the scheme varies depending on the protocol used. Our proxy supports the "http" and "ws" schemes defined by the HTTP and WebSocket protocols respectively. They are translated directly to the "coap" scheme. The authority part of the URI contained in the Web application request corresponds to the combination of the IP address and TCP port of the proxy. The authority part of the CoAP device that hosts the requested resource is derived after the translation process of the URI path, which identifies the resource target of the request.

At present, the CoRE working group is defining best practices for HTTP-CoAP mapping [96]. The document includes several proposals that seek to establish a common URI format to be used in HTTP request. However, these proposals do not define a shared format able to identify the targeted resource or the request to establish an observe relationship.

In this paper, we propose and adopt a novel format for the URI path. Its structure is derived from the Core Resource Directory (RD) [97] specifications. The RD is a Web links repository that allows hosting the information related to CoAP devices and the resources they expose. We include the RD repository on our

proxy and use the information hosted in it to map the HTTP and the CoAP URI paths. We propose and adopt the following format:

"observe" +"/" + "domain" +"/"+ "target_node" +"/ "+ "target_resource" (1)

The first part of the format is optional and indicates the willingness to establish an observe relationship. A short-lived request must be sent without the *observe* part. The *domain* is included to support WSNs with complex topologies. A *domain* is considered as a logical grouping of nodes [97]. For example, a WSN could be deployed over different floors of a building; in this case we can group the nodes of the same floor in the same *domain*. The *domain* part is optional. The name of the CoAP device and that of the requested resource are expressed as *target_node* and *target_resource* respectively. A *target_node* is unique within a domain while a *target_resource* is unique in a *target_node*. As an example we consider a node, which we refer as node_1, placed on the first floor of a building that measures temperature and exposes it as a resource. The resulting URI path to observe this resource is shown as follows:

observe/floor_1/node_1/temperature

The complexity of the URI path composition is hidden from the Web application. It learns the URI performing a resource discovery on the CoAP proxy. CoAP defines a URI for this purpose which is called ".well-known/core" [19]. Any request directed to that URI results in a response containing the resources offered by the proxy. The CoAP proxy hosts a repository that contains the information related to the resources and the CoAP devices hosting them. The repository is designed following the Core Resource Directory (RD) specifications [97]. The CoAP proxy uses the information hosted in the RD to assemble the URI queried by a Web application. The URI format, as shown in (1), differs from that used by the CoAP proxy or by another node to interact directly with the CoAP device and query the same resource. The only part that is equivalent is the *target_ resource*. In a direct interaction, the observe relationship is established including the observe option in the CoAP header and not in the URI path. Furthermore, the *domain* and *target_node* are replaced by the combination of the IPv6 address and UDP port of the server. The *domain* part could be introduced by the proxy and could not correspond to any composition of the real CoAP URI. The CoAP proxy translates the URI in (1) into the equivalent used for direct interaction as shown in Figure 48.

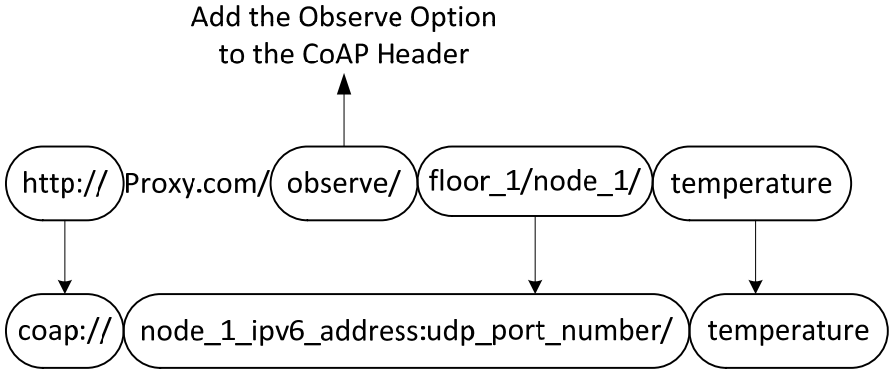


Figure 48 Translation of the HTTP URI into a CoAP one. The URI used by WebSocket has the same format of the HTTP URI except for the scheme. WebSocket used the "ws://" scheme.

6.3. Proxy Design and Implementation

A CoAP proxy can be classified in three categories depending on its role in the network architecture [19]. In particular, a proxy is classified as a forward-proxy when it performs requests on behalf of the client. Instead, a proxy that behaves as if it were the real server is classified as a reverse-proxy. Finally, a proxy that translates between different protocols is defined as a cross-proxy.

In our design process we choose to group all of these roles inside the same proxy. They are, in fact, complementary and not opposite. As an example, a reverse-proxy could be at the same time a cross-proxy since it may have to translate the received request. Furthermore, the concept of reverse and forward proxy can co-exist in the same device. We consider these roles as services offered by the CoAP proxy more than strict categories. We implement the reverse-proxy service to establish end-to-end data transactions between a Web application and a CoAP device. The forward-proxy is implemented to allow proxing operations inside the WSN. End-to-end connectivity between disjointed CoAP networks can be provided by both services.

The CoAP proxy is composed by three modules which, at run-time, are executed in separate processes. UNIX sockets are used to establish communication between them. The first module is the Lighttpd server [98]. It is in charge of receiving the incoming HTTP long-polling requests. The 6LoWPAN interface, instead, allows the proxy to communicate with the WSN. Finally, the main proxy module implements the core functionalities provided by the CoAP proxy. This module is composed by the following components:

- CoAP module.
- Web server module
- Cache.
- Resource Directory.

Figure 49 shows the composition of the proxy.

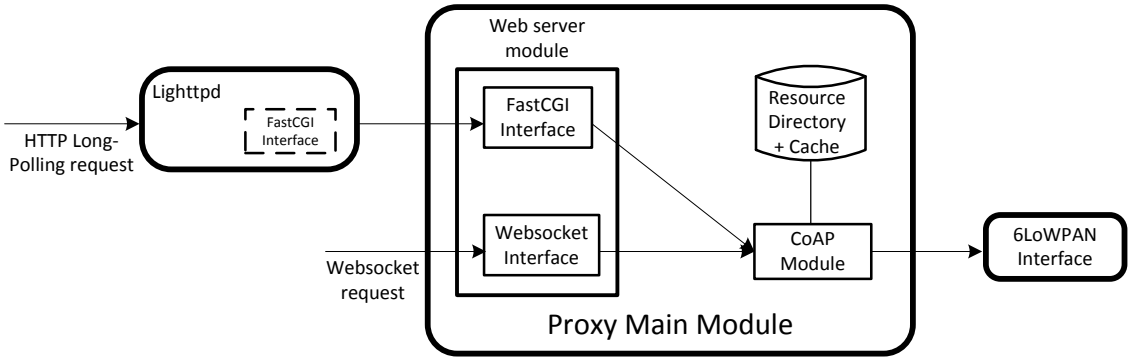


Figure 49. CoAP proxy design. The CoAP proxy is composed by three modules.

6.3.1. 6LoWPAN Interface

As previously mentioned, the CoAP proxy has also the function of 6LoWPAN edge router. This is in charge of forming the 6LoWPAN network and route between the 6LoWPAN network and other IP networks.

The edge router provides the IPv6 network prefix to nodes. These learn the prefix using the 6LoWPAN neighbor discovery [99]. In this mechanism a node sends a broadcast message at start-up, which is called router solicitation message. The edge router replies with a message containing the IPv6 network prefix, which the node uses to auto-configure its address. The node learns the address of the edge router from the same message.

In our implementation, the address of the edge router corresponds to that of the CoAP proxy. Therefore a CoAP device does not need to use any proxy discovery mechanism. This allows reducing the number of messages interchanged by the CoAP device and the CoAP proxy and therefore saving the device's battery power.

The CoAP proxy is connected to the WSN via an 802.15.4 interface with a 6LoWPAN layer. 6LoWPAN functionalities are provided by Blip. Blip is also embedded in CoAP devices to enable IPv6 networking.

6.3.2. Lighttpd Module

As previously mentioned, we include WebSocket and HTTP long-polling in the CoAP proxy. The module responsible for handling WebSocket requests is integrated in the main proxy module while that of HTTP long-polling is split in two parts. In this sense, an HTTP server is in charge of establishing the HTTP connection while the main proxy module performs the relative URI translation process. There are several implementations of HTTP servers that already exist and are commonly used. Between these, we opt for Lighttpd. It has, in fact, a lower memory footprint and requires less CPU usage respect to other solutions. Both aspects are of paramount importance to reduce the resource consumption. The CoAP proxy, in fact, could be embedded in constrained hardware where the minimization of the resource consumption is essential.

6.3.3. Main Proxy Module

The main proxy module provides the core functionalities of the CoAP proxy. As mentioned before, this module includes the RD repository, cache memory and the CoAP and Web server modules. The translation process of the HTTP and WebSocket URIs to CoAP ones is implemented in this module. In particular, both the Web server and CoAP modules perform this process. The CoAP module is also responsible for handling the observe protocol and maintains the RD and cache memory. The Web server module includes the interfaces needed to receive WebSocket requests and to communicate with the Lighttpd server.

The composition and the functions of the main proxy module are detailed in the rest of this section.

6.3.3.1. Web Server Module

The Web server module is one of the essential building blocks of the main proxy module. It allows Web applications to communicate with the CoAP proxy. Its main role is to establish the connection between them and handling the incoming requests. HTTP long-polling and WebSocket are implemented as separate modules. As mentioned before, the Lighttpd module is responsible for receiving the incoming HTTP long-

polling requests. The Web server module and Lighttpd interact using the FastCGI protocol [100]. The use of FastCGI has been preferred to that of the simple CGI protocol. FastCGI has better performance respect to CGI and it is able to reduce the process overhead. It uses long-lived processes to handle multiple requests instead of creating new processes for each request as done by CGI.

The requests received by the Web server module are stored in a data structure, which we refer to as *connection_t*. This contains the information related to the connection established by the Web application with the CoAP proxy. Furthermore, *connection_t* stores the CoAP PDU resulting from the translation process of the URI. The *connection_t* structure is composed as follows:

```
typedef struct {
       int type;
       char *path;
       void *data;
       coap_pdu_t *pdu;
} connection_t;
```

Type indicates the protocol used by the Web application. This could be Websocket or FastCGI if the connection has been established through HTTP long-polling. *Path* contains the URI targeted by the request. Finally, the socket number of the connection is stored in *data* while *pdu* contains the CoAP message created after the translation process.

The correct translation of the URI contained in HTTP and WebSocket messages requires checking its validity and correctness through the analysis of the scheme and path parts. This can be considered as the first step of the translation process. At this level, it focuses on verifying the presence of the *observe* part in the URI and the existence of the *domain* and *target_node* in the RD. Instead, the validity of the *observe* request as well as the *target_resource* are verified by the CoAP module. This is subsequent to the validation of the *target_node*. The structure of the RD is presented and discussed in the next section. Algorithm 1 illustrates the procedure used by the Web server module to accept and validate a Websocket request. The algorithm used for handling HTTP long-polling request is similar except for the fact that the TCP and HTTP connection are established by the Lighttpd server. In this case, the Web server module receives the information about the Web application and the URI through the FastCGI protocol.

Algorithm 1: The Web server module receives a Websocket request, checks its validity and passes it to the CoAP module.

Input: Websocket request from Web client.
Output: node entry from RD.
for *each incoming Websocket request* ***do***
    accept TCP connection;
    URI ← *Websocket_do_handshake(socket);*
    if *URI contains "observe/"* ***then***
        observe ← *1;*

```
        remove "observe/" from URI;
    end if
    node_id ← get_node_id(URI);
    node_entry ← get_node_entry_from RD(node_id)
    connection ← create a connection_t structure;
    pass the requests to CoAP module (connection, node_entry, observe);
end
```

6.3.3.2. Resource Directory

Direct resource discovery using the *.well-known/core* URI is a useful mechanism only when the discovery is relative to the WSN. It becomes inefficient when a device or application external to the WSN wants to learn the resources hosted in its nodes. In this context, the use of an RD entity simplifies this task. The RD hosts the description of the resources provided by the WSN nodes. The resource description in CoAP is achieved using the Core link format [101] that is equivalent to the Web linking [102] used by HTTP. The WSN nodes are responsible for registering and keeping updated their entries in the RD. The RD provides the interfaces for registering, updating and deleting entries as well as for performing resource discovery. The entry point of the RD is the *.well-known/core* URI.

We organize the RD hierarchically so that to each node correspond a single entry. The resources are linked to the corresponding node entry. In fact, a single node could offer several resources. Therefore, indexing the RD using resources instead of nodes could lead to an intricate structure that complicates the resource discovery and the translation of the URI. Using the node description as index allows simplify the RD structure and the operation that are performed on it. The node description includes the node name and type, the IPv6 address and UDP port and optionally the domain. The node name and domain correspond to the *target_node* and *domain* part of the URI as shown in (1).

