Wireless Sensors Networks (WSN) monitoring : application to secure interoperability

Raul Armando Fuentes Samaniego

NNT : 2017SACLL005

These de doctorat
de
L'Universite Paris-Saclay
preparee a
TELECOM SudParis

Ecole Doctorale N° 580
Sciences et technologies de l'information et de la communication

Spécialité de doctorat: Informatique

Par

M. Raul Armando Fuentes Samaniego

Monitorage des réseaux des capteurs sans fils (WSN) - Application a l'interopérabilité sécurisée

Thèse présenté et soutenue à Evry, le 27/02/2017 :

Composition du Jury :

M. Garcia-Alfaro, Joaquin	Professeur, Télécom SudParis - Univ. Paris Saclay	Président
M. Lochin, Emmanuel	Professeur, ISAE	Rapporteur
Mme. Merayo, Mercedes	Professeur, Universidad Complutense de Madrid	Rapporteur
M. Natalizio, Enrico	Maître de conferences, Université de Compiègne	Examinateur
M. Montes de Oca, Edgardo	Montimage	Examinateur
Mme. Cavalli, Ana	Professeur, Télécom SudParis - Univ. Paris Saclay	Directeur de thèse
M. Nolazco, Juan	Professeur, Instituto Tecnologico de Monterrey	Invité

Abstract

Wireless Sensors Networks (WSN) monitoring - Application to secure interoperability

by: *Raul Armando Fuentes-Samaniego*

The denominated "Internet of Things" (IoT) has been getting relevance in both public and research communities. The main reason is that on 2011, the number of "objects" connected to the Internet surpassed the number of humans online, and is expected that for 2020, the number of objects exceeds the amount of 20 billion.

Because of the high number of heterogeneous platforms that composed the IoT, our interest is centered around the Wireless Sensors Networks (WSN), which are composed by hundred, or even thousands, of small devices with constrained resources (small amount of memory, small power processor, and small power supply) that collect one or more type of data. Almost all the research conducted to date relies on standardizing the communication protocols, ameliorating the performance, optimizing the resource consumption, etc.

Security has been relegated to a second plane, due mainly to the low resources available on the sensors. However, the data collected in many scenarios can be highly sensitive, e.g. when the sensors collect health parameters, driving plates, biometrical data from pedestrian, etc. The data must be stored in a safe way and must be transmitted in a safe approach from the origin to the destiny, in a similar way than in traditional networks.

The work developed in this dissertation defines mechanisms to guarantee the safety of the communication between sensors, following especially the well-known HTTP Secure (HTTPS) protocol of traditional networks. And also providing native tools for the monitoring of the communication, in order to validate these mechanisms directly on the network.

Résumé

Monitorage des réseaux des capteurs sans fils (WSN) - Application a l'interopérabilité sécurisée

par: *Raul Armando Fuentes-Samaniego*

La formule "Internet of Things" a pris du sens à la fois au sein de la communauté public et de recherche. La raison principale est qu'en 2011, le nombre d'objets connectés à Internet surpassent le nombre d'humains en ligne, et il est attendu qu'en 2020, le nombre d'objets connectés dépassent les 20 billions.

Etant donné la présence d'un grand nombre de plateformes hétérogènes qui composent l' IoT, notre intérêt se focalise sur les Reseaux de Capteurs (WSN), qui sont composés de cent, voire mille, petits dispositifs avec des contraintes de ressources (capacité de mémoire faible, processeur de faible puissance et faible puissance matérielle) qui collectent un ou plusieurs types de données. Presque toutes les recherches menées à ce jour reposent sur la standardisation de protocoles de communication, l'amélioration de la performance, l'optimisation de la consommation de ressources, etc.

La sécurité a été relégué au second plan, dû principalement au faibles ressources disponibles sur les capteurs. Cependant, les données collectées dans de multiples scénarios peuvent être très sensibles, cf. les paramètres de santé, les plaques d'entrainement, les données biométriques de piétons, etc. Les données doivent être stockées de manière sûre et doivent être transmises de manière sûre d'un point à un autre, comme c'est le cas pour les réseaux traditionnels.

Le travail développé dans cette thèse définit les mécanismes permettant de garantir une communication sûre entre les capteurs, reposant sur un protocole très connu, le HTTP Secure (HTTPS) des réseaux traditionnels. Et aussi fournissant des outils natifs pour le monitorage des communications, afin de valider ces mécanismes directement sur le réseaux.

Acknowledgments

I want to thank my supervisor, Prof. Ana Cavalli by giving me the opportunity for this journey and research and by all her support, guidance and encouragement through my doctoral studies. I also express my gratitude to Prof. Juan Nolazco and Dr. Raul Ramirez for helping me to start this journey. I want to express my gratitude to Prof. Mercedes Merayo and Prof. Emmanuel Lochin for kindly accepting to be the reviewers for this thesis. I would also like to thank Prof. Joaquin Alfaro, Prof. Enrico Natalizio and Dr. Edgardo Montes de Oca for accepting to be part of my jury committee. I also want to express my gratitude to Prof. Javier Baliosian, Dr. Natalia Kushik and Dr. Anis Laouiti who have helped me in various ways while preparing the present thesis work. I also thank the staff of Telecom SudParis for always helping me with gratitude, among them Veronique Guy, Brigitte Laurent and Valerie Mateus.

I want to thank the staff of Montimage, with whom I had the opportunity to work while preparing my thesis: Edgadrdo, Bachar, Wissam and Antonio. Their help concerning practical design and implementation is invaluable. I also thank to the people from the IDOLE project (Investigation et Détection Opérées à Large Echelle) from which my PhD research was funded. I really appreciate the chance to participate in the project which provided me a practical view and contributed to prepare the basis for this thesis

I must thank all my colleagues and friends of from the team of former Logiciels-Réseaux (LOR) department (currently joined to RS2M) at Telecom SudParis for the joint cross-topic work we have done together and that has made a difference to this research: Xiapoing, Khalifa, João, Jorge, Olga, Fabrice, Fatma, Huu Nghia, Kun, Ichrak, Jeevan, Asma L., Diego, Pablo, Jose (Chepe), Denisa, Sarah, Asma B., Pamela, Mohamed H., Stephane and Nina. In special, I want to thank Vinh Hoa La by all the time we spent discussion about our collaborative works. Also, I want to thank to the community around the RIOT project: Martine, Francisco, Oliver, Jose, Alexander, between many others.

Finally, I truly thank my family, which has supported me through all my academic trajectory. Many thanks to my mother and father. I want to thank Erika and Eloi; Jeanne and Mathias; Pascalle and Alain; for all their help and support when I arrived to France. I also thank by the support given by my friends Lirida, Diana, Barbara an Dulce. Special thanks to all the friends I have met here: Hector, Elsa, Luis, Alice, Vlad, Pratysh, Jay, Alynka, Caroline and many others I was unable to mention here due to the reduced space but I truly thank you my unnamed friends!

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

6BR 6LoWPAN Border Router.

6LoWPAN IPv6 over Low power Wireless Personal Area Networks.

CBOR Concise Binary Object Representation.

CoAP Constrained Application Protocol.

CoAPS CoAP Secure.

CSMA/CA carrier sense multiple access/collision.

DDoS Distributed Denial of Service.

DPI Deep Packet Inspection.

DTLS Datagram Transport Layer Security.

ETSI European Telecommunications Standards Institute.

EXI Efficient XML Interchange.

HTTP Hypertext Transfer Protocol.

HTTPS HTTP Secure.

ICMPv6 ICMP for IPv6.

IETF Internet Engineering Task Force.

IoT Internet of Things.

JSON JavaScript Object Notation.

M2M Machine to Machine.

MMT Montimage Monitoring Tool.

MTU Maximum Transmission Unit.

NDP Neighbor Discovery Protocol.

PKI Public Key Infrastructure.

ReST Representational State Transfer.

RFID Radio-frequency identification.

RPL IPv6 Routing Protocol for Low-Power and Lossy Networks.

SLAAC Stateless Auto-configuration.

TLS Transport Layer Security.

WSN Wireless Sensor Networks.

XML Extensible Markup Language.

Chapter 1

Introduction

Contents

1.1 Motivation

The denominated Internet of Things (IoT) has been getting an amazing relevance in both the public and research communities as in 2011 the number of "objects" connected to the Internet surpassed the number of humans online, and it is expected that for 2020, the number of objects will exceed the amount of 20 million [13, 14]. The IoT can be defined today as *a world-wide network of interconnected objects uniquely addressable, based on standard communication protocols* [15] which can provide real-time information. Examples of platforms using this technology are: Radio-frequency identification (RFID) tags, sensors, actuators, smart watches, Raspberry devices, surveillance devices, etc.

Those devices connected to the Internet are composed of a highly heterogeneous variety of architectures and some of them even lack any traditional Internet protocols. As a consequence many devices are improperly configured making it easy to steal information or even manipulating them. This situation begins to be referred by some professionals as the "Internet of Vulnerabilities" or "Internet of Threats" [16]. In September 2016, a new record for a Distributed Denial of Service (DDoS) attack took place. Previous to this event, the record for such attacks was 363 Gbps, but in this attack, a peak of 620 Gbps was reached against a network security popular author's website [17]. This record was surpassed just some days later at an attack against OVH, a major web hosting provider based in France with a peak attack of more than 1Tbps [18]. In October 2016 Dyn, a company that provides core Internet services for different popular sites such as Netflix, Twitter, Airbnb, Spotify and Reddit, was the victim of a DDoS attack with a peak of 1.2 Tbps [19]. These three attacks have something in common: the malicious use of compromised IoT devices. The attacks were not complex and are even considered "script kiddie" level [20] (too little knowledge required for attackers). However, due

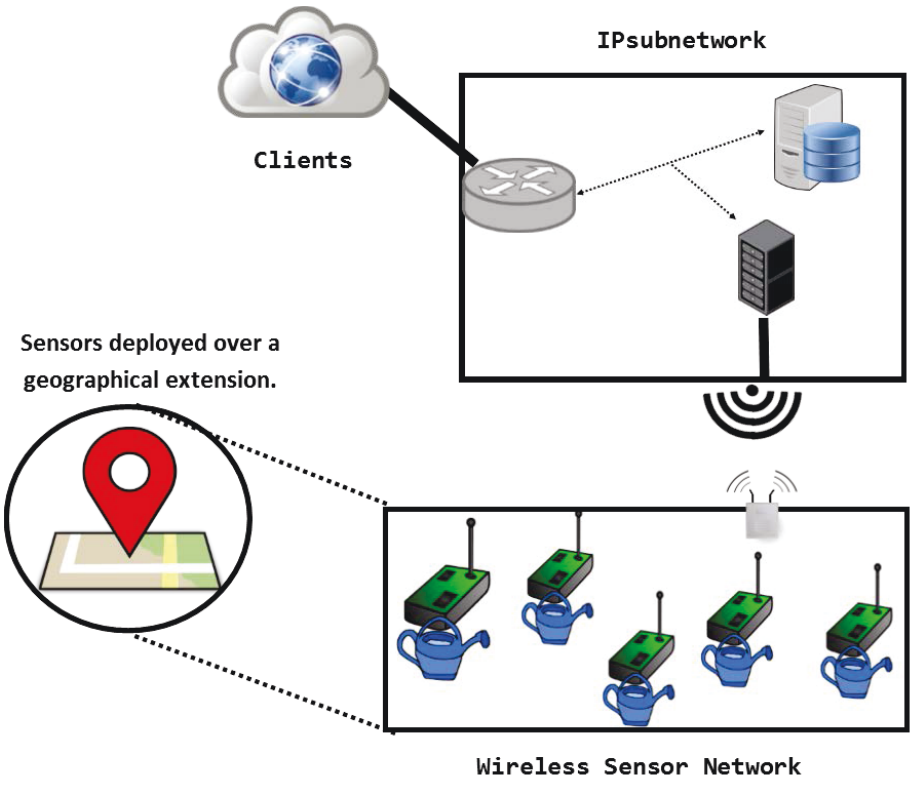


Figure 1.1: A general representation of an IoT environment using WSN.

to the almost ridiculously high number of unprotected IoT devices that got under an attack control, this has become a serious threat.

Many of the devices compromised for reaching the previous DDoS attacks fall under the surveillance category while some of them are even considered intermediary devices; however they are just a window to the current problem involving any architecture considered as IoT: security integration as a part of the core design is often ignored. To find a solution for this however, it is crucial to start focusing only on one of the many available architectures.

Of particular interest for us is the Wireless Sensor Networks (WSN) architecture, which can benefit from a high number of tasks related to monitoring, such as the smart homes, traffic congestion, waste management, health and military applications among others. The WSN has a wide range of architectures with their own protocols and focus (industry, home, wearable). Examples of said WSN architectures are: Zigbee, WirelessHART, Bluetooth Low-Energy (BLE), ISA1000.11a, 802.11ah, Z-wave, INSTEON, Wavenis and ANT+. See Section 2.2 for more details.

A representation of a WSN architecture for a watering system can be seen in Figure 1.1. The total amount of sensors varies according to the scenario and can reach a number close to a thousand. If those sensors were compromised, they could be utilized in a similar approach to the previously described DDoS attacks. Not only the malicious use for performing unauthorized actions is the sole problem but also the data captured by the sensors could be sent to unauthorized parties. This can cause a serious problem when the data is more than simple metrics for a watering system, for example a health parameter including identification of the patient, full plate numbers, data collected by military service, etc.

Name	Data size (e.g., RAM)	code size (e.g., Flash)
Class 0, C0	<<10 KiB	<<100 KiB
Class 1, C1	≈ 10 KiB	≈ 100 KiB
Class 2, C2	≈ 50 KiB	≈ 100 KiB

Table 1.1: Classes of Constrained Devices (KiB = 1024 bytes) [9].

Most of the WSN architectures consist of small devices, the sensors, able to do a measurement of at least one type of data. Sensors are distributed in an ad hoc manner over a specific area. The sensors transmit data collected to a main server, or directly to clients who requested the data.

The sensors are characterized by having a low economic cost. They provide a limited amount of resources such as a small amount of volatile and non-volatile memory, a low power microcontroller and a small power supply. The sensors can be categorized in three categories as shown in Table 1.1. The C0 devices usually cannot be configured for secure communication with the Internet as they are required to handle in a special approach due to their extremely limited resources. C1 devices are potentially able to communicate to the Internet and are used mainly for this purpose, however they do not usually implement any security system. C2 sensors are more powerful and are susceptible to the same benefits and weakness as the C1. Consequently, most of the IoT research projects instead of giving attention to the security are focused on the standardization of the communication protocols, improvement of the performance of the IoT systems, optimization of the resource consumption, etc.

1.2 Contributions

The main contributions of this dissertation are 1) the proposal and implementation of a security monitoring tool for WSN; and 2) the proposition and testing of a WSN platform with standardization and security as the bases, including the controls for detecting and preventing misbehavior or wrongful configurations.

Figure 1.2 shows the intended definition of an "IoT environment". This environment is the result of five criteria that help to understand the requirements and challenges for IoT:

1. Things: The classification of the "Interconnected objects" according to their architecture (WiFi, WSN, Bluetooth, RFID, etc.). For this dissertation, our main interest relays on WSN.

2. Semantics: This involves all the resources handled internally by the objects (sensors): data representation, processing of the internal commands, etc.

3. Internet: A "standard communication protocol" to be able to connect objects over a "world-wide network". While "Internet" part is self evident, not all the WSN have used the TCP/IP for communication.

4. Privacy: Legal and ethical implications of the data collected, processed and transmitted by the objects, if required.

5. Security: how standard security is achieved in traditional networks and the way that permits to achieve Confidentiality, Integrity, Availability (CIA), Authentication and non-Repudiation.

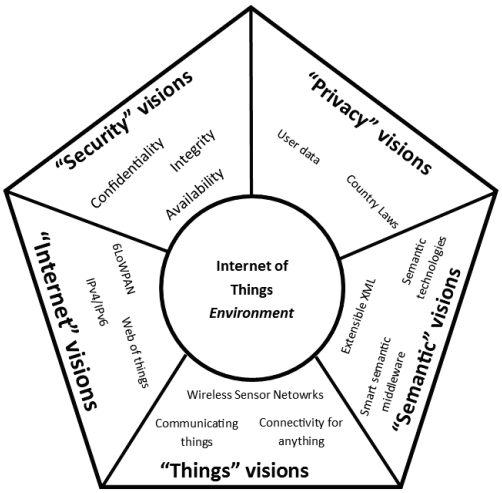


Figure 1.2: The five issues to consider for developing an real IoT with security on mind.

The first three criteria have been widely studied in multiple works in the last years with performance as the main objective. They have been a challenge because of the restriction of constrained nodes, as shown in Table 1.1. However, because of this same reason, the last two criteria are usually left out even if they are not new at all: handling sensitive data between devices without the intervention of humans has been done for years in the denominated Machine to Machine (M2M) communication. Nonetheless, they require more resources and an extra overhead in the communication so they are usually left as secondary goals for the IoT.

The requirements for the security criteria are the same as for standard entities (yet they are not easy to achieve in the IoT [21] as it will be seen throughout this dissertation):

- Confidentiality: The data collected by the sensors must be accessible only to legitimate devices; this also involves commands that affect the sensor performance.
- Integrity: The accuracy and reliability of the data transmitted over the network must be guaranteed.
- Availability: The resources of the WSN must be available even under attack. This ensures reliability and survivability of the resources in the WSN.
- Authentication: The nodes involved in the WSN must be able to verify the identity of other nodes.
- Authorization: Only authorized devices should have access to the WSN resources.
- Non-repudiation: A node cannot deny in the future the authorship of a previous message sent.
- Freshness: Avoid old messages from being re-sent without being detected.

Therefore, this brings simple questions: Is it possible to bring the same techniques to guarantee CIA on well-known M2M communication to the WSN? Does the impact on the performance of the sensors worth it? Is it possible to ensure that all the communication is being done safely? The second question has already been answered with the DDoS attacks: even if the

data is not relevant, the risk of an unauthorized use of the sensors must be reduced as much as possible. For the first question, on a standard M2M communications, Hypertext Transfer Protocol (HTTP) is employed with Secure Socket Layer (SSL) and Transport Layer Security (TLS) to reach HTTP Secure (HTTPS). Yet, SSL and TLS are not infallible, misconfigurations in the protocols, even those intentionally created by nation entities [22] force us to monitor such protocols for detecting the exploits as soon as possible.

Considering the sum of the five criteria, an architecture for WSN based on the standard IEEE 802.15.4 and IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) has been selected. This WSN is able to have a native presence over the Internet (TCP/IP) and all the protocols involved in reaching a secure M2M communication in this WSN. The list of WSN platforms and the reason for our selection are discussed in Section 2.2 meanwhile most of the internal characteristics of the WSN 802.15.4/6LoWPAN are discussed in detail in Section 3.2.

Monitoring and analyzing the communication over WSN brings unique challenges: it is too easy to lose access to the communication of the more distant nodes in the mesh topology. Also, the constrained nature of the nodes, makes it hard to use some of them as secondary devices for the monitoring. This is strongly discussed in many works related to this topic, as it is seen in Section 2.5. Even for 802.15.4/6LoWPAN it is not an easy task, as the adaptation on 6LoWPAN for the IPv6 network makes it hard to monitor the native traffic. To solve this problem, Montimage Monitoring Tool (MMT) [8] is used as the core of our proposed monitoring activity. To the best of our knowledge, there are not other monitoring tools able to understand directly 6LowPAN traffic for processing the upper layers. Also, this work proposes a number of techniques for detecting misconfiguration in the communication of the sensors together with a series of experiments to validate our monitoring over an IoT environment.

As already stated, a monitoring tool is useless if the configuration of the nodes involved in the communication is wrongly done. The communication inside the WSN is classified as M2M. This is because the sensors inside are autonomous and the external clients will operate without human intervention, though some exceptional clients could be under direct human control. This type of communication and its threats are well known and there are entities that have identified the main threats and their countermeasures (the use of HTTPS is one of them). For the proposed IoT environment, we will follow the ETSI Technical Committee M2M specifications for the M2M countermeasure together with the Internet Engineering Task Force (IETF) proposal of standardization with the use of Constrained Application Protocol (CoAP) and Datagram Transport Layer Security (DTLS) for reaching CoAP Secure (CoAPS) (a near equivalent to HTTPS).

Therefore, our monitoring tool will be first adapted to be able to work natively with all the protocols involved in our IoT environment. And after this, in concordance with the ETSI Technical Committee M2M, we will define our monitoring rules for identifying any configuration that can weaken the communication of the sensors.

1.3 Publications

The list below shows (in ascending order by date of publication) the articles developed from this dissertation work, together with a brief description about each one.

- R. Fuentes-Samaniego, A. Cavalli, J. Nolazco-Flores, J. Baliosian, *A survey on wireless sensors networks security based on a layered approach*, In proceedings of the 13th International Conference on Wired/Wireless Internet Communications (WWIC 2015), pp. 77

- 93, May 2015, Malaga, Spain.

  This survey follows a revision of the state of art in a layer-by-layer systematic analysis. The purpose of this paper was to generate a review of the state of the art regarding to the IoT, taking as a platform the WSN whose sensors work with the IEEE 802.15.4 standard. The main contribution for the dissertation work is the proposed ensemble of the protocols 802.15.4, 6LoWPAN, DTLS, CoAP and EXI as a solution for a safe M2M communication.

- R. Fuentes-Samaniego, A. Cavalli, J. Nolazco-Flores, *An analysis of secure M2M communication in WSNs using DTLS*, In proceedings of the 2nd IEEE International Workshop on Security Testing and Monitoring (STAM 2016), June 2016, Nara, Japan

  This paper introduces the analysis of well-known threats related to the M2M communication and the possible mitigation inside of the WSN (802.15.4/6LoWPAN), with the restriction related to the resources of the available devices. Of particular interest is the analysis of the Datagram TLS protocol and the proposed monitoring rules to validate the mitigation are being taken.

- V. La, R. Fuentes-Samaniego, A. Cavalli, *A novel monitoring solution for 6LoWPAN-based Wireless Sensor Networks*, In proceedings of the 22nd Asia-Pacific Conference on Communications (APCC 2016), August 2016, Yogyakarta, Indonesia

  This collaborative work is focused on the development of a monitoring solution, MMT (Montimage Monitoring Tool), which enables data capture, events extraction, statistics collection, traffic analysis and reporting. The tool is applicable to 6LoWPAN-based (IPv6 over Low power Wireless Personal Area Networks) WSNs. MMT integrated with new 6LoWPAN plugins over a real test-bed in analyzing real-world 6LoWPAN traffic are validated. The main contribution for the dissertation work was the development and support of MMT for the protocols 802.15.4 and the 6LoWPAN, with the latest being specifically the support of the dispatches related to fragmentation and routing.

- [***under review***] R. Fuentes-Samaniego, V. La, A. Cavalli, J. Nolazco-Flores, *Secure M2M Communication Monitoring in 6LoWPAN-based Wireless Sensor Networks*, In International Journal of Computers and Applications.

  In this paper, MMT for 6LoWPAN is used for the analysis of the traffic in the upper layer and to detect security threats over a real WSN as testbed with sensors using DTLS for protecting the communication. The contribution of this dissertation is the development of the security rules for monitoring the communication between sensors. Said rules are based on the mitigation identified by the ETSI.

1.3.1 Other publications

The following is a list of works published that are not directly related to WSN. They are listed in ascending order by date of publication.

- R. Fuentes-Samaniego, A. Cavalli, M. Wissam, J. Baliosian, *Monitoring-based Validation of Functional and Performance Aspects of a Greedy Ant Colony Optimization Protocol*, in

IEEE: Network Operations and Management Symposium (LANOMS), 2015 Latin American.

Delay Tolerant Networks (DTN) are well adapted for situations where the network nodes suffer from intermittent communications due to the high mobility of the nodes and the constantly changing environment. In this paper we firstly assess GrAnt performance in much more challenging conditions than those presented by its authors, and secondly, we present a generic methodology based on MMT that enables real-time analysis of network traffic, to correlate the performance of a DTN protocol (such as GrAnt) with the information stored in its messages in order to validate the reliability of the protocol.

- V. La, R. Fuentes-Samaniego, A. Cavalli, *Network monitoring using MMT: an application based on the user-agent field in HTTP headers*, AINA 2016: 30th International Conference on Advanced Information Networking and Applications, IEEE, 23-25 march 2016, CransMontana, Switzerland, 2016, pp. 147-154, ISBN 978-1-5090-1857-4.

  In this paper, we use MMT to deal with a practical case-study in which we analyze the usefulness of MMT for detecting abnormal activities over the HTTP User-Agent field. We compare MMT performance against other popular IDS such as Suricata and Snort. In the context of our research, MMT provides an automated detection of malicious traffic abusing vulnerable User-Agent field. Analyzing abnormal User-Agent strings is also useful to rapidly detect existing evil objects in the network (bots).

- J. Visca, R. Fuentes-Samaniego, A. Cavalli, J. Baliosian, *Opportunistic media sharing for mobile networks*, in NOMS 2016: IEEE/IFIP Network Operations and Management Symposium, IEEE Computer Society, 25-29 April 2016, Istanbul, Turkey, 2016, pp. 799-803, ISBN 978-1-5090-0223-8.

  In this work we propose a publish/subscribe, opportunistic network protocol aimed at offload infrastructure network nodes. It contributes to optimize the cellular network usage among mobile devices and to reduce the costs and energy consumption induced by the reproduction of media that are temporally popular.

- J. Visca, R. Fuentes-Samaniego, A. Cavalli, J. Baliosian, *Path sampling: a robust alternative to gossiping for opportunistic network routing*, in WIMOB 2016: 12th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, IEEE, 17-19 October 2016, New York, United States.

  In this second paper about opportunistic networks, we show that Gossiping-based solutions suffer from pathological behaviour in some simple network scenarios. Under certain conditions almost all the data transmitted by some nodes may get lost in the network, not reaching its destination. To address this problem we have proposed an algorithm that responds in a more robust manner while staying relatively simple. In this work, we show that our algorithm achieves delivery rates comparable to gossiping-based algorithms, while being more robust and providing better fairness.

1.4 Dissertation outline

The remainder of this dissertation has the following organization: Chapter 2 reviews the literature of related topics including monitoring tools for WSN, the different known platforms, and

the approaches and tools for intrusion detection. Chapter 3 presents our selected architecture for WSN, with the purpose of achieving the maximum standardization possible and to be as close as possible to the real Internet. Further discussed are all the components, terms and security requirements for the platform. Additionally, an analysis for identifying threats and risks for M2M communication inside the WSN is discussed. Chapter 4 introduces our selected monitoring tool, and our work for achieving a native monitoring over our WSN platform. Chapter 5 presents the testings with the tool in the WSN architecture and their results. Finally, in Chapter 6 the conclusions of this work are presented.

Chapter 2

Background and related work

Contents

2.1 Introduction

Chapter 1 introduced the problematic in the scenario of this dissertation. This chapter compares the solution proposed in this dissertation with related state-of-the art approaches. Because of the highly heterogeneous nature of WSN and the multiple challenges on those types of networks, said approaches leave many gaps. This also has a high impact on the current tools available that are able to offer a certain degree of monitoring over one or more types of WSN.

This chapter is divided into four main sections. As stated in the introduction, the first contribution of this dissertation is to provide a feedback on the different architectures that can be considered WSN. This is discussed in Section 2.2. Following, general treats for any type of WSN architectures are discussed in Section 2.3, which are used as a point of reference to compare the purposes and effectiveness of the IDS tools mentioned in Section 2.4. Finally, as our goal is to monitor a proper configuration to provide a secure IoT environment, works that try to protect the M2M communication or to identify one or more type of possible issues over its communication are discussed in Section 2.5. Many of those works are using well-known protocols such as DTLS, but at the same time are limiting themselves to execute a simple measurement of power performance.

2.2 Classification of WSN

The objective of this section is to analyze different architectures that fall in the classification of WSN. Figure 1.1 shows the four typical components of any WSN [23]: 1) A network constituted by the wireless sensors, 2) An internal mesh network as a typical form. 3) A central point for clustering the gathered data. The bridge node between the cluster network and the sensor network is usually referred as a sink point. 4) A set of computing resources in the cluster network for handling the collected data.

There are many vendors and platforms that handle the different market targets related to WSN, including its open source alternatives. This list is not exhaustive, but covers very popular platforms in the research, industry and home domains.

The content of this section was strongly utilized for the elaboration of the work presented at WWIC 2015 [24] as well as a foundation of the background section of the other articles.

2.2.1 Radio Frequency Identification

The RFID was one of the first technologies using the term of “Internet of things”. The RFID allows to easily build highly populated networks with a low or no-power consumption mode at a reasonable cost [25]. A RFID network is composed of RFID tags and RFID readers. The tags are able to provide a unique 96-bit code for tracking a single object in the network. In the standard configuration, this tag is used as an index for accessing an external database at the request of the reader; nowadays, there are certain tags able to use 256-bits for their ID code.