The RD is designed as a tree-structure where the node description is placed at the top and the resource descriptions are the branches of each entry. The resource description includes the path and the semantic type of the resource, the estimation of the payload size and a flag to indicate if it is observable or not. The resource path corresponds to the *target_resource* part of (1). The RD structure is shown in Figure 50. To simplify the management of the resources we add an extra layer at the bottom of the resource description. The purpose is to store the lists of observers of each resource. The cached response is also linked to the corresponding resource entry. This allows simplifying the use of caching. It is, in fact, strictly related to the resource and separating its representation from the resource entry may complicate its use. Cache is explained in the next section.

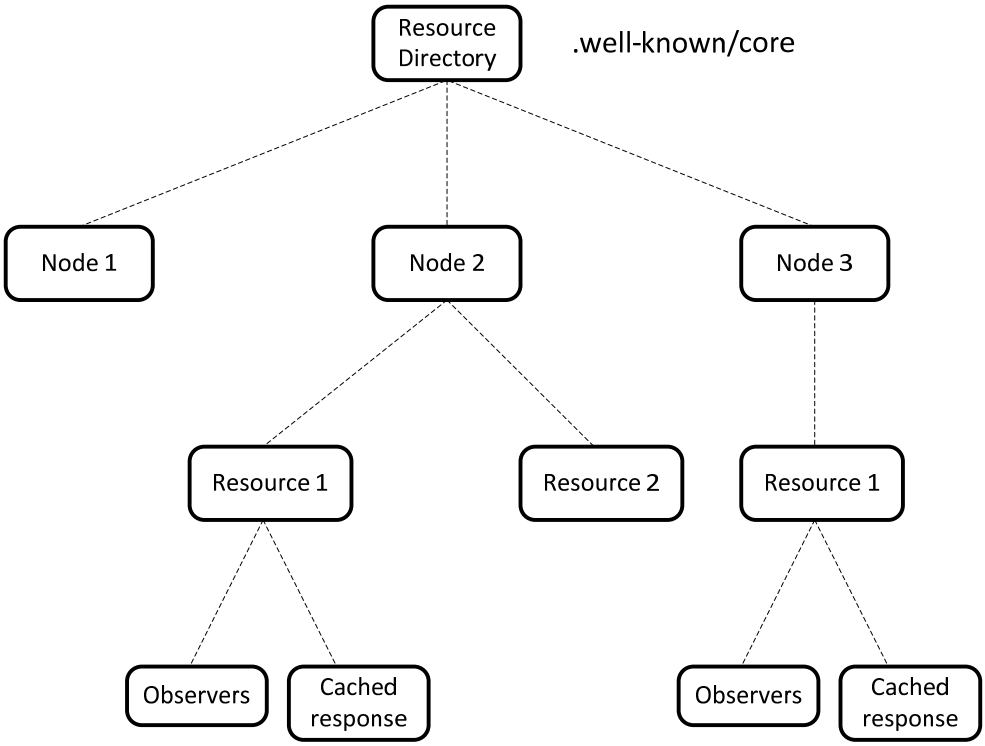


Figure 50 RD structure. The RD is designed as a tree-structure and it is indexed by the node description.

6.3.3.3. Cache

CoAP is a protocol specifically designed for constrained devices where the minimization of resource consumption is crucial. As a consequence, it is expected that some devices might be in sleeping mode the majority of the time in order to save as much energy as possible. Furthermore, the bandwidth available in WSN links is limited. The reduction of the number of interactions between the CoAP proxy and the WSN is therefore necessary.

Under these conditions, a caching system is essential in the CoAP proxy. The presence of cache allows reducing the number of queries that the proxy sends to a CoAP device to obtain the state of a resource. CoAP defines a freshness and a validation model for caching [19]. A CoAP device indicates explicitly whether a response or an observe update is cacheable or not. Furthermore, it can indicate the expiration time of the cached response using the Max-age option. When the cached data is no more valid the proxy can ask the server to renew its validity. This is accomplished by using the ETag option. Caching is particularly useful for the observe updates. A CoAP device may use caching to avoid sending periodical updates containing unchanged data. Cached data could also be used to send the first update to a new observer if the CoAP device has not sent a new update after its registration.

The CoAP proxy stores the payload of the response and the timestamp. The timestamp refers to the time at which the cached message has been received and it is used to check the validity of the cached payload. As

previously mentioned, the cache is contained in the RD structure. The CoAP module is responsible for the management of the cache. Next we illustrate the design and functions of this module.

6.3.3.4. CoAP Module

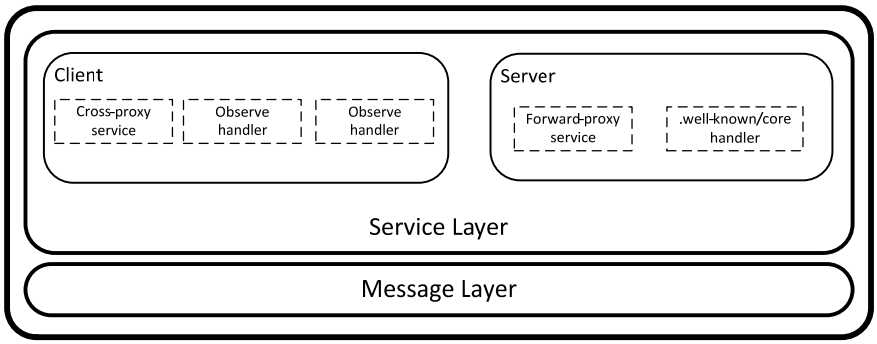


Figure 51. CoAP module overview

As can be seen from Figure 51, the CoAP module is split in two layers. The message layer interacts with the 6LoWPAN interface allowing sending and receiving messages to and from the WSN. The functionalities needed to create and manipulate CoAP messages are also implemented in this layer. The service layer handles CoAP requests and responses. It is split further in two sub-modules: a client and a server. Both client and server operate on the RD and the cache with different purposes. The functionalities of both sub-modules are described as follows.

a) Client sub-module

The client sub-module provides the cross-proxy service. As previously mentioned, the operation performed by this service focuses on the translation between protocols. The result of this process is a CoAP message that matches the request of the Web application as specified in the *connection_t* structure, which is passed by the Web server module. The client's functions also include the management and establishment of observe relationships. The management of observe requests as that of simple requests is compliant with the characteristics of the reverse-proxy service. The client sub-module is also in charge of validating the cached responses and using its content to reply to a request.

The creation of a CoAP request message is subsequent to the validation of the resource targeted by the original request. As mentioned above, the *domain* and *target_node* parts of the URI are validated by the Web server module. Therefore, the client sub-module only checks if any resource linked to the *target_node* matches the resource specified by the *target_resource*. Should a match be found, the client sub-module verifies if the request is intended to establish an observe relationship. In that case, it checks that the CoAP device has labeled the resource as observable and, if positive, it adds the observe option to the CoAP message.

The client sub-module sends the observe request to the CoAP device only if an observe relationship with the same resource does not exist yet. This relationship is, therefore, established only when the first Web client request it. The CoAP proxy is, in fact, the only observer registered directly at the CoAP device. Instead, the

Web clients are registered only at the CoAP proxy. As a consequence, the CoAP device sends the updates only to the CoAP proxy that will distribute them to Web clients. This allows reducing the resource consumption of the CoAP device and the bandwidth usage of the wireless link. A further implication of this choice is that the number of Web clients that can observe a resource increases. The wireless link of the WSN is, in fact, unable to sustain the high-traffic that is generated by multiple observers. Moving this traffic to the more capable link between the Web clients and the CoAP proxy ensures the sustainability of a higher number of observers and the scalability of the solution.

The CoAP request message and the information related to it are stored in the *coap_context_connection_t* structure. This allows the client sub-module to match the response received by the CoAP device with the requests it has sent. A linked list is used to implement the queue containing these structures, which we refer to as *pending_queue*. The *coap_context_connection_t* elements stored in this queue are indexed by the token value contained in the CoAP request.

As shown in algorithm 3, the client sub-module extracts the token contained in the response to find a match between the responses stored in the *pending_queue*. The list of observers of a resource is also implemented using a linked list. In this case, it contains the *connection_t* structure received by the Web module. The *coap_context_connection_t* is implemented as follows:

```
typedef struct {
    unsigned int n_retransmit;
    time_t timestamp;
    unsigned int is_observe;
    unsigned int is_separate_response;
    connection_t *connection;
} coap_context_connection_t;
```

The *n_retransmit* and *timestamp* fields are used by the CoAP retransmission mechanism. In particular, *timestamp* stores the time at which the message has been sent and it is used to calculate the value of the retransmission back-off. The number of retransmissions is maintained by the *n_retransmit* counter. The *connection* pointer is used to access the CoAP message and the information on the request received by the Web client, which are contained in the *connection_t* structure. The *is_observe* and *is_separate_response* flags are used to specify the kind of the data transaction established with the CoAP device. The *is_separate_response* flag indicates that the CoAP device will not reply immediately to the request. CoAP defines a separate response mechanism to enable devices to delay the reply if they cannot process the request directly.

Algorithm 2 Translation process followed by the client sub-module. The CoAP request message is sent after verifying the existence of the resource and the validity of the observe request.

Input: connection_t, node entry in RD, observe flag

Output: CoAP request, observe request, cached response

```
resource ← get_resource_from_node_entry(connection_t)
If resource does not exist then return error
if observe == 1 AND resource is observable then
    if resource.observer_list[] is empty then
        connection_t.pdu ← create CoAP request;
        insert observe and URI options →connection_t.pdu;
        send observe request to CoAP device;
        coap_connection ← create a coap_connection_t structure;
        coap_connection → pending_queue[];
        connection_t → observer_list[];
    else
        connection_t → observer_list[];
    end if
else
    if cached response is valid then send cached response to Web client
    else
        connection_t.pdu ← create CoAP request;
        insert URI options → connection_t.pdu;
        send request to CoAP device;
        coap_connection ← create a coap_connection_t structure;
        coap_connection → pending_queue[];
    end if
```

Algorithm 3: response_handler

```
Input: response from CoAP device
Output: response message or observe update to Web client
token ← get token from response;
coap_connection ← get_connection_from_pending_queue(token);
if response is cacheable then
    update cache;
end if
if coap_connection.is_observe == 1then
    get observer_ list[] from the RD;
    for each observer in the observer_ list[] do
        send observe update;
        if send observe update fail then remove observer from observer_list[];
    end for
else
    send response to Web client;
    remove coap_connection from pending_queue[];
end if
```

b) Server sub-module

The server sub-module provides the forward-proxy service. This enables CoAP devices belonging to the same WSN to interact with each other through the CoAP proxy. This service is particularly useful when they are several hops away from each other and direct interaction could squander precious resources. The forward-proxy service is requested including the Proxy-URI option in the CoAP request message. The server sub-module can forward the request to the destination or reply with a valid cached response. The forward-proxy service can also be used to observe a resource through the CoAP proxy.

The server sub-module provides the interface required to handle queries directed to the *.well-known/core* URI and, consequently, to manage the RD. It also initializes the cache memory for a given resource. The initialization is subsequent to the creation of a new resource inside the RD. The client sub-module is, instead, responsible for validating and updating the cache, as well as for replying to a request with a cached response.

6.4. Performance Evaluation

As mentioned, we evaluate the performance obtained by the CoAP proxy in a real implementation. The tests consider short and long lived communications. In both situations, we evaluate the CoAP proxy according to the use of WebSocket and HTTP long-polling. This allows gaining insight into the performance that the CoAP proxy can achieve with both protocols and evaluating which is the best option to use according to the kind of data transaction.

In long-lived communications, the CoAP proxy uses the observe protocol to receive updates from the CoAP device. A long-lived communication ends when the Web client receives 10 observe update. In the short-lived case the request sent by the Web client implies a single response message from the CoAP proxy. Therefore, a short-lived communication ends after the Web client receives the response.

The HTTP and WebSocket requests sent by the Web client and the CoAP requests sent by the CoAP proxy use the GET method. Furthermore, CoAP requests and observe updates are sent as CON messages. The response or observe update sent by the CoAP device contains a data payload composed by a sequence of bits of fixed size. The composition and length of the CoAP request and response are specified in Table 6.

Table 6 Composition and length, in Bytes, of the messages interchanged between the CoAP proxy and the CoAP device.

CoAP Request	***Observe Request***	***CoAP Response***	***Observe update***
CoAP Header: 5 B	CoAP Header: 5 B	CoAP Header: 5 B	CoAP Header: 5 B
Token : 1 B	Token : 1 B	Token : 1 B	Token : 1 B
URI: 5 B	Observe option: 2 B	Payload: 5 B	Observe option: 2 B
	URI: 5 B		Payload: 5 B

Our experiments are focused to evaluate simple short and long lived communications between the Web client, the CoAP proxy and the CoAP device. Thereby, we can keep the test-bed network simple and avoid deploying complex architectures. In this sense, we choose to implement a single-hop WSNs, which is shown

in Figure 52. The Web client and the CoAP proxy are located in two different PCs. These feature 2 GB of RAM and use Linux as OS.

The proxy sends CoAP requests through a 6LoWPAN base station attached to its USB port. The CoAP device and the 6LoWPAN base station are embedded in TelosB motes. The CoAP device is located one-hop away from the base station. We use TinyCoAP as the CoAP implementation embedded in the CoAP server.

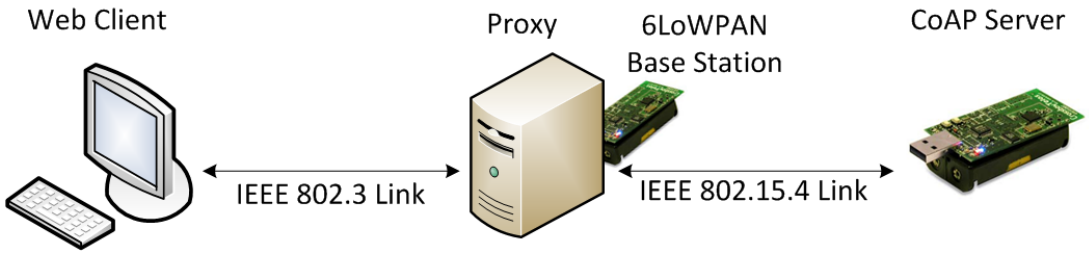


Figure 52 Test-bed network

Our study evaluates two parameters. First we evaluate the RAM and ROM footprints of the CoAP proxy. This analysis helps to evaluate the code complexity and the efficiency in terms of memory that the proxy has at run-time. Then, we measure the latency in order to evaluate the proxy response time. For all the tests, the results are reported according to the number of Web clients that simultaneously request data from the CoAP device. As mentioned before, the CoAP proxy uses the observe protocol to deal with long-lived communications. In this case the CoAP device sends observe updates each 500 ms. This value is consistent with the update frequency of typical temperature monitoring in industrial processes [103]. The proxy establishes an observe relationship with the CoAP device when the first Web client requests it. The first notification sent to the subsequent observers is the value of the observed resource that is currently in the cache.

6.4.1. Memory Footprint

The CoAP proxy is designed to be embedded in any kind of hardware. However, its resource consumption has to be minimized to allow its use also in constrained hardware. In particular, the RAM memory used by the proxy at run-time has to be tailored to the characteristics of constrained devices where the available RAM could be in the order of few tens of mega bytes. The analysis of the ROM footprint allows evaluating the complexity of the code that implements the COAP proxy. An evaluation of the ROM footprint can be carried out by analyzing the size of the executable files generated by the main proxy, Lighttpd and 6LoWPAN processes. The ROM footprint of the CoAP proxy is therefore the sum of the size of each executable. The evaluation of the 6LoWPAN takes into account the memory that the CoAP proxy allocates to communicate with the 6LoWPAN interface. It does not consider the memory allocated by the 6LoWPAN interface. It is, in fact, embedded in a TelosB mote, which make impossible the use of valgrind. Table 7 shows the results of the ROM memory consumption evaluation. The size of the ROM footprint is equal to 1466,9 KB, which is compliant with the characteristics of embedded applications.

Table 7 ROM occupation of the CoAP proxy. Three modules compose the proxy.

Main proxy	Lighttpd	6LoWPAN
302 KB	805,2 KB	359,7 KB

The modules that compose the CoAP proxy are implemented using the C language. Therefore, we analyze the RAM footprint according to the typical layout of a C program, which is shown in Figure 53. At run-time, memory is allocated dynamically in the *heap* and *stack* fields of the RAM using the *malloc()* and *free()* functions. We use the *valgrind* profiling tool to evaluate the size of the memory allocated dynamically at run-time. The *text*, *bss* and *data_segment* fields compose the part of the RAM that is allocated statically. Therefore, they have fixed sizes that do not vary at run-time. The *text* field contains the instructions that rule the execution of the program. The *bss* is used to store the variables that are not initialized. Finally, the *data_segment* contains the variables that have an initialization value. The static components of the RAM memory are evaluated using the *size()* command.

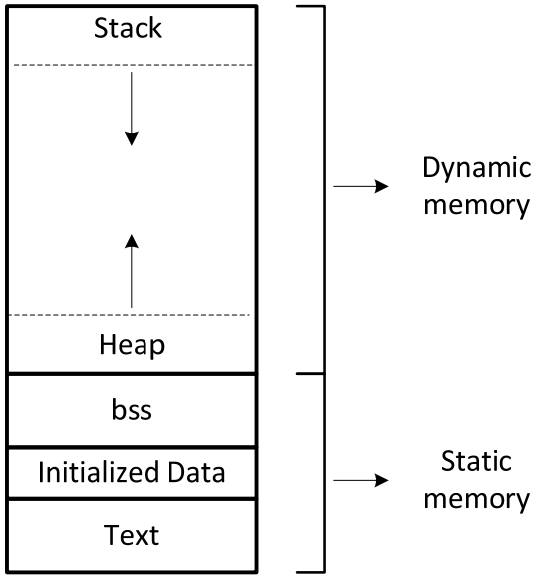


Figure 53 Layout of RAM memory.

The RAM footprint is evaluated according to the number of Web clients that simultaneously request the representation of a resource or receive updates as observers. We evaluate short-lived as well as long-lived data transactions. We kept the concurrency level low considering the presence of 2, 10 and 50 simultaneous clients. In fact, it is rather unlikely that in WSNs there is a very high workload of hundreds of simultaneous client requests. The variation of the concurrency level only affects the size of the memory allocated dynamically.

We evaluate the RAM footprint measuring the memory allocated by the static and dynamic components of each of the processes composing the proxy. The sum of the memory consumed by each process is the resulting size of the RAM footprint. The results are differentiated according to the use of Websocket and HTTP long-polling. We include the memory consumption of the Lighttpd process also in the WebSocket case.

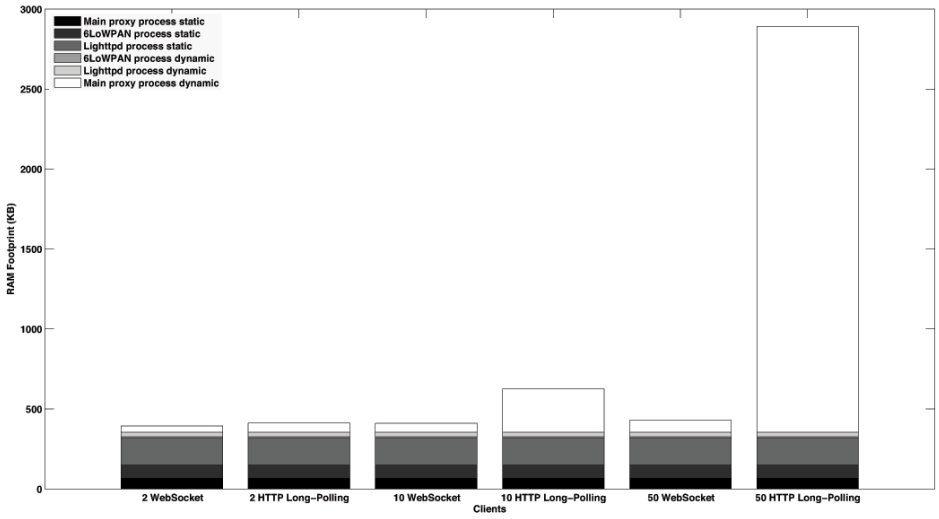
This process, in fact, is always present at run-time unless it is used only in HTTP long-polling. In this test, the RD is composed by a single node entry and the relative resource description. The cached response and the observer list are present only in the long-lived case.

As can be seen from Figure 54, the CoAP proxy has a lower RAM footprint with long-lived communications. The use of the observe option allows reducing the size of the dynamic memory that the main proxy process allocates to create and sends CoAP requests. The CoAP proxy, in fact, establishes only a single observe relationship with the CoAP device. This implies that the main proxy process has to allocate only the memory required to translate, create and send a single CoAP observe request and to acknowledge the subsequent updates. Further memory is allocated to send the updates to Web clients, to update the cache memory and to create and maintain the RD. In the short-lived case, instead, the CoAP proxy sends a CoAP request for each WebSocket or HTTP long-polling request that it receives. The main proxy process allocates, therefore, dynamic memory for each of these CoAP requests. Further memory is allocated to store the responses sent by the CoAP device and to send the relative acknowledgments.

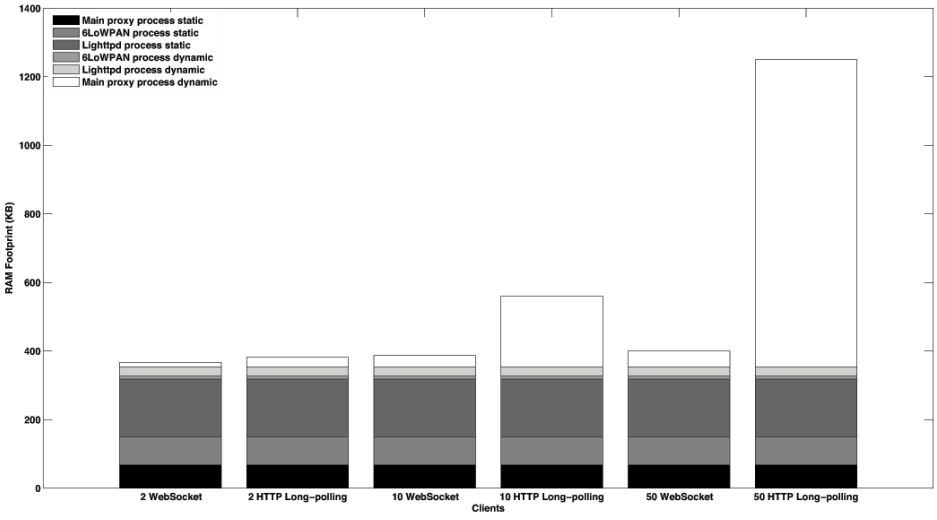
In both cases, The CoAP proxy yields a better performance using WebSocket instead of HTTP long-polling. The handling of WebSocket requests, in fact, implies less complexity and overhead than that required by HTTP long-polling requests. In this sense, we found the use of the FastCGI protocol as the main cause of the high consumption reached by HTTP long-polling. The FastCGI requests intercepted by the Lighttpd server are multiplexed into a single connection and are processed by the main proxy process in a multi-threaded style. In that way we can achieve better performance respect to creating a single thread to process each request. However, the presence of several concurrent processes increase the memory consumed by FastCGI. The dynamic memory allocated by the Lighttpd process does not vary with the concurrency level. As mentioned before, the Lighttpd server has a very low memory footprint that made it suitable for being embedded in constrained devices. The memory footprint of the Lighttpd server will increase only with a higher concurrency level.

In the Websocket case, the RAM footprint of the CoAP proxy is dominated by the memory allocated statically. The slight increase of the RAM footprint is due to the growth of the concurrency level. This has effect only on the memory allocated dynamically by the main proxy process to store the information related to each WebSocket requests. The size of the dynamic memory allocated by the 6LoWPAN process is constant. The 6LoWPAN interface, in fact, receives by the main proxy process only a request at a time. The subsequent CoAP request is sent to the interface only when the previous has been sent. The dynamic memory allocated to create and send the 6LoWPAN packet is therefore constant.

Figure 54 RAM footprint of the CoAP proxy. It has a low memory footprint when using WebSocket. The FastCGI protocol used in HTTP long-polling requires more complexity that results in a growth of the memory consumption. a) RAM footprint of CoAP proxy in short-lived communications b) RAM footprint of the CoAP proxy in long-lived communications.



(**a**)



(**b**).