The tags can be passive and powerless with a price per reader lesser than a dollar. Yet, their reach is less than one meter. They also work at different frequency channels: 124 KHz, 135 KHZ, and the range of 860 - 960 MHz and the power is supplied by the TAG reader. The optimum ranges of passive RFID tags are less than 20 cm (424 kbps), however, with the lowest data rate available (106 kbps) they can reach up to one meter [26].

There are also semi-passive and active RFID tags which are supplied with a battery and are able to increase the reach of their signal, but those tags have a considerably higher price per unit,

and inclusively their RFID tags can become very expensive (close to thousand of dollars) [27]. The original RFID does not have any mechanism for collision avoidance, however the European implementation introduced the method "Listen-Before-Talk" to the standard [28].

Several approaches to implement RFID in WSN environments have been made [25, 29–31] yet, they rely on active tags and their benefits are less than their complexity and cost.

The work presented by Demirkol et al. [29] presents a study of the implementation of passive RFIDs as a wake-up radio through a simulation of sensors using passive TAG readers and a series of MULEs as TAGS readers. While on their experiment the power efficiency saved on a long term was validated, it was also recognized that it is a trade-off with the cost of the implementation as it is required to increase the number of the MULEs in order to make this approach economically advantageous. Other problems are that the sensors only measure the data when they are awakened by the mule and that they are only able to transmit a small chain of data at the encounter (12 bytes).

The work presented by Catarinucci et al. [25] is a multi-ID tag able to be attached to the sensors and to deliver measurement values in up to 96 bits by request by any standard RFID reader. This work is a proof-concept and does not give a good power performance, and it also points out that it is not practical for distances above of 3.5 meters between the tag and the reader.

The work presented by Pileggi et al. [30] proposes a multimode wireless sensor node (MM-WSNode), a hybrid between RFID tags and a sensor. Within RFID components, this sensor should be able to transmit any type of value as it has a Point-to-Point Protocol (PPP) as a base for communication. Additionally, the sensors are able to use the IEEE 802.15.4 standard to communicate the measured values to other stations. This work is mostly a definition of the architecture; however, there is not sufficient evidence of a real benefit on cost efficiency or performance.

The work presented by Zhang et al. [31] is an analysis and a comparison of three common integrations of RFID to a WSN. The integration with the best power performance has as a negative factor in terms of complexity while the ones with a lower complexity lack real-time performance. The contribution of this work is an alternative mix between performance and complexity, but they are basically RFID readers working as sensors with extra nodes working as the RFID tags.

Other important factors that the previously mentioned works are unable to answer are the security of the network and the transmission chart data of only the original 96 bits of the tag values. Therefore, it is of our opinion that RFID is not a suitable architecture for our environment.

2.2.2 IEEE 802.11 Low Energy WiFi

The IEEE 802.11x is under the control of the WiFi Alliance and most of the standards are not intended to work with constrained networks, due those networks were covered by the IEEE 802.15.4 standard. Yet, some companies have made some modifications to the hardware that is available for the 802.11 standard by adding a duty cycle. They assume that a WiFi node can achieve a very low power consumption with a native support for IPv4 and IPv6 while also providing protocols for security, such as the WPA2 [32]. Thanks to the AP, their configuration can be a star topology, avoiding relay agents.

The work of Tozlu et al. [32,33] is an evaluation of one modified hardware (G2M5477) for the low-power WiFi against more popular implementations of the 802.15.4/6LoWPAN for TinyOS

and Contiki O.S. The conclusions are two: I) the low-energy WiFi provides an improvement over the original standard on latency and energy consumption with limited impact on battery supplies due to the introduced duty cycle. II) Both standards have a similar performance when working with a small amount of data to transmit. However, due to the difference in the MTU of both, Low-power WiFi handles the transmission of bigger frames such as 1024 MB in a better way. A minor problem with this work is the wrongly estimated value of MTU for the 802.15.4 of 102 bytes instead of the 127 bytes; while most probably this discrepancy comes from the sum of the headers of the 802.15.4 and 6LoWPAN protocols, it is not specified. Also, the value of 8 bytes used in the tests is too small for any type of transmission: a test with a value of 45 bytes would have been better.

In the work of Hulea et al. [34] the RN-131C WiFi chip is presented. This work is a proof-concept and it's main contribution is the estimation of the power supply based on the duty-cycle, which is around 2 years of life in a best case scenario.

The main disadvantage of these works is the non-standardization of the current implementations. All the previous works have implemented their own product with a result unique to them. Another problem is that no one considered scenarios where the area to cover is superior to the default range of the WiFi products, and therefore the devices are forced to work in a mesh topology instead of a star topology. One last issue is the high price of the hardware compared to other options such as those for the IEEE 802.15.4.

2.2.2.1 IEEE 802.11ah standard

Another issue to consider is the official IEEE 802.11ah HaLoW, a standard defined by the WiFi Alliance, which is a redesign of the PHY and MAC layers aimed for sensor networks; it will use the channels under the Gigahertz frequency that are subject to the regulations of each country to ensure a widespread range with a minimum speed of 150 Kbps. Yet this standard is not expected to be certified until 2018 [35] and consequently there are no works to validate its benefit against other protocols such as 802.15.4.

2.2.3 Bluetooth Low Energy

The BLE is optimized for low energy devices: it uses a master-slave model which supports the star topology. The masters of each topology can communicate with other masters in a distance up to 46 hops. The BLE groups the first four layers of the OSI model into a layer called "controller layer" which has the following characteristics:

- BLE operates in the 2.4 GHz frequency with 40 channels of 2 MHz for each one (instead the 1 MHz size of the standard Bluetooth).
- BLE MTU consists of 23 bytes.
- BLE has three advertising channels for broadcasting functions.
- BLE makes use of Frequency Hopping Mechanism (FHP) with a relaxed timing request.

The BLE has an improved state machine compared to the standard Bluetooth. This also causes that a BLE device is unable to communicate directly to a standard Bluetooth device without the intervention of a device with both protocols.

BLE offers two places where security can be implemented: I) the link layer where it is possible to add an authentication using an AES 128-bits block chippering with an overhead of 4-bytes of the packet. II) with the use of AES for data-integrity on the GATT sub-layer (a sub-component of the equivalent presentation layer of the OSI model) which adds an overhead of 12 bytes. Although, some works suggest that both of them are mutually exclusive [36].

The work presented by Wang et al. [37] is an adaptation of 6LoWPAN for BLE, which is generated combining the open source stack of Bluez, Linux platform, with the IPv6 stack of Contiki. With this, it is possible to compress the IPv6 and UDP headers to 5 bytes, leaving 18 bytes for the data. It is important to remark the fact that this work is ad hoc for the Bluetooth standard.

Finally, BLE does not support IPv4 or IPv6, however Bluetooth SIG has stated that official support to IPv6 will be provided in the near future [38]. This incompatibility and the small amount of MTU for the data application and security makes the use of BLE not appropriate for WSN.

2.2.4 Other minor proprietary protocols

There are also wireless proprietary protocols for WSN, while some of them fall in the category of Wireless Home Automation (WHA) and are strongly similar to WSN; in the home automation market it is possible to find Z-wave [39], INSTEON [40], Wavenis [40] and ZigBee. With exception of ZigBee these protocols are hard to be used in an open-field environment as they have little resistance to noise or a low data rate. Another minor inconvenience is the lack of a general documentation.

Z-Wave communicates using a low-power wireless technology designed specifically for remote control applications [39]. The Z-Wave wireless protocol is optimized for reliable and low-latency communication of small data packets with data rates up to 100 kbps in the sub-gigahertz frequency channel (900 MHz). Currently, Z-Wave supports the AES with 128 bits-key and has a native support for IPv6. Its lower layers are based on the ITU-T G.9959. It is important to mention that Z-wave has been inconsistent regarding security as originally it was using Triple DES with 56 bits key of length, but this was removed from certain products "*as it was unnecessary and not useful for the clients*" until eventually the AES 128-bits was introduced [41].

The protocols ANT and ANT+ [42] have a considerable success in the industry. Developed by the Dynastream Innovations Company, they are focused on delivering a high interoperability among different devices and for different applications like environment, health, sport and audio. Both protocols operate on the 2.4 GHz channels and seem to have a certain type of security based on keys of 64 bits and its own mechanisms to ensure co-existence with other technologies. The ANT communication architecture is based on the OSI model, but uses only 8 bits for addressing each node in the network. The data rate is directly related to the size of the data being transmitted and the higher chip supports a 60 kpbs for a MTU 24. Something similar happens with encryption, which is only supported with certain chips.

Wavenis has been developed by Coronis System and it is under direct support of the Wavenis Open Standard Alliance [40]. This protocol focuses on the control and monitoring of applications and operates on the sub-Gigahertz channels (433, 868 and 915 MHz), and, with some products working in the 2.4 GHz, it can reach up to 100 kbps. However, a typical value for speed would be of 19.2 kbps, with the minimum being 4.8 kbps. For the noise control, Wavenis offers a mix of TDMA and CSMA but the documentation of this protocol is very limited and it is difficult to access.

INSTEON is another WHA developed by Smart Labs and it is promoted by the INSTEON alliance [40]. It has its own mechanism to provide a mesh topology and RF communication using the frequency shift keying (FSK) on the channel of 904 MHz achieving a data rate of 38.4 kbps. INSTEON makes use of a special time division hops mechanism with a certain tolerance to become jammed based on probability.

2.2.5 IEEE 802.15.4

The IEEE 802.15.4 standard for Wireless Personal Area Network (WPAN) [43] is ideal to convey information over relatively short distances, involving little or no infrastructure. This feature allows small, power-efficient, inexpensive solutions to be implemented for a wide range of devices with small battery consumption requirements and typically operating in the reach range of 10 meters.

This standard defines two layers called the physical layer (PHY) and the medium access control (MAC) layer. Both of them are closely equivalent to their counterparts in the OSI model layer. The following list are the general characteristics of both layers:

- The PHY layer offer 3 frequencies: 2.4 GHz with 16 channels (universal), 868 MHz for Europe and 915 MHz for America.
- The 2.4 GHz frequency can reach at a transmission rate of 250 kbps.
- On the sub-GHz frequencies, the transmission was originally in a range between 10 to 20 kbps but after the 2006 revision it was improved to 100 kbps.
- The PHY layer supports only frames of up to 127 bytes, including data from upper protocols.
- The MAC layer uses Carrier Sense Multiple Avoidance - Carrier Sense (CSMA-CA) or ALOHA for controlling the access to the RF.
- The MAC layer can provide an access control by means of AES 128-bits keys.
- The MAC layer defines 16 bits for identifying PAN nodes, plus 64 bits for other types of addresses.
- The MAC layer can support frame validation, also it guarantees time slots and handles node association.
- The MAC layer is able to operate on two topologies: Star topology and mesh topology.

The IEEE 802.15.4 incorporates features aimed to minimize the power consumption, in addition to handling the duty cycles for saving the battery life. This standard also can make a smart selection of channel frequency for scenarios where other services are using the 2.4 GHz channel [44, p. 295].

As the IEEE 802.15.4 is a very robust open standard, it has been adopted by different entities for their own WSN devices. Each one of them handles their own upper layers. The ZigBee Alliance is the most popular. Other entities like WirelessHART and ISA, have also adopted it for the industrial environment. It is also popular with different open source platforms such as TinyOS, Contiki and RIOT.

2.2.5.1 ZigBee

ZigBee is a specification for a suite of high level communication protocols used to create personal area networks built from small, low-power digital radios. ZigBee is based on the IEEE 802.15.4 standard and is under the control of the ZigBee Alliance [45].

The ZigBee Alliance has defined a number of important ZigBee variances or specifications: ZigBee (2004), ZigBee PRO (2007), ZigBee Green Power (2012) and ZigBee IP (2013). Each one has their own defined layers, having in common the use of the 802.15.4 standard. Only the ZigBee IP can support IPv6 networks by means of 6LoWPAN. The others require to make use of additional devices to have access to any IP network.

The typical nodes of the WSN are represented by the ZigBee IP host and the ZigBee IP router which are the sensor and sink nodes respectively. The ZigBee IP border router will translate packets from 6LoWPAN network to an IPv6 network and vice versa.

The equivalent to the session, presentation and application layers of the OSI model is the proprietary part of ZigBee IP. They are presented as a single layer under the name of the application profile layer and the application layer. The profiles are based on the ZigBee Smart Energy Version 2 (SEP2) with the purpose of reaching the maximum standardization possible between different chips manufacturers and ZigBee applications developed by the clients. The current specification of ZigBee IP is not compatible with nodes using the other ZigBee specifications, at least not without a special intermediary nodes.

2.2.5.2 WirelessHart

The HART Communication Foundation has adopted the IEEE 802.15.4 as the bases for the WirelessHART networks. The networks are composed of sensors, routers, hand-hold devices, Gateways and a network manager [46].

The PHY and MAC layers of the 802.15.4 standard have the following modifications:

- The CSMA-CA mechanism is substituted by the Time Division Multiple Access (TDMA) for achieving the Time Slotted Channel Hopping (TSCH) mechanism for continual transmission without collision.

- The implementation of super-frame: a collection of cells, which repeat at regular intervals to help to minimize the risk of collision and to solve the hidden node problem.

- The MAC layer is able to block some channels at the frequency of 2.4 GHz when it detects other wireless networks making use of them.

- The sleep state of the nodes is impacted, as WirelessHART will be giving feedback each 20 ms.

- It is still possible to use the AES 128-bits, however the overhead on the packet size will be at the limit of the transmission.

- Even if the 16 bits of the MAC layer are provided, in practice it is probable that it will not support more than 100 nodes at the same time, due to the TDMA limitation.

It is also important to note that WirelessHart assumes that the nodes will have a very large or even permanent power supply.

HART is proprietary of all the layers above the network layer. On the network layer, there are different routing mechanisms to achieve the communication from all the nodes. Yet, there is not native support for the TCP/IP protocol.

The upper layers of the OSI model are grouped together in the WirelessHART application layer, and this layer responds to HART's designs to guarantee some compatibility among their products. The communication between devices is based on a defined set of commands grouped on 4 types: universal, common practice, device families and device-specific commands.

The work of Alcaraz2010 et al. [47] analyses the security of WirelessHART focused on the confidentiality, integrity and availability, and concludes that it has a strong protection against attacks like Sybil, replication, flooding and isolation. As well it preserves data integrity with authentication using some automation mechanisms to renew the keys.

Because of the constant data transmission, the standard sleep state of the chips of ZigBee is not suitable for WirelessHART; this can be a restriction as the current chips supporting 802.15.4 are intended to be used with the ZigBee specifications [48].

2.2.5.3 ISA100.11a

The ISA100.11a is a standard for processes and systems control released by the International Society of Automation (ISA). The ISA100.11a provides a reliable and secure wireless operation for a variety of control processes as non-critical monitoring, alerting, supervisory control, open-loop control and closed-loop control applications [49].

The physical and data-link layers (OSI) are the adaptation of the PHY and MAC layer of the 802.15.4 standard with the following modifications to the MAC layer [50]:

- The TDMA mechanism replaces the CSMA-CA. Although it is still possible to use CSMA-CA if the network will coexist with other types of networks [50].
- ISA100.11a operates on the 2.4 GHz frequency with the 26 channels defined by the IEEE 802.15.4 standard.
- ISA100.11a has a backward support for other ISA100 devices.
- The sleep mode is also affected due to the constant transmission.

ISA100.11a has a very bad level of coexistence with other WLAN technologies that made use of the same frequency channel. ISA100.11a is composed by sensor nodes, routers, gateways, backbone routers (for subnets) and two special managers: a system manager and a security manager [47].

The equivalent of the network layer at the OSI model supports the mesh topology and has a certain degree of compatibility with 6LoWPAN, for instance it supports nodes with 16 or 128 bits of addresses, however it is not fully implemented. This is also the case for the transport layer, as it has some compatibility with UDP.

The application design of ISA is divided into two sub-layers. The first is dedicated to the control of the devices and the second to handling the hardware resources available to the node.

2.2.5.4 IPv6 packet delivery in Low Power Wireless Personal Area Networks

The IEEE 802.15.4 defines only the PHY and MAC layers of the TCP/IP protocol. Even if it was possible to implement directly the TCP/IP stack over this standard, this would not be feasible as IPv4 does not have enough space addresses for a lot of scenarios, and the use of IPv6 with TCP could generate headers of 100 bytes before any real data, forcing a fragmentation in almost all of the messages due to the restriction of the Maximum Transmission Unit (MTU).

This is the reason for the devolpment of 6LoWPAN, to allow that a native IPv6 network can run on the WSN and to cover the restrictions of memory and MTU. 6LoWPAN is the definition of a series of dispatches for compressing the IPv6 and UDP headers and also for handling the fragmentation at the Network layer. The development of 6LoWPAN by the IETF also involved a routing protocol for a mesh topology denominated IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL). The revision IEEE 2014.15.4-2006 included the support for 6LoWPAN.

All the dispatches used in 6LoWPAN add a minimum of 2 bytes of overhead to the packets, however with the best possible configuration an IPv6 packet with UDP as next header can reduce the header size from 48 bytes to only 3 bytes, including the 2 bytes of the dispatch.

When 6LoWPAN is used over IEEE 802.15.4, some specific configurations at the MAC layer are set: the long addresses mode of 64 bits is used instead of the 16 and the beacon messages are not used. While it is possible to change these configurations it is usually not required. The sink point is also now a bridge between the 6LoWPAN network and the standard IPv6 network denominated 6LoWPAN Border Router (6BR).

It is also important to note that the use of 6LoWPAN does not require the IEEE 802.15.4, however this is the initial configuration proposed by the IETF. It is expected that 6LoWPAN will be officially adapted to other architectures, such as the BLE, in the near future. For this reason 6LoWPAN should never be considered as a specific type of WSN, because one WSN could be configured with IEEE 802.15.4/6LoWPAN and another with BLE/6LoWPAN. These are totally different architectures even if both of them would support native IPv6.

The WirelessHART, the ISA100.11a and the ZigBee IP branch, have their own proprietary adaptations of 6LoWPAN. All the same, popular open source platforms for WSN under the IEEE 802.15.4 have created their own stacks for 6LowPAN, such as Contiki, TinyOS and RIOT.

2.2.6 Host Identify Protocol

There are several works focused on the Host Identity Protocol (HIP). This protocol is proposed and standardized by the IETF in the RFC 4423 and introduces a framework to dissociate the computer from the routing purposes and the data exchange [51]. This model is useful for handling mobility and multi-homing. Basically, HIP tries to separate the host identifier from the possible IP addresses that are used to reach it. To achieve this, the hosts are identified using a pair of both private and public keys. This association with the host is referred as the Host Identifier (HI). The HI uses PKC signatures for authentication, a Diffie-Hellam (DH) is involved in the process together with a protection mechanism against DoS attacks on this process and finally made use of IPSec. In total, previous to sending any real data, an overhead of four messages using DH is required.

The reason HIP has not been a topic of discussion for WSN in the past is because it is linked to the IP protocols and has a high demand of resources. The former is solved by the use of 6LoWPAN, but the later is still an issue. There have been efforts to make HIP more lightweight and those derive on HIP Diet Exchange (DEX). Yet HIP DEX cannot support the major features of 6LoWPAN and it requires to run on all the nodes involved in the communication [52].

The work of Khurri et al. [51] makes use of an Imote2 wireless sensor node platform (256 KB RAM and SRAM, 32 MB FLASH and 32 MB SDRAM) and of TinyOS. On this work an analysis of energy consumption using a different key algorithm for generating the HIs is performed concluding that while RSA and DH are some of the most friendly nodes, the consumption of resources is still high.

The work of Meca et al. [53] presents an integration of HIP for WSN with the use of Multimedia Internet Keying (MIKey) to "provide enhanced key management using polynomials, which allows to generate pairwise keys with any node based on its identity". This work adapted an 6LoWPAN/CoAP application from the popular Contiki OS using a Red Bee Econotag as prototype (32-bit CPU, 128 KB of ROM, 128 KB of RAM).

Another use of HIP DEX is the project Usable Trust in the Internet of Things (uTRUSTit) which has two objectives [54–57]: I) Create a guideline to identify, produce and manage trustiness on the IoT environments. II) Develop the trust feedback toolkit (TFT) which aims to be embedded in smartphones and IoT applications for providing privacy settings and feedback to the final user.

In addition to uTRUSTit, there is also the paradigm Social Internet of Things (SIoT). With SIoT, the "smart" device "evolves" to "social" device with the purpose of fostering resource visibility from the devices [58]. SIoT not only wants the devices to be able to communicate inside of a specific IoT environment, but also between multiple environments. Allowing the nodes to interact among themselves, as people do, to deliver data and to find paths [59]. Also, SIoT has developed a trustiness based on social interaction and P2P techniques [60].

The main limitations of SIoT are that all their works are theoretical and only based on simulations. They assume a mix of devices (RFID, mobile devices, sensors, etc.) where the nodes without enough resources can request said resources from other nodes. Consequently, all the simulations are made without considering the hardware limitations.

An adaptation to SIoT for adding Quality of Service (QoS) is proposed by Nasiraee et al. [61]. Similar to the original work on SIoT, the testing does not take into account the current hardware limitations. More research is needed to define a method to handle QoS on 6LoWPAN networks as key elements for this were removed.

The work presented by Machara et al. [62] is able to interconnect devices of different environments; however, it takes into consideration the privacy of data as a part of Quality of Context (QoC) for defining the trustiness of the nodes. Yet, again, this work is still based solely on simulations and does not consider the current hardware constraints.

For this dissertation we do not have an interest in the work with HIP as they cannot easily be deployed in a large scale. The devices used in the previous works [51,53] have more resources than those in the C2 category of the Table 1.1; this has an impact on the price and the life of the power supply. Works related to uTRUSTit and SIoT are based on the idea that they will become more feasible in the far future as the platforms begin to be more powerful for a lesser price.

2.2.7 Final observations about the available WSN architectures

The Table 2.1 reflects a comparative of the protocols discussed previously. The IPv6 field is there because it tries to reach the maximum standardization possible.

RFID is not suitable because of the high complexity required for working as a WSN. The Low Energy Wifi, although very tempting due to its high transmission rate is not standardized and the performance can vary highly from vendor to vendor. The 802.11ah should be considered

Technology	Battery time life	Range (meter)	Data rate (Kbps)	Frequency	Size addresses	MTU	Noise resistance	IPv6
RFID	Infinite[1]	<1	106	Varies	96, 128, 256		Low (Varies)	No
ZigBee	1-10 years	10-100	250	2.4 GHz	16 bits	127	CSMA-CA	Partial
WirelessHART	1-10 years	10-100	250	2.4 GHz	50 - 100 devices	127	TDMA	No
ISA100.11a	1-10 years	10-100	250	2.4 GHz	16 bits	127	CSMA-CA, TDMA	Partial
802.15.4 / 6LoWPAN	1-10 years	10-100	250	2.4 GHz	128 bits	127	CSMA-CA, TDMA	6LoWPAN
Low Energy Wifi	<1 year	10-100	>1000	2.4 GHz	128 bits	1280	CSMA-CD	Native
WiFi 802.11ah[2]	1-10 years	N/A[2]	>=150	Sub-Ghz[2]	128/16 bits	1280	CSMA-CD	Native
Bluetooth LE	1-10 years	10-100	10000	2.4 GHz	16 bits	23	TDMA	No
Z-wave	<5 years	30 / 100	40/250	Sub-Ghz / 2.4GHz	32 bits	64	none or /CSMA-CA	Yes
INSTEON	<1 year	45	38.4	904 MHz	24 bits	14	TDMA, simul-cast	No
Wavenis	1-10 years	200 / 1000	19.2	Sub-Ghz / 2.4GHz	48 bits	N/A	CSMA, TDMA	No
ANT/ANT+	N/A	<5	<= 60	N/A	8 bits	8/16, 24	TDMA	No

Table 2.1: A comparative of WSN architectures.

[1] Only when using passive RFID tags, otherwise it can vary.

[2] The 802.11ah is still under development and it will not be ready before 2018. It is announced that it will work on the 915 MHz, 868 MHz or 780 Mhz due to the laws restriction on each country.

once the specifications are released, however, this will not happen in the near future. Other proprietary protocols are simply not reliable enough nor have support for IPv6.

Finally, those using the IEEE 802.15.4 are viable candidates. ZigBee products have contributed to make this architecture very popular, yet it is not an open standard, and devices among different branches of ZigBee are not compatible with each other, at least not without the intervention of additional devices. ZigBee IP, WirelessHART and ISA100.11a are using IEEE 802.15.4/6LoWPAN with modifications. The last two made heavy modifications to the IEEE 802.15.4 as they assumed the sensors have a permanent power supply and because of the challenges related to a factory environment. ZigBee IP is considered a candidate, but its standardization in the upper layers is not very convincing.

For this reason, open source platforms with support for IEEE 802.15.4, 6LoWPAN and popular upper protocols such as HTTP and CoAP are excellent candidates. In Section 3.2 the previous protocols are discussed in detail, while in Section 3.5 a discussion about the open source platform is done.

2.3 Common threats for WSN

The use of IPv6 and well-known TCP and UDP with upper protocols in WSN combines threats unique to WSN and those related to TCP/IP. The objective of this section relies on identifying the types of attacks unique to WSN. The following is a list of threats [63]:

1. "Cloning of things": At some point, the sensor can be cloned or duplicated. Normally this can happen when the node is being manufactured.

2. Eavesdropping attack: This attack can happen anytime but it is more dangerous right after the sensor is added to the network, especially if operational keying materials, security parameters, or configuration settings are exchanged in clear.

3. Man-in-the-middle attack: This attack can also happen anytime. It can be potentially dangerous when other nodes are being added to the network as they can intercept the passing of operational keying materials, security parameters, or configuration settings.

4. Firmware Replacement attack: This is very similar to the Cloning. However, this can happen also during the normal operation if the network permits the load of a new firmware (remote updates of the sensors).

5. Tempering attacks (extraction of security parameters): The sensor can be compromised physically by one or more attackers and its content could be cloned, erased or modified.

6. Routing attack: Routing information can be spoofed, altered or replayed with the purpose of creating routing loops, to attract the network traffic (maybe for a main-in-the-middle attack) or to provoke the loss of the traffic.

7. Privacy threat: Depending on the type of data gathered, the tracking of the location of one or more sensors and usage may lead to a privacy risk to the users as the attacker could infer information based on them.

8. Denial-of-Service (DoS) attack: Because of the limitation of resources, the sensors are vulnerable to resource exhaustion attack. Attackers can continuously send requests for

any resource (CoAP Request message). The mesh topology is particularly vulnerable to this attack. This type of attack can also be provoked by jammering attempts, some of them unintentionally as a result of being crowded by other wireless networks using the same 2.4 GHz frequency.

The sensors deployed can have one or more of the following characteristics: I) untenanted by a long period of time, II) exposed to the physical environment (rain, heat, snow), III) the deployment can be random, and IV) new nodes can be added at any time. Because of this, several threats became more serious. The sensors are particularly susceptible to eavesdropping, tempering (including unintentional damage) and DoS based on jamming attacks.

As a result of the mesh topology and the sink point, one of the risks almost impossible to avoid is the "hot-spot" dilemma [64]. This means basically, listening to the medium to detect the nodes in the mesh topology that are closer to the sink point. It is not necessary to know the infrastructure and it is irrelevant if the communication is protected. The sensors closer to the sink are those that require to transmit more often as they are carrying the data of the most distant neighbors. This knowledge can facilitate DoS attacks based on hammering or inclusive tempering attacks, which at the same time can lead to a DoS attack due to the subtraction or damage of the key nodes in the mesh topology.

The work presented by Vlajic et al. [64] proposed mobile sink nodes for the reduction of the hot-spot risk, and it concluded that using a path-constrained mobile sink may increase the network lifetime. However, it is not always possible to create mobile sink points. And, if they are implemented, the velocity needs to be considered as Zen et al. [65] determined that the IEEE 802.15.4 standard is not able to maintain a node's connectivity for fast moving nodes. Another alternative is placing a higher capacity battery on the key nodes.

The passive and active protection can handle the risk associated to tampering attacks [66,67]. The passive mechanisms are those who do not need additional power and are composed by technologies that protect a circuit from being detected, such as protective coatings and tamper seals. Active defenses are related to special hardware circuits to prevent sensitive data from being exposed but those usually carries a higher cost, making them little suitable for sensors [67].

The sensors should be able to resist the physical environment to guarantee their correct performance. Simple practices such as placing them out of people reach at the open spaces or begin the communication when there is a smaller chance to jam the sources can increase the reliability with little extra costs. Also, the selection of the sensor hardware and its placement in the area defines the resistance against intentional tampering attacks.

It is possible to defend against jammering using various forms of spread-spectrum or frequency hopping communication. However, those mechanisms require more complex hardware and a permanent power supply so the low-cost and low-power sensors will be limited [67, 68]. Still, the same work concluded than an antenna polarization well designed could handle the jammering attacks properly.

The survey of Mpitziopoulos et al. [69] also suggests a series of measures for having a better control against jammering: I Detection techniques: deploying elements able to instantly detect a jamming attack. II) Proactive countermeasures: software measures, such as changing the MAC protocol for adding FHSS. Some of the techniques are compatible with the IEEE 802.15.4 standard. III) Reactive countermeasures: enable reactions only when a jamming attack is detected, many techniques are compatible with the IEEE 802.15.4 standard. IV) Mobile Agent-based solution: Where special mobile-agents are defined and used as autonomous programs with the ability to move from host to host, in this case for finding new paths free of jamming

attacks.