6.4.2. Latency

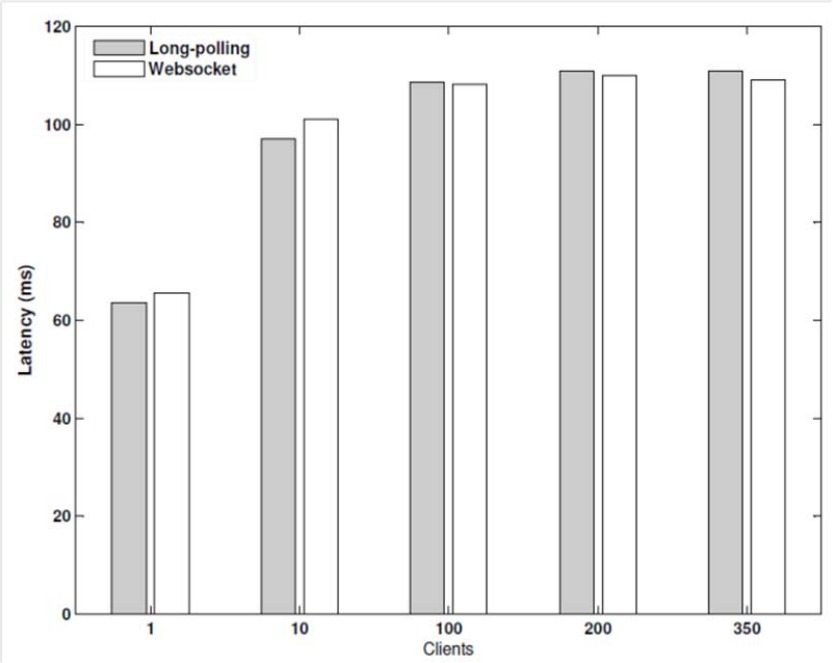
The latency experienced by a Web client to retrieve data from the CoAP device is perhaps the most important parameters used to evaluate the goodness of the CoAP proxy implementation. In short-lived communications, we define latency as the time elapsed from the moment the Web client sends a request until the moment it receives the response. The latency considers the delay introduced to establish and close the related TCP connection. In the long-lived case, the latency is the time elapsed from the moment the Web client requests to observe a resource until the moment it receives 10 updates. The latency, therefore, measures the length of the entire data transaction and not that of sending single updates. In fact, measuring the latency

of each single update would not allow highlighting the difference between WebSocket and HTTP long-polling in long-lived data transaction. Instead, it would measure the latency as if the data transaction were short-lived. Also in this case we include the delay experienced to establish and close the TCP connection. Figure 55 shows the latency of each tested solution.

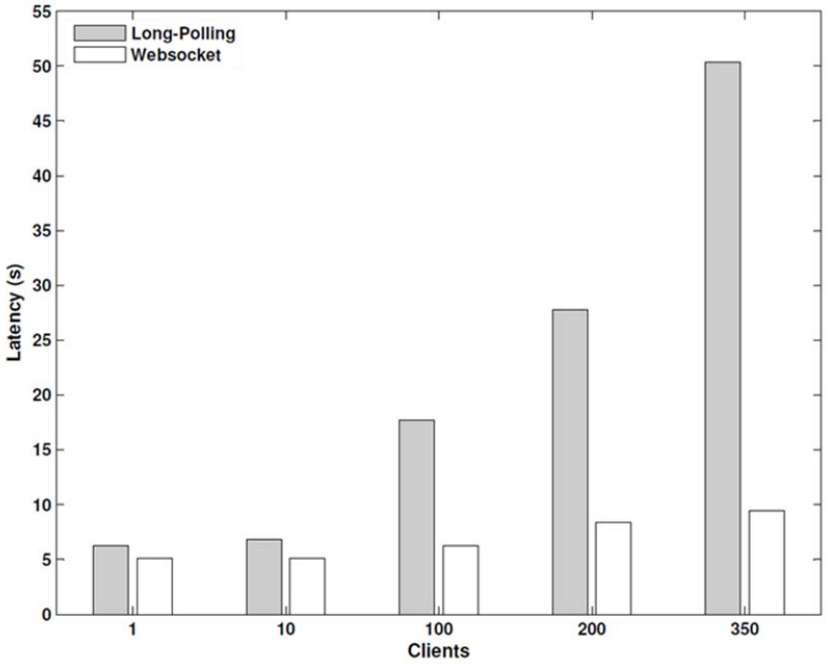
In long-lived communications, the CoAP proxy yields a better performance with WebSocket. In this context, the use of a persistent connection and the low overhead caused by the WebSocket protocol allows reducing significantly the latency. The use of HTTP long-polling allows an acceptable performance when the level of concurrency is low. Instead, the latency undergoes a sharp rise when this level grows. In this case, the CoAP proxy is subject to a high workload that causes a rapid worsening of the performance. The high rate at which the Web clients requests the observe updates is the cause of this high workload. The use of the WebSocket protocol, instead, reduces significantly this workload. Consequently, the growth of the concurrency level causes only a small increase of the latency.

In short-lived communications, the CoAP proxy has almost the same performance with WebSocket and HTTP long-polling. The use of HTTP long-polling allows a slight improvement for low concurrency values. The difference between the performance of WebSocket and HTTP long-polling is mainly due to the delay introduced by the opening and closing handshake phases of the WebSocket protocol. However, this is partially compensated by the delay caused by the Lighttpd module to handle HTTP long-polling requests. This delay becomes predominant when the concurrency level is high causing a slight deterioration of the performance. WebSocket requests, instead, are received directly by the main proxy module, which allows reducing the delay caused by their handling.

Figure 55 Latency for short and long lived communications. The CoAP proxy benefits from the use of WebSocket in long-lived communications. c) Latency for short-lived communications d) Latency for long-lived communications



(**a**)



(**b**).

6.5. Conclusions and Contribution

We have presented the design of a CoAP proxy able to interconnect Web applications to CoAP devices located in WSNs. Its design is consequent to the analysis of the communication patterns that are used by the CoAP proxy to communicate with the CoAP device and the Web application. In this sense, we have considered the use of protocols able to deal with short and long lived communications.

The data flow of long-lived communications requires adopting protocols able to reduce the overhead and the workload that it could cause. The WebSocket protocol was included in the CoAP proxy to establish persistent connections with the Web applications. The observe protocol was used for the same purpose but to communicate with the CoAP device. The CoAP proxy also supports Web applications that use the traditional HTTP long-polling technique.

We have included a cache memory and a RD repository in the CoAP proxy. The RD is used to host the description of the resources offered by the CoAP device. Its presence simplifies the resource discovery process. Cache, instead, allows reducing the number of messages interchanged between the CoAP proxy and the CoAP device. We also have defined a convention format for the URI that should be used by a Web application to access a CoAP resource. The proposed format is compliant with the design principles of the RD and simplifies the translation process.

We have evaluated the performance of the CoAP proxy considering long and short lived communications according to the use of HTTP long-polling and WebSocket. The evaluation has been done in terms of latency and memory consumption. Results show that, in long-lived communications, the CoAP proxy yields a better performance using the WebSocket protocol. Its use minimizes the communication overhead between the Web

application and the CoAP proxy, which reduce the latency and lower the RAM footprint consumption. HTTP long-polling, instead, causes a high workload that result in a worsening of both metrics especially for high concurrency levels. In the short-lived case, however, HTTP long-polling shows a good performance in terms of latency and, respect to WebSocket, reduces it slightly when the concurrency level is low. For higher levels, instead, the latency is very similar with WebSocket having a slight better result. The CoAP proxy has a lower RAM footprint when using the WebSocket protocol. In this case, the FastCGI protocol used in HTTP long-polling entails further complexity to manage the HTTP connections that result in a high demand of RAM. The observe protocol proves to be able to reduce the RAM footprint of long-lived communications by reducing the messages interchanged between the CoAP proxy and the CoAP device.

In conclusion, the design proposed for the CoAP proxy offers an effective solution for interconnecting Web applications and CoAP devices. The use of different communication patterns provides flexibility and enables the CoAP proxy to work in different application scenarios. In particular, the adoption of the WebSocket and the observe option allows improving the performance of long-lived applications. HTTP long-polling, instead, permits the use of the proxy in applications where short-lived communications are most used and the concurrency level is low.

From the results of the research presented in this chapter derives the paper *A Proxy Design to Leverage the Interconnection of CoAP Wireless Sensor Networks with Web Applications* [P3].

7. QoS Support for Timeliness

The research on CoAP follows with our proposal to enable QoS support for timeliness in the observe option. It is based on delivery priority, update selection and QoS negotiation. Nodes are able to express the priority order with which they wish to be updated. Furthermore, they are also able to select which updates they want to receive. We classify updates in two categories: Critical and Non-Critical. The provided QoS is the result of a negotiation between the client and server. The client demands a certain degree of QoS according to its role. The server could accept or negotiate it. This choice depends on the availability of server and network resources. The proposal is evaluated in a real WSN. In particular, we choose the requirements of an e-health application as target of our tests. The performance evaluation is done in terms of average delay, energy consumption and delivery ratio.

Common solutions used in publish/subscribe protocols provide reliability and timeliness as QoS parameters. The first is used to ensure to subscribers the reception of updates. The timeliness is used to guarantee the on-time delivery of packets. Timeliness is of paramount importance for WSN applications such as e-Health. The data generated by sensor nodes should be received within a deadline in order to be processed quickly and to react immediately to critical situations. Moreover, data delivery should be guaranteed to those nodes that have key roles in the application.

As mentioned, the observe option provides reliability by using CoAP CON messages. Regarding timeliness, it only provides the possibility to indicate the validity of an update over a period of time. This is achieved using the freshness model of CoAP caching. However, it does not specify how to guarantee on-time delivery. Meeting the deadline requirements of an update depend in large part to the delivery process. In this sense, the order in which the nodes are updated is of paramount importance. The observe option model gives to the server the faculty to choose in which order clients are updated.

The current delivery model could be inefficient for many application domains. We argue that the server should be able to prioritize the delivery of updates to nodes requiring it. Our approach is motivated by the fact that WSN nodes could have distinct roles and requirements. As a consequence, timeliness requirements could vary depending on the characteristics of a node. For instance, in e-emergency applications the notification of a critical event must be prioritized to those nodes in charge of reacting to it. Furthermore, these nodes could only be interested to be updated when a critical event occurs. Therefore, the server should avoid sending them each update. Other nodes, however, could be interested in receiving all of them. Therefore, a server should be able to distinguish which information sends to a particular client.

7.2. QoS Support in the Observe Option

As mentioned, the observe option supports QoS for reliability by using the end-to-end reliable data transfer defined in CoAP. QoS, therefore, can be divided in two categories: best-effort using NON messages

or persistent when using CON ones. The use of NON or CON messages depends exclusively on the subject. For each individual update it can decides which message use.

Regarding timeliness, the observe option allows to specify the validity of an update but not to guarantee its on-time delivery. The indication of the validity is achieved using the caching model of CoAP. As mentioned in chapter 6, this defines a freshness model to indicate the period of time in which the cached response is valid. The subject is responsible for deciding the length of this period. Its value is contained in a CoAP option, which is called max-age. An observer can cache and re-use an update until it is not expired. Moreover, observers can use the max-age value to control if they are still registered to the subject. If an observer does not receive an update after the max-age has expired, it may assume that it has been cancelled.

The lack of support for on-time delivery could not allow meeting the different delay requirements of observers. As previously mentioned, the delay experienced by an observer when receiving an update is strongly affected by the delivery order followed by the subject. The current observe definition leaves to the subject the decision of the delivery order. We argue that this approach is inefficient for many applications that wish to use the observe option. The delivery order should, in fact, be differentiated depending on the requirements of each observer. In this sense, we proposed to modify the observe option for supporting timeliness. Our contribution seeks to define a simple mechanism to establish the delivery order based on the priority expressed by observers. Furthermore, the observers can express their interest in which kind of update they want to receive.

7.3. Proposal of QoS Support for Timeliness

As mentioned, the subject is the only node able to manage the delivery order of updates. Therefore, its choices can only be based on the information available at server side. Commonly, it concerns the characteristics of the resource that the subject is monitoring. The requirements of observers are not considered in the current model. Although the deadline of an update depends on the type of data that it contains, the delivery should be differentiated depending on the particular observer. The total delay of an update, in fact, undergoes a pronounced variation depending on the delivery order.

Depending from the particular WSN application domain, each observer could have different roles with distinct delay requirements. In a WSN used for fire detection, for instance, a node in charge of the fire-extinguishing process should be the first to be notified when a fire is detected. Furthermore, an observer could only be interested in receiving only critical updates. For instance, in the e-health domain an alarm is only interested in knowing critical states of the patient rather than each single state. Furthermore, it could require to be notified only when the critical situation is detected and when it finishes. The adoption of a mechanism that allows an observer to explicit the notifications it wishes to receive, allows minimizing the resource consumption of the subject and the observers.

In our proposal, observers can request a priority level in the update delivery. Moreover, they can indicate which kind of updates they wish to receive. As previously mentioned, an update can be classified as critical or non-critical. The subject is the only component that has the authority to make this decision. This choice

depends exclusively from the WSN application domain. Usually, these are not multi-purpose networks. They are deployed to perform a specific task. Each node is aware of its role and of the resource it is monitoring. Therefore, it is expected that the subject will have the necessary information to establish the criticality of an update.

Our proposal supports four levels of QoS: low, medium, high and highest. The subject prioritizes the delivery to observers requiring the highest level. This level is intended for observers that are interested in being notified when a critical event is detected and when it finishes. The high level is required by those nodes that are interested in receiving each critical update. They are notified immediately after the observers with highest priority. Observers interested in receiving both critical and non-critical updates can demand a low or medium level. The medium level has priority on the low one. Should more than one observer require the same level, the subject notifies them following the order at which their requests arrived.

The value of the QoS level is specified using the two most significant bits of the observe option value. We refer to those bits as QoS field. This modification affects only the requests sent by observers for registering and the consequent response of the subject. The bits used by the QoS field do not affect the subsequent updates and, therefore, the maximum value of the observe option does not change. The definition of the four QoS level and the relative QoS field values is the following:

- Level 1: The subject sends non-critical and critical updates with low priority. The value of the QoS field is 00.
- Level 2: Both non-critical and critical updates are sent with medium priority. The value of the QoS field is 01.
- Level 3: The subject sends only critical notification with high priority. The value of the QoS field is 10.
- Level 4: The subject notifies with the highest priority only the start and the end of a critical state. The value of the QoS field is 11.

The model we propose expects that the observer and the subject negotiate the QoS. A subject, in fact, should have the faculty to reject or negotiate the demanded QoS. It could not have enough resources to satisfy all the requests. An observer could demand the highest priority but the subject could have already other observers with the same level. Therefore, the subject would not meet the delay requirement of the observer and should offer a lower priority level. Furthermore, the subject could recognize that the energy level of its batteries is under a certain threshold. Therefore it could not satisfy a new request from an observer that requires each update. Should the subject accept the request, it replies with the observe option containing the QoS value requested. The subject expresses its willingness to negotiate the QoS by replying with the offered QoS. Should the observer accepts, it sends a response containing the offered QoS. The observer can reject the offer by ignoring it.

The definition of a routing strategy that a subject can use to reduce the delay is out of the scope of our proposal. We focus on reducing the delay by defining a simple mechanism for establishing the delivery order

of updates. A further optimization is achieved allowing observers to register their interest according to the updates they want to receive.

7.4. Experimental Set-up

The effectiveness of our proposal is evaluated in a real WSN. We run tests to calculate the average delay of updates, their delivery ratio and the energy consumption of the observer. The proposal has been implemented as part of TinyCoAP.

Our test-bed network meets the requirements of an e-health application used to monitor the cardiac rate of a patient. We choose this scenario because of its criticality. E-health applications, in fact, have strong deadline requirements. The results, however, are general and valid for different application domains. We assume that several observers are interested in receiving updates about the state of a patient. Each observer has different requirements in terms of priority and interest. In our experiments, we consider the presence of six observers: an alarm, a doctor, a nurse, an Intra Venous (I.V.), a general and a patient-specific monitor. The QoS requirements for each observer are the followings:

- Alarm: It needs to be notified when a critical event occur and when it ends. The level of the QoS is 4.
- Doctor: He needs to be notified only when the patient enters or leaves the critical state. The level of the QoS field is 4.
- Nurse: He has a tablet to monitor each state of the patient. The level of the QoS field is 2.
- General monitor: It receives the updates of many patients. The level of the QoS is 1.
- Personal monitor: It is exclusive for the patient. The level of the QoS is 2.
- I.V.: It is active in case of emergency. It has to receive each critical notification. The level of the QoS is 3.

The characteristics of the updates sent to monitor the cardiac rate are specified in Table 8. The values reported in this table are based on the characteristics of this parameter [104]. The deadline of an update corresponds to its periodicity. Should the update be critical, the deadline corresponds to the sampling rate of the cardiac rate. In non-critical updates, the data collected is aggregated and sent as data log in a full 802.15.4 frame. Considering the presence of protocol overhead, the maximum payload size available is of 74 bytes. Therefore each notification will contains 37 samples of the cardiac rate and will have a periodicity of 3700 ms. The use of data logs allows to a subject reducing the number of updates to sent to observer and, therefore, minimize the resource consumption.

Table 8 Characteristic of cardiac rate updates

Parameter	*Patient state*	*Payload*	*Periodicity*
Cardiac Rate	Normal	74 bytes	3700 ms
	Critical	2 bytes	100 ms

The test-bed used TelosB motes to implement the subject and the observers. The WSN adopts a star topology. For simplicity we assume that the observers are equally spaced between each other and at the same distance from the subject. Figure 56 shows the topology of the WSN. In the next section we present and discuss the results of our performance evaluation.

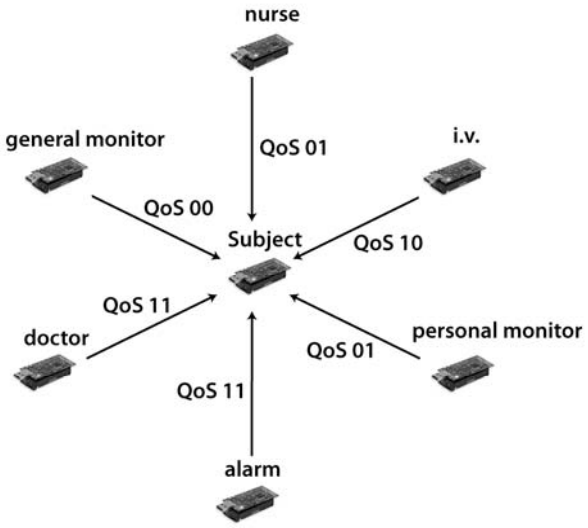


Figure 56 Topology of the test-bed network

7.5. Results and Discussion

The performance evaluation focuses on evaluating the delay and the delivery ratio of updates. Moreover, we also evaluate the energy consumption of the subject. Our tests consider a subject sending both critical and non-critical updates. For each case we differentiate the tests according to the reliability mode used.

7.5.1. Latency

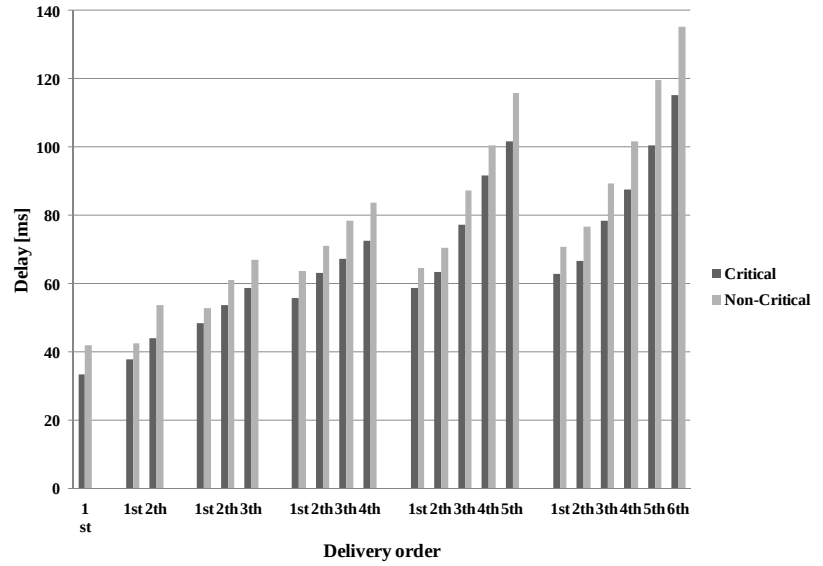
Figure 57 shows the delay of updates as a function of the delivery order. Each value represents the average delay of 100 delivered updates. We define delay as the time elapsed from the moment the update is created until the moment the observer acknowledges its reception. We consider both the use of best effort and persistence mode. In persistence mode the update is acknowledged at CoAP layer while in best effort only at MAC layer. For each reliability mechanism we run different tests according to the number of observers that receive the updates.

As one may observe from Figure 57, the delivery order has a strong influence on the delay experienced by an observer. In non-critical updates, however, this delay is considerable lower than the deadline of the update. A further reduction of the delay is achieved by allowing the subject to avoid sending non-critical updates to

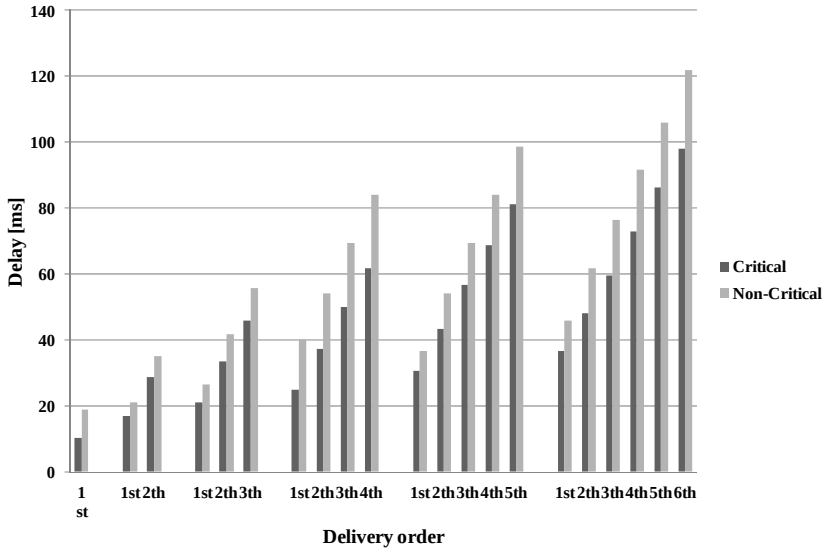
all the six observers. According to the QoS required by the observers of our test-bed, only three nodes receive non-critical updates. More observers, however, could require these updates.

The delivery order is of paramount importance in critical updates. According to the size and topology of our test-bed network, the delay experienced by observers with lower priority is greater or equivalent to the deadline of an update. We expect that this could be even greater in case more observers have to be notified. As a consequence, the current delivery mechanism of the observe option could not guarantee the on-time delivery. When delivering critical updates, especially in e-emergency applications, the subject has to follow a delivery order based on the characteristics and requirements of observers. As our results show, the use of a priority-based delivery is necessary to meet the timeliness requirements of observers with critical roles. Furthermore, we further reduce the delay by allowing to observers to receive only the first and the last update of a critical situation. As shown in Figure 56, during the time the cardiac rate is in a critical condition the subject sends updates only to four observers. Therefore, the notification delay of each observer is lower enough to meet the deadline requirements.

A further analysis has to consider the reliability support of the observe option and the effects this has on delay. As one may expect, the use of the persistence mode implies a growth of the delay. In fact, the processing time taken by an observer to reply to an update with a CoAP ACK is higher than that required for sending a MAC ACK. Besides the packet processing time, a further delay is introduced by the CSMA-CA mechanism of the 802.15.4. An observer that sends a CoAP ACK, in fact, does the same channel access procedure of the subject sending the update. In best effort, instead, the observer sends the MAC ACK without using CSMA-CA. Therefore, in persistence mode the observers and the subject compete for accessing the channel. The probability that one of them finds the channel busy could be high. In this case, they could perform several attempts for transmitting the packet. As a consequence, the delay will increase. Due to the high number of nodes competing for the channel, the probability that an ACK collides with the transmission of an update could be high. In case of collision the subject could retransmit the update. However, as specified in [27] a subject should avoid retransmitting an update if a new one is available. Therefore a collision could imply either an extra delay caused by retransmissions or the lost of the update.



(**a**)



(**b**).

Figure 57 Delay as a function of the delivery order. A) Persistence B) Best Effort

7.5.2. Delivery Ratio

Figure 58 shows the delivery ratio as a function of the delivery order. The values obtained refer to the transmission of critical updates using persistence or best effort as reliability support. We define delivery ratio as the probability that an observer receives an update correctly. We fix the retransmission timeout to 38 ms. This value is equivalent to the sum of the average delay experienced in presence of a single observer and its standard deviation. The retransmission timer is activated when the notification has been sent. Therefore it does not consider the time required to generate it. The results reported in this test are only valid for WSN s

adopting a star topology. We expect that the persistence mode will achieve better results in multi-hop networks. The study of this topology is, however, out-of-the-scope of this evaluation.

The results demonstrate that observers notified with high or highest priority have also a high delivery ratio. In persistence mode, an update sent with these priorities find a less congested channel and therefore has less chance to collide with an ACK. Even if the update or ACK does not arrive correctly, the subject has more time to retransmit it before a new one is ready. As mentioned above, an update is not retransmitted if a new one is available and therefore it is considered as a lost packet. Updates sent in best effort yield a better performance in terms of delivery ratio. Here, the subject is the only node transmitting packets and therefore it does not compete for the channel. As a consequence, the channel is less congested and the probability of collision is negligible. The performance reached in best effort is more suitable for an application domain as e-health where having a reliable communication is of paramount importance.