The tempering and jammering attacks are not in the scope of this dissertation. The routing attacks are located in the network layer. Typical attacks are Bogus routing information, Selective forwarding, Sinkhole attacks, Sybil attacks, Wormholes and HELLO flood attacks [70, 71]. 6LoWPAN handles those threats by means of RPL but this is also outside of the scope of this dissertation. However, because our real interest lies in the upper layers, DoS attacks focused on the resources of upper-layer applications and risk of threats to privacy are further discussed in Section 3.4.

The content used in this section was utilized for the elaboration of the work presented at WWIC 2015 [24].

2.4 Intrusion Detection Systems over WSN

An intrusion detection system (IDS) is defined as "a system that tries to detect and alert the occurrence of potential intrusions into a system or a network" [21]. The identification of the intrusion is based on the monitoring of nodes and/or the network. The systems that make implementations on nodes are referred as IDS agents [72]. IDS can be classified on intrusion detection or intrusion protection: the former only reports the intrusion meanwhile the latest also reacts to it.

An IDS is, conventionally, classified as Host IDS (HIDS) or Network IDS (NIDS) [21]. HIDS monitors specific activities by each host. The NIDS monitors the network's activity by capturing the packets (sniffing). An important drawback of the HIDS is the requirement of installing it in all the nodes of the network. The NIDS also require to have dedicated hosts, but in a smaller amount than HIDS.

IDS helps to identify the threats defined in Section 2.3. They are the first line of defense [71]. However, WSN are not an easy area to deploy an IDS. The use of a sink point (or bridge router) would give a central point for collecting data, and while this is true for all the traffic coming and leaving the WSN, all internal messages can be missed or manipulated. If a DoS is affecting a far portion of the WSN, in relation to the BR, this probably would pass inadvertent, except maybe for the sudden drop of the bandwidth use.

2.4.1 Impact of IDS over WSN

The study presented by Doddapaneni et al. [73] is focused on measuring the impact of IDS over WSN. Yet it has not specified the type of architectures tested as the simulation is executed over OMNET. The study remarks that the use of a promiscuous mode in the sensors (be listening to all the packets reaching them) could drain their power supply too fast. Also, one of its main conclusions is that a "novel intrusion detection system should not eavesdrop the network all the time nor should run a Byzantine agreement involving too many nodes".

A small size WSN can make use of IDS based on the Byzantine agreement, however as the number of sensors increases, the network lifetime could decrease dramatically. Other techniques must be careful not to force the sensors to be promiscuous nodes as they can drain very fast their battery.

2.4.2 PAD, a probabilistic diagnosis for WSN

The work presented by Liu et al. [74] is the proposal of an online diagnostic tool that passively monitors the network from the 6BR, trying to deduce root causes of any abnormal symptom. It deploys the monitoring over a sea environment. Although the types of sensor and platform are stated (TelosB, TinyOS), the exact architecture for the WSN is never stated. An additional overhead for the packets was added to have the possibility of tracing sources and destinations to be able to assemble the full topology. Consequently, the objective is to detect anomalies in the topology but not the cause of it

An important observation made by the authors is the fact that the WSN topology is highly dynamic and it is constantly modifying the routes, though this is directly linked to their type of deployment.

2.4.3 Foren6

The tool Foren6 [75] is designated for 80215.4/6LoWPAN. This tool also subtracts information from the original packets, without adding extra overhead in the original message. It is capable of reconstructing a visual and a textual representation of the WSN based on the analyzed packets.

Foren6 is available as an open source software for Linux and MAC OS. Foren6 comes with an integrated sniffer with native support for Contiki OS; it is able to process packets captured from other platform such as RIOT, but they need to be saved as pcap files.

Finally, Foren6 is focused over on-site diagnosis, detecting routing problems and debugging of the network. Therefore is not well suited for detecting attacks not related to routing.

2.4.4 LHSFW

The work presented by Sedjelmaci et al. [76] proposes a Lightweight Hybrid Security Framework in a cluster-based WSN (LHSFW); this is an active IDS that also works by forcing the sensors to cipher their communication. Their active component is based over the monitoring of anomaly traffic (specification-based model) or traffic that is not normal. The ciphering component is the passive protection against eavesdropping attacks. They separate the sensors in clusters to try to reduce the impact on the lifetime of the WSN. Once that a local agent detects an anomaly in its cluster, it will notify the root so that it tries to isolate the problem. This solution is only tested with the simulator TOSSIM.

There is not enough information about the type of architecture used in the tests. The use of an additional layer of protection provokes a negative impact on the complexity of the network and its standardization. In the case of 802.15.4/6LoWPAN, it is possible to protect the upper layers and the communication between the sensors, reaching the same objective as LHSFW without adding any extra protocols. However, the concept of a cluster separation and the detection method used on this work is in fact, quite interesting

2.4.5 LoMoM

The work presented by Bhuiyan et al. [77] proposes an approach to Local Monitoring and Maintenance for a WSN called LoMoM. In short terms, is a parallel 3G network over the WSN working as an IDS. All the nodes check their immediate neighbors to detect any anomaly. And, if it is the case, then trigger the alert using the 3G network. Their work is still proof of concept

and simulation, they estimate to be able to detect 97% of the attacks meanwhile preserving the sensor's battery.

However, an implementation like the suggested for LoMoM would increase significantly the cost of the sensors because they need to support two different types of networks.

2.4.6 NIDS over WSN

The work presented by Amaral et al. [21] is an IDS for WSN supporting the 6LoWPAN. TinyOS was used for developing their own IDS based on traffic signatures and abnormal behaviors. The contributions of this work were added also to the official repository of TinyOS. On this work, "watchdogs" nodes are generated in some of the sensors. The watchdogs are listening to the node communication in their wireless area. The NIDS compares it with its own set of rules and in case of matching an attack, it will notify a central entity located at the sink node.

The study also developed a system based on UDP commands for updating and populating the NIDS policies based on its research. One setback is the fact that in order to save resources on the watchdogs nodes, once processed the nodes never store the packages. Meaning that if an attack takes place in more than one single step, it will probably be lost to the NIDS. Even if the work presents screen-shots of the IDS being configured, it lacks an analysis of a power performance and well defined tested scenarios.

A minor point of discussion is, if the full deployment of this IDS and the WSN are chained to the use of TinyOS or if it can work as expected with different platforms for WSN.

2.4.7 SVELTE

SVELTE is an IDS for WSN using 802.15.4/6LoWPAN and presented by Raza et al. [78]. It was implemented and evaluated to detect attacks against the RPL routing protocol (spoofed, sinkhole and selective-forwarding). SVELTE is divided into three components: I) 6Maper, present in the sensors it sends protected data with information about the routing tables. An additional small overhead with ciphered data in relation to the WSN routing data is sent in the messages. II) The IDS in the bridge router, which collects and process the previous data to resemble the topology and searches for attacks. And III) a distributed "mini-firewall" in every sensor to try to prevent unwanted UDP traffic.

SVELTE was tested only in simulations using the Contiki platform and with its simulator tool, called Cooja. SVELTE active monitoring seems to provide a small overhead to the sensors and its rate of success against the routing attacks is acceptable. It is also possible to update and add extensions to SVELTE to add protection against new or different types of attacks.

SVELTE distributed mini-firewall can be discussed as a useful tactic. It is implemented under the assumption that the upper traffic will be protected with a DTLS session, hiding it from the external firewall. However, the firewall would be strongly constrained in resources as the sensors already are limited in resources.

Finally, even with the extensions, SVELTE is only able to detect Network layer issues as the IDS bases its choices over the 6Mapper module.

2.4.8 Adaptation of Suricata for WSN

The work presented by Kasinathan et al. [71] is the IDS integrated to the EU FP7 project ebbits (European project - enabling the business-based internet of things and services). This

is an architecture proposed for the WSN to guarantee a successful IDS implementation able to detect all types of attacks.

The IDS proposed here is Suricata, a popular multi-threading signature-based open-source IDS. This architecture proposes a second parallel wired network where each one of its nodes is monitoring the WSN channel. All traffic on the channel is captured and translated to standard IPv6 to be able to send it to Suricata. This is also its main weakness: This model of parallel wired network is only feasible for small areas due to the increment in the cost; also it is possible to miss specific attacks to the 6LoWPAN dispatches when "translating" the packets

Still, the choice for the parallel network is not arbitrary, with this approach it is possible to detect any successful DoS over any portion of the WSN in real-time which makes it easier to take proper countermeasures.

2.5 Works related to security for 802.15.4/6LoWPAN

The works discussed in this section are implementations made with the purpose of mitigating the threats discussed in Section 2.3. They are characterized by their focus on the power performance of the sensors. This is because their discussion is about if the sensors in the C1 and C2 categories are able or not to support the standard security implementations (TLS, DTLS or IPsec) with their limited resources. These implementations are also different from the discussed on the previous section due to the lack of a security evaluation.

For the works that make use of DTLS we have a special interest for cipher suites `TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8` and `TLS_PSK_WITH_AES_128_CCM_8` as they are specifically requested to be supported by the CoAP specifications.

Some of those works are making explicit use of the Trusted Platform Module (TPM) embedded chip. This chip can execute operations related to encryption helping the micro-controller to save resources [49]. This contributes reducing the time required for the DTLS handshake process.

2.5.1 IPSec for 6LoWPAN

The work presented by Granjal et al. [79] is an evaluation of power performance when security is implemented at the network layer and application layer. Their main contribution was the proposition, implementation and evaluation of a custom 6LoWPAN dispatch for integrating IPSec to 6LoWPAN [80], with support for the Authentication Header (AH) and Encapsulating Security Payload (ESP). Their testbed is composed of two TelosB sensors and a Linux host where one of the TelosB acts as a bridge router. TinyOS is used as a platform for both sensors; the TelosB sensors specifications are a 16-bit RISC MSP 430 micro-controller with 48 KB of ROM and 10 KB of RAM. This work evaluated eleven different modes of security implementations divided into 3 different protocols: 6LoWPAN with the dispatches for AH and ESP; and CoAP using three different cipher suites, including `TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8` and `TLS_PSK_WITH_AES_128_CCM_8`. It is important to emphasize that the tests with AES were executed with the TPM chip implemented in the sensors.

In this evaluation is not defined precisely which stacks were used for DTLS or CoAP. Apart from the implementations using 3DES, CoAP with DTLS has a higher impact on memory compared to the network layer. For power performance it was concluded that when using ECC, the demand of energy is too high and should be considered only for applications where the transmission rate is low. Another important observation on this work is the fact that even with

the use of TPM, the application's lifetime is impacted by the use of these implementations, yet this was expected. The final conclusion is that the use of IPsec or CoAP with DTLS are viable according to the "most appropriate security mode". Still, the approach with DTLS is better suited for scenarios where foreigner clients request information from the sensors.

Although these works are focused on validating the use of custom dispatches, the comparison with DTLS is a good reference. They compare against robust cipher suites which are not suitable for the sensors as well as with those better suited for them. Even in their study, regarding energy consumed and time requirement, the most friendly cipher suite with DTLS requires much less resources than the other alternatives. This work lacks detailed information about the size achieved using compression.

The work of Raza et al. [81] is another adaptation of IPsec for 6LoWPAN. It focuses over the NHC compression for adding support for EH, AH and ESP headers. Based on the open library of MIRACL, it only supports SHA1 and AES. Here the Contiki OS generates the test and the sensors used are Tmote Sky (MSP430 microcontroller with 10k RAM and 48k Flash). The testbed involved a Linux host with WSN using 1 to 5 sensors, with IP datagrams ranging from 16 to 512 bytes which guarantees enough capcity to test 6LoWPAN with fragmented messages. They also used TPM to make the comparison. This work only tests their own implementation.

Their results have similarities with those of Granjal et al. [79] as both of them evaluated the configuration with AES-CBC+AES-XCBC-MAC-96 for AH and ESP. However, the results and observations between both works differ in some aspects: the ROM space for the stack for those MACs differs by 60% or more, the reported RAM usage differs greatly (e.g. ESP is given an approximately 40% for Granja et al. and 89% for Raza et al.). This probably is a direct consequence from the way TinyOS and Contiki OS report the resource consumption. When comparing the energy consumption, ESP reports a similar behavior but it is not the case for AH, where Granjal et al. report a significantly lesser amount of required energy as their ESP counterpart. On the contrary, Rosa et al. report that ESP and AH are closer in consumption.

The inconsistency found on both works on power performance probably is because of the tools available for TinyOS and Contiki OS for dissecting the components plus the approach for measuring power consumption. Nonetheless, both works offer a non-standardized implementation of IPsec over 6LoWPAN.

2.5.2 Lithe, a DTLS implementation for 6LoWPAN

The previous work of Raza et al. [81] about IPSec as a dispatch for 6LowPAN, was followed by the adaptation of DTLS as another type of dispatch for 6LoWPAN called Lithe [82]. Here, Contiki OS 2.7 was used together with the stack TinyDTLS 0.3.2 in beta state. This was not as easy to adjust as IPsec mostly because DTLS runs over the application layer instead of the network layer of the TCP/IP model. For the testbed Wismote sensors were used as a hardware platform (16 KB RAM and 128 KB ROM). Once the DTLS was integrated in the 6LoWPAN dispatch, it was tested with the Contiki erbium CoAP library.

The main advantage here is the reduction of the overhead on the DTLS packets and consequently in the energy consumption of the nodes. There are at least 12 bytes that can be omitted by integrating DTLS fragmentation on the network layer; this is greatly appreciated by the MTU of 127 as well as by the reduction of the power consumption required for the transmission. It is not possible to compress all the traffic generated by DTLS before sending real data, however at least 40% of the bytes are recovered for each type of DTLS record.

There is a difference in the Lithe tests when compared to the previous works with IPsec:

The TPM chip is not used, probably because they made use of the Cooja simulator, which also includes a power consumption tool called MSPSim. The measurement of power performance only relates to the comparison of standard DTLS, supporting only the cipher suite `TLS_PSK_WITH_AES_128_CCM_8`.

Therefore, Lithe is a very interesting work for us as it offers an alternative to the use of DTLS and CoAP, saving more resources. However, even if it offers a dissection of the memory used for the adaptation, it was not possible to retrieve the same values when the source code was tested[1]. Still, this could be because of modifications to the code after the release of the publication.

There is another issue with this approach of adapting DTLS as a dispatch: it is highly complex to test with new releases of TinyDTLS or Contiki. All our efforts to update the tools for supporting new versions of each one were futile. The difference between TinyDTLS 0.3.2 and TinyDTLS 0.8.2 is substantial and it is not backward compatible. There are also benefits from Contiki 2.7 to Contiki 3.0, but the heavy modifications to the erbium library and the 6LoWPAN stack make them not compatible. There is also a secondary issue of a small Ethernet tunnel in the traffic generated in Cooja on Contiki 2.7 that is not present anymore in Contiki 3.0.

2.5.3 CoAPS: CoAP and DTLS implementations

The works to be discussed in this section have implemented DTLS over sensors, including some that have been combined with CoAP for reaching CoAPS. Lithe was discussed in Section 2.5.2 because of the high number of modifications made to DTLS in order to adapt it to the 6LoWPAN dispatches.

The negotiation of a DTLS session is called a DTLS handshake process. The DTLS session is a secure channel between the nodes (client and server) involved. The use of different cipher suites has an impact on the type of keys, the required mechanism for the process, and on the size of each message (called DTLS records) required for the handshake process.

The work of Kothmayr et al. [83] is another comparison of DTLS for WSN. This work was one of the first adapting DTLS to WSN, using public cryptography keys such as RSA. As a testbed is used TinyOS 2.x, with an integrated TPM chip and the OpenSSL 1.0.0d for the DTLS stack. The hardware used for the sensor is an Opal Sensor (48 KB RAM and 256 KB Flash) which falls in the C2 category. The power was measured with an oscilloscope using a 10 Ohm resistor. "This yielded a value for the electric potential which can be converted into a value for the current draw by dividing it through the value of the resistance".

As one of the conclusions, a required energy of 485.2 mJ was estimated for completing a fully DTLS session. This number seems to be in accordance with the work of Granjal et al. [79] where the cipher suite `TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8` required 10.89 mJ. As the differences between cipher suites are huge, the CoAP specifications request to avoid the use of RSA with the sensors. In general, this work validates the use of only the cipher suites `TLS_PSK_WITH_AES_128_CCM_8` and `TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8` for WSN.

The work of Vucinic et al. [84] is a performance test of TinyDTLS 0.8.2 with Contiki 3.X. Its main contribution is the evaluation of the duration of the DTLS handshake with three duty cycles models: preamble sampling, IEEE 802.15.4 beacon-enabled and IEEE 802.15.4e time slotted channel hopping. The measurement is done by means of the MSPSim tool and the Cooja simulator. Around 100 DTLS negotiations with the cipher suite `TLS_PSK_WITH_AES_128_CCM_8`

[1] Lithe source code is available at http://www.shahidraza.info/resources/CoAP-DTLS.zip

were executed. The hardware tested were the Wismote and ST GreenNet platforms (32 KB of RAM and 256 KB of FLASH). The ST GreenNet platform also made use of TPM.

This work identifies that each DTLS session handled by TinyDTLS requires 400 bytes or more. The session includes enough data for identifying its states which are the IPv6 and port addresses of the nodes, their role, the master secret for the keys, and other variables depending on the cipher suite. According to their results, DTLS has a poor performance of in radio duty-cycle networks. For the preamble sampling protocol the DTLS handshake process can be larger than 50 seconds. In IEEE 802.15.4 beacon-enable mode, the duration was of up to 35 seconds. Also, they concluded that DTLS is acceptable for applications where the DTLS handshake occurrences are limited during the sensor lifetime and that applications that are expecting a high number of DTLS client for the DTLS server should be considered carefully.

There seems to be a small mistake in their calculation as this work assumes an overhead of 29 bytes for each datagram sent after a DTLS session is established. At least for the DTLS records where the data of the application is sent, the overhead is just of 13 bytes. This overhead is discussed in Section 3.2.4.1. Additionally, the number of DTLS sessions handled by the sensors is indeed restricted, therefore, it is recommended to limit the number of devices that are able to request data (clients) to the sensor (server). The time required for reaching a DTLS session can be considered as debatable. As it can be configured once the first message of the DTLS handshake process is received, the server does not go to the sleep state until the session is completed. Yet, it is true that the number of DTLS handshake processes should be limited to save resources involved in calculating new keys.

The work presented by Van den Abeele et al. [85] tries to resolve the issue marked by the previous work by creating a DTLS proxy between external clients and the sensors inside the WSN. This DTLS proxy is a trusted Gateway able to mitigate the high cost of the DTLS handshake process. In summary, the clients begin the process using the most powerful and secure cipher suites for the communication with the sensors, but in reality they are communicating directly with the DTLS proxy which handles their DTLS session and at the same time they establish another DTLS session with the sensors with friendlier cipher suites. Their testbed consisted in the use of Contiki with Cooja and the RM090 as the hardware for the sensors (16 KB RAM and 256 KB ROM). In the case of the DTLS stack, TinyDTLS with `TLS_PSK_WITH_AES_128_CCM_8` inside the WSN was used. It is not specified which versions of Contiki and TinyDTLS were implemented.

Their evaluation shows that up to 60% of the energy required for the DTLS handshake process can be recovered with this trusty Gateway. This is in concordance with the previous works. The time for completing the process is also reduced by more than half. The most important contribution of this work is the possibility to use the X.509 certificates for the external clients. In general, this work can be seen as complementary to our work as it uses standard protocols defined for WSN, and the DTLS proxy will be transparent for the sensors.

2.6 Conclusion

Although the concept of a secure M2M communication is not new, all the challenges that WSN faces makes it hard to achieve. Even harder is to develop a security tool able to monitor such a communication over WSN. This is reflected throughout this chapter.

In Section 2.2 some of the most popular architectures available for WSN were discussed, where some of them are ad hoc adaptations of well-known protocols. All this diversity and the fact that the sensors have very constrained resources, provoked that many manufacturers and

researchers ignored the security factor in the development of their architectures. Those that considered the security factors were more interested in the DoS and routing attacks over WSN, as discussed in Section 2.3. Those threats are not to be underrate, and the tools for detecting them have a mixed rate of success, as discussed in Section 2.4. Yet, the protection of the data transmitted was left as something desired to do, but uninteresting to the researchers. One of the reasons for this, is the fact that the techniques to guarantee the CIA and the non-repudiation are well-known and therefore of "little research value".

The researches that started to consider the use of protocols for data protection and association of the sensors were more inclined to validate the feasibility of such protocols in the constrained sensors. The works discussed in Section 2.5 consider only standardized and well-known protocols for this. Other works were left out as they have a high risk of being insecure, even if their resources are friendlier than DTLS.

The work of the researchers is also affected by the tools available to them: TinyDTLS, TinyOS and Contiki were the most common tools used, in part because they are open-sourced. Yet, in the last two years, many of these open source tools have become more efficient. This causes that the conclusion of the works focused on power performance falls back quickly, and where once a C1 node was unable to compile TinyDTLS over Contiki, it is now able to do it, yet the energy required for transmission probably will remain the same. This is also true for this dissertation in some points: TinyDTLs is expected to upgrade the libraries related to the ECC cryptography before the end of 2017, once the version 0.9.2 is released. *Therefore, the analysis of power performance should be seen just as a simple reference as even now, it has already proved that it is valid to use the well-known standard DTLS to protect the communication of the sensors.*

The monitoring tools available for WSN are also affected in a similar way than the protection of the data. Again, this is because the challenge of implementing and even guaranteeing a reliable monitoring over WSN is very different than in any traditional networks. But also, because each WSN has its own unique standard. Even popular and well-known IDS such as Snort, Suricata and BRO are still not able to process traffic coming from 802.15.4/6LoWPAN WSN and those who work over such WSN are mostly focused on the implementation and routing attacks. Therefore, *there is a lack of a native monitoring tool able to watch the data transmitted by the sensors.*

Chapter 3

The proposed architecture

Contents

3.1 Introduction

The purpose of this section is to identify and discuss all the components involved in the design of the proposed infrastructure for a secure IoT environment, as it was discussed in Section 1.1. Said infrastructure is composed of an IPv6 network and the WSN. External clients are able to inquire the data collected by the sensors by means of an intermediary proxy as showed in Figure 3.1. Based on the information collected through Chapter 2, we propose the use of the IEEE 802.15.4 standard and 6LoWPAN to conform our WSN architecture, together with the

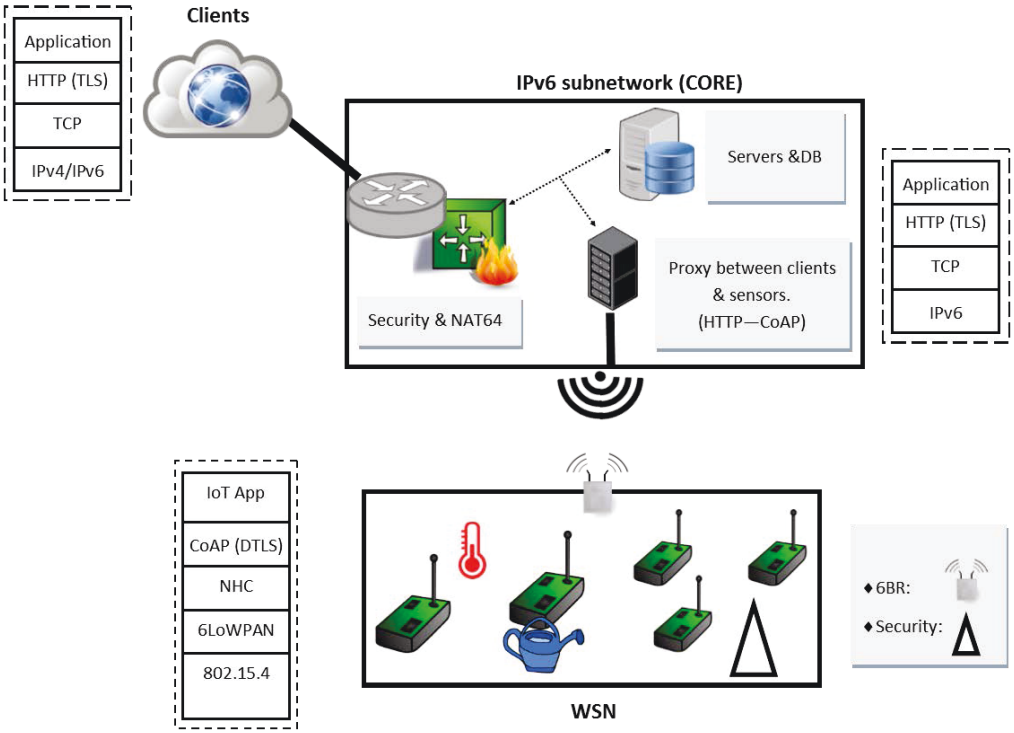


Figure 3.1: Model proposed for supporting a WSN architecture.

use of the upper protocols of DTLS and CoAP. Although even if briefly, those protocols were already discussed, their particularities require them to be further explored with the purpose of identifying the requirements for our monitoring tool. Also, the open source tools used for generating the testbed are as well introduced in this section, with a brief discussion of the work developed for each one of them.

This chapter is divided into five sections, in addition to this introduction. Section 3.2 is focused in the discussion of each protocol involved in the architecture of the WSN (IEEE 802.15.4/6LoWPAN) in terms of the TCP/IP model layer. Once the protocols are discussed, the different types of services, devices and communication between both networks are identified and classified in Section 3.3. At this point, enough information will be available for an analysis of risk for the M2M communication between the nodes (sensors, border routers, proxies, firewalls, etc.). This analysis will be further discussed in Section 3.4. Section 3.5 involves the discussion of the open source tools used for generating our proposed WSN. Finally, in Section 3.6 final remarks of this adaptation are discussed.

The content of this chapter was strongly used for the elaboration of the work presented [***under review***] at the International Journal of Computers and Applications and STAM 2016 [86].

3.2 The model layer

WSN has become a complex scenario, where more than one protocol is used to guarantee the interconnection of the "objects uniquely addressable" [15]. However, the standard Internet uses the TCP/IP model due to a similar complexity. An analysis of the requirements of WSN

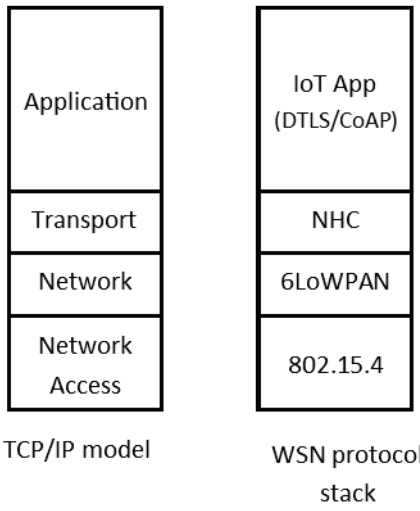


Figure 3.2: Comparison between the TCP/IP model and the stack proposed for WSN.

according to this model can thus be useful to disclose the different key components of the network, as shown in Figure 3.2.

As previously discussed in Section 2.2, many platforms or architectures fall under the WSN category. The platform selected for our WSN is the IEEE 802.15.4, due to its 2006 revision with native support for 6LoWPAN. For the Network layer, our objective is to have native access to the real Internet, consisting of IPv6 networks, being expected that IPv4 will fade in the upcoming years [87].

Because of the requirements of sensors on the WSN's transport-layer level, UDP was prefered to TCP. It is nevertheless required to make use of compression in the last two layers in order to save bandwidth and resources of the sensors. These compressed versions of IPv6 and UDP appear in this work as 6LoWPAN and NHC respectively. In a similar approach, protocols over application layer, optimal for constrained devices, are also being discussed.

The content of this section was used for the elaboration of the works published in WWIC2015 [24] and [***under review***] in the International Journal of Computers and Applications.

The following section will discuss all protocols involved in the stack used for the WSN. The Section 3.2.1 will discuss the protocol 802.15.4. Section 3.2.2 will discuss the techniques involved in the compression of IPv6 over 6LoWPAN. In a similar approach, Section 3.2.3 is related to the transport layer. The discussion in Section 3.2.4 will be divided into three steps: The DTLS protocol, the CoAP protocol and the presentation of the data.

3.2.1 Network Access layer

The protocol IEEE 802.15.4 consists of two sub-layers: the physical (PHY) sub-layer and the Medium Access Control (MAC) sub-layer. The former is related to all the physical components, such as the transmission, frequency, the method used for transmission and the bandwidth. Meanwhile, the MAC sub-layer contains the type of messages, beacon management, channel access, frame validation, acknowledged frame delivery, and association.

3.2.1.1 The physical (PHY) sub-layer

The 802.15.4 specification can operate with two types of topologies: star and peer-to-peer, as shown in Figure 3.3. A third type of topology, cluster-tree, can also be considered, but is rather a specific case of a peer-to-peer topology [88].

In both cases the nodes, or devices, can be assigned the roles of *Full-Function device* (FFD) or of *Reduce-Function device* (RFD) [1]. The RFD can be operated in three different modes: *PAN coordinator*, *coordinator* or just *device*. Only one *PAN coordinator* is possible per network.

In the star topology, the communication is established solely between each *device* and the *PAN coordinator*. It is usually considered for very simple scenarios with a main device having high availability of resources (peripheral devices communicating to a PC). In the peer-to-peer topology messages are transmitted directly between all devices by means of a multi-hop communication.6LoWPAN is greatly benefited from the Peer-to-peer topology, because of the easiness for making extensive use of multicast messages. Therefore, for this dissertation, only the peer-to-peer topology was utilized.

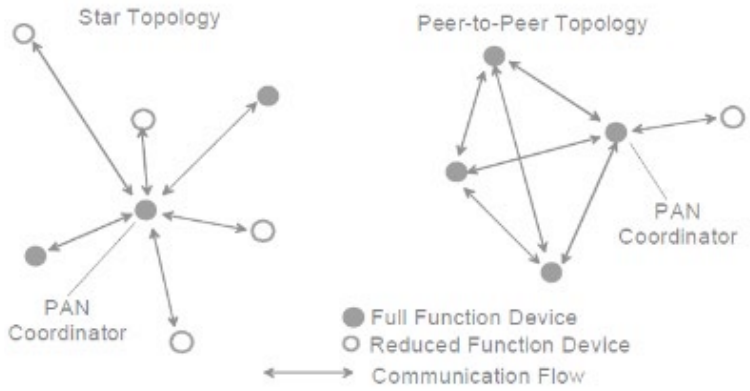


Figure 3.3: Star and peer-to-peer topology examples [1].

The use of the *direct sequence spread spectrum* (DSSS) modulation allows to guarantee robustness against interference from other sources [89]. Additionally, the PHY sub-layer provides three operational frequencies:

- 2.4 GHz world wide ISM band (16 channels) with a bandwidth of 250 kb/s.
- 868 MHz band for Europe (1 channel) with a bandwidth of 20 kb/s or up to 100 kb/s if the DSSSS O-QPSK modulation is used.
- 915 MHz band for America (10 channels) with a bandwidth of 40 kb/s or up to 250 kb/s if the DSSSS O-QPSK modulation is used.

3.2.1.2 The medium access control (MAC) sub-layer

The MAC sub-layer provides two services: the MAC data service and the MAC management service. The former makes the transmission and reception of MAC protocol frames across the PHY sub-layer possible.

Carrier sense multiple access/collision (CSMA/CA) was selected as the channel access method, despite the fact that it does not make use of the *Request to Send/Clear to send* (RT-

S/CTS) frames, contrary to other standards [1] such as IEEE 802.11. The use of slotted or unslotted CSMA/CA defines how the devices will try to access the channel to transmit frames.

In *beacon-mode*, a superframe is available, with which the nodes access the channel using slotted CSMA/CA, and the frame is started with a beacon frame [90]. In *non-beacon mode*, if the channel is idle after waiting a random time, the nodes are free to transmit data frames by means of unslotted CSMA/CA. Because the use of the Peer-to-Peer topology makes mandatory the use of the *non-beacon* mode, this dissertation only focus over said mode.

The MAC sub-layer handles four different types of frames, each of them with a maximum size of 127 bytes. These frame-types are [2]:

- Beacon frame, used by a *coordinator* to transmit beacons.
- Data frame, used for all transfers of data.
- Acknowledgment frame, used to confirm a successful frame reception.
- MAC command frame, used to handle all MAC peers entities control transfers.

A data or MAC command frame can request the use of an acknowledgment frame as validation for the reception of the frame. If such frame is not received after a given time, the sender assumes the packet was not delivered and it is re-transmitted. In case such a frame is not requested, the sender assumes the packets are delivered correctly. The beacon frame is used only in *beacon-mode* and is therefore out of scope in the dissertation.

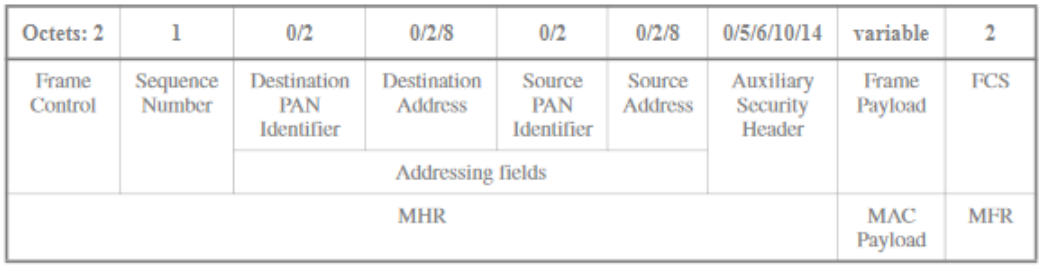


Figure 3.4: General MAC frame format [2].

Figure 3.4 shows the general format of a MAC frame. It is dynamic in size due to the possibility to compress or even omit the destination and/or source fields, as well to use a certain degree of security. Following is the description of each component:

1. `Frame Control` - All the control flags are handled in this two-bytes field.
2. `Sequence Number` - It is the ID of the frame's sequence identifier.
3. `Destination PAN Identifier` - This field is present only if the `Destination Address` is in the frame. The size is about 16 bits.
4. `Destination Address` - The size of this field is defined in the Frame Control. Possible values are: zero (suppressed), 16 or 64 bits.
5. `Source PAN Identifier` - The size of this field is defined in the Frame Control. Possible values are: zero (suppressed) or 16 bits.

6. `Source Address` - The size of this field is defined in the Frame Control. Possible values are: zero (suppressed), 16 or 64 bits.
7. `Auxiliary Security Header` - The size of this field is defined in the Frame Control. Possible values are: zero if the payload is in plain text. Otherwise the size can be about 30, 48, 50 or 112 bits.
8. `Frame Payload` - In case of the data frame, this is the payload of the upper layers. It is dynamic in size.
9. `FCS` - This is the check-sum of the full frame. Fixed at two bytes (16-bit IT U-T CRC).

The first seven fields must be in this precise order and are part of the MAC header, denominated MHR in the Figure 3.4. The FCS field is considered part of the header as well, and it will always be at the end of the frame.

The `Sequence Number` field helps to identify which frame an acknowledgment frame is being sent to. In case of a data or MAC command frame requesting an acknowledgment of delivery, the sequence number will be present in the acknowledgment frame.

The `Destination/Source PAN identifiers` are used to identify the *PAN coordinator* in the topology; when `source` and `destination` addresses are present, only the `destination PAN Identifier` is necessary.

The `Auxiliary Security Header` provides enough information to handle the security processing at this layer, which basically means ciphering the payload with AES 128-bits-long symmetric keys. This topic will be left out of our scope.

The source and destination address fields contain the MAC addresses of the source and destination in case they are present in the frame. The destination address field can in particular contain the broadcast short address (0xFFFF). The full size of the MAC addresses is about 64 bits. For some specific configuration in the upper layers, these can be the only fields available to identify the source and destination device in the WSN, see Section 3.2.2 for more information.

Bits: 0–2	3	4	5	6	7–9	10–11	12–13	14–15
Frame Type	Security Enabled	Frame Pending	AR	PAN ID Compression	Reserved	Dest. Addressing Mode	Frame Version	Source Addressing Mode

Figure 3.5: Format of the `Frame Control` field [2].

The bits used to handle the eight flags in the `Frame Control` are shown in Figure 3.5. Here shortly described:

- `Frame type` - 3-bits flag to identify the type of the MAC frame.
- `Security enabled` - 1-bit flag to identify whether the frame is protected or not.
- `Frame pending` - 1-bit flag to indicate if more frames are pending from the sending device to the destination device.
- `Acknowledgment Request` (AR) - 1-bit flag to request an Acknowledgment frame from the destination.

- `PAN ID Compression` - 1-bit flag to handle the presence of the Destination/source PAN Id fields.
- *Reserved* - Two bits reserved for future use, to be set to zero.
- `Destination Addressing Mode` - 2-bits flag to identify the presence and size of the destination PAN and the destination address field.
- `Frame Version` - 2-bits flag to indicate the version of the Field.
- `Source Addressing Mode` - 2-bits flag to identify the presence and size of the source PAN and the destination address field.

In the case of Destination/Source addressing mode, the possible three values are: I) fields suppressed; II) length of address fields set to 16-bits (short address); III) length of address fields set to 64-bits (extended address).

The purpose of the `Frame version` flag is to identify which revision of the IEEE 802.15.4 is handled by the nodes. Currently, the values of zero and one are employed for revisions 2003 and 2006 respectively. Other values are, for now, reserved.

The Acknowledgment and Data frames will contain all the MAC fields, although the former will suppress everything related to the source and destination, and cannot have any type of payload. The Beacon frame is used only in *beacon-mode* and it is beyond the scope of this dissertation. In a similar way, the use of 6LoWPAN does not require to send MAC frames, which also are beyond of our scope.

For the purpose of the MTU, a data frame will include at most 25 bytes for the header, leaving 102 bytes for the payload of the upper layers.

3.2.2 Network layer

The IPv6 protocol is the successor of the IPv4, and has for main intention to replace it in the Internet architecture in the upcoming years, by means of a gradual migration. It was released in 1998 and has been introduced to the market slowly since then. In the context of this dissertation, the most relevant differences between IPv6 and IPv4 are the following:

- The address space available evolves from 32 bits (approximately 4.3 billion addresses), to 128 bits (approximately $3.4 \text{x} 10^{38}$ addresses).
- A new, simpler header was created for IPv6, as shown in Figure 3.6.
- ICMP for IPv6 (ICMPv6) includes the sub-protocol Neighbor Discovery Protocol (NDP) to handle the configuration of the nodes, replacing the previous ARP protocol.
- A single node can have multiple IPv6 addresses simultaneously, with different scopes.
- The NDP makes it possible for the nodes to auto-configure their own IPv6 addresses by means of Stateless Auto-configuration (SLAAC).

IPv6 networks operate with the same rules as IPv4 networks, when referring to routing and configuration. Some changes can nevertheless be noticed in the address' syntax and the way the nodes are able to discover others nodes. In the syntax of IPv4, the addresses are

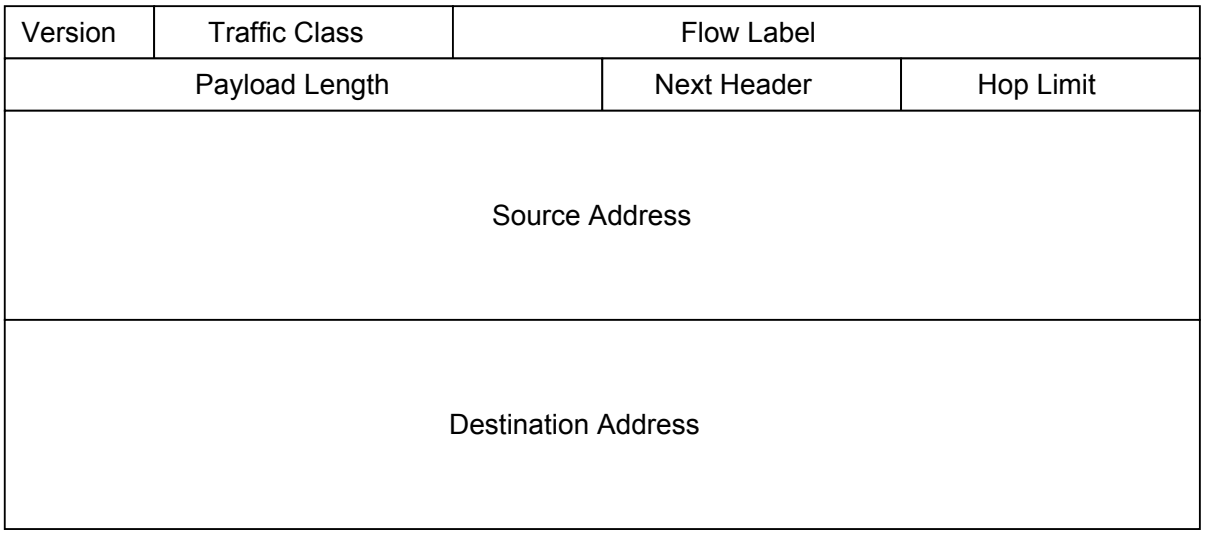


Figure 3.6: The IPv6 header [3].

divided in four segments called octets, of 8 bits each, expressed in decimal values. IPv6 dedicates eight segments of 16 bits each, without a special name, and all expressed in hexadecimal values. An IPv6 address can nevertheless be shortened following certain rules, for example FE80:0000:0000:0000:0000:0000:0000:0001 can be shortened to FE80::1. IPv6 also includes a prefix, sometimes known as "subnet mask" on IPv4, which has the same purpose as in IPv4.

A very important change of philosophy on IPv6 networks is the fact that a single node can have multiple IPv6 addresses at the same time [87], each from different sub-networks or scopes. This dissertation will only focus on the mandatory link-local and global scope. The link-local scope makes it possible for the nodes inside the network to identify each other without the intervention of a third entity. The operation is made easier by the fact that all nodes start with the prefix FE80::/64; It is not possible to route this scope to the outside. In contrast, the global scope has public addresses than can be routed to the outside, and requires the use of a router device, the prefix used for this scope typically is the 2000::/3.

In IPv6 all nodes can configure their IPv6 addresses by means of the stateless or stateful mode, regardless of the scope. The stateless mode makes use of SLAAC, which allows for an auto-configuration of an IPv6 address with the prefix "64", based on the MAC address of the Network Access protocol. With SLAAC, no extra service such as DHCP servers is further required for the nodes. In the stateful mode, it is possible to give other prefixes than "64", but additional services, such as DHCPv6 stateful, are required [87].

The increased address-space is the main reason why IPv6 can be used for IoT. The IPv6 protocol was however designed for an environment where the majority of the devices have a high amount of resources and, most importantly, enough power to access constantly the transmission-mean. WSN is however quite the opposite. And the IPv6 header alone requiring 40 bytes, makes it all the more complicated. Considering that the 802.15.4 data frame requires up to 25 bytes of the MTU's 127, the space for the IPv6 payload will be at most 63 bytes, which must be shared between the transport layer and the application layer.

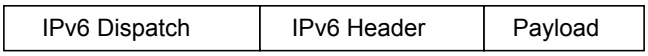


Figure 3.7: Generic 6LoWPAN Dispatch structure.

Pattern (bits)	Header Type	Description
00 XXXXXX	NALP - Not a LoWPAN frame	The next header is not 6LoWPAN or IPv6.
01 000001	IPv6 - Uncompressed IPv6 Addresses	The next header is an uncompressed IPv6 header.
01 010000	LOWPAN_BC0 - LOWPAN BC0 broadcast	Used for mesh broadcast and multicast support.
01 000010	LOWPAN_HC1 - Compression for IPv6	Deprecated compression of the IPv6 header.
01 1X XXXX	LOWPAN_IPHC - Compression for IPv6	Compression of the IPv6 header.
10 XXXXXX	MESH - Mesh header	Used for route messages from more than one hop.
11 000XXX	FRAG1 - Fragmentation Header (first)	Used for fragmentation of messages.
11 100XXX	FRAGN - Fragmentation Header (subsequent)	Used for fragmentation of messages.

Table 3.1: List of dispatches headers available for 6LoWPAN [10, p. 8].

The best way to achieve IPv6 communication over constrained networks such as WSN has been an important topic investigation over the last past years, ultimately leading to the development of 6LoWPAN. In its most basic concept, 6LoWPAN aims at compressing the IPv6 headers from their original 40 bytes to a significantly lesser dynamic value. In some cases, the header is compressed to lengths of three or six bytes.

To achieve the compression, 6LoWPAN makes use of dispatches, special frames that precede the IPv6 header and deliver information about the fields of the IPv6 header present in the transmission. The usual representation is shown in Figure 3.7. These dispatches do not replace IPv6, and the nodes still need to be able to configure their own IPv6 addresses and to know their own scopes.

Some of the dispatches available for 6LoWPAN [10] are listed in Table 3.1. The dispatches of BC0, MESH are not of interest in this work due to their link to the lower layers. NALP is used when the nodes have more than one network protocol and are thus outside this work's scope. NCH dispatch was the original proposal for the compression, it is nevertheless being lately deprecated by the IPHC compression.

The IPv6 dispatch is used to communicate that the next header is a full IPv6 header, giving an extra overhead to the original packet. It is relevant to use it in order to avoid confusion with the possible value for the `Version` field of IPv6. Some faulty configurations might ignore this dispatch. Additional precautionary measures must be used to distinguish the value 0x6 of the IPv6 `Version` from the dispatch IPHC. In this dissertation, when a node makes use of this faulty configuration it will be referred to as "raw IPv6 header".

The focus of this dissertation is on the dispatches IPHC, FRAG1 and FRAGN. The first dispatch is the compression selected for our testbed and will be handled in Section 3.2.2.1. The latest two are related to the fragmentation over 6LoWPAN, and will be discussed in Section 3.2.2.2 [10].

3.2.2.1 IPHC compression

The IPHC dispatch is defined in [4]. It was developed aiming at a more efficient compression than the HC1 dispatch, defined in [10].

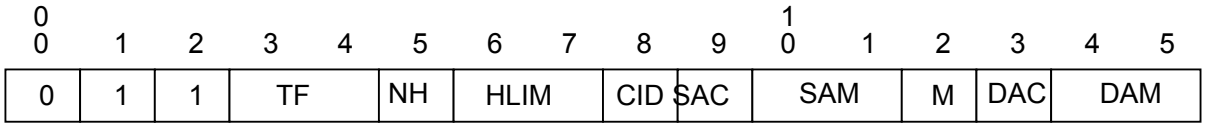


Figure 3.8: The LoWPAN IPHC Dispatch [4].

The dispatch consists of the fields shown in Figure 3.8, and of an optional third octet to define special contexts. The fields of this dispatch affect one or more of the fields of the IPv6 header, shown in Figure 3.6.

The field `TF` (2 bits) is used to compress the fields `Traffic Class` (8 bits) and `Flow Label` (20 bits). The Traffic class is split into two fields [4]: the 2-bits long Explicit Congestion Notification (ECN) and the Differentiated Services Code Point (DSCP). The possible values and their effects over those two fields are the following:

- 00: All fields (ECN + DSCP + 4-bit Pad + Flow Label) carried in-line.
- 01: ECN + 2-bit Pad + Flow Label (3 bytes), DSCP is ignored.
- 10: ECN + DSCP (1 byte), `Flow Label` is ignored.
- 11: `Traffic class` and `Flow Label` are ignored.

The field `NH` is a single bit that controls the field `Next-header` (8 bits) of the IPv6 header. The possible values and their effects are the following:

- 0: Full `Next-header` field carried in-line.
- 1: Field ignored, NHC dispatch at the end of this dispatch. The NHC dispatch is further discussed in Section 3.2.3.

The field `HLIM` are two bits that handle the field `Hop-Limit` (8 bits) of the IPv6 header. The possible values and their effects are following:

- 00: Full field `Hop-Limit` is carried in-line.
- 01: Field elided and Hop limit set to 1.
- 10: Field elided and Hop limit set to 64.
- 11: Field elided and Hop limit set to 255.

The field Context Identifier Extension (`CID`) does not handle any field of the IPv6 header but indicates if a special context follows this dispatch. The possible values are following:

- 0: No additional 8-bit `CID` present on this packet.
- 1: Additional 8-bits field after the `DAM` field. This field will not be discussed further within the present dissertation, as it is not relevant for the investigations conducted.

The Source Address Compression (`SAC`) is a 1-bit flag that controls if the source address compression is using stateless or stateful compression. This field impacts the field `SAM`. The possible values are following:

- 0: `SAM` using stateless compression.
- 1: `SAM` using stateful compression.

The Source Address Mode (`SAM`) is a 2-bits flag that controls the compression of the `source address` field of the IPv6 header. The `SAC` has an impact on the interpretation of the possible values.

- If `SAC` is set 0:
  - 00: The full source address (128 bits) is carried in-line.
  - 01: The last 64 bits of the source address, which belong to the host portion, are carried-inline. The other 64 are assumed to be the link-local high portion (FE80::/64) and are thus elided.
  - 10: The last 16 bits of the source address, which belong to the host portion, are carried-inline. The higher 112 bits are assumed to be the link-local high portion (FE80::FF:FE00:XXXX/112)
  - 11: The full source address (128 bits) is elided. It is assumed that the high portion is FE80::/64 meanwhile the lower portion is taken directly from the Source MAC address field on the Network Access layer.
- If `SAC` is set 1:
  - 00: The full source address (128 bits) is elided. It is assumed that is The UNSPECIFIED addresses is used (All the bits are set to zero, which is represented by "::")
  - 01: 64 bits of the source address are carried-inline. *The address is derived using context information and the 64 bits carried in-line* [4]. The bits not covered by context information are taken from the bits carried in-line.
  - 10: 16 bits of the source address are carried-inline. *The address is derived using context information and the 64 bits carried in-line* [4]. The bits not covered by context information are taken from the bits carried in-line and are assumed to be the last 16 bits (0000:00FF:FE00:XXXX).
  - 11: The full source address (128 bits) is elided. The address is derived by using context information and from the Source MAC address field on the Network Access layer.

The Multicast Compression (`M`) is a 1-bit field that has a direct impact on the `destination addresses` (128 bits) field in the IPv6 header. The possible values and their impact over the `DAC` and `DAM` are following:

- 0: IPv6 destination address is unicast or anycast.
- 1: IPv6 destination address is multicast.

The Destination Address Compression (DAC) is a 1-bit field that has a direct impact on the destination addresses (128 bits) field in the IPv6 header. The possible values and their impact on the DAM are following:

- 0: DAM uses stateless compression.
- 1: DAM uses stateful compression.

The Destination Address Mode (DAM) is a 2-bits flag that controls the compression of the source address field of the IPv6 header. The M and DAC fields have an impact on the interpretation of the possible values:

- If the IPv6 destination is unicast (M is set to 0).
  - If the DAC is set to 0 (stateless):
    - 00: full destination address (128 bits) carried in-line.
    - 01: first 64 bits of the IPv6 address elided. They are assumed to be the link-local padded with zeros (FE80::/64). The remaining 64 bits are carried in-line.
    - 10: first 112 bits of the IPv6 address elided. They are assumed to be the link-local padded with zeros (FE80::FF:FE80:XXXX/64). The remaining 16 bits are carried in-line.
    - 11: full destination address (128 bits) is elided.
  - If the DAC is set to 1 (stateful):
    - 00: Reserved.
    - 01: 64 bits carried in-line. The address is derived using context information and the carried in-line bits.
    - 10: 16 bits carried in-line. The address is derived using context information and the carried in-line bits.
    - 11: full destination address (128 bits) elided. The address is derived using context information and the carried in-line bits.
- If the IPv6 destination is multicast (M is set to 1).
  - If DAC is set to 0 (stateless):
    - 00: full destination address (128 bits) carried in-line.
    - 01: destination address takes the form FFXX::00XX:XXXX:XXXX, where the X's are 48 bits carried in-line.
    - 10: destination address takes the form FFXX::00XX:XXXX, where the X's are 32 bits carried in-line.
    - 11: destination address takes the form 02::00XX, where the X's are 16 bits carried in-line.
  - If DAC is set to 1 (stateful):
    - 00: 48 bits carried in-line. Those bits will be used to assemble a Unicast-Prefix-based IPv6 multicast address [4]. This multicast address has following structure: FFXX:XX:LL:PPPP:PPPP:PPPP:PPPP:XXXX:XXXX, where the X's are the bits carried in-line, in strict order. Meanwhile, the P's and L's (listening) bits are inferred from the specified context.

* 01: Reserved.
* 10: Reserved.
* 11: Reserved.

The fields `Version` (4 bits) and `Payload Length` (16 bits) are always elided when using the IPHC dispatch. The `Payload Length` will be inferred based on the `FCS` field of the IEEE 802.15.4 frame. *It is therefore possible to elide all the fields of the IPv6 header when the source and destination are unicast addresses.* The small overhead of the IPHC header and the NHC header still remain, but they are only about 3 bytes, 2 bytes of the IPHC and the remaining one of the NHC. NHC is further discussed in Section 3.2.3.

The use of unicast messages with fully elided fields is only possible when the communication remains within the WSN. The 64 bits of the MAC addresses of the sensors are already on the IEEE 802.15.4 and the link-local and global scopes can be inferred. If external nodes try to reach the sensors, their packets will necessarily requires to use a full IPv6 address, as this external global address cannot be inferred. The other case are the multicast messages, those are still required by the NDP and the custom routing protocol RPL.

The choice of stateless configuration for the sensors over the stateful one has an impact on the average size of the header. It can be considered a good practice to avoid using the stateful configuration and thus avoiding the complexity of adding a third service over the network. However, the stateful mode could be considered in cases where the space for host addresses needs to be superior to 64 bits (approximately $18x10^{18}$ addresses). Another theoretical scenario could be to give the same space as other architectures for WSN, such as RFID which requires 98 bits for the host space.

In general, if the `Traffic Class` and `Flow Label` fields are suppressed and stateless configuration is used, it is possible to reduce the full communication between the sensors to ten bytes or less, except when they need to communicate with the exterior.

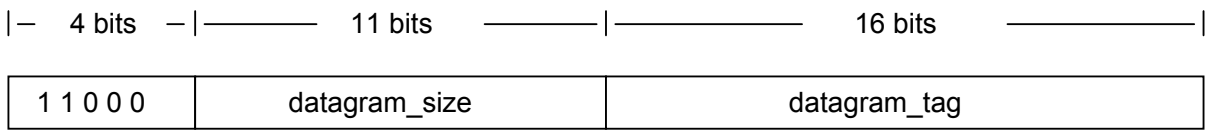
3.2.2.2 Fragmentation over 6LoWPAN

Because the IEEE 802.15.4 frame only supports 127 bytes for the MTU, messages exceeding this size will need to be fragmented. In that case, the nodes will use the dispatches FRAG1 and FRAGN. It is irrelevant if the IPv6 packet to fragment is already compressed. The dispatches are shown at the Figure 3.9a and Figure 3.9b.

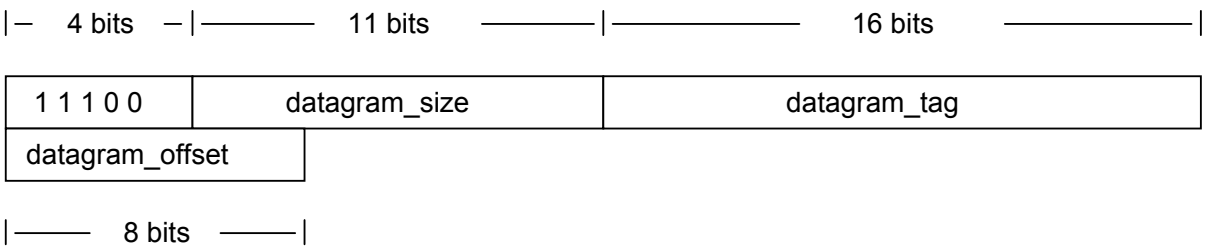
The FRAG1 dispatch will always mark the beginning of a new fragmented message with an ID defined in the field `Datagram Tag`. It is mandatory for this first message to include all the information concerning the dispatch used in the original message, including the header for the transport layer protocol. If the size of all headers combined exceeds the one of the MTU, then the message to transmit will be the raw IPv6 packet, without any type of dispatch preceding it [4, p.4].

FRAGN uses the field `Datagram Offset` to re-assemble the fragments of a single message. The field `Datagram Size` is used in both dispatches to help identifying each message and, most importantly, assembling the fragments correctly. The nodes must handle the reception of each FRAGN and calculate if the current amount of frames received fits the datagram size. If it is the case, then the node proceeds to reassemble the original packet and process it.

To identify which FRAGN corresponds to a specific FRAG1, the IEEE 802.15.4 source and destination addresses are used together with the `Datagram Tag`. The node that receives the FRAG1, will start to assemble the new packet, with a size equal to `Datagram Size`. Each new



(a) First Fragment (FRAG1).



(b) Subsequent Fragments (FRAGN).

Figure 3.9: The dispatches for handling fragmentation.

FRAGN received from the same `Datagram Tag` will be assembled according to its `Datagram Offset`. In the case of FRAG1 it is assumed that the offset is zero. If two FRAGN with the same offset are received during the assembling-process, the latest will prevail while the oldest will be discarded.

It is important to notice that the size that will be in the field `Datagram Size` will always be the sum of the raw IPv6 header and its payload, even in the case of a packet which is already compressed. The values in the `Datagram Offset` field are according to the raw size. This is why the FRAG1 must contain all the possible headers or else a raw IPv6 packet. Otherwise, the nodes are unable to reassemble the rest of the fragments correctly due to the inconsistency with the offset.

3.2.3 Transport layer

UDP and TCP protocols are the standard protocols used over the Internet. They are supported as well for WSNs based on the 802.15.4/6LoWPAN. However, TCP-overhead are usually regarded as excessive for the sensors [13, 83, 91]. A cause for this perception is the requirement of extra communication of TCP to guarantee a reliable channel, plus the impact that it has over the sensors resources. Further drawback which could be considered is its size of at least 20 bytes against the 8 bytes of UDP.