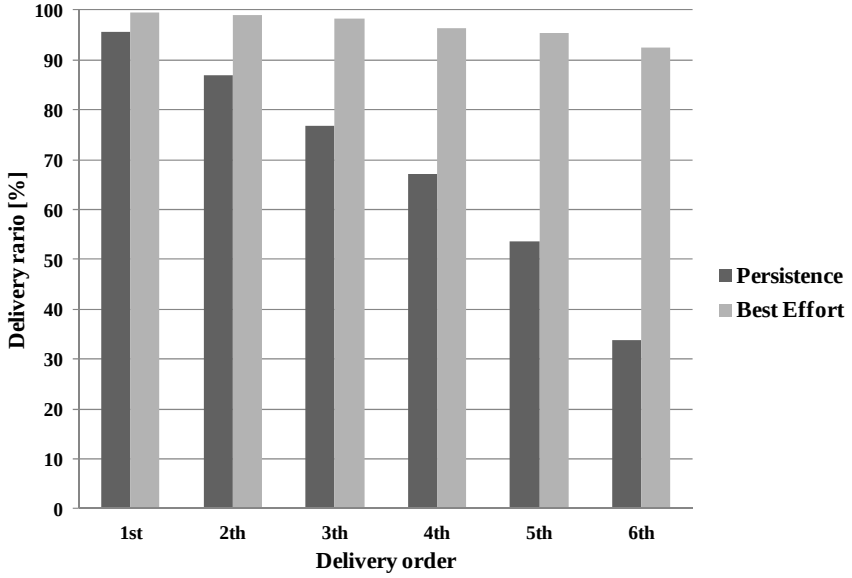


Figure 58 Delivery ratio as a function of the delivery order. A delivery order based on priority allows guarantying high delivery ration to observers requiring high priority.

7.5.3. Energy Consumption

Figure 59 shows the results obtained in the energy consumption test. This is focused to evaluate the energy consumed by a subject when delivering critical or non-critical updates. The results of this test do not take into account the energy consumed to listen the channel. The measures are only inherent to the energy consumed to process, send and receive updates or acknowledgments. As a consequence, our evaluation does not need to consider power-saving protocols for radio duty cycling. The energy is sampled each 0.02 ms. The device used for these measures is the Agilent Technologies DC power Analyzer N67705A.

Updates sent in persistence mode involve a higher number of messages respect to those sent in best effort. As a consequence, the subject consumes more energy. The increased consumption is due mainly to the energy used by the radio chip to receive the ACK and, to a lesser extent by that consumed to process them. As one

may expect, the subject consumes less energy when sends non-critical updates. In this sense, our proposal of enabling an observer to choose which updates receive allows to reduce the energy consumption of the subject. Regarding critical updates, the consumption is significantly reduced by avoiding sending all of them to those observers that choose the QoS level 4.

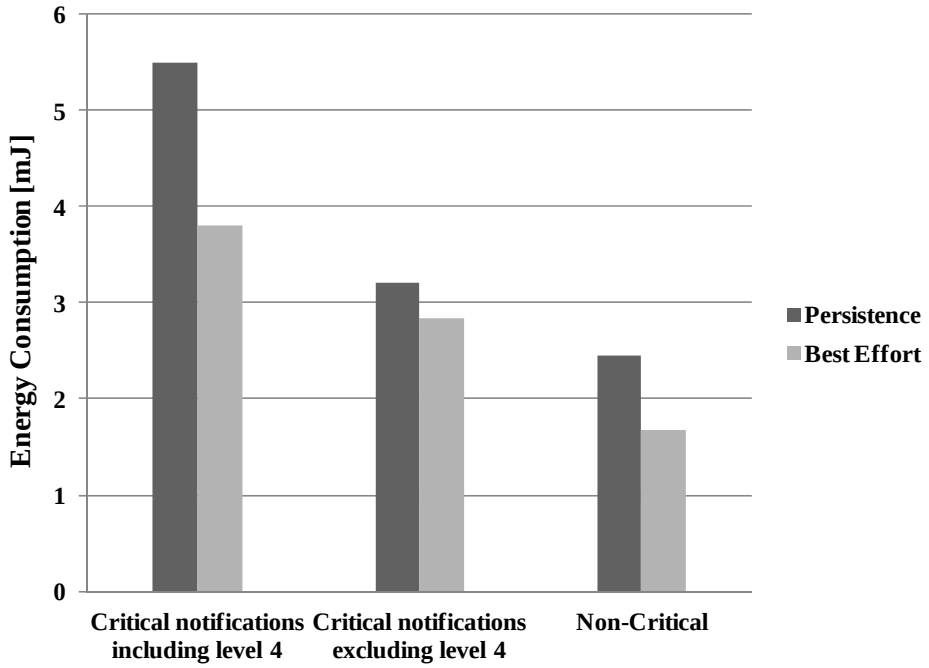


Figure 59 Energy consumption of the subject. The subject saves energy by avoiding sending all the critical updates to all observers.

7.6. Conclusions and Contributions

We have presented our novel proposal for adding QoS support for timeliness to the observe option of CoAP. In this perspective, we have suggested the adoption of a simple update delivery mechanism based on priority. The level of priority is requested by the observers. This level establishes the order in which the subject sends an update. We have defined four level of priority. Each level is associated to a class of updates that the observer wishes to receive. We have classified the updates as critical or non-critical. The subject is the only authority able to distinguish the class of an update. At present, the definition of the observe option leaves to the subject the decision of which updates send and in which order. This approach could have limitations since it does not consider that observers could have different requirements. A subject could have no information about the observers and, therefore, it would be unable to meet their requirements.

We run tests focused on evaluating the effects that our proposal has on a WSN adopting the observe option. The tests are done in a real WSN. The requirements of the WSN application are those of an e-health network used for monitoring the cardiac rate of a patient. The performance evaluation is done in terms of delay, delivery ratio and energy consumption. The results confirm that the delivery order of an update influences both the delay and delivery ratio. The first updates sent by a subject experience less delay and have

a high delivery ratio. A delivery order based on priority is therefore essential for providing on-time delivery to observers requiring it. The energy consumption of the subject is considerably reduced if observers are able to express the notification class of interest.

In conclusion, our proposal offers an effective solution for adding timeliness support to the observe option. The priority-based delivery proposed allows to the observe option to work better with applications with deadline requirements. Furthermore, our proposal is able to maintain the simplicity of the observe option by requiring only two bits of its option value. These are used only when an observer registers its interest to a subject. The option value of the subsequent updates is not affected by these bits.

From the results of the research presented in this chapter derives the paper *Adding QoS support for timeliness to the observe extension of CoAP* that has been presented at the 8th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob) [P7].

8. Conclusions and Future Works

During the course of this thesis we have studied various solutions that allow optimizing the application of Web services in WSNs. Based on the constraints that characterize these networks, we identified some important key areas that allow reducing the resource consumption as well as improving the performance. These included the design of optimized embedded software solutions and that of communication techniques for large data transactions. We analyzed Web service architectures and chosen the most appropriate for the constraints of WSNs, which is REST. Based on this analysis, we reviewed the state-of-the-art of protocols that allows implementing REST Web services. To this end, we adopted the IEEE 802.15.4 standard for the physical and data-link layers, 6LoWPAN for the network layer and CoAP for the application layer.

We analyzed 6LoWPAN forwarding techniques and individuated their drawbacks when applied in communications requiring packet fragmentation. We designed a novel 6LoWPAN forwarding technique able to solve these limitations. In particular, we proposed to add control to MU forwarding by monitoring the fragmentation header of the incoming fragments. This allowed ensuring the correct delivery order of the fragments as well as the correctness of each one. In addition, this solution avoids forwarding fragments of packets that cannot be reassembled. We referred to our proposal as CMU. We demonstrated through a performance evaluation that CMU is able to enhance the performance of MU by reducing its packet loss and end-to-end delay. In this evaluation, we also considered a forwarding technique based on RO, which improved significantly the end-to-delay of RO at cost of augmenting its packet loss.

Our study on 6LoWPAN forwarding allowed identifying the criticality of 6LoWPAN fragmentation, which we individuated into the lack of support for end-to-end reliability. The loss of a fragment would force the retransmission of the entire fragmented packet, which has high costs in terms of bandwidth and energy consumption. CoAP defines CoAP blockwise transfer to overcome the limitations of 6LoWPAN fragmentation. We designed a novel analytical model for CoAP large data transactions in order to study and analyze both techniques. This model included the effects of fragmentation on the contention level at MAC layer and has been validated through Monte Carlo simulations. Both techniques have been compared in terms of reliability and delay. The results obtained shown that 6LoWPAN fragmentation yields a better performance in terms of delay. Both techniques had a good performance in terms of reliability with small difference between them. However, CoAP blockwise transfer proved to be more reliable when the traffic conditions cause a congestion of the WSN. 6LoWPAN fragmentation, instead, is preferable when the WSN is less congested.

To the best of our knowledge, the model presented in this thesis is the first to evaluate and compare analytically the performance of CoAP blockwise transfer and 6LoWPAN. Furthermore, it considers the presence of both saturated and un-saturated traffic condition at MAC layer, which was not included in the existing models of the 802.15.4 protocol.

In part of this thesis, we have designed and developed optimized software solutions for CoAP. To this end, we presented our original library for TinyOS, which we referred to as TinyCoAP, and the design and

implementation of a CoAP proxy. In both cases we illustrated their design principles, described their implementations and presented their evaluations.

We compared TinyCoAP to CoapBlip, which is the CoAP implementation distributed with TinyOS. TinyCoAP proved to be able to reach a high code optimization and to reduce the impact over the memory of WSN nodes. The evaluation included also the analysis of the reliability mechanism defined by CoAP, which was still uncovered in the literature. As a novelty, we also compared CoAP with HTTP considering different solutions for the transport layer protocol such as UDP and persistent TCP connections.

CoAP is still under standardization and its diffusion is limited. For the same reason, Web applications with a dual HTTP-CoAP stack are not diffused. Therefore, it is of paramount importance the presence of systems able to interconnect Web applications with IoT devices. With these considerations in minds, we designed a CoAP proxy. It enables Web applications to transparently access the resources hosted in IoT devices based on CoAP. The CoAP proxy supports long-lived communications by including the WebSocket protocol. It also supports Web applications that use the traditional HTTP long-polling technique. Finally, one of the main contributions of the proxy design is the proposal of a standard URI path format to be used by Web applications to access to CoAP resources.

Our research on CoAP continued with the analysis of the observe option and its QoS support. Existing QoS in the CoAP observe option supports reliability and, partially, timeliness. Regarding timeliness, the observe option only allows to specify the validity of an update but not to guarantee its on-time delivery. This approach is inefficient and does not consider applications that require the delivery of an update within a deadline. With this limitation in mind, we have designed a novel mechanism to add QoS support for timeliness in the observe option. This is based on a simple update delivery method based on priority. We evaluated our proposal in a real WSN applied in an e-health application. The evaluation proved that the delivery order influences both the delay and delivery ratio. In particular, we found that the first updates experienced less delay and had a high delivery ratio, which proved the effectiveness of our proposal. Furthermore, we were able to reduce the energy consumption allowing observers to express the class of notifications that they wish to receive.

In conclusion, with this thesis we contributed to the analysis of IoT and WoT protocols and their effects on the performance and resource consumption of WSNs. Due to the novelty of these protocols there is little research over important problematic as large data transactions or uncovered evaluation of aspects such as QoS support. To this end, WSN applications used in monitoring of critical domain field, i.e. e-health, can benefit from our proposal for QoS support for timeliness. Furthermore, applications that require packet fragmentation, i.e. those that works with data logging or data aggregation, can benefit from the proposed forwarding mechanism as well as from the analytical analysis of CoAP blockwise transfer and 6LoWPAN fragmentation. TinyCoAP, instead, contributes to overcome the limitations encountered in the CoAP implementations of TinyOS. Finally, the design proposed for the CoAP proxy offers an effective solution for interconnecting Web applications to CoAP devices. The use of different communication patterns provides flexibility and allows the CoAP proxy to work in different application scenarios.