When the dispatch IPHC is used, it is possible to use NHC to compress the headers after the IPv6 header. Although NHC can be used for different headers, the one presented here is for the UDP one. The standard UDP header is shown in the Figure 3.10 and the header for NHC in the Figure 3.11. The NHC header is basically made of two variable fields:

- The `C` or `checksum` field is a flag that handles the checksum of UDP:

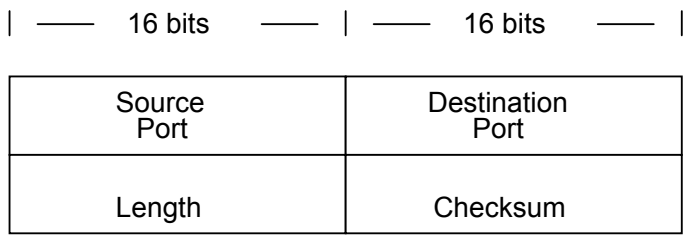


Figure 3.10: UDP Header.

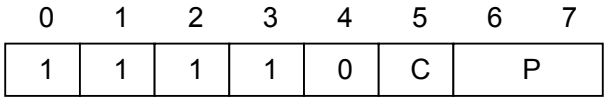


Figure 3.11: UDP Header Encoding with LoWPAN NHC [4].

  - Set to 0: The 16 bits of checksum are carried in-line, after this header.
  - Set to 1: The 16 bits of checksum are elided. Only the checksum of the 802.15.4 is available.

- The `P` or `ports` is a field of two bits handling the possible combinations for the port addresses:
  - Set to 00: The 16 bits of the source and destination addresses are carried-in line.
  - Set to 01: The 16 bits of the source address are carried-in line. Only the lower 8 bits of the destination address are carried in-line; its higher portion is assumed to be 0xF0.
  - Set to 10: The 16 bits of the destination address are carried-in line. Only the lower 8 bits of the source address are carried in-line; its higher portion is assumed to be 0xF0.
  - Set to 11: Only the lowest 4 bits of both addresses are carried in-line. The higher 12 bits elided are assumed to be 0xF0B.

The `checksum` field should not be enabled unless the sensors are able to check its integrity, such as with the use of the `Field Checksum` field of 802.15.4, or because in the UDP payload is provided a similar ability, such as in the case of tunneling over UDP [4, p. 17].

The benefits of this compression are only visible when the port addresses are compressed, in particular when both addresses are compressed to a simple nibble because the final header is reduced from 8 bytes to only 2. However, if the range of bits to be used are lesser than the original 16 bits, the configuration of the applications must been made with a lot of care. By default, the ranges of 0xF0XX and 0xF0BX are used for the configuration of 8 bits and 4 bits. The sensors must be able to operate accordingly, unexpected errors in the communication of the sensors may otherwise occur.

3.2.4 Application layer

The application layer of the TCP/IP model can be considered as ambiguous in comparison to the OSI model. Mostly, because it does not make a distinction from the session between the nodes (the session layer of the OSI Model), the way the data is presented (the presentation layer of the OSI Model) and the interface with the human (the application layer of the OSI Model). For this dissertation, the first two are very important, the session established between the sensors must be secured, and the way to present and transmit the data must be optimized for the sensors.

For securing the communication over UDP in a feasible approach, and as close as possible to the traditional TCP, it is possible to use the DTLS, defined in [5], originally designed for applications running over UDP, such as VoIP, and not for constrained devices. As a consequence, the overhead of DTLS can be too large for the small amount of memory (RAM and ROM) and even for the processor power. Because of this, special considerations for this protocol must be made.

Protocols based on the paradigm Representational State Transfer (ReST) have been developed in the last years to reach a reliable communication using an intrinsic unreliable protocol, such as UDP. Usually they are similar to HTTP, but are designed for constrained networks requirements, for instance 6LoWPAN [91]. These protocols are ideal for M2M communication over constrained networks.

CoAP is a ReST protocol defined by the IETF [92] and has the same type of messages as HTTP (GET, PUT, POST and DELETE), which makes them compatible. In a similar way to HTTPS, composed by the use of HTTP with TLS, it is possible to reach a CoAPS using the DTLS protocol. With CoAPS, only the lowest channel of DTLS will be visible, meanwhile the CoAP communication will be encrypted and thus hidden to any monitoring tool.

The remaining of this sub-section is composed as following: All the characteristics of DTLS are discussed in Section 3.2.4.1. Meanwhile, details of the CoAP protocol are specified in Section 3.2.4.2. Lastly, formats for representing the data captured by the sensors are mentioned in Section 3.2.4.3.

3.2.4.1 Datagram Transport Layer Security

The DTLS provides the security equivalent to TLS (confidentiality, integrity, authentication and non-repudiation) [5]. Currently defined are the DTLS version 1.0 and the DTLS Version 1.2 which are equivalent to the TLS version 1.1 and TLS version 1.2 respectively. The focus in this dissertation is only with DTLS 1.2, though the differences between them are minimal.

In terms of TLS and DTLS, before sending any data related to the upper layers, in this case the CoAP messages, a secure channel must be first established; the channel must operate only for a certain time and normally its lifetime is the time required for transmitting the CoAP message. This is typically referred as a DTLS session.

DTLS is able to provide a DTLS session by means of one of the following modes [24, 49]:

- `Nosec` - In this mode, although the DTLS session is established, the upper layer is not encrypted. All the communication is in plain text.

- `PreSharedKey` (PSK) - In this mode, all the devices have pre-loaded symmetric cryptographic keys for all the DTLS sessions. Changing those keys is outside of DTLS functionality.

- `RawPublicKey` (RPK) - This mode allows the use of a pair of asymmetric keys for authentication. However, a third entity is not necessary.
- `Certificate`. This is the equivalent of the infrastructure for HTTPS where the X.509 certificated mode is used. It supports digital certificates and public key encryption.

In the `PreSharedKey` mode, one or more nodes share the same keys; this permits to authenticate members as a part of one group without the use of public-key cryptography [79]. The entropy of the keys should be sufficient to mitigate brute force attacks and dictionary attacks [93]. This mode can be used in deployment scenarios where the sensors operate unattended and without security infrastructure [49, 79].

As the keys can be pre-loaded in the nodes, it is possible to generate a list of members of specific groups, sharing the same keys, generating a relation of *members:groups* [94]. It is possible to reach a rate of 1:1, where each node has its own key, but memory could be a limitation.

Although the `RawPublickKey` mode can guarantee a better protection, some works have identified this mode as too demanding for the resources. In particular it has a high impact on the battery life of the sensors [49, 79]. The overhead is still smaller than the one from the `certificate` mode, but at the same time, it requires extra work for ensuring the binding between the identifier and the key [95]. In the work of Granjal et al. [79] it was identified that cipher suites using the PreShared Keys (PSK) have a noticeably larger lifetime than those using RSA, almost 50% more. However, the CoAP requirements indicate that the `RawPublickKey` should be available for the sensors [49].

Any of the four previous modes are reached by means of a handshake process in which one or more cipher suites will be enumerated according to the modes that are available to the devices. The handshake is started by the client, with a list of the available cipher suites. The server then will have the final choice, including the possibility to reject the communication. Once the handshake is completed, the CoAP datagrams will be integrated as a type of messages of DTLS.

All the steps involved in establishing a secure channel over DTLS will be explained next. First the specifications and concepts related to different types of DTLS messages are introduced. After this, the different types of cipher suites available to the aforementioned four modes are introduced. Next to it, follows a detailed handshake process for the session. At last, we close the discussion with observations and conclusions of DTLS.

DTLS records

All the messages involved in the operation of DTLS are referred as DTLS records. There are only four types:

1. Handshake - Used for generating the DTLS session.
2. Application Data - Used for transmitting the data from the upper layers.
3. Alert - Used for terminating a DTLS session.
4. ChangeCipherSpec - Used for updating a DTLS session.

The structure of a DTLS record is very similar to its counterpart of TLS. However, due to the unreliable nature of UDP, DTLS requires new fields to handle the fragmentation at this level. The following is the common portion of the four types of messages:

- `Content Type` - Field of 8 bits for identifying one of the four types of DTLS messages.
- `Version` - Field of 16 bits identifying which version of DTLS generated this record.
- `Epoch` - New to DTLS. Field of 16 bits used as a counter for every cipher state change.
- `Sequence Number` - New to DTLS. Field of 48 bits for identifying the sequence number of this record.
- `Length` - Field of 16 bits for indicating the total size of this record. Max value possible is 2^{14} or 16,384.
- `Fragment` - Field of variable size. The content of the record is located here. The content is in plain text only in the messages of the type handshake.

Additionally, if the MTU allows it, more than one single DTLS record can be combined inside of a single datagram. In this dissertation, the term "jumbo datagram" is used as a reference to those datagrams. When a jumbo datagram is present, the total size of the packet will come only in the UDP header, or NHC for WSN. Therefore, the field `Lenght` of each separated DTLS record is required to identify the correct portion of each record.

In a similar way, a single DTLS record can be broken into multiple smaller DTLS records of the same type; this could happen if the MTU was too small for the size of the message. This scenario is specially probable with the handshake type.

Cipher suites

```
<ciphername>
<Protocol Version>
Kx=<Key exchange>
Au=<Auhtentication>
Enc=<Symmetric encryption method>
MAC=<MAC code>
<Export code>
```

Figure 3.12: A generic composition for Cipher suites in DTLS [5]

Basically, what defines each one of the 4 modes of DTLS are the cipher suites. Those suites are the stack of 4 different protocols for establishing a session, sharing secrets and authentication. In Figure 3.12, the general structure of any cipher suites is shown and it is explained below:

- The type of *key exchange* to use. Examples are RSA, Diffie-Hellman (DH), ECDH, SRP and PSK.
- The type of symmetric encryption to use, referred as *bulk encryption*.
- The mechanism for authentication, referred as a *message authentication code* (MAC)[1]and involves well-known digital signature algorithms.
- A *pseudorandom function*, for TLS, and DTLS, a hash-based message authentication such as MD5 or SHA.

DTLS and TLS share the same cipher suites list with the exception of the stream ciphers which do not work well with UDP [83]. Said list has unique names composed normally by the name of the main protocol that is using it, TLS in this case, and the four components previously identified. Each cipher suite has a unique ID composed of 16 bits. Examples of cipher suites are[2]:

- `TLS_RSA_WITH_3DES_EDE_CBC_SHA` (ID 0x0035)
- `TLS_RSA_EXPORT_WITH_RC4_40_MD5` (ID 0x0003)
- `TLS_DHE_PSK_WITH_RC4_128_SHA` (ID 0x008E)
- `TLS_PSK_WITH_AES_128_CCM_8`(ID 0xC0A8)
- `TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8`(ID 0xC0AE)
- `TLS_NULL_WITH_NULL_NULL` (ID 0x0000)

The first one in the list is a mandatory cipher suite for TLS, but not suitable for DTLS because of the CBC component. The following two are standard cipher suites for DTLS v. 1.2 (and TLS 1.2). The next two are non-standard cipher suites available for DTLS 1.2. The last one, is a cipher suite which does not add protection at all in the DTLS records but will complete the full handshake process.

Because the cipher suites can have very specific requirements, some of the DTLS records will have additional fields for providing said requirements. Those extra fields are referred as "extensions" and are highly dynamic in content and size.

The power performance of sensors supporting DTLS has been the main interest of multiple researches in the last years as discussed in Section 2.5.3

Handshake process

Before transmitting the upper layer protocol, a DTLS session must be established and for this a handshake process is required. The handshake is broken down into six steps which are referred as flights. The flights can be composed of multiple DTLS records. The full process of this handshake is shown in Figure 3.13. In total, the 6 flights can generate up to 15 messages and are described briefly below:

- The DTLS session is always requested by the client, a DTLS Client Hello record is used for this flight, which includes all the cipher suites available by the client.
- The server responds to the client with the DTLS Hello Verify Request record. This record includes a cookie to mitigate the repetition of attacks. This is the only record of the full handshake that should always be DTLS 1.0.
- The client confirms the process with a new DTLS Client Hello record, this too confirms which version of DTLS is to be used. Also, the client begins to include the cookie in the record.
- The server answers the client request, selecting one of the cipher suites offered by the client and preparing the exchange of keys. This flight can be composed of multiples DTLS records, but it will always begin with the DTLS Server Hello record that carries the cipher suite selected and ends with the DTLS Server Hello Done record.

[1]This MAC protocol must not be confused with the MAC layer of the IEEE 802.15.4 protocol.
[2]A full list is avaliable at http://www.thesprawl.org/research/tls-and-ssl-cipher-suites/

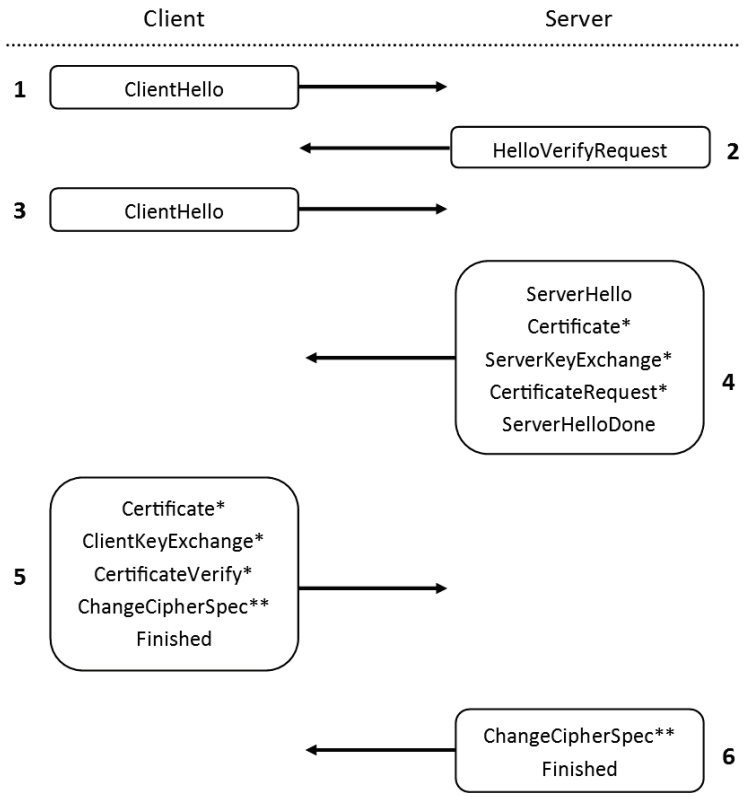


Figure 3.13: The six flights of DTLS handshake process.

- The client will process the selected cipher suite and continue with the process of the exchange of keys with the DTLS Client Key Exchange record. Once the client has processed all its part of this handshake, it will notify the server with the DTLS ChangeCipherSpec and Finished records.
- The server will use the previous DTLS ChangeCipherSpec record to finish its part and will notify the client with its own DTLS ChangeCipherSpec and Finished records.

The flights 4 and 5 are dynamic in size due to the extensions that bring the special requirements for the preferred cipher suite of the client and the one selected by the server. Also, the cookies play a role as their size varies from 0 to 255 bytes. The first DTLS Client Hello record does not include a cookie.

In total, there are eleven types of messages for the handshake process. Their structures are the following:

- `Handshake type` - 8 bits for identifying the type of handshake message.
- `length` - 24 bits for the total size of the DTLS record, before any possible fragmentation.
- `message_seq` - 16 bits to help to identify a new message generated from a repeated message.
- `fragment_offset` - 24 bits to help to reassembled a fragmented DTLS record.
- `fragment_length` - 24 bits to identify the size of a specific fragment. If the DTLS record is not fragmented, this field has the same value than the `length` field.

- Specific to each handshake type:
  1. `HelloRequest` - It is not expected to have a body at all.
  2. `ClientHello`
     - `client_version` - 4 bytes for the DTLS version supported by the client.
     - `random` - 32 bits for a EPOCH time plus 28 bytes of entropy (random bytes).
     - `session_id` - 8 bits to identify a specific DTLS session.
     - `cookie` - Dynamic in size, from 1 byte to 256 bytes.
     - `cipher suites` - Dynamic in size, multiple of 2 bytes. Minimum 4 bytes.
     - `compression_methods` - Dynamic in size, minimum 2 bytes.
     - `Extensions` - Zero, one or more extensions can be added to this record. Each one is dynamic in size.
  3. `ServerHello` - Same composition than the `ClientHello` record.
  4. `HelloVerifyRequest`
     - `server_version` - 4 bytes for the DTLS version supported by the server.
     - `cookie` - Dynamic in size, from 1 byte to 256 bytes.
  5. `Certificate` - Highly dynamic, according to the cipher suite selected.
  6. `ServerKeyExchange` - Highly dynamic, according to the cipher suite selected.
  7. `CertificateRequest` - Highly dynamic, according to the cipher suite selected.
  8. `ServerHelloDone` - The body itself is empty.
  9. `CertificateVerify` - Highly dynamic, according to the cipher suite selected.
  10. `ClientKeyExchange` - Highly dynamic, according to the cipher suite selected.
  11. `Finished`
      - Cipher text dynamic in size, although default is 12 bytes.

DTLS messages can carry optional extensions, which are features requested by clients and that extend the DTLS functionality; some of the most important are the cookies, which help to prevent denial-of-services attacks. Additionally, some of the DTLS messages in each flight can be omitted; this will happen when the cipher suite chosen by the client does not require them.

The DTLS Hello Request record is not shown in Figure 3.13 because it is used only by the server to request the re-transmissions of the Hello messages.

The DTLS ChangeCipherSpec record is not strictly speaking a handshake record. Its function is to indicate that subsequent DTLS records will be protected with a new couple of symmetric keys. Therefore, the communication cannot continue until the other part finishes the preparation of the new set of keys, this is indicated by the second DTLS ChangeCipherSpec record. The DTLS Finished records indicate that the handshake is finished and that they are the first DTLS records to be indeed protected by the DTLS session.

The DTLS Certificate, Server Key Exchange, Certificate Request and Certificate Verify records are used in the flights if the cipher suites are using the mode `RawPublickKey` or `Certificate`. If the client and server are using different modes in their configuration, it can happen that not a single cipher suite is accepted and in consequence the DTLS session cannot be established.

Each one of the DTLS records can be broken into multiple smaller DTLS records; this can happen if the MTU is too small for the size of the message. This behavior normally occurs on the

```
157.159.103.42    157.159.103.56    DTLSv1.2    270 Server Hello, Certificate (Fragment)
157.159.103.42    157.159.103.56    DTLSv1.2    270 Certificate (Fragment)
157.159.103.42    157.159.103.56    DTLSv1.2    270 Certificate (Fragment)
157.159.103.42    157.159.103.56    DTLSv1.2    247 Certificate (Reassembled)
157.159.103.42    157.159.103.56    DTLSv1.2    130 Certificate Request, Server Hello Done
```

Figure 3.14: DTLS flight 4 with jumbo datagrams and fragmentation (Over ethernet and IPv4).

`Certificate` mode. This is something to consider because of the fragmentation over 6LoWPAN. As already mentioned, the element to differentiate a full DTLS record from a fragmented one are the fields `length` and `fragment_length`. If they are of different size, then the message is fragmented and future records of the original message are recognized by having the same values for fields `length` and `message_seq` and different values for the field `fragment_offset`.

Figure 3.14 shows the capture of traffic between a server and a client. The popular OpenSSL stack is being used. Those messages are specific to the fourth flight. The first and last records are jumbo datagrams where each one carries two separate handshake records. The fragmentation over UDP with the DTLS Certificate record is also visible and it is divided into four fragments.

General performance

Although the implementation of DTLS over WSN is highly attainable, the cipher suites are the culprits of the negative impact on the sensors: those that fall under the `certificate` mode can require more than one single megabyte to transmit the digital certificate, and this size has an impact on the sensors resources from each node. The overhead is inclusive higher due to the presence of additional architecture. This brings as a consequence to only consider the modes `RawPublickKey` and `PresharedKey`. However, even if the `PresSharedKey` mode is used, the size of the keys for the symmetric encryption can still drain the batteries of the nodes quickly.

This overhead in the communication and upon the device's resources has been already discussed in the past. An alternative that has been analyzed is the use of non-standard cipher suites for DTLS but friendlier for the sensor, or heavy modifications to the DTLS protocol [49, 82, 83].

An alternative is offered in the work of Kothmayr et. al [83] which supports the notion of using DTLS and recommends the use of a Trusted Platform Module (TPM) embedded chip, which performs the RSA algorithm operations in hardware, as it is the case for the DTLS Certificate mode.

3.2.4.2 The Constrained Application Protocol

While DTLS takes care of establishing a secure session, CoAP takes care of the presentation of the data. In a brute comparison, CoAP can be seen as a generic web protocol almost equivalent to HTTP but designed for constrained networks requirements, as for instance 6LoWPAN [24]. The CoAP is a protocol based in the ReST and defined by the IETF [40]. It has the same types of messages as HTTP (GET, PUT, POST and DELETE)

CoAP provides a paradigm of request and response similar to HTTP, with devices interacting by sending a request and receiving a response [94]. The access to its resources is done by means of URIs which use standard addresses and port numbers.

An example is shown in the Figure 3.15, where a client requests a specific resource with the path `sensors/temperature`. The default UDP port for CoAP is 5683. Once the client sends

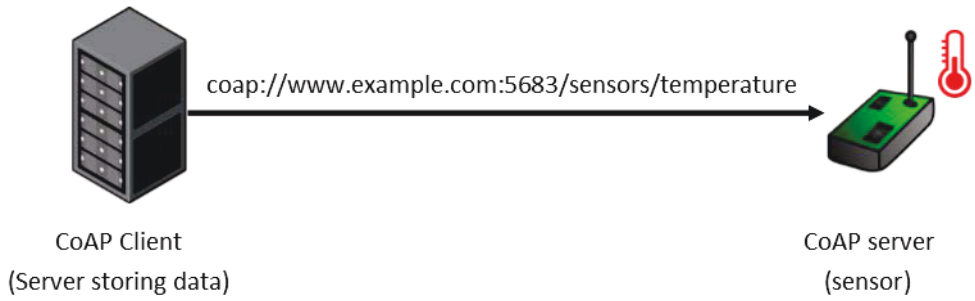
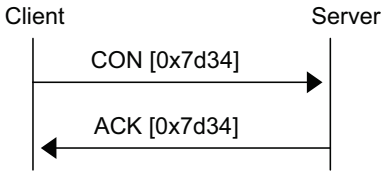
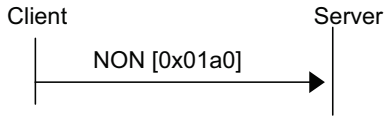


Figure 3.15: Example of CoAP Client and Server.



(a) Reliable Message Transmission.



(b) Unreliable Message Transmission.

Figure 3.16: Examples of a CON and NON message for CoAP [6].

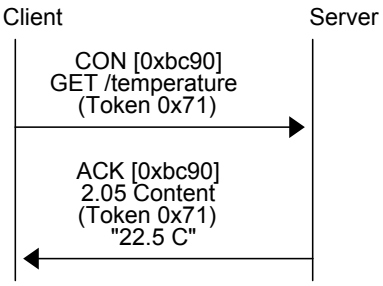
the request it will receive a response in one type of format, such as CBOR, JSON, XML or plain text [94].

The following is a brief summary of the types of messages available to CoAP [6, 94]. They are shown in Figure 3.16:

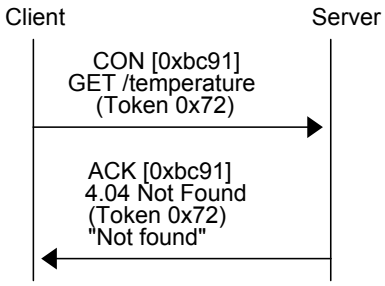
- CON - Confirmable message which requires a response. Otherwise will re-send the message until an answer is received.
- NON - Non-confirmable message. Useful for unreliable connection as the loss of one message has a low impact.
- ACK - Acknowledgment. These messages are used in response to CON messages. This type of message can carry the answer, referred as a piggybacked response [6].
- RST - Negative acknowledgment. It means reset and can indicate a failure.

In a standard operation, when the client requests a resource by means of a CON message, the client will receive an ACK with the piggybacked response. This answer can contain the resource, as in Figure 3.17a, or an error message as in Figure 3.17b. In scenarios where the server cannot respond immediately, it will send an empty ACK to the client with the purpose to avoid unnecessary retransmissions from the client. Once the server is able to answer, it will send a CON message carrying the response to the client. This behavior is referred as a separate response and is shown in Figure 3.17c. Finally, it is also possible to avoid the use of CON/ACK messages and only transmit a couple of NON messages, as shown in Figure 3.17d.

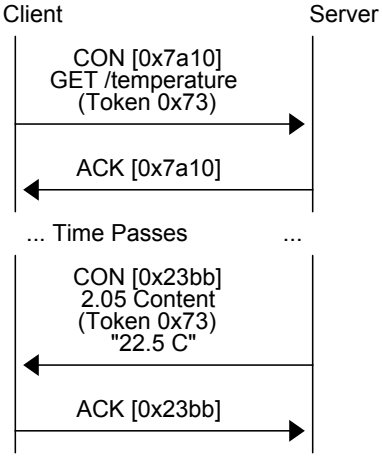
As already sown in Figure 3.15, the URI starts with `CoAP://`, but similar to HTTP Secure (HTTPS), it is possible to start it with `CoAPS://` equivalent to CoAP Secure (CoAPS). While



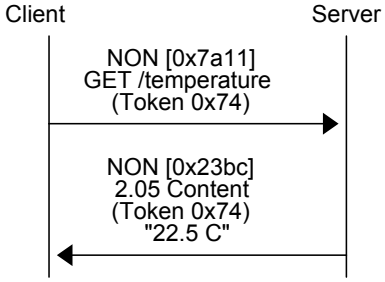
(a) GET Requests with Piggybacked Responses.



(b) GET Requests with Piggybacked Responses (not resource available).



(c) A GET Request with a Separate Response



(d) Request and a Response Carried in NON messages.

Figure 3.17: Examples of GET requests over CoAP [6].

HTTPS make uses of SSL and TLS, CoAPS make uses of DTLS. This brings as a consequence that the monitoring of packets with CoAPS will be hidden to us. Therefore, the CoAP protocol is not extensively detailed in this section in comparison of the protocols involved in previous layers. When CoAPS is used, it is mandatory that the DTLS stack supports the following cipher suites [49]:

- The `TLS_PSK_WITH_AES_128_CCM_8` for the `PreSharedKey` mode.
- The `TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8` for the `RawPublicKey` mode.
- The `TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8` for the `Certificate` mode.

All the CoAP messages shown in Figure 3.18 shared the same format. The header is properly formed by 2-bytes, however it will indicate the final size of the rest of the message. The message fields are:

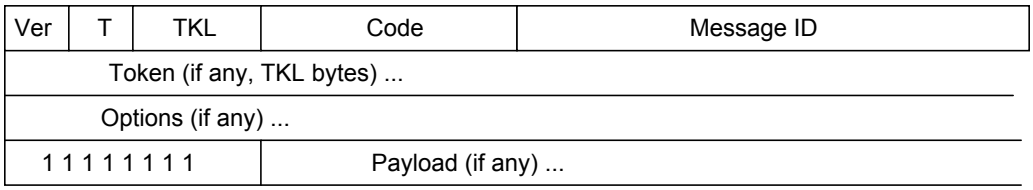


Figure 3.18: General architecture of a CoAP message.

- `Version` (Ver) - 2 bits to indicate the current version of CoAP. Currently, only a value of 1 is valid.
- `Type` (T) - 2 bits to determine if the message is of type CON (0), NON (1), ACK (2) or RST (3).
- `Token Length` (TKL) - A field of 4 bits to indicate the length of the Token field.
- `Code` - A field of 8 bits to identify the type of class of the request method and the response method (GET, PUT, POST, NOT FOUND, etc.). Or, if the message is empty.
- A 16-bit field to identify the message, this is used to detect message duplication and to match the messages ACK and RST.
- `Token` - An optional field of up to 8 bytes. Used for correlating requests and responses.
- `Options` - An optional field of dynamic size for indicating possible options. The most typical are the members of the URI path.
- `Payload Maker` - An optional static field of 8 bits set to 1s (0xFF) to indicate the start of the payload. The absence of this field indicates the absence of it.
- `Payload` - The payload of this message. The size will be inferred from the UDP length field.

The tokens are used to co-relate messages that participate in single related transaction, even when the identifiers are different [94].

There is also an extension for CoAP to add support to the publisher and subscribers model. In this model, one or more clients send a special GET request with the option of Observer to monitor constantly the resource requested [13]. The server will notify periodically, or upon events, of the current status of the resource and provided an identifier to help the clients to discern the latest update. An example of this model is shown in Figure 3.19.

Finally, although CoAP is able to work alone, it is considered than standard clients making use of HTTP should be able to communicate directly with CoAP resources. This is done by means of a HTTP-CoAP intermediary or a cross proxy [91]. This cross proxy should straightforwardly translate the GET and REQUEST message between both protocols in a transparent way.

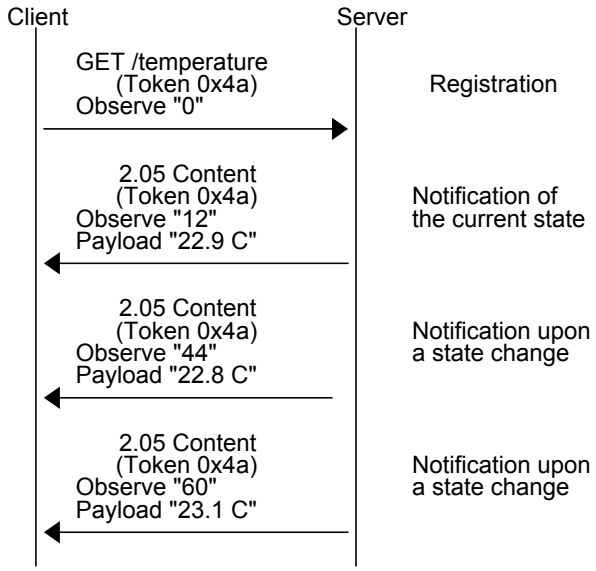


Figure 3.19: Observing a Resource in CoAP [7].

3.2.4.3 The data format

How to present the data is equally important as the complete previous stage. They are abstract entities with freedom of specific formats [96]. However, the standardization helps to decentralize a system so that it can interact better. For web services, there are already well-known standards, such as HTML [96]. Yet, for the communication over sensors, where the data is the measured resource, protocols with a minimum of verbosity are better suited; in this particular case popular representations are with Extensible Markup Language (XML) and JavaScript Object Notation (JSON) [97, p. 242]. Furthermore, there is a more recent protocol called CBOR worth to be considered. The content in this entry focuses only on the general description of those three standards.

The XML has been used to represent the data on traditional devices [91], still, it has a weakness due to its "verbosity" to define the data, as it is not ideal for constrained networks. Yet, is possible to use Efficient XML Interchange (EXI) as a compact and less verbose representation of XML; although it was not originally designed for devices on WSN [98]. The original objective of EXI was to give high performance to XML applications, with a reduced cost to the bandwidth overhead [89].

EXI has been validated in different works [99–101] and can be configured in two modes or schemes: I) schema-less encoding where it is generated from the XML data, allowing other nodes to decode it without prior knowledge about the structure. II) schema-based which requires that the nodes share the same XML schema to be able to encode and decode the transmitted data. In both modes, a wrong formatted EXI on any part of the communication could provide false information to the receptor, risking a unexpected behavior from the sensors or the clients.

JSON is a human-readable format based on a subset of JavaScript, and it is able to be parsed correctly to any JavaScript compiler; though it is possible to use it for other languages [102]. Any JSON data can be seen as a string, making it easy to read its components.

The work of Pacini et al. [103] is a comparison of JSON and EXI, both scheme-less and scheme-based, for WSN. They concluded that for purposes of Bandwidth, the EXI scheme-based outperforms the other two in terms of energy consumption, execution, time and bandwidth. Yet, it is JSON who has a significantly lower memory requirement, making it very desirable for heavily constrained sensor nodes.

Concise Binary Object Representation (CBOR) is a relatively new standard, released in October 2013 [104]. It is similar to JSON. Its purpose is to give an extremely small size of code, able to fit in the size of a simple message and to have extensibility without the requirement of negotiation between versions. Some of its major additions are the binary byte strings, and that it is able to fit on constrained devices. At the moment of writing this dissertation, there are not enough works done that involve CBOR. Yet, due to the lesser amount required than with JSON, CBOR it is ideal to be used for devices of the categories C1 or C2 of Table 1.1.

3.3 Classification of communication between the nodes in the architecture

In Section 3.2 the purpose was to discuss all the different protocols involved in the WSN. Meanwhile, the purpose of this section is to identify and classify the different types of services and devices used in the proposed architecture based on IEEE 802.15.4/6LoWPAN. As explained in Section 1.1, this architecture is considered to protect the sensors from unauthorized use independently of the sensitivity of the data.

Figure 3.1 shows the whole proposed infrastructure. As previously commented, this model is separated into three categories:

1. The WSN, already discussed in Section 3.2.1. With sensors measuring one or more type of data.

2. The core network that is proposed to be an IPv6 network with key components such as proxies and firewalls.

3. The external clients which are not under our control.

Figure 3.1 shows sensors with different metrics for measurement. However, this is only to reflect the fact that it is possible to have multiple sensors deployed in a wide area with different purposes. It is also possible to have an area big enough to have two different mesh networks, or even have multiple WSNs with very specific requirements for each one. Yet those scenarios are outside of the scope of this dissertation.

The requirement of proxies is mandatory to achieve a communication between WSN and the core network, and with the external clients. The first proxy required is for the compression and decompression of the IPv6 header between the core network and the WSN. This proxy is reflected as the 6BR in Figure 3.1. This is furthermore discussed in Section 3.3.1.

The secondary proxy is theoretically not required. Yet it is recommended to be present as it makes the translation between HTTP and CoAP. This proxy helps to reduce the complexity for the external clients, avoiding the necessity of installing any additional software on the clients. This is further discussed in Section 3.3.2.

The core network must make use of traditional security considerations, which are also outside of the scope of this dissertation. It also needs to be able to take away as many possible threats to the sensors as possible, this is in order to save their resources and the constrained bandwidth in the WSN. This is further discussed in Section 3.3.3.

3.3.1 The communication inside the WSN

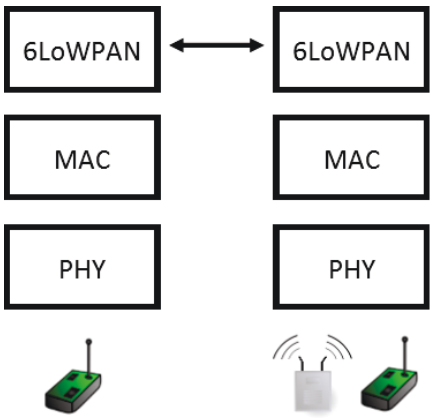


Figure 3.20: Communication between a sensor and the 6BR

It is expected that almost all the communication inside of the WSN will have as a destination the exterior, but because of the mesh topology, the messages will be transferred from sensor to sensor until they reach the 6BR. Figure 3.20 shows the layers involved in the communication.

Additionally to the function of relaying messages from other sensors, the sensors must also answer to RPL and ICMPv6 messages. This mix of messages can use different configurations for the dispatches.

A third type of node can also be found inside the WSN, which carries special functions for the security. An example of this, are the nodes that provide the PANA service on the MAC layer. The PANA is an asymmetric encryption with AES [105, 106]. There may be also nodes monitoring the traffic in one or more portions of the mesh topology to be able to detect different type of threats, such as those discussed in Section 2.3.

The internal monitoring of the WSN should really be considered, as rogue nodes are easy to get inside. However, an effective monitoring of the complete topology is not always feasible. Yet, all the traffic destined to the exterior must go through the 6BR. Thus, the 6BR becomes a mandatory fixed point for monitoring.

3.3.2 The communication in the Core network

Although the external clients can use any type of network for reaching the Internet, the core is proposed to work with IPv6, for integrating the WSN to their internal routing tables. This brings as advantage the integration with the internal routing tables of the WSN. Even if IPv4 is also feasible, it is left outside of the scope of this dissertation due to the redundancy and the fact that it is of no use to the WSN. The use of a NAT64 to support external IPv4 clients can be considered, particularly if they connect only to the proxy HTTP/CoAP.

The communication between the core and the WSNs should be under a strict control. This is the final point of protection against remote attacks to the sensors. The nodes in the core have a better bandwidth and much more resource available to them. Therefore, they should be used to detect and stop malicious messages destined to the sensors. Because of this, firewalls, honeypots and other devices destined to the security should be implemented in this network. This of course, excludes rogue nodes able to connect directly to the WSN. Yet, malicious traffic originating from them and leaving the WSN, could also be detected.

Inside the core, probably the most relevant node for the WSN is the proxy for HTTP and CoAP. This should be the "sensor" available to the external clients; this is in part, because it is able to attend high peaks of demand and in part because it is a last filter for analyzing the content of the HTTP header, or even the payload of CoAP. However, its most practical purpose is to give a more transparent service to the external clients [6, p. 72]. This change of session will provoke an overhead in the time of the communication, but it will also be easier to provide stronger cipher suites from the core to the external clients.

The traffic that arrives and leaves the WSN should be generated only when data is requested from the sensors, or alternatively, from system administrators taking special inspection of the sensors.

3.3.3 The communication between the Core and the WSN

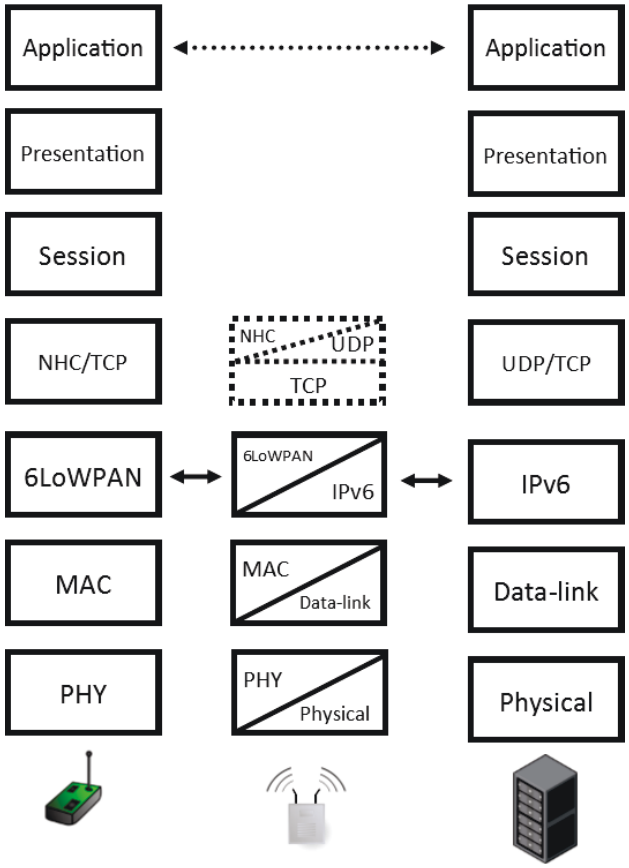


Figure 3.21: Communication between a sensor and proxy HTTP/CoAP.

Figure 3.21 reflects the expected communication between any node in the Core, or from external clients, reaching to or coming from the WSN. The 6BR is the most important node in the WSN because of its function as the translator between IPv6 and 6LoWPAN network. It will compress the IPv6 address of the involved sensor in the communication, while the external IPv6 address remains untouched.

The 6BR also verifies if the transport layer is UDP, and if so it will try to compress it with NHC. Anything else on this layer, and upper layers are not touched by the 6BR.

The sensors are constrained in resources. Therefore, the TCP stack cannot be compiled to save resources from the sensors. And, in a similar approach, only the CoAP protocol can be compiled for the upper layers. Yet, this does not guarantee that other types of traffic are not going to appear in the WSN. This could happen because nodes are misconfigured or because they are rogue nodes.

3.4 Threats to the Machine to Machine communication

The IoT is composed of a highly heterogeneous collection of devices, where many of them are constrained in resources. Yet, they are expected to transfer data, sometimes highly sensitive, between themselves to a central core or multiple external clients. Therefore, it is necessary to guarantee that the M2M communication is equally safe as in any other traditional network for the scenarios where the HTTPS protocol is used.

The objective of this section relays in the analysis of the threats to the communication in order to identify key elements in the protocol CoAPS, discussed in Section 3.2.4, to guarantee a secure M2M communication.

This section begins with the analysis of the main threats related to the M2M communication in Section 3.4.1. It is also followed by the general conclusions in Section 3.4.2.

The content of this section was strongly used for the elaboration of the work presented at STAM 2016 [86].

3.4.1 Analysis of risks

A Risk analysis for the WSN, or a M2M environment, can be obtained from the report generated by the ETSI Technical Committee M2M [11]. The ETSI is an entity with a predominance over the standards and regulation for the IoT [15,23,91,103,107–109]. Although the analysis of ETSI takes into consideration all the devices inside the M2M architecture, *we are only interested in those that affect directly the communication from sensors to sensors, to or from the proxies as well as the border router for the WSN*. Also, in concordance with the consideration of the scenario proposed in Section 3.3, for this analysis it is considered that the sensors are carrying sensitive data, or that the interception, duplication or suppression of one or more messages can alter the normal operation of the WSN.

The ETSI Technical Committee (TC) M2M has identified a total of 21 threats in two types: I) threats specific to the M2M Service layer (the applications itself) and its interfaces. And, II) threats affecting the functional requirements of the M2M environment. All the threats of interest to us are found in the first category. This category has identified 19 threats that encompass the alteration of the hardware or the communication. The first seven threats are related to the extraction and adulteration of keys stored in the memory of the nodes. Threats 9, 11, 12, 13 and 14 are related to the alteration of applications stored in sensors or key devices in the network infrastructure. The treats 18 and 19 are related to the unrestricted access to resources by one or more nodes in the infrastructure. Only the threats 8, 10, 15, 16 and 17 are directly related to risks in the communication of the nodes inside the WSN. A full list of the threats and countermeasures identified by the ETSI TC M2M is available in the Appendix A.

The original number of the threat (ET#) and the countermeasure ID (CM#) are respected. The following topics are considered for the analysis:

- The likelihood to exploit a specific threat, measured as low, moderate, substantial or severe.

- The level of impact over the WSN and the reputation of its provider if the threat is exploited.
- A grade of seriousness (risk) of the threat if it is exploited. The ranges are minor, major and critical.
- A risk value based on the factor of likelihood.
- Required countermeasures to mitigate the threat.

The following list of terms is defined by the ETSI M2M for their M2M platform with the specific translation for WSN [11, 109]:

- *M2M service layer* - This is the application making use of the service inside the WSN. For our scenario, it is CoAP, as discussed in Section 3.2.4.
- *M2M device* - The sensors.
- *M2M area network* - This refers to the WSN.
- *M2M Core* - This refers to the IPv6 network and all the key devices inside of it. In Section 3.3 it is referred as the core (Fig 3.1).
- *M2M Gateway* - This is the 6BR, the border between the IPv6 and 6LoWPAN networks.

For each analyzed threat, there are threat agents which impact on the likelihood of the threat. Different threat agents can have the same interest for specific threats. The list of threat agents and their likelihood value identified by the ETSI TC M2M is the following [11, p 10]:

a) Individual criminal: A opportunistic individual with simple profit motive. Likelihood of 1 or low.
b) Hacker: An individual with thrills from intrusion or destruction of property, without a strong agenda. Likelihood of 1 or low.
c) Commercial Competitor: A business adversary who competes for revenues or resources. Likelihood of 2 or moderate.
d) Organized crime syndicate: any type of organized crime organization with a significant amount of resources. Likelihood of 3 or substantial.
e) Extremist and Hacktivist: Highly motivated but non-violent supporter of a cause. It is also possible to find individuals with a potentially destructive supporter of cause. Likelihood of 3 or substantial.
f) Terrorist: A person who relies on the use of violence to support personal socio-political agenda. Likelihood of 3 or substantial.
g) Nation state: State-sponsored attacker with significant resources to affect major disruption on national scale. Likelihood of 4 or severe.

The impact of seriousness, or risk, of each threat are identified as follows [11, p 12]:

a) Minor impact: minor or no effect on the stakeholder, with a resulting inconvenience and very localized. There is not external impact or visibility of any type of problem.
b) Serious impact: there are failures on one or more important processes or systems, they can also affect the revenue of the enterprise, penalty payments, market share and customer confidence.

c) Enterprise: there is an irreparable damage to key systems and/or process with high chance of a widespread impact. The ability of the enterprise to continue operations may be impacted. There are also major regulatory, licensing and legal implications. The impact will be highly visible, probably with a highly severe cash flow problems and a large-scale defection of major customers.
d) National: Equivalent to Enterprise, but on this case the severe damage to the systems and/or process, can have an impact on important infrastructure requirements for the nation or the state.

3.4.1.1 ET8: Discovery of keys by eavesdropping the communications between entities

"*Keys are discovered by eavesdropping on messages at the M2M service layer*" [11, p. 27]. In our proposed scenario, this can happen between the 6BR, the proxies and the sensors; it also includes devices in middle of the communication between entities on the exterior communicating directly with the sensors. Physical scenarios where the eavesdropping may occur are:

- A rogue node in the peripherals of one or more of the wireless sensors.
- A rogue node inside of the LAN with the proxies.
- Outside of our network, but between our resources and the external clients.

As stated by the ETSI TC M2M [11, p. 27]:

> The description here assumes that the network layer, or the network access layer, does not provide any protection against this threat. The attack may therefore exploit the lack of protection in the communications, or the vulnerabilities in protected communications, at the M2M service layer. The attack may discover keys from an examination of communications used in the provisioning of credentials but also may infer keys by examining normal operational communications which use those keys. The method of attack may involve the exploitation of vulnerable algorithms, or the incorrect usage of algorithms, so as to be able to infer the keys used in encrypted communications.

In general, this threat must be considered for all types of scenarios as any node inside of the network can be a victim.

Assessment of Risk

If this threat is exploited, there is not an impact on the resources of the sensors because they are only being monitored. However, it is difficult to detect this monitoring and the recoverability could be highly complex due to the potentially high number of deployed sensors affected.

- Likelihood of a successful attack: 3 ("substantial risk"). It can be higher if no countermeasures are in place.
  - Threat agents score: 3, commercial competitor, hacktivitist, criminal, hacker.
  - Motivation score: 2, probably financial but limited by risk of detection.
  - Opportunity score: 2, probably monitoring of IP communication, but attacking one device at a time.

  - Capability of attackers: 2, the threat agents can have access to different manuals or tools, and/or have knowledge skills.

- Seriousness of the threat: 3 "Enterprise" (if the keys discovered are in large numbers).
- Risk (Seriousness * likelihood): 9 (low end of "critical risk"). Counter measures are required to minimize this risk, with a high priority
- Mitigation is required.

Potential countermeasures

CM9: "*A security association is established between communicating entities, which provides for mutual authentication and confidentiality*" [11, p. 28].

- Advantages
  - Resistance to the attack.
  - It is a well-established countermeasure.
- Disadvantages
  - Provokes an overhead in the communication.

CM10: "*the security association between the communicating entities uses protocols which are proven to resist man-in-the-middle attacks*" [11, p. 28].

- Advantages
  - Resistance to the attack.
  - It is a well-established countermeasure.
- Disadvantages
  - Provokes an overhead in the communication.

CM11: "*M2M service-layer keys in a provisioning message are encrypted for confidentiality, independently of any confidentiality provided by the messaging protocol*" [11, p. 28].

- Advantages
  - Resistance to the attack.
  - It is a well-established countermeasure.
- Disadvantages
  - A significantly overhead with the use of encryption, including the possibility of the use of Public Key Infrastructure (PKI).

CM12: "*during provisioning of M2M service-layer keys, the protocol end-points for the encryption/decryption of those M2M service keys are Secured Environments*" [11, p. 28].

- Advantages
  - Resistance to the attack.
- Disadvantages
  - Increase the complexity and cost.

CM13: "*communications whose security is anchored in M2M service-layer keys use session keys, i.e. keys with a limited lifetime which can be set by security policy. Session keys can be derived from M2M service-layer keys*" [11, p. 29].

- Advantages
  - Resistance to the attack.
  - It is a well-established countermeasure.
- Disadvantages
  - Increase the number of encryption operations.

CM14: "*secured communications use only those cryptographic algorithms which are assessed by cryptography experts as being fit for purpose, e.g. the length and randomness of cryptographic parameters is sufficient to resist a brute-force attack*" [11, p. 29].

- Advantages
  - Resistance to the attack.
  - It is a well-established countermeasure.
  - Easiness for the testing.
- Disadvantages
  - Can lead to restriction for some stakeholders.

CM15: "*industry-accepted recommendations for the use of cryptographic algorithms in secured communications are followed*" [11, p. 29].

- Advantages
  - Resistance to the attack.
  - It is a well-established countermeasure.
  - Easiness for the testing.
- Disadvantages
  - Can lead to restriction for some stakeholders.
  - The recommendations may change over time.
  - It can be a challenge to specify a complete set of recommendations.

3.4.1.2 ET10: Provisioning of non-legitimate keys

This threat is related to the impersonation of servers provisioning the keys or for generating non-usable keys for denying the services to legitimates nodes. In our proposed scenario this can happen inside of the WSN and in the IPv6 network, and it will usually be present by intrusion of one or more rogue nodes.

Assessment of Risk

If this threat is exploited the impact on the service can be major as one or more sensors can be denied the capacity to transmit the data measured. The detection of the exploit can be hard if there are no countermeasures for detecting unauthorized software. This risk can lead to the loss of the privacy of the data measured by the sensors.

- Likelihood of a successful attack: 4 or severe likelihood.
  - Threat agents score: 4 (organized crime syndicate, hacktivitist, nation state).

  - Motivation score: 3, Important objective for the attacker but limited by risk of detection.
  - Opportunity score: 3, Devices reachable over IPv6 networks, key elements behind properly configured firewalls.
  - Capability of attackers: 4, particularly for unprotected systems, the threat agents are assumed to have good amount of acknowledgment of the protocols involved.
- Seriousness of the threat: 3 "Enterprise" (or 4 "National infrastructure" if the WSN compromised is of national interest).
- Risk (Seriousness * likelihood): 12 or 16 (medium or high end of "critical risk"). Depending in the type of owner of the WSN it can be considered. In any case, countermeasures are required to minimize this risk, with a high priority
- Mitigation is required.

Potential countermeasures

The CM9, CM10 and CM11, already defined for the ET8, are mitigation for this threat.

3.4.1.3 ET15: General eavesdropping on M2M Service-Layer messaging between entities.

This threat is related to the discovery of confidential or private information by eavesdropping the messages to the M2M service layer between nodes keys to the infrastructure. This excludes the risk of eavesdropping messages for discovering or inferring the value of keys as are already noted in the ET8.

Attacks for exploiting this threat can be based on the lack of protection in communications, or vulnerabilities in the protected communication, at any layer.

Assessment of Risk

If this threat is exploited the impact can be significant to the reputation of the WSN provider. This risk can lead to the loss of the privacy of the data measured by the sensors. Once the confidentiality is lost, it is hard to re-establish it.

- Likelihood of a successful attack: 4 or severe likelihood.
  - Threat agents score: 2 (individual criminal, hacker, commercial competitor).
  - Motivation score: 2, mainly financial (moderate motivation).
  - Opportunity score: 4, It can focus over concentration points, such as the Gateway, the monitoring could be carried out over long periods of time, without too much difficulty.
  - Capability of attackers: 3, The attackers must have substantial capability.
- Seriousness of the threat: 2 "serious impact".
- Risk (Seriousness * likelihood): 8 (high end of "major risk"). Countermeasures are required to minimize this risk as soon as possible.
- Mitigation is required. With a high priority.

Potential countermeasures

The CM13, already defined for the ET8, can be a mitigation for this threat.

CM25: "*Communications between entities in the M2M system are protected by end-to-end security associations which provide end-to-end confidentiality*' [11, p. 41]'.

3.4.1.4 ET16: Alteration of M2M service-layer messaging between entities

This threat is related to the forgery of messages related to the communication of the sensors. Either because they captured a valid message and modified it or because they generated fakes one. This attack can affect all nodes inside of the WSN and the IPv6 network.

Attacks for exploiting this threat can be based on the lack of protection in communications, or vulnerabilities in the protected communication, at any layer.

Assessment of Risk

If this threat is exploited, it can be of significant loss if it is able to make a wide-scale attack against the sensors or M2M Gateways. It is also a threat against the M2M core. It is hard to detect or prevent if there are not already countermeasures implemented. Finally, the recovery of the original messages is almost impossible.

- Likelihood of a successful attack: 3 or substantial likelihood.
  - Threat agents score: 3, (individual criminal, organized crime syndicate, commercial competitor).
  - Motivation score: 2, primarily financial (moderate motivation).
  - Opportunity score: 3, the nature of WSN made it susceptible to this type of attack, particularly if the sensors use the CoAP Observer communication, where periodically transmits the data measured.
  - Capability of attackers: 3, The attackers must have substantial capability.
- Seriousness of the threat: 3 "enterprise". Because to the nature of the sensors.
- Risk (Seriousness * likelihood): 9 (low point of "critical risk"). Countermeasures are required to minimize this risk, with a high priority.
- Mitigation is required. With a high priority.

Potential countermeasures

The CM9, CM10, CM13 and CM25 already defined for the ET8, are a mitigation for this threat.

3.4.1.5 ET17: Replay of M2M Service-Layer Messaging Between Entities

This threat is related to the replay of valid messages sent in the past to any component inside the WSN and IPv6 network.

Attacks for exploiting this threat can be based on the lack of protection in communications, or vulnerabilities in the protected communication, at any layer.

ETSI Threat Number (ET#)	Likelihood	Impact	Risk	Countermeasures
(ET8) Discover Keys by Eavesdropping on Communications Between Entities.	Substantial	High	Critical	CM9, CM10, CM11, CM12, CM13, CM14, CM15
(ET10) Provisioning of non-Legitimate Keys.	Severe	High	Critical	CM9, CM10, CM11
(ET15) General Eavesdropping on M2M Service-Layer Messaging Between Entities.	Severe	High	Major	CM13, CM25
(ET16) Alteration of M2M Service-Layer Messaging Between Entities.	Substantial	High	Major	CM9, CM10, CM13, CM25
(ET17) Replay of M2M Service-Layer Messaging Between Entities.	Severe	High	Major	CM26

Table 3.2: Countermeasures classification based in threats defined by the ETSI [11].

Assessment of Risk

If this threat is exploited, there can be a significant loss of revenue if it occurs between the exterior and the M2M Core or if it is able to make a wide-scale attack against the sensors or M2M Gateways. It is also a threat against the M2M core. It is hard to detect or prevent it, if there are not already countermeasures implemented. Finally, the recovery of the original messages is almost impossible.

- Likelihood of a successful attack: 3 or substantial likelihood.
  - Threat agents score: 3, (individual criminal, organized crime syndicate, commercial competitor).
  - Motivation score: 2, primarily financial (moderate motivation).
  - Opportunity score: 3, the nature of WSN made it susceptible to this type of attack, particularly if the sensors use the CoAP Observer communication, where periodically transmits the data measured.
  - Capability of attackers: 3, The attackers must have substantial capability.
- Based on previous values, the likelihood of a successful attack: 3 or substantial likelihood.
- Seriousness of the threat: 3 "enterprise". Because to the nature of the sensors.
- Risk (Seriousness * likelihood): 9 (low point of "critical risk"). Countermeasures are required to minimize this risk, with a high priority.
- Mitigation is required. With a high priority.

Potential countermeasures

CM26: "*The protocol includes functionality to detect if all or part of a message is an unauthorised repeat of an earlier message or part of a message.* " [11, p. 44].

- Advantages
  - Resistance to the attack.
  - It is a well-established countermeasure.
- Disadvantages
  - A significantly overhead to the communication and processing of the packets.

3.4.2 Final observations about the countermeasures proposed

The threats for the M2M communication identified as crucial to mitigate are reflected in the Table 3.2 with their level of likelihood, impact, risk and their respective countermeasures.

The CM9 to CM15, CM25 and CM26 are achieved with the correct implementation of CoAPS. The CM11, CM12 and CM13 are achieved by the cipher suites, a key component of DTLS. The CM9 is achieved when CoAPS is fully implemented. The use of cookies, an optional component for DTLS, helps to achieve the CM10. At last, the CM14, CM15, CM25 and CM26 are guaranteed if all the communication between sensors and other devices, (e.g. the 6BR and the clients) is by means of CoAPS. Therefore, the elements that weaken the strength of DTLS must be monitored.

If CoAPS is configured correctly, the CM9 is achieved with an implicit achieving of the CM10 to CM12. If a unique DTLS session is generated each time a client request information from the sensors arrives, the CM13 will be achieved. The CM14 and CM15 are guaranteed when using DTLS with official cipher suites. Thus the threat ET8 got mitigated.

The CM9 helps against the threat described as ET10, but the risk can be mitigated even more if in addition it is validated that the nodes initiating the CoAPS communication are authorized to transmit. For example, it could be that the sensors answer requests from the 6BR or from a specific proxy server. The way to store the keys in the nodes for the PSK mode is also relevant.

As shown in the Table 3.2, mitigating the threat ET8 will mitigate the ET15 and ET16; however the ET16 requires in addition special attention to the cookies; they are intended to resist replay attacks against DTLS. Therefore, an adequate cookie size should be in consideration

For the CM11 and CM13 the monitoring of the cipher suites selected by the client and server after the first three flights are relevant. For example, if the server accepts the `TLS_NULL_WITH_NULL_NULL` cipher suite there will not be any confidentiality at all, yet, it will be a valid DTLS session.