A deep analysis of the problematic related to the application of IoT and WoT paradigms in WSNs could not be addressed in a single thesis. Therefore, the research conducted here constitutes the ground on which design and develop the future of the IoT. Future developments of this thesis are individuated as:

- *Cloud Computing:* Considering the increasing presence of sensor devices and the huge amount of the data they produces, the design and development of a platform based on cloud computing is of paramount importance to manage the sensor nodes and process data. This platform could be an extension of our work on CoAP proxy.
- *Multi-radio devices:* Progress in microelectronics is leading to a new generation of sensor devices which can rely on less constrained hardware. This would affect also the wireless communications technology that can be used in WSN. In this sense, a single sensor device could support multi-radio wireless communication, i.e. Bluetooth, IEEE 802.11 and IEEE 802.15.4. The device, therefore, would be able to switch between different wireless technology standard. Next generation protocols, therefore, should be able to adapt dynamically to these changes. This thesis only considers the use of 6LoWPAN and CoAP over single-radio devices. A future extension could be the optimization of these protocols in multi-radio devices.
- *Large data transaction in Multi-hop WSNs:* The model presented in Chapter 4 can be extended to analyse both 6LoWPAN fragmentation and CoAP blockwise transfer in multi-hop networks.

References

Contributions

Journal papers:

[P1] Ludovici, A.; Calveras, A.; Casademont, J. Forwarding Techniques for IP Fragmented Packets in a Real 6LoWPAN Network. Sensors 2011, 11, 992-1008.

[P2] Ludovici, A.; Moreno, P.; Calveras, A. TinyCoAP: A Novel Constrained Application Protocol (CoAP) Implementation for Embedding RESTful Web Services in Wireless Sensor Networks Based on TinyOS. J. Sens. Actuator Netw. 2013, 2, 288-315.

[P3] Ludovici, A.; Moreno, P.; Calveras, A.; Gimeno, X. A Proxy Design to Leverage the Interconnection of CoAP Wireless Sensor Networks with Web Applications.

[P4] Ludovici, A.; Di Marco, P.; Calveras, A.; Johansson, K. H. Analytical model for large CoAP data transactions.

Conference papers:

[P5] Ludovici, A.; Calveras, A. Implementation and evaluation of Multi-hop routing in 6LoWPAN. In proceedings of the 9th conference of telematics engineering, pp. 123 -128, Valladolid, Spain, 29 September – 1 October 2010.

[P6] Ludovici, A.; Calveras, A. 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, September 7-10, 2010.

[P7] Ludovici, A.; Garcia, E.; Gimeno, X.; Calveras Auge, A., Adding QoS support for timeliness to the observe extension of CoAP," IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), pp.195-202, Barcelona, Spain, 8-10 Oct. 2012

Bibliography

[1] Gershenfeld, N.; Krikorian, R.; Cohen, D. “The Internet of Things”. Scientific American, vol. 291, no. 4, pp. 76–81, October 2004.

[2] Hoebeke, J.; Carels, D.; Ishaq, I.; Ketema, G.; Rossey, J.; Depoorter, E.; Demeester, P. “Leveraging upon standards to build the Internet of Things”. In *2012 IEEE 19th Symposium on Communications and Vehicular Technology in the Benelux (SCVT),* (pp. 1-6).

[3] Bojkovic, Z.; Bakmaz, B.; Bakmaz, M. “Some Challenging Issues for Internet of Things Realization”. In *Proc.12th International Conference on Data Networks, Communications, Computers (DNCOCO'13)*, Limassol, Cyprus, March 2013, pp. 63-70.

[4] Zorzi, M.; Gluhak, A.; Lange, S.; Bassi, A. “From today's INTRAnet of things to a future INTERnet of things: a wireless- and mobility-related view”. *Wireless Communications, IEEE*, vol.17, no.6, pp.44,51, December 2010

[5] Ishaq, I.; Carels, D.; Teklemariam, G.K.; Hoebeke, J.; Abeele, F.V.; Poorter, E.D.; Moerman, I.; Demeester, P. “IETF Standardization in the Field of the Internet of Things (IoT): A Survey”. *J. Sens. Actuator Netw.***2013**, *2*, 235-287.

[6] Palattella, M.; Accettura, N.; Vilajosana, X.; Watteyne, T.; Grieco, L.; Boggia, G.; Dohler, M., “Standardized Protocol Stack for the Internet of (Important) Things”. *Communications Surveys & Tutorials, IEEE*, vol.PP, no.99, pp.1,18, 2013

[7] Montenegro, G.; Kushalnagar, N.; Hui, J.; Culler, D. RFC 4944. Transmission of IPv6 Packets over IEEE 802.15.4 Networks.

[8] IEEE, Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs). In IEEE Standard 802.15.4-2006. Part 15.4; IEEE Computer Society: Los Alamitos, CA, USA, 2006.

[9] IP for Smart Objects Alliance (IPSO). http://www.ipso-alliance.org/. (accessed on 07 April 2014)

[10] Stirbu, V. "Towards a restful plug and play experience in the web of things." *Semantic computing, 2008 IEEE international conference on*. IEEE, 2008.

[11] Luckenbach, T.; Gober, P.; Arbanowski, S. “TinyREST - a Protocol for Integrating Sensor Networks into the Internet”, In Proc. of REALWSN, 2005

[12] Fielding, R.T.; “Architectural Styles and the Design of Network-based Software Architectures”. University of California, Irvine, 2000.

[13] Gudging, M.; Hadley, M.; Mendelsohn, N.; Moreau J.; Nielsen, H.F.; Karmarkar, A.; Lafon, Y. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). World Wide Web Consortium (W3C) Recommendation, April 2007.

[14] Bray, I. T.; Paoli, J.; Sperberg-McQueen, C.M. XML Extensible Markup Language 1.0. W3C Recommendation, http://www.w3.org/TR/1998/REC-xml-19980210. (accessed on 07 April 2014).

[15] Fielding, R.T.; Gettys, J.; Mogul, J.; Frystyk, H.; Masinter, L.; Leach, P.; Berners-Lee, T. RFC 2616. Hypertext Transfer Protocol -- HTTP/1.1.

[16] Christensen, E.; Curbera, F.; Meredith, G.; Weerawarana, S. Web Services Description Language (WSDL) 1.1. W3C Recommendation, http://www.w3.org/TR/wsdl. (accessed on 07 April 2014).

[17] CoRE IETF Working Group; Available online: http://datatracker.ietf.org/wg/core/charter/ (accessed on 07 April 2014).

[18] Shelby, Z. “Embedded web services”. Wireless Communications, IEEE, vol.17, no.6, pp.52-57, December 2010

[19] Shelby, Z.; Hartke, K.; Bormann, C.; Frank, B. Constrained Application Protocol (CoAP). draft-ietf-core-coap-18. IETF Internet-Draft.

[20] Villaverde, B. C.; Pesch, D.; De Paz Alberola, R.; Fedor, S.; Boubekeur, M.; “Constrained Application Protocol for Low Power Embedded Networks: A Survey”. *Sixth* IEEE *International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012,* (pp. 702-707)..

[21] Bormann, C.; Castellani, A.P.; Shelby, Z. "CoAP: An Application Protocol for Billions of Tiny Internet Nodes," *Internet Computing, IEEE* , vol.16, no.2, pp.62,67, March-April 201

[22] Chebrolu K.; Raman, B.; Mishra, N.; Valiveti, P. K.; Kumar. R.; “Brimon: a sensor network system for railway bridge monitoring”. In Proceeding of the 6th international conference on Mobile systems, applications, and services, Proceedings of The International Conference on Mobile Systems, Applications, and Services (MobiSys), Breckenridge, CO, USA, 2008.

[23] Kim, S.; Pakzad, S.; Culler, D.; Demmel, J.; Fenves, G.; Glaser, S.; Turon, M.; “Health monitoring of civil infrastructures using wireless sensor networks”. In Proceedings of the International Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN), pages 254–263, 2007

[24] Werner-Allen, G.; Lorincz, K.; Ruiz, M.; Marcillo, O.; Johnson, J.; Lees, J.; Welsh, M.; “Deploying a wireless sensor network on an active volcano”. *Internet Computing, IEEE*, *10*(2), 18-25, 2006.

[25] Mesrinejad, F.; Hashim, F.; Noordin, N.K.; Rasid, M. F A; Raja Abdullah, R.S.A., "The effect of fragmentation and header compression on IP-based sensor networks (6LoWPAN),"*17th Asia-Pacific Conference on Communications (APCC),* pp. 845-849, 2-5 Oct. 2011

[26] Bormann, C.; Shelby, Z. CoAP blockwise transfers in CoAP. draft-ietf-core-block-14. IETF Internet-Draft.

[27] Hartke, K.; Observing Resources in CoAP. draft-ietf-core-observe-11. IETF Internet-Draft.

[28] Levis, P.; Madden, S.; Polastre, J.; Szewczyk, R.; Woo, A.; Gay, D.; Hill, J.; Welsh, M.; Brewer, E.; Culler, D. TinyOS: An Operating System for Sensor Networks. In *Ambient Intelligence*; Springer; Berlin, Germany, 2005; pp. 115–148.

[29] Mulligan, G. "The 6LoWPAN architecture". In Proceedings of the 4th workshop on Embedded networked sensors (EmNets '07). ACM, New York, NY, USA, 78- 82, 2007.

[30] Hui, J.; Culler, D.; "Extending IP to Low-Power, Wireless Personal Area Networks," IEEE Internet Computing, pp. 37-45, July/August, 2008

[31] Hui, J.; Culler, D. "IPv6 in Low-Power Wireless Networks", Proceedings of the IEEE , vol.98, no.11, pp.1865-1878, Nov. 2010

[32] Kim, E.; Kaspar, D.; Gomez, C.; Bormann, C. , RFC6606, Problem Statement and Requirements for 6LoWPAN Routing.

[33] Chowdhury, A.H; Ikram, M.; Cha, H.; Redwan, H.; Saif Shams, S.M.; Kim, K.; Yoo. S. "Route-over vs mesh-under routing in 6LoWPAN". In Proceedings of the 2009 International Conference on Wireless Communications and Mobile Computing: Connecting the World Wirelessly (IWCMC '09). ACM, New York, NY, USA, 1208-1212.

[34] Hincapie, D.F.R.; Cespedes, S. "Evaluation of mesh-under and route-over routing strategies in AMI systems," *2012 IEEE Colombian Communications Conference (COLCOM),* pp.1-6, 16-18 May 2012.

[35] Oliveira, L. M. L.; de Sousa, A. F.; Rodrigues, J. J. P. C. "Routing and mobility approaches in IPv6 over LoWPAN mesh networks". International Journal of Communication Systems, 2011.

[36] Zhu, Y. H.; Chen, G.; Chi, K.; Li, Y.; The Chained Mesh-Under Routing (C-MUR) for Improving IPv6 Packet Arrival Rate over Wireless Sensor Networks. In *Advances in Wireless Sensor Networks* (pp. 734-743). Springer Berlin Heidelberg, 2013.

[37] Bhunia, S.S.; Sikder, D.K.; Roy, S.; Mukherjee, N. "A comparative study on routing schemes of IP based wireless sensor network". *9th International Conference Wireless and Optical Communications Networks (WOCN), on* , vol., no., pp.1,5, 20-22 Sept. 2012

[38] Shelby, Z.; Bormann, C. "6LoWPAN: The Wireless Embedded Internet". 1th ed.; John Wiley & Sons Ltd: Chichester, UK, 2009; pp. 40-41.

[39] Hui, J. "An Extended Internet Architecture for Low-Power Wireless Networks— Design and Implementation". PhD Thesis, University of California, Berkeley, CA, USA, 2008; pp 85-87.

[40] Gopinath, R.S.; Khan, I.; Suryady, Z., "Optimized web service architecture for 6LoWPAN". International Conference on Information Networking, ICOIN 2009, pp.1-3, 21-24 Jan. 2009

[41] Moritz, G.; Zeeb, E.; Golatowski, F.; Timmermann, D.; Stoll, R. "Web Services to improve interoperability of home healthcare devices". 3rd International Conference on Pervasive Computing Technologies for Healthcare. PervasiveHealth 2009, pp.1-4, 1-3 April 2009

[42] Lerche, C.; Laum, N.; Moritz, G.; Zeeb, E.; Golatowski, F.; Timmermann, D. "Implementing powerful Web Services for highly resource-constrained devices". 9th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), pp. 332-335, 21-25 March 2011

[43] Chan, S. et al. "Devices Profile for Web Services (DPWS) Specification". 2006.

[44] Groba H.; Clarke, S. "Web services on embedded systems - a performance study". 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), pp.726-731, March-April 2 2010

[45] Guinard D.; Trifa V. “Towards the Web of Things: Web Mashups for Embedded Devices||”. In Proc. Workshop Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM '09), April 2009.

[46] Tolle, G. Embedded Binary HTTP (EBHTTP), draft-tolle-core-ebhttp-00, IETF Internet Draft.

[47] Dawson-Haggerty, S.; Jiang, X.; Tolle, G.; Ortiz, J.; Culler, D. “sMAP – a Simple Measurement and Actuation Profile for Physical Information”. In Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys’10), Zurich, Switzerland, 3-5 November 2010.