For the CM13, one specific DTLS session is defined by the socket address of the client as well as the combination of the fields `g` and `Sequence_Number` which would be updated by the pass of flights and the `ChangeCipherSpech` message. Therefore, to guarantee the CM13 it is necessary to be able to track the communication between the client and the server, validating the session, and thus the private symmetric keys.

3.5 Alternatives for the implementation of the WSN

The sensors require running over a platform that handles their internal resources, different stacks for reaching communication over the Internet, and running applications such as CoAP in real time. In other words, the sensors require an operating system. The majority of the platforms discussed in Section 2.2 include their own private pseudo operating system. Yet, there are also open source alternatives, some of them created for more conventional embedded systems, yet they are functional, or even adapted for WSN. For the IEEE 802.15.4 standard, the most well-known proprietary operating system is Zigbee from the Zigbee Alliance [97, p. 93]; for the industry, WirelessHART and ISA100.11a are also well-known standards [47, 110].

Table 3.3 has listed four popular operating systems for WSN. There are other OS such as FreeRTOS with Nabto [111] or OpenWSN [112], but they are designed for other specific standards or require additional steps to support 6LoWPAN.

Although many of those operating systems offer support for 6LoWPAN, and even for CoAP,

O.S.	Min. RAM	Min. ROM	C/C++ Support	Multi-threading	MCU/MMU	Modularity	Real-Time
Contiki	<2kB	<30kB	Partial C	Partial	Yes	Partial	Partial
TinyOS	<1kB	<4KB	None	Partial	Yes	No	No
RIOT	≈ 1MB	≈ 1MB	C/C++	Yes	Yes	Yes	Yes
Linux	≈ 1.5kB	≈ 5kB	C/C++	Yes	No	Partial	Partial

Table 3.3: comparison between O.S. available for the IEEE 802.15.4 [12]

an implementation of the DTLS stack is not yet integrated by default. In the work of Bergmann et al. [113], an integration and testing of the open stack TinyDTLS was performed. TinyDTLS is a DTLS stack which supports two of the three cipher suites requested by CoAP specifications. TinyDTLs is written in language C with native support for Linux, and, up to a certain degree, Contiki. TinyDTLS is further discussed in Section 3.5.1.

Of particular interest are Contiki and RIOT. Contiki was used originally for the work of this dissertation, but was eventually dropped by RIOT. The former is further discussed in Section 3.5.2 meanwhile RIOT is discussed in Section 3.5.3. In addition to these two operating systems, the FIT IoT-Lab platform, an academic testbed for WSN, is introduced in Section 3.5.4.

3.5.1 TinyDTLS, an open source stack for the IoT

DTLS is a suitable candidate for securing the M2M communication between the sensors. However, the resources required for establishing a standard DTLS session can be too much for the sensors. A certificate could be more than a single megabyte to transmit and to be processed by the sensors. Even if the `PresSharedKey` mode is used, the size of the keys for the symmetric encryption can still drain the power supply of the nodes rather quickly. This is why the specifications of CoAP take in consideration friendlier cipher suites, as discussed in Section 3.2.4.2 Relevant works related to this issue are Kothmayr et al. [83], Granjal et al. [49, 79] and Raza et al. [82].

A popular DTLS implementation for WSN is TinyDTLS, which is a stack written in language C. The main characteristics of TinyDTLS for the WSN are:

- The use of the cipher suites `TLS_PSK_WITH_AES_128_CCM_8`for the `PreSharedKey` mode. And, the cipher suite `TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8`for the `RawPublicKey` mode.
- Jumbo datagrams are disabled.
- Cookies are configured to have a size of 16 bytes by default.
- A light implementation of the protocol.

At the moment of this dissertation, TinyDTLs is still marked as under development (available version is the 0.8.2). Therefore, it is important to remark that changes from a version to another can lead to a low backward compatibility. This is the case with the work of Raza et al. [82] which is compatible only with TinyDTLS 0.3.0.

It is also required to make the observation that for the next release of TinyDTLS, the V. 0.9.0, the ad-hoc ECC library will be changed by the micro-ECC library. To be considered, is also the fact that the generation of random values used by TinyDTLs is too weak; in addition, the EPOCH value transmitted in the DTLS records related to the handshake process will start at zero (the first of January of 1970). This EPOCH time is used in combination with other 24 random bytes to generate a portion of the master key for the symmetric encryption. Although

157.159.103.95	157.159.103.95	DTLSv1.2	123 Server Hello
157.159.103.95	157.159.103.95	DTLSv1.2	161 Certificate[Malformed Packet]
157.159.103.95	157.159.103.95	DTLSv1.2	210 Server Key Exchange
157.159.103.95	157.159.103.95	DTLSv1.2	75 Certificate Request
157.159.103.95	157.159.103.95	DTLSv1.2	67 Server Hello Done

Figure 3.22: TinyDTLS flight 4 without jumbo datagram or fragmentation (Over ethernet and IPv4).

there is no requirement to use the correct date, if the first session always uses a static value, the resistance of the temporary keys to an attack could be reduced.

Also important is the suppression of the DTLS Finished record from the handshake process discussed in Section 3.2.4.1. It is possible that this is done to reduce the overhead of the communication of the constrained devices, as this DTLS record can be seen as redundant.

In the example of the fourth flight of Figure 3.14, the size of the DTLS header and its payload is near to 886 bytes, once the lower layer protocols are discarded. In comparison, TinyDTLS using `TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8`requires only 362 bytes for the same flight as showed in Figure 3.22. If the PSK cipher suite is used, the size decreases to 172 bytes.

3.5.2 Contiki operating system

Contiki is a very popular operating system for the IoT. It was released in 2003 with its own stack for IPv6 and uIP, focusing on having the minimum possible number of components for IP, TCP, UDP and ICMP. In the version 2.6 of Contiki, the support for IPv6 and 6LoWPAN was already added. Contiki has developed its own MAC protocol for the 802.15.4 [114]; this version also offers support for a large number of platforms, it added support to the Cooja simulator and gives support for at least two stacks for CoAP, erbium and libcoap.

The work with Lithe of Raza et al. [82] included the integration of TinyDTLS to the erbium CoAP library for Contiki 2.6 and for TinyDTLS 3.0. This integration was removed from the official repository, partly because the library got significantly upgraded. Finally, the work of Raza et al. used the Z1 and wismote hardware platforms.

Our works published on WWIC 2015 [24] and APPC 2016 [115] used Contiki as a base. Additionally, an implementation of TinyDTLS 0.8.2 for Contiki 3.0 was developed by the author of this dissertation[3], but it was eventually dropped for RIOT.

The integration of TinyDTLs and Contiki came with many issues. Many of the well-known works were used by Contiki 2.6 and older versions of TinyDTLS. Also, as TinyDTLS is still under development, the jump between version 0.3.6 and 0.8.2 makes them incompatible. Besides adding the `TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8`cipher suite, many structures were modified. The extra cipher suite makes this unsuitable for the Z1 platform.

The default example of TinyDTLs running over Contiki [113] had conflicts with the MEMB functions of Contiki. This provoked memory overflow when handling the DTLS session in Contiki, but only when TinyDTLs was compiled for supporting the modes *PreSharedKey* and *RawPublicKey*. For the tests with only the *PreSharedKey* mode, the performance was too poor as it had too many retransmissions of the DTLS records involved in the DTLS handshake process. There are additional implementations using TinyDTLS over Contiki, such as the

[3]The source code for Contiki 3.0 and TinyDTLS 0.8.2 is available at https://zenodo.org/record/48524

6LBR project[4]. However, even with their own implementation of TinyDTLS the conflict with the memory allocation remains in the example code.

Another minor reason for dropping the work with Contiki is that the testbed platform only supports Contiki 2.7, which makes all the work done with TinyDTLS and CoAPS not backtrack compatible.

3.5.3 RIOT operating system

RIOT is a new operating system released in 2013 [12]. RIOT has less requirements than Contiki. RIOT takes care of the protocols below the Transport layer in a transparent approach to the user. When RIOT is compiled for a specific board, the correspondingly stack will also be compiled. On Contiki, the user has more flexibility, which can lead to unexpected issues, particularly when configuring the IPHC dispatch.

RIOT did not have an official support for TinyDTLS, however *one of the main contributions of this dissertation is the addition of the support of TinyDTLS 0.8.6 to RIOT release 2016.10*[5]. The testing in the native platform, a Linux thread, for the two DTLS modes is successful on RIOT. For real hardware, TinyDTLS over RIOT was tested with the M3 platform with mixed results. The `RawPublicKey` mode has an issue with memory when the transmission is too fast to be handled. Yet, probably this is a limitation of hardware rather than to the implementation of TinyDTLS.

3.5.4 FIT IoT-Lab platform

The FIT IoT-Lab[6]is an *open large-scale testing infrastructure for systems and applications on wireless and sensor communications* [116]. The platform is provided by the FIT consortium, formed by Université Pierre et Marie Curie (UPMC), Inria, Université de Strasbourg, Institut Mines Télécom and CNRS.

The FIT IoT-Lab provides more than one thousand sensors, shared between multiple points in Grenoble, Lille, Paris and Rennes. These sensors are distributed among WSN430, the M3 and the A8 hardware platforms, together with other mobile devices. The access to the sensors is remote and allows users to load specific programs. This includes a sniffer packet and an analyzer packet on each available node in the platform. More information about the different types of sensors is available in Appendix B.

The sensors available in the platform can support the operating systems listed in Table 3.3. However, for Contiki this is only for the version 2.7 as the experiments for compiling Contiki 3.0 on those nodes were unsuccessful. RIOT has a native support for the M3 platform.

3.6 Conclusion

In Section 3.2 all the details of the protocols involved in reaching the M2M communication between the sensors were discussed; particularly, the benefit of the compression of IPv6 and UDP. In Table 3.4 a comparison of unicast message between CoAP entities messages requesting one type of data was made; this considers the following configuration for the sensors: I) sensors are using SLAAC for their IPv6 addresses. II) The sensors are using only a nibble for their

[4]The project 6LBR (with TinyDTLS integrated) is available at https://github.com/cetic/6lbr
[5]https://github.com/RIOT-OS/RIOT/tree/2016.10
[6]FIT IoT-Lab is available at https://fit-equipex.fr/

	WSN	IPv6 over 802.15.4	IPv6 over Ethernet
MTU	127 bytes	127 bytes	1,500 bytes
802.15.4 vs. Ethernet	23 bytes	23 bytes	14 bytes
6LoWPAN IPHC vs. IPv6	2 bytes	40 bytes	40 bytes
NHC vs UDP	1 byte	8 bytes	8 bytes
DTLS	13 bytes	13 bytes	13 bytes
CoAP	8 bytes	8 bytes	8 bytes
Remaining bytes for data	80 bytes	35 bytes	1,417 bytes
Percentage	63%	28%	94%

Table 3.4: Comparison between compression and non-compression.

port addresses. and III)The UDP payload is a DTLS Data Application record with a CoAP CON REQUEST for the URI `./actuators/toggle`. After all the headers, there are 81 bytes remaining of the MTU, this is around 63% of the available space against the 28% without any compression. Before closing the discussion of the involved protocols, it is important to emphasize that the 6LoWPAN is not a WSN architecture by itself. 6LoWPAN was originally designed to support the IEEE 802.15.4 standard, which is the WSN architecture. Yet it can also be integrated with other architectures, such as those discussed in Section 2.2.

In Section 3.2.4.1 was stated that the cipher suites have a major impact over the DTLS protocol; particularly, the standard suites that are available for DTLS have a negative impact for WSN. However, the CoAP specifications have already specified 3 cipher suites for each mode, except the `NoSec` mode. This is a good reason to avoid making ad-hoc modifications to DTLS records which can lead to non-standard implementations.

In Section 3.3, the different components for generating a fully architecture were defined; The key components are: An IPv6/6LoWPAN proxy and a HTTP/CoAP proxy.

An risk analysis for the sensors and the devices in the IPv6 network is discussed in Section 3.4.1. For the M2M communication, a list of threats and their possible countermeasures was produced; the majority of the countermeasures are achieved with the use of DTLS. However, the grade of success will depend directly on the configuration, cipher suites, and options used for DTLS. Therefore, the traffic monitoring becomes crucial to guarantee a secure M2M communication.

In Section 3.5 the RIOT operating system and the FIT IoT-Lab were introduced. The former is the operating system where the implementation of TinyDTLS as our DTLS stack for the 6LoWPAN was developed. FIT IoT-Lab is an open testbed for the IoT used for the elaboration of experiments and results of this dissertation.

Even if not all of the WSN platforms will generate sensitive data, the protection of the application layer must still be considered. Not all the threats to the communication are involved with the data leakage, but also with the risk of having sensors under the control of someone else. That why the use of DTLS should be considered to protect the CoAP messages.

Finally all the key components of the communication, the platform and stacks are defined. The next steps are the implementation of a monitoring tool, and the real-time tests with the testbed.

Chapter 4

Montimage Monitoring Tool for 6LoWPAN

Contents

4.1 Introduction

As already mentioned in Section 2.5, many of the works related to the WSN are focused on making them realistic and practical. Security tends to be a second factor. In part because of the complexity of achieving a viable monitoring over a network composed of thousand of wireless devices in an extensive territory. There are some designs, discussed in Section 2.4, but they are still under development or they have high restrictions. Other solutions focus on routing problems (Foren6, SVELTE), or in detecting the system performance. In the particular case of the WSN based on the IEEE 802.15.4 and 6LoWPAN, some monitoring solutions are based on the translation from 6LoWPAN to standard IPv6. However, this concealed some possible issues present in the same 6LowPAN, different types of dispatches, raw packets, malformed headers or event tunnel traffic hidden in in the dispatches.

To our best knowledge, as of this moment there is not an official monitoring solution to such kind of networks. Most of the propositions concentrate over routing issues and this is not enough

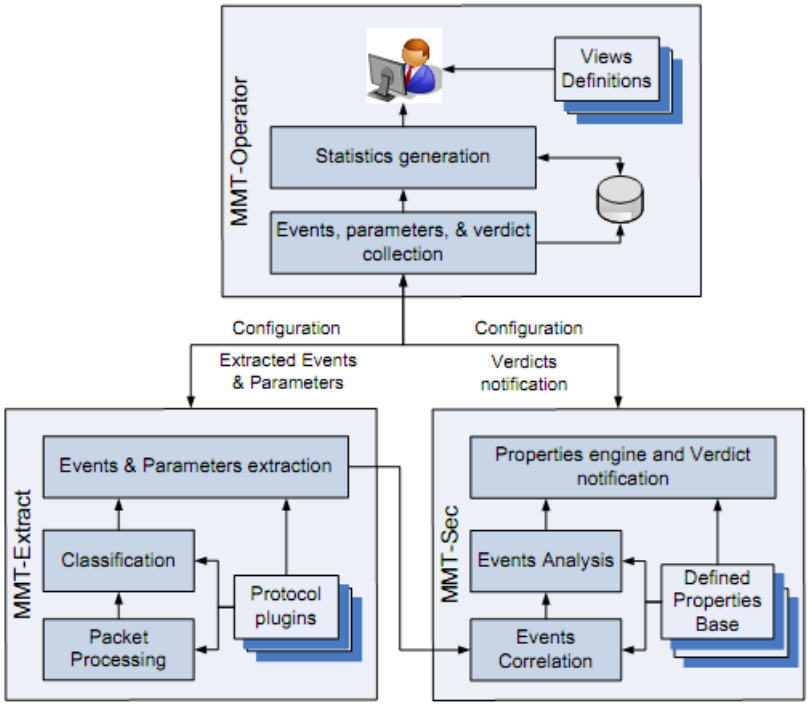


Figure 4.1: MMT global architecture [8].

to allow a Deep Packet Inspection (DPI) on the network traffic. Therefore, our intention is to cover this gap of the monitoring. Our proposition is the use of MMT, which has been utilized with success for TCP/IP networks [8, 117–120]. Our goal is to be able to execute real time monitoring in a passive way over real 802.15.4/6LoWPAN topologies with emphasis on the M2M communication of the sensors to any other nodes, without affecting the normal operation of the WSN. The main contributions of this chapter are: I) the introduction of MMT for 6LoWPAN as our native monitoring tool for WSN based on the architecture 802.15.4/6LoWPAN. And, II) The development of security rules to validate a secure M2M communication over the WSN.

The rest of this chapter is structured as follows: In Section 4.2 the architecture of MMT is presented. Afterwards in Section 4.3 the adaptation of MMT for our WSN is developed and explained. Next to them, in Section 4.4 our proposition for monitoring the correct mitigation of the threats identified in Section 3.4 is presented. Finally, in Section 4.5 final remarks of this adaptation are discussed.

The content of this chapter was strongly used for the elaboration of the work presented [***under review***] in the International Journal of Computers and Applications.

4.2 Architecture of MMT

MMT is a monitoring tool that allows capturing and analyzing network traffic in both on-line and offline manners. MMT supports inspection of network traffic by extracting the necessary attributes and referring to a set of security rules. MMT uses Deep Packet/Flow Inspection (DPI/DFI) techniques and consists of three principal modules, as shown in Figure 4.1:

- **MMT-Extract** is a C library that enables the extraction of network protocol fields of not

only offline structured traffic (e.g., PCAP files) but also real-time on-line network traffic from an interface. It is possible to build a new plug-in for the addition of new protocols and the parsing of proprietary structured data. In practice, this module permits monitoring different applications, systems, or networks. In the case of an application, the input could be a message exchange or an event log.

- **MMT-Security** contains security rules written in XML that refer to both expected and unexpected behaviors. MMT-Security model is inspired from Linear Temporal Logic (LTL). Different rules can be correlated in order to detect security incidents.

- **MMT-Operator** collects and aggregates extracted data, generates network and application statistics, and presents them via a graphical user interface.

MMT facilitates network performance monitoring and operation troubleshooting. With its advanced rules engine, MMT can correlate network and application events in order to detect performance, operational and security incidents. MMT is not based on only pattern matching (i.e., signature-based) as SNORT nor requires writing executable scripts as in BRO [115]. MMT is a flexible solution that can integrate pattern matching, statistics and machine learning [118] techniques depending on the actual problem.

Additionally, MMT has a high adaptability by means of plugins and modules written in C that specify the structure of the new data input (e.g., the fields of one or more new protocols). The plugins operate in the MMT-Extract and they give support to different protocols on any layer of the TCP/IP model.

The use of XML syntax of the security rules has as purpose to provide the advantage of simple and straightforward structure verification. A property is an IF $<$ *context* $>$ THEN $<$ *trigger* $>$ relation. The trigger is verified if and only if the context is valid. If the trigger is found valid, then the property is satisfied. Otherwise, the property is violated. Embedded functions can also be added to pre-process the data input before passing it to the MMT-Security rules.

The three components of the MMT architecture are shown in Figure 4.1. There is a fourth component referred as the user program. In this dissertation it is also denominated as a probe.

4.3 The adaptation to 6LoWPAN

Figure 4.2 illustrates the interaction of a user program and a MMT-Extract. In a nutshell, MMT-Extract uses the "Events Analysis and Parsing" library to process all events, which are captured in each new packet. The events are parsed by using the integrated or added plugins. With the plugin, MMT can classify the event to determine the type of message, extracting the needed data and reporting it back to the user program.

Our probe employs a variety of plugins for each layer identified in Section 3.2. Each one of those plugins is handled by MMT-Extract and helps extracting the values of the fields. These values are transferred to MMT-Security to make an analysis of the security rules.

In the work presented at LANOMS 2015 [121], a unique plugin was developed for working with very specific requirements, where the number of fields to extract in a single message was fixed. Yet, for the WSN the requirements are different, and the complexity is higher. The main issue is that for the majority of the layers, a single header can have a variable size with a variable number of fields.

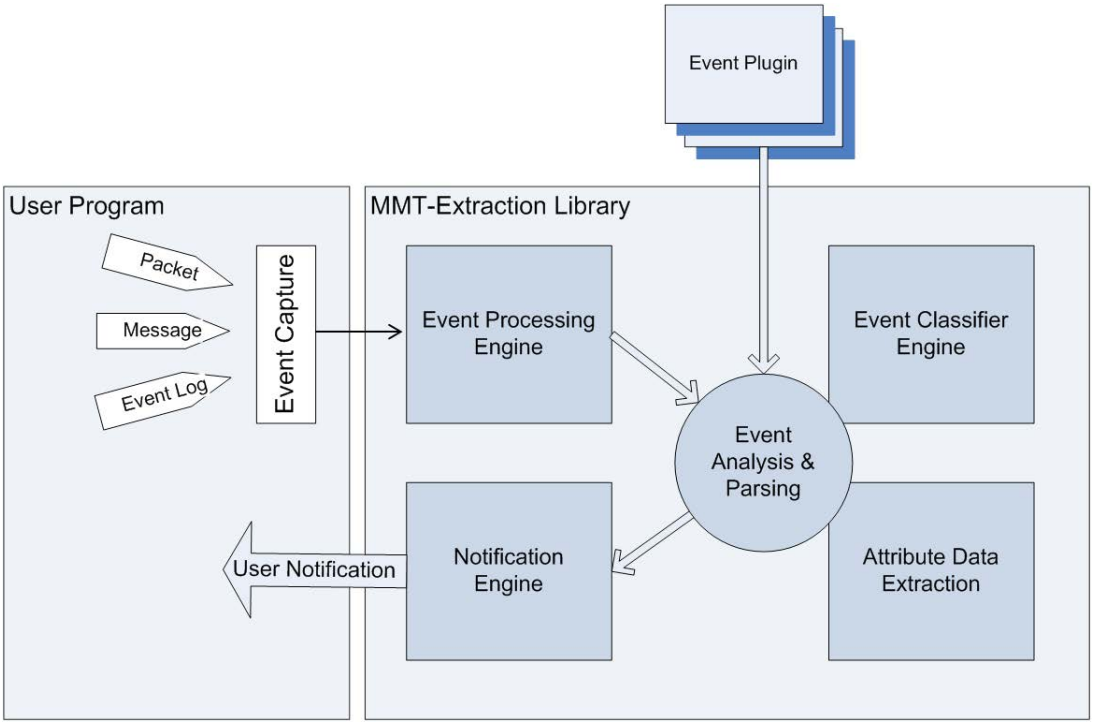


Figure 4.2: MMT-Extract architecture.

The rest of the section is composed with a general description for each plugin that was developed for our WSN. In Section 4.3.1 the development of the frames of the 802.15.4 protocol is discussed. Section 4.3.2 explains the development for capturing all the different types of packets according to the dispatch. In Section 4.3.3 the plugins involving the datagrams are disclosed. Finally, in Section 4.3.4 the processing of the DTLS records is discussed.There Because the payload of upper protocols (such as CoAP and its data) is encrypted there are no more plugins.

4.3.1 Plugin: 802.15.4

The characteristics of the network access layer for the WSN are detailed in Section 3.2.1. There are four different frames to capture: beacon, MAC command, data and acknowledgment. However, the first two are currently ignored by our plugin. The acknowledgment frame is fixed in size. The data frame is of a dynamic size because the destination addresses could be broadcast or unicast or not be present at all.

Figure 3.4 shows the general MAC frame for the four types. The only constant component in all of the 4 frames is the `Frame Control` field. Algorithm 1 shows the approach for our first custom plugin denominated `802.15.4_Frame_Plugin`. The objective is to validate the `Frame Control` field for identifying the possible sizes and then loading the correspondingly plugin.

The structures defined as `802.15.4, 802.15.4_Multicast_Body, 802.15.4_Unicast_Body, and 802.15.4_ACK_Body` are the data to be passed from MMT-Extract to MMT-Security. This is why their sizes are very specific, `uint_16` instead of `int` or another more generic data type. Also, their content is on the host byte order instead of their original network byte order.

The probe is the beginning the process, passing as arguments in the original packet, and user arguments to MMT-Extract. MMT-Extract will load at the first plugin with the previous

Algorithm 1: MMT-Extract plugin IEEE 802.15.4 standard.

```
input : packet, offset, User_Args ;  // Original packet, provided by MMT-Extract
output          : struct {
  | uint8_t SeqNum; uint16_t FCS;
} 802.15.4_ACK_body;
struct {
  | uint16_t SeqNum; uint64_t Destination_Address, Source_Address; uint16_t FCS;
} 802.15.4_Unicast_body;
struct {
  | uint8_t SeqNum; uint16_t Destination_PAN, Destination_Address; uint64_t
  | Source_Address; uint16_t FCS;
} 802.15.4_Multicast_body;
struct {
  | uint8_t Frame_Control; struct 802.15.4_ACK_body, 802.15.4_Unicast_body,
  | 802.15.4_Multicast_body
} 802.15.4;
802.15.4_Plugin    :
1  index ←0;
2  802.15.4→Frame_Control ← MMTExtract_Get_Fields_Plugin(index, offset);
3  switch ExtractFrameControl( packet [index ]) do
4  |  case FrameControl & Dest. Address Mode == Long/64 bits do
5  |  |  MMTExtract_Load_Frame_Unicast_Plugin(packet, offset + FrameControl_Size,
   |  |    UserArg) ;
6  |  end
7  |  case FrameControl & Dest. Address Mode == Long/16 bits: do
8  |  |  MMTExtract_Load_Frame_Multicast_Plugin(packet, offset + FrameControl_Size,
   |  |    UserArg) ;
9  |  end
10 |  case FrameControl & Type == ACK: do
11 |  |  MMTExtract_Load_Frame_Unicast_Plugin(packet, offset + FrameControl_Size,
   |  |    UserArg) ;
12 |  end
13 |  otherwise do
14 |  |  MMTExtract_Finsh_Plugin
15 |  end
16 end
802.15.4_ACK     :
17 index ← MMTExtract_Calculate_Index_Plugin(offset) ;
18 802.15.4_ACK_body→SeqNum ← MMTExtract_Get_Fields_Plugin(index, offset);
19 MMTExtract_Load_Plugin(6LoWPAN_Plugin, offset + SizeFrameACK) ;
802.15.4_Unicast   :
20 index ← MMTExtract_Calculate_Index_Plugin(offset) ;
21 802.15.4_Unicast_body→SeqNum ← MMTExtract_Get_Fields_Plugin(index, offset);
22 MMTExtract_Load_Plugin(6LoWPAN_Plugin, offset + SizeFrameUnicast) ;
802.15.4_Multicast:
23 index ← MMTExtract_Calculate_Index_Plugin(offset) ;
24 802.15.4_Multicast_body→SeqNum ← MMTExtract_Get_Fields_Plugin(index, offset);
25 MMTExtract_Load_Plugin(6LoWPAN_Plugin, offset + SizeFrameMulticast) ;
```

argument, plus the offset, at this time with a value of zero. The user argument is used later for handling the DPI property as discussed in Section 4.3.2.

At line 1 the index is calculated based on the current offset, in this case zero. The purpose of the index is to indicate the position of the next field to extract in the packet. In line 2 the frame control field is loaded into the structure *802.15.4*. In line 3 the plugin check which is the following plugin to load by means of the content of the frame control field.

Once the frame control is extracted, the flags `frame type` and `dest. addressing mode` are used to identify the plugin to load. Because ACK has a fixed size, only the former is used in line 10. In contrast, when a data frame is identified, it needs to be confirmed if the destination address is unicast or multicast (64 bits or 16 bits) as shown in lines 4 and 7.

The three possible plugins to load do the same main operation: populate their structures with the specific fields. And in the case of the two data frame plugins: they load the 6LoWPAN plugin. For the ACK frame, there are no more bytes to extract and thus MMT-Extract finished the plugin.

There is a special field which is the `FCS`. This field is always at the end of the packet and its offset is calculated as `packet.size -2`. Also, when the function `MMTExtract_Finish_Plugin` is called the plugins are stopped. This is used on three scenarios: I) it is an unsupported frame (such as the beacon frame), II) it is a malformed packet, or III) it is something else.

When MMT-Extract passes the data to MMT-Security, there are always two empty structures. However, MMT-Extract security rules are able to operate under such situation. A native plugin probably could remove the unnecessary structures.

4.3.2 Plugin: 6LoWPAN

The plugin for all the dispatches is discussed in Section 3.2.2. The steps taken for this first version are in a beta state and it only focuses on identifying packets that carry UDP datagrams, ICMPv6 packets or that are fragmented. This is because only with IPHC there are more than 100 different sizes for the IPH dispatch and the IPv6 header.

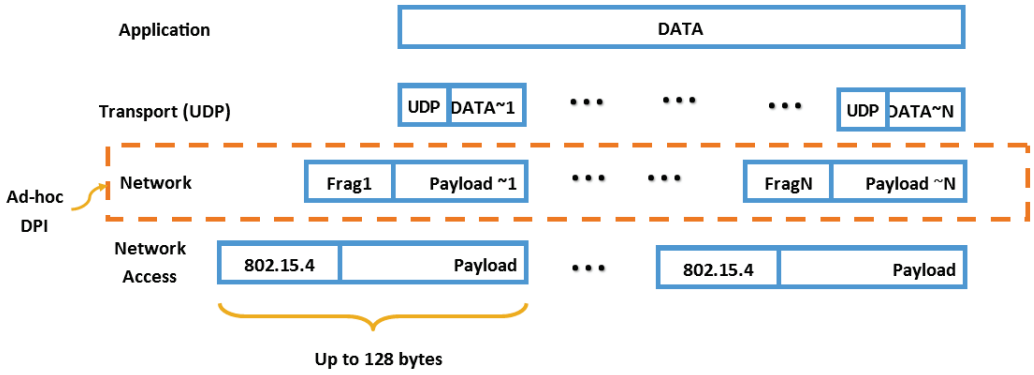


Figure 4.3: Adding DPI to MMT for 6LoWPAN packets.