[48] Frank, B. Chopan - Compressed HTTP Over PANs, draft-frank-6lowapp-chopan-00, IETF Internet Draft.

[49] Lerche, C.; Hartke, K.; Kovatsch, M. “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), Kraków, Poland, 17-21 September 2012.

[50] Kovatsch, M.; Duquennoy, S.; Dunkels, A. “A Low-Power CoAP for Contiki”. In Proceedings of Eighth IEEE International Conference on Mobile Ad-Hoc and Sensor Systems, Valencia, Spain, 17–22 October 2011; pp. 855–860.

[51] Kuladinithi, K.; Bergmann, O.; Pötsch, T.; Beckera M.; Görg, C. “Implementation of CoAP and its Application in Transport Logistics”. In Proceedings of Extending the Internet to Low power and Lossy Networks (IP+SN 2011), Chicago, IL, USA, 11 April 2011.

[52] Castellani, A.P.; Gheda, M.; Bui, N.; Rossi, M.; Zorzi, M. “Web Services for the Internet of Things through CoAP and EXI”. In Proceedings of IEEE International Conference on Communications Workshops (ICC), Kyoto, Japan, 5–9 June 2011; pp 1–6.

[53] Potsch, T.; Kuladinithi, K.; Becker, M.; Trenkamp, P.; Goerg, C. “Performance Evaluation of CoAP Using RPL and LPL in TinyOS”. In Proceedings of 5th International Conference on New Technologies, Mobility and Security (NTMS), Istanbul, Turkey, 7–10 May 2012

[54] Harvan, M.; Schoenwaelder, J. TinyOS Motes on the Internet: IPv6 over 802.15.4 (6lowpan). Praxis der Informationsverarbeitung und Kommunikation (PIK), **2008**, *31*, 244–251.

[55] Silva, R.; Sá Silva, J.; Boavida, F. “Evaluating 6LoWPAN Implementations in WSNs.” In Proceedings of the 9th Conference on Computer Networks, Oeiras, Portugal, 15–16 October 2009.

[56] Yibo, C.; Hou, K.; Zhou, H.; Shi, H.; Liu, X.; Diao, X.; Ding, H.; Li, J.; De Vaulx, C. “6LoWPAN Stacks: A Survey”. In Proceedings of 7th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM), 2011, Wuhan, China, 23–25 September 2011; pp.1–4.

[57] Colitti, W.; Steenhaut, K.; De Caro, N.” Integrating Wireless Sensor Networks with the Web”. In Proceedings of Extending the Internet to Low power and Lossy Networks (IP+SN 2011), Chicago, IL, USA, 11 April 2011.

[58] Colitti, W.; Steenhaut, K.; De Caro, N.; Buta, B.; Dobrota, V. “Evaluation of Constrained Application Protocol for Wireless Sensor Networks”. In Proceedings of 18th IEEE International Workshop of Local and Metropolitan Area Networks (LanMan), Chapel Hill, NC , USA, 13-14 October 2011.

[59] Duquennoy, S.; Wirstom, N.; Tsiftes, N.; Dunkels, A. “Leveraging IP for Sensor Network Deployment”. In Proceedings of Extending the Internet to Low power and Lossy Networks (IP+SN 2011), Chicago, IL, USA, 11 April 2011.

[60] Chander, R.P.V.; Elias, S.; Shivashankar, S.; Manoj, P. “A REST based Design for Web of Things in Smart Environments”. In Proceedings of 2nd IEEE International Conference on Parallel Distributed and Grid Computing (PDGC), Waknaghat, Solan, Himachal Pradesh, India, 6–8 December 2012; pp. 337–342.

[61] Colitti, W.; Steenhaut, K.; De Caro, N.; Buta, B.; Dobrota, V. “REST Enabled Wireless Sensor Networks for Seamless Integration with Web Applications”. IEEE 8th International Conference on Mobile Ad hoc and Sensor Systems (MASS), pp.867-872, 17-22 Oct. 2011

[62] Young Ki, P.; Ngoc-Thanh, D.; Younghan, K. “A network monitoring system in 6LoWPAN networks”. Fourth International Conference on Communications and Electronics (ICCE), pp.69-73, 1-3 Aug. 2012

[63] Bergmann, O.; Hillmann, K.T.; Gerdes, S. "A CoAP-gateway for smart homes." IEEE *International Conference on Computing, Networking and Communications (ICNC),* 2012.

[64] Evangelatos, O.; Samarasinghe, K.; Rolim, J. “Syndesi: A Framework for Creating Personalized Smart Environments using Wireless Sensor Networks”. Available online. http://www.hobnet-project.eu/files/papers/iotip2013.pdf (accessed on 07 April 2014).

[65] Bandyopadhyay, S.; Bhattacharyya, S. “Lightweight Internet Protocols for Web Enablement of Sensors using Constrained Gateway Devices”. 2013 International Conference on Computing, Networking and Communications, Workshops Cyber Physical System.

[66] MQ Telemetry Transport, Available online: http://mqtt.org/ (accessed on 07 April 2014).

[67] Carvalho, N.; Araújo, F.; Rodrigues, L. “Scalable QoS-Based Event Routing in Publish-Subscribe Systems”, In Proc. of the Fourth IEEE International Symposium on Network Computing and Applications (NCA 2005).

[68] Fengyun. C.; Jaswinder, P.S. Medym: match-early with dynamic multicast for content based publish-subscribe networks”. In Proceedings of the ACM/IFIP/USENIX ’05 International Conference on Middleware, pages 292–313, New York, NY, USA, 2005.

[69] Cugola, G.; Di Nitto, E.; Fuggetta, A. “The jedi event-based infrastructure and its application to the development of the OPSS WFMS”, IEEE Transactions on Software Engineering, pages 827-850, 2001.

[70] Jacobsen, H.A.; Cheung, A.; Li, G.; Maniymaran, B.; Muthusamy, V.; Kazemzadeh, R.S. “The PADRES Publish/Subscribe System” In Principles and Applications of Distributed Event-Based Systems, pages 164-205, IGI Global, 2010.

[71] Gruber, B.; Krishnamurthy, E.; Panagos, E. “The architecture of the READY event notification service” Electronic Commerce and Web-based Applications/Middleware, 1999. Proceedings. 19th IEEE International Conference on Distributed Computing Systems Workshops on , pp.108-113, 1999.

[72] Pietzuch, P.R.; Bacon, J. “Hermes: A Distributed Event-Based Middleware Architecture”, Proceedings of the 22nd International Conference on Distributed Computing Systems, p.611-618, July 02-05, 2002.

[73] Esposito, C.; Platania, M.; Beraldi, R. "Reliable and Timely Event Notification for Publish/Subscribe Services Over the Internet," *IEEE/ACM Transactions on Networking*, vol. no.99, pp.1,1, 2013.

[74] Hunkeler, U.; Truong, H.L.; Stanford-Clark, A. “MQTT-S — A publish/subscribe protocol for Wireless Sensor Networks”, COMSWARE 2008. 3rd International Conference on Communication Systems Software and Middleware and Workshops, Jan 2008, pp 791-798.

[75] Davis, E.; Calveras, A.; Demirkol, I. “Improving Packet Delivery Performance of Publish/Subscribe Protocols in Wireless Sensor Networks”. Sensors 2013, 13(1), 648-680.

[76] Sharifi, M.; Taleghan, M.A; Taherkordi, A. “A Publish-Subscribe Middleware for Real-Time Wireless Sensor Networks”, International Conference on Computational Science; ICCS 2006, LNCS, 2006, Volume 3991/2006, Page(s): 981-984.

[77] Sharifi, M.; Taleghan, M.A; Taherkordi, A. “A Middleware Layer Mechanism for QoS Support in Wireless Sensor Networks”, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, 2006. ICN/ICONS/MCL 2006, pp. 118, 23-29 April 2006.

[78] Chen, J.; Díaz, M.; Rubio, B.; Troya, J. M. “PS-QUASAR: A publish/subscribe QoS Aware Middleware for Wireless Sensor and Actor Networks”. *Journal of Systems and Software.* (2013).

[79] Ketema, G.; Hoebeke, J.; Moerman, I.; Demeester, P.; Li Shi Tao; Jara, A.J. "Efficiently Observing Internet of Things Resources”, *IEEE International Conference on Green Computing and Communications (GreenCom), 2012*, pp.446,449, 20-23 Nov. 2012.

[80] Li, S. T, Conditional observe in CoAP, draft-li-core-conditional-observe-04, IETF Internet Draft.

[81] Sammarco, C.; Iera, A. “Improving Service Management in the Internet of Things”. Sensors 2012, 12, 11888-11909.

[82] Deering, S.; Hinden, R., RFC2460, Internet Protocol, Version 6 (IPv6) Specification;

[83] Hui, j.; Thubert, P, RFC 6282, Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks.

[84] Crossbow Technology Inc.; *TelosB Datasheet*; Available online: http://www.willow.co.uk/TelosB_Datasheet.pdf (accessed on 07 April 2014).

[85] Blip; Available online: http://smote.cs.berkeley.edu:8000/tracenv/wiki/blip (accessed accessed on 24 July 2013).

[86] Gay, D.; Levis, P.; Von Behren, R.; Welsh, M.; Brewer, E.; Culler, D. The nesC language: A holistic approach to networked embedded systems. *ACM SIGPLAN Not.* 2003, *38*, 1–11.

[87] Shelby, Z, RFC6990, CoRE Link Format. Levis, P. TinyOS Programming. Available online: http://csl.stanford.edu/~pal/pubs/tinyos-programming.pdf (accessed on 07 April 2014).

[88] Lauwens, B.; Scheers, B.; Van de Capelle, A. Performance analysis of Unslotted CSMA/CA in wireless networks. *Telecommun. Syst.* **2010**, *44*, 109–123.

[89] Paxson, V; Allman, M.; Chu, J.; Sargent, M., RFC6298, Computing TCP’s Retransmission Timer.

[90] Goyal, M.; Prakash, S.; Xie W.; Bashir, Y.; Hosseini, H.; Durresi, A. Evaluating the Impact of Signal to Noise Ratio on IEEE 802.15.4 PHY-Level Packet Loss Rate. In Proceedings of 13th International Conference on Network-Based Information Systems, Takayama, Japan, 14–16 September 2010.

[91] S. Duquennoy, G. Grimaud, J.H Vandewalle, "Consistency and scalability in event notification for embedded Web applications," 11th IEEE International Symposium on Web Systems Evolution (WSE), pp.89,98, 25-26 Sept. 2009

[92] I. Fette; A. Melnikov, RFC 6455, The WebSocket Protocol.

[93] T. Berners-Lee, R. Fielding, L. Masinter, RFC 3986, Uniform Resource Identifier (URI): Generic Syntax.

[94] P. Leach, M. Mealling, R. Salz, RFC4122, A Universally Unique IDentifier (UUID) URN Namespace.

[95] A. Castellani, S. Loreto, A. Rahman, T. Fossati, E. Dijk, Best Practices for HTTP-CoAP Mapping Implementation. draft-castellani-core-http-mapping-07, IETF Internet Draft.

[96] Z. Shelby, S. Krco, C. Bormann, CoRE Resource Directory, draft-shelby-core-resource-directory-05, IETF Internet Draft.

[97] Lighttpd. Available online: http://www.lighttpd.net/ (accessed on 07 April 2014).

[98] Z. Shelby, S. Chakrabarti, E. Nordmark, C. Bormann, RFC 6775, Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs).

[99] FastCGI. Available online: http://www.fastcgi.com/ (accessed on 07 April 2014)

[100] A. Shelby, RFC 6690, CoRE Link Format.

[101] M. Nottingham, RFC 5988, Web Linking.

[102] J. Åkerberg, M.M. Gidlund, M. Björkman, Future Research Challenges in Wireless Sensor and Actuator Networks Targeting Industrial Automation, *Proc. 9thIEEE Int'l Conf. Industrial Informatics* (INDIN 11), IEEE Press, 2011, pp. 410–415.

[103] O. Gama, P. Carvalho, J. Afonso, P. M. Mendes, "Quality of service in wireless e-emergency: main issues and a case-study," In proc. of 3rd UCAmI, Salamanca, Spain, Oct. 2008.

[104] Di Marco, P; Pangun, P.; Fischione, C.; Johansson, K.H.; "Analytical Modeling of Multi-hop IEEE 802.15.4 Networks". IEEE Transaction on Vehicular Technology, vol 61, no.7, pp.3191-3208, September 2012

[105] Jianliang Gao; Jia Hu; Geyong Min, "Performance Modelling of IEEE 802.15.4 MAC in LR-WPAN with Bursty ON-OFF Traffic," Computer and Information Technology, 2009. CIT '09. Ninth IEEE International Conference on, vol.2, no., pp.58-62, 11-14 Oct. 2009