There is another complexity on this layer. To guarantee a full native DPI support to MMT over WSN, the fragmentation must be handled correctly, as shown in Figure 4.3. This was particularly complex because it was required to integrate a communication from the probe to the plugins for FRAG1 and FRAGN. This integration required also an age mechanism for avoiding exhaustion of memory. The plugins related to the fragmentation are further discussed in Section 4.3.2.1.

Because of all the requirements to perform in this layer, the starting point is very similar to those on the 802.15.4. There is a master plugin which only extracts the first two bytes to try to identify which scenario is handled as shown in line 2 Algorithm 2.

Algorithm 2: MMT-Extract plugin 6LoWPAN, first part.

```
   input : packet, offset, User_Args ;  // Original packet, provided by MMT-Extract
   output                : struct {
   |  uint16_t Dispatch ;
   } 6LoWPAN;
   6LoWPAN_Dispatch  : index
 1 ← MMTExtract_Calculate_Index_Plugin(offset) ;
 2 6LoWPAN→Dispatch ← MMTExtract_Get_Fields_Plugin(index, offset);
 3 if 6LoWPAN→Dispatch & 0xFF00 == Uncompressed_IPv6 then
 4 |  MMTExtract_Load_Plugin(Proto_IPv6, packet, offset- 1, User_Args) ;
 5 else if 6LoWPAN→Dispatch & 0xF800 == FRAG1 then
 6 |  MMTExtract_Load_Plugin(Plugin_FRAG1, packet, offset- 1, User_Args) ;
 7 else if 6LoWPAN→Dispatch & 0xF800 == FRAGN then
 8 |  MMTExtract_Load_Plugin(Plugin_FRAGN, packet, offset- 1, User_Args) ;
 9 else if 6LoWPAN→Dispatch & 0xE000 == Compression_IPHC then
   |  // WARNING: This also can be a raw IPv6 header.
10 |  Check_IPHC_Dispatch packet,offset,User_Args,index;
```

The structure `6LoWPAN` has an unique field of `uint16_t` this because all the dispatches are of dynamic size and because the IPHC dispatch is composed of 16 bits, without the additional context header. The dynamic size of the dispatches is listed in Table 3.1 and it is the cause of the use of the *if-else-if* clauses instead of a `switch` clause.

For the specific case of the dispatch uncompressed IPv6, MMT can make use of its native support for IPv6. This is reflected in line 4 of Algorithm 2.

The potential presence of raw IPv6 packets, discussed in Section 3.2.2, adds another complication. The IPHC dispatch begins with the value of 0x6 with the last bit being part of the field `TF`. A raw IPv6 packet also starts with the nibble 0x6 as shown in line 9 of Algorithm 2. A DPI property to differentiate both is required, yet not supplied with the beta release.

The second part of the beta plugin is the process for identifying the IPHC dispatch as shown in Algorithm 3. This involves the identification of the type of the IPv6 header and the next header. In any implementation of Contiki or RIOT, the upper traffic and the ICMPv6 traffic, mostly the RPL traffic, are handled with a different configuration for the IPHC dispatch.

The first step is to differentiate the upper traffic from ICMPv6, for this beta version, it is assumed that only NHC, UDP compressed, is to be used. This is the reason why line 1 of Algorithm 3 checks if the flag for `NH` is set. Later, in line 2 the function `Confirm_Unicast` is taking the IPHC dispatch and identifying the configuration for the fields SAC, SAM, M, DAC and DAM to confirm if both IPv6 addresses are unicast and fully elided (go to Section 3.2.2.1 for more details). If this is not the case we are before a type of traffic not expected and not supported yet, therefore we finish loading plugins.

After this point, we are breaking the standard mechanism for the MMT plugins as we are moving away from the boundary of this plugin. Line 4 checks the third byte after the offset of this plugin, which is not considered in the structure `6LoWPAN`. This byte corresponds to the

Algorithm 3: MMT-Extract plugin 6LoWPAN, second part.

```
input : packet, offset, User_Args ;  // Original packet, provided by MMT-Extract
output                : struct {
    uint16_t Dispatch ;
} 6LoWPAN;
Check_IPHC_Dispatch:
1  if 6LoWPAN→Dispatch.NH == 1 then
2      if Confirm_Unicast6LoWPAN→Dispatch != 1 then
3          MMTExtract_Finsh_Plugin() ;
4      else if Extract_NHC_ChecksumBit(Packet[index + 2]) == 1 then
5          MMTExtract_Finsh_Plugin() ;
6      else
7          switch Extract_NHC_Port_Flag(Packet[index + 2]) do
8              case SRC_16_DST_16 do
9                  MMTExtract_Load_Plugin(NHC_16_16_Plugin, offset + Size6LoWPAN) ;
10             case SRC_8_DST_16 do
11                 MMTExtract_Load_Plugin(NHC_8_16_Plugin, offset + Size6LoWPAN) ;
12             case SRC_16_DST_8 do
13                 MMTExtract_Load_Plugin(NHC_16_8_Plugin, offset + Size6LoWPAN) ;
14             case SRC_4_DST_4 do
15                 MMTExtract_Load_Plugin(NHC_4_4_Plugin, offset + Size6LoWPAN) ;
16 else
17     switch 6LoWPAN→Dispatch & 0x00FF do
18         case SRC_UNICAST_DST_UNICAST do
19             MMTExtract_Load_Plugin(ICMPv6_Unicast, offset + Size6LoWPAN) ;
20         case SRC_UNICAST_DST_MULTICAST_08 do
21             MMTExtract_Load_Plugin(ICMPv6_M08, offset + Size6LoWPAN) ;
22         case SRC_UNICAST_DST_MULTICAST_16 do
23             MMTExtract_Load_Plugin(ICMPv6_M16, offset + Size6LoWPAN) ;
24         otherwise do
25             MMTExtract_Finsh_Plugin
```

NHC header. The reason for this choice is because NHC is not only used for UDP, but also for other protocols. So, for this beta it is necessary to confirm that only NHC for UDP are being retrieved.

Even if NHC is for UDP there is still a condition where the traffic can be generated from a tunnel, as discussed in Section 3.2.3. Therefore, the line 4 confirms that the `checksukm` field is carried in the NHC header. Finally, the switch on the line 7 is used for loading the following plugin, which is further discussed in Section 4.3.3.

As a closure of this discussion, this plugin gives to MMT-Security the full IPHC dispatch. Although in its current state, the detection of anomalies is relatively low for MMT-Security.

4.3.2.1 Handling the fragmentation

Algorithm 2 also loads the plugins for the dispatches FRAG1 (line 5) and FRAGN (line 7). As already discussed, handling those two plugins properly guarantees the DPI for MMT. Usually this done in three parts:

1. Plugin FRAG1 - Detects a new packet to assemble, avoiding duplicates. Populates the array.

2. Plugin FRAGN - Detects new fragments for already stored packets.

3. The probe - Age mechanisms is executed to handle the array of packets. Also, it detects when a packet is fully received for assembling it and passing it to MMT-Extract.

A complexity is added as the size calculated for the field `Datagram size` for FRAG1 and FRAGN is based on an original brute IPv6 header and its payload, instead of the compressed packet transmitted. As an example, a DTLS ClientHello message with the cipher suite `TLS_PSK_WITH_AES_128_CCM_8`has 172 bytes, this is superior to the 102 bytes remaining after IPHC and NHC and it requires to be fragmented. The message fragmented contains the IPHC and NHC headers, and thus the final size is 175 bytes, yet, the value stored in `Datagram size` is 220, which represents the difference with the IPv6 and UDP headers. Therefore our plugins must be able to calculate the correct offset when assembling the original 6LoWPAN packet.

An alternative to the conflict of assembling the packet, is to translate directly the message to IPv6, and by doing this the offset will be calculated correctly. However, the content of the IPHC and NHC is lost to MMT-Security; As a consequence of this a minor conflict will come up among the open source implementations of 6LoWPAN: Some of them, such as Contiki 2.4, do not calculate the raw IPv6 size, but instead the size of the already compressed packet. If all the sensors operate with the same version, this is not a problem at all. However, if the network has a diversity of nodes with different versions, there could be unexpected errors in the transmission of the packets. To our best knowledge, the behavior of Contiki 2.4 was passed to early adaptations of 6LowPAN on the Linux kernel, but it is already fixed. Contiki 2.7 and superior are already free of this behavior.

Algorithm 4 illustrates the interaction of the probe with the plugins. The probe is the one that initializes the structure array with a fixed size and the age mechanism. The age is based on 60 seconds as in the original 6LoWPAN specifications [10, p. 12]. This algorithm only shows the part of the probe where each packet captured and passed to MMT is processed as shown in the line 16.

Lines 6 and 15 handle the special structures required for this plugin. `Fragment_block` is a structure for storing each fragment. The offset field is expected to be the original value from the

Algorithm 4: The fragmentation handled by the probe.

```
   input : packet
   Structures :
 1 struct {
 2 |   timeval timestamp ;
 3 |   uint8_t offset ;
 4 |   uint32_t len ;
 5 |   uint8_t data[];
 6 } fragment_block;
 7 struct {
 8 |   uint8_t assembling ;
 9 |   uint16_t datagram_size ;
10 |   uint16_t datagram_tag ;
11 |   uint64_t mac_src ;
12 |   uint64_t mac_dst ;
13 |   uint8_t type_fragment ;
14 |   fragment_block Blocks[] ;
15 } Fragments;
   Proccesing packets :
16 while Non-empty packet do
17 |   User_Args ← Fragments ;
18 |   packet_process( mmt_handler, User_Args, packet ) ;
19 |   timeout_clean_registry(Fragments) ;
20 |   PacketID = check_status_asm_packet(Fragments) ;
21 |   if (PacketID has valid ID then
22 |   |   packet ← generate_packet(Fragments[PacketID]);
23 |   |   clean_registry(Fragments[PacketID]) ;
24 |   else
25 |   |   packet ← pcap_next();
   check_status_asm_packet:
26 for z ← 1 2 3 ... Fragments.size do
27 |   Sum ← Sum_Packet(Fragments[z]) switch Fragments[z]→type_fragment do
28 |   |   case TYPE_FRAGMENT_IPHC_NHC_16_16: do
29 |   |   |   Sum← Sum + OFFSET_NHC_16_16
30 |   |   case TYPE_FRAGMENT_IPHC_NHC_8_16:
   |   |    TYPE_FRAGMENT_IPHC_NHC_16_8: do
31 |   |   |   Sum← Sum + OFFSET_NHC_8_16
32 |   |   case TYPE_FRAGMENT_IPHC_NHC_4_4: do
33 |   |   |   Sum← Sum + OFFSET_NHC_4_4
34 |   if Sum >Fragments[z]→datagram_size then return z;
35 |   else return −1;
```

FRAGN `datagram_offset` and the timestamp is used for the age mechanism. The `Fragments` structure has all the elements required to identify each packet to assemble what is captured in the WSN. Line 8 is used for identifying the available space for storing a new packet. And, line 13 is used to help the probe to determine the correct offset for assembling the packet.

The state of `Fragments` is checked each time a packet has been processed by MMT in line 20. Although a timer could be more resource friendly, doing this allows MMT-Security to process the packet as soon as the last fragment is delivered. If there is a new packet ready to be assembled the position in the array (`PacketID`) is returned and the packet is assembled in line 22. Otherwise, the following packet captured by the sniffer application (pcap) is processed in line 25.

The age mechanism happens at the 19 and 23. In the former, `timeout_clean_registry()` is used to search for a timestamp older than a minute and to delete its full entry from the array. With `clean_registry()` the last assembled packet is erased from the array. In this first beta release, `timeout_clean_registry()` is called constantly but in future adaptations it will be possible to call it on only specific times.

Everytime `check_status_asm_packet` is called, it is required to verify each possible packet to be assembled. This is done by the sum of all the blocks. Yet, this sum will always be lesser than the brute value in the datagram_size field. To compensate this problem, it is necessary to identify the type of compressed packet for adding an offset to the sum. Once the sum is done, its value will be compared against the datagram_size in line 34. For this first beta, the offset is for NHC and ICMPv6. Additionally, any brute IPv6 packet will be also detected by this mechanism.

Line 17 in Algorithm 4 loads the structure `Fragments` to MMT-EXtract plugins, but it is used only by the FRAG1 and FRAGN plugins. Those plugins still have the same behavior than the previous ones used to extract the fields of the packet for MMT-Security but they also populate the structure to assemble a new packet.

Algorithm 5 is the general architecture of FRAG1. The first two lines are the standard plugin while the followings are for population `Fragments`. Lines 4 and 5 are making inspection in the Packet at the level of the MAC layer protocol for subtracting the source and destiny MAC addresses from the packet.

Line 6 guarantees the detection of duplicated FRAG1 messages. On WSN it is not unexpected to find re-transmission of packets. Meanwhile, line 9 tries to allocate a space in the `Fragments` array. If no space is available the plugin will finish correctly, but it will lose this packet, and probably further ones until the age mechanism has an opportunity to recover some space in the array.

Line 14 explores the payload of FRAG1 to try to identify the type of content, which can be a unicast IPHC with NHC (in any of its four variants), IPHC with ICMPv6, or a brute IPv6 header. Finally, in line 15 the payload is being allocated in the first available space in the array `blocks`.

Algorithm 6 is the general architecture for the FRAGN plugin. It is highly similar to its counterpart FRAG1. The first difference is in line 6 where the search for a previous populated entry for this specific packet is done. In an ideal environment, where all the packets are captured this should always be true, however, it is possible that the original FRAG1 was lost, or even that this FRAG1 is a duplicated one. Therefore, ending the plugin at this point is a normal operation.

Line 9 does the most critical part, storing the payload of the FRAG1 in an available space of the `Fragments` structure. If there is not enough space for this, an alert must be triggered.

Algorithm 5: MMT-Extract FRAG1 plugin.

```
input  : packet, offset, User_Args
output                  : struct {
    uint16_t datagram_size ;
    uint16_t datagram_tag ;
    uint8_t payload ;
} FRAG1_Dispatch;
FRAG1_Dispatch      : index
1  ← MMTExtract_Calculate_Index_Plugin(offset) ;
2  FRAG1→Dispatch ← MMTExtract_Get_Fields_Plugin(index, offset);
3  Fragments←User_Args ;
4  mac_src← Extract_MAC_src_From_Packet(packet, offset);
5  mac_dst← Extract_MAC_dst_From_Packet(packet, offset);
6  if Entry_Duplicated(datagram_tag,mac_src,mac_dst ) then
7      MMTExtract_Finsh_Plugin();
8  NewID←Avaliable_Entry_Fragmets(Fragments[]);
9  if NewID is a valid range for the array then
10     Fragments[NewID]→datagram_size← FRAG1_Dispatch→datagram_size;
11     Fragments[NewID]→datagram_tag← FRAG1_Dispatch→datagram_tag;
12     Fragments[NewID]→mac_src ← mac_src;
13     Fragments[NewID]→mac_dst ← mac_dst;
14     Fragments[NewID]→Type_Fragment ← InspectPayload(FRAG1_Dispatch→Payload);
15     Fragments[NewID]→Blocks[0]← FRAG1_Dispatch→Payload;
16 else
17     Trigger_Alert("Not enough space in the array");
18 MMTExtract_Finsh_Plugin() ;
```

Algorithm 6: MMT-Extract FRAGN plugin.

```
   input  : packet, offset, User_Args
   output           : struct {
   |   uint16_t datagram_size ;
   |   uint16_t datagram_tag ;
   |   uint8_t datagram_offset ;
   |   uint8_t payload ;
   } FRAGN_Dispatch;
   FRAGN_Dispatch: index
 1 ← MMTExtract_Calculate_Index_Plugin(offset) ;
 2 FRAGN→Dispatch ← MMTExtract_Get_Fields_Plugin(index, offset);
 3 Fragments←User_Args ;
 4 mac_src← Extract_MAC_src_From_Packet(packet, offset);
 5 mac_dst← Extract_MAC_dst_From_Packet(packet, offset);
 6 for z ← 1 2 3 ... Fragments.size do
 7 |   if Match_Fragment(Fragments[z], mac_src, mac_dst,FRAGN_Dispatch←datagram_tag)
   |    then
 8 |   |   OffsetID ← Detect_Duplicate(Fragments→blocks.size,
   |   |    FRAGN_Dispatch→datagram_offset) ;
 9 |   |   if OffsetID is a valid range for the array then
10 |   |   |   Fragments[z]→Blocks[OffsetID]← FRAGN_Dispatch→Payload;
11 |   |   else if OffsetID provokes overflow then
12 |   |   |   Trigger_Alert("Not enough space in the array");
13 MMTExtract_Finsh_Plugin() ;
```

While the plugin can still operate without conflicts, it is highly probable that a packet has been lost and will remain in the structure until the age mechanism removes it.

There is one last observation for this first approach to the fragmentation. It is assumed that the sensors are not re-sending fragmented packets with different values, such as offset and content of the payload. In the protocol references [10, p. 10], this can happen and the sensors are forced to substitute the oldest fragment with the new data. However, this scenario is not typical.

As a small summary to close the discussion of these couple of plugins, MMT-Extract is receiving the full packets, and therefore it is possible to try to inspect the content. It is also possible to determine if a rogue node is sending poisoned fragmented packets.

4.3.3 Plugin: NHC

Because of the approach used in the 6LowPAN dispatch the plugins for the transport layer, more specifically, UDP using the NHC dispatch, are straightforward. Each field is well-known and the only reason there is more than one single plugin is because of the possible sizes for the port addresses. These are the four plugins:

- `NHC_FF_Plugin`: Both port addresses are using the 16 bits.
- `NHC_8F_Plugin`: Source port is using 16 bits meanwhile the destiny port is using 8 bits.
- `NHC_F8_Plugin`: Source port is using 8 bits meanwhile the destiny port is using 16 bits.
- `NHC_44_Plugin`: Both port addresses are compressed to 4 bits.

The general architecture of the four plugins is shown in Algorithm 7. Lines 4 and 6 are the special cases for the plugin where both port addresses are compressed to four bites; this is because they require to be stored in `uint8_t` and the offset must be the same for both of them. Line 8 is used to recover the correct host order type.

The main advantage of the plugins for MMT-Security is the ability to detect the right use of NHC (e.g. it would be known if the nodes are compressing or not the source address). In the current state, these plugins are only loaded for DTLS (line 3), however, it is required to grant the same flexibility as the standard UDP plugin for detecting other application protocols than DTLS.

4.3.4 Plugin: DTLS

There are fifteen plugins for DTLS due to the many types of DTLS records. As discussed in Section 3.2.4.1 some of the DTLS records are highly dynamic in size and number of fields and others are very straightforward. A master plugin is developed for capturing the first fields (`Type`, `Version`, `Epoch`, `Sequence Number` and `Length`) is used for loading the correct plugin as showed in line 3 of Algorithm 8.

The plugins for the DTLS records alerts, application data and ChangeCipherSpec are easy to develop; this is in part because all of them are opaque to any monitoring tool as their content is already encrypted. Their structures and a general representation are shown in Algorithm 9. Conversely, the type of handshake requires to be explored further in the DTLS record to identify the specific DTL handshake record for loading the correct plugin as shown in line 14 of Algorithm 8.

Algorithm 7: MMT-Extract NHC plugins (four cases).

```
input  : packet, offset, User_Args
output                                  : struct {
    uint8_t ID_CP ;
    uint16_t or uint8_t source ;
    uint16_t or uint8_t destiny ;
    uint8_t UDP_Chksum ;
} NHC_XX_Plugin;
NHC_XX_Plugin                           :
1 index ← MMTExtract_Calculate_Index_Plugin(offset) ;
2 NHC_XX_Plugin ← MMTExtract_Get_Fields_Plugin(index, offset);
3 MMTExtract_Load_Plugin(DTLS_Plugin, offset + SizeNHC_XX_XX) ;
  NHC_44_Offset_Source_Extraction : N
4 HC_XX_Plugin→source ← MMTExtract_Get_Fields_Plugin(index, offset);
5 NHC_XX_Plugin→source ← NHC_XX_Plugin→source && 0x0F ;
  NHC_44_Offset_Destiny_Extraction: N
6 HC_XX_Plugin→source ← MMTExtract_Get_Fields_Plugin(index, offset-1);
7 NHC_XX_Plugin→source ← NHC_XX_Plugin→destiny && 0xF0 ;
8 Rotate_4_Bits_Right(NHC_XX_Plugin→Destiny) ;
```

Lines 13 and 26 of Algorithm 8 indicate the ending of the master plugin as this is not able to detect a proper DTLS record; this can happen when the UDP payload under analysis is not a DTLS record at all. For future works, the plugins must be able to support other upper layer protocols.

The structure of the DTLS record is shown in Algorithm 9, although it is valid from Algorithm 10 to Algorithm 14. Also, all the algorithms are leaving out the DPI property for reassembling DTLS records and separating DTLS jumbo datagram. This is in part because TinyDTLS does not implement them (see Section 3.5.1 for more details).

The DTLS records for the Hello Client handshake message are shown in Algorithm 10 which is one of the most complex to extract correctly. One of the reasons for this difficulty is because the fields `length`, `fragment_offset` and `fragment_length` have an irregular size of 24 bits; this can be solved using arrays of 8 bits, but that would cause a bigger complexity for MMT-Security. Another alternative is parsing them to the next valid data type of unit32_t, but getting a higher complexity for extracting correctly the following fields due to the extra byte in the offset. The second was the choice for the plugin and is reflected in the lines 3, 5, and 6 of Algorithm 10.

The remaining complexities are the cookies, the compression methods and the extensions. All of them have dynamic sizes; therefore, calculating the correct offset for those fields, and the offset for the next field, requires an extra analysis over the packet. This is reflected in the lines 10, 12, 14 and 16 (Algorithm 10) with the use of the function `Calculate_Extraction_Size` which updates the index and the offset.

The DTLS record of the `Server Hello` handshake message is highly similar as its client counterpart, though its `cipher suite` field is fixed in size. This record is shown in the Algorithm 11.

The plugins for the DTLS Hello Verify Request record, and those related to certificates are the following in complexity. The former because made use of cookies which have a dynamic

Algorithm 8: MMT-Extract DTLS master plugin.

```
   input : packet, offset, User_Args
   DTLS_Plugin :
 1 index ← MMTExtract_Calculate_Index_Plugin(offset) ;
 2 switch DTLS_Type ← Packet[offset ];
 3  do
 4     case Handshake do
 5        DTLS_Handshake_Type_Extractionpacket, index, offset
 6     case Alert do
 7        MMTExtract_Load_Plugin(DTLS_Alert_Plugin, offset)
 8     case Application data do
 9        MMTExtract_Load_Plugin(DTLS_App_Data_Plugin, offset)
10     case ChangeCipherSpec do
11        MMTExtract_Load_Plugin(DTLS_ChangeCipherSpec_Plugin, offset)
12     otherwise do
13        MMTExtract_Finsh_Plugin()
   DTLS_Handshake_Type_Extraction :
14 DTLS_Hshk_T ← Packet[offset + 4];
15 if DTLS_Hshk_T == Hello Request then
    MMTExtract_Load_Plugin(DTLS_Hello_Request_Plugin, offset);
16 else if DTLS_Hshk_T == Hello Verify Request then
    MMTExtract_Load_Plugin(DTLS_Hello_Verify_Request_Plugin, offset);
17 else if DTLS_Hshk_T == Client Hello then
    MMTExtract_Load_Plugin(DTLS_Client_Hello_Plugin, offset);
18 else if DTLS_Hshk_T == Server Hello then
    MMTExtract_Load_Plugin(DTLS_Server_Hello_Plugin, offset);
19 else if DTLS_Hshk_T == Server Hello Done then
    MMTExtract_Load_Plugin(DTLS_Server_Hello_Done_Plugin, offset);
20 else if DTLS_Hshk_T == Certificate then
    MMTExtract_Load_Plugin(DTLS_Certificate_Plugin, offset);
21 else if DTLS_Hshk_T == Certificate Request then
    MMTExtract_Load_Plugin(DTLS_Certificate_Request_Plugin, offset);
22 else if DTLS_Hshk_T == Certificate Verify then
    MMTExtract_Load_Plugin(DTLS_Certificate_Verify_Plugin, offset);
23 else if DTLS_Hshk_T == Client Key Exchange then
    MMTExtract_Load_Plugin(DTLS_Client_Key_Exchange_Plugin, offset);
24 else if DTLS_Hshk_T == Server Key Exchange then
    MMTExtract_Load_Plugin(DTLS_Server_Key_Exchange_Plugin, offset);
25 else if DTLS_Hshk_T == Finished then
    MMTExtract_Load_Plugin(DTLS_Finished_Plugin, offset);
26 else MMTExtract_Finsh_Plugin();
```

Algorithm 9: MMT-Extract DTLS Alert, App. Data and ChangeCipherSpec plugins.

```
input  : packet, offset, User_Args
output                                  : struct {
    uint8_t Content_Type ;
    uint16_t Version, Epoch ;
    uint48_t sequence_number ;
    uint16_t length ;
    DTLS_* body ;                  // The body of each DTLS record (Alert/App.
    Data/ChangeCipherSpec).
} DTLS_Record;
struct {
    uint8_t * data ;
} DTLS_application_data;
struct {
    uint8_t * data ;
} DTLS_changecipherspec;
struct {
    uint8_t * data ;
} DTLS_alert;
DTLS_Plugin                             :
1 index ← MMTExtract_Calculate_Index_Plugin(offset) ;
2 DTLS_XXXX_Plugin ← MMTExtract_Get_Fields_Plugin(index, offset);
3 MMTExtract_Finsh_Plugin
```

Algorithm 10: MMT-Extract DTLS Handshake plugins (Client Hello).

```
input  : packet, offset, User_Args
output                          : struct {
    uint8_t Handshake_Type ;
    uint32_t length ;
    uint16_t msg_seg ;
    uint32_t fragment_offset, fragment_length ;
    uint16_t Version ;
    uint32_t random_epoch_time ;
    uint8_t session_id_length ;
    uint8_t cookie[];
    uint8_t ciphersuite[];
    uint8_t compression[] ;
    uint8_t extension[] ;
} DTLS_Hshk_Client_Hello;
DTLS_Client_Hello_Plugin:
1  index ← MMTExtract_Calculate_Index_Plugin(offset) ;
2  DTLS_Hshk_Client_Hello→Handshake_Type←Extract_Field(index, offset);
3  DTLS_Hshk_Client_Hello→length←Extract_Field(index, offset-1);
4  DTLS_Hshk_Client_Hello→msg_seg←Extract_Field(index, offset);
5  DTLS_Hshk_Client_Hello→fragment_offset←Extract_Field(index, offset-1);
6  DTLS_Hshk_Client_Hello→fragment_length←Extract_Field(index, offset-1);
7  DTLS_Hshk_Client_Hello→version←Extract_Field(index, offset);
8  DTLS_Hshk_Client_Hello→random_epoch_time←Extract_Field(index, offset);
9  DTLS_Hshk_Client_Hello→session_id_length←Extract_Field(index, offset);
10 Size←Calculate_Extraction_Size(packet,index);
11 DTLS_Hshk_Client_Hello→cookie←Extract_Field(index, offset-Size);
12 Size←Calculate_Extraction_Size(packet,index);
13 DTLS_Hshk_Client_Hello→ciphersuite←Extract_Field(index, offset-Size);
14 Size←Calculate_Extraction_Size(packet,index);
15 DTLS_Hshk_Client_Hello→compression←Extract_Field(index, offset-Size);
16 Size←Calculate_Extraction_Size(packet,index);
17 DTLS_Hshk_Client_Hello→Extension←Extract_Field(index, offset-Size);
18 MMTExtract_Finsh_Plugin
```

Algorithm 11: MMT-Extract DTLS Handshake plugins (Server Hello).

```
input : packet, offset, User_Args
output :
struct {
    uint8_t Handshake_Type ;
    uint32_t length ;
    uint16_t msg_seg ;
    uint32_t fragment_offset, fragment_length ;
    uint16_t Version ;
    uint32_t random_epoch_time ;
    uint8_t session_id_length ;
    uint16_t ciphersuite ;
    uint8_t compression[] ;
    uint8_t extension[] ;
} DTLS_Hshk_Server_Hello;
DTLS_Server_Hello_Plugin:
1  index ←MMTExtract_Calculate_Index_Plugin(offset) ;
2  DTLS_Hshk_Server_Hello→Handshake_Type←Extract_Field(index, offset);
3  DTLS_Hshk_Server_Hello→length←Extract_Field(index, offset-1);
4  DTLS_Hshk_Server_Hello→msg_seg←Extract_Field(index, offset);
5  DTLS_Hshk_Server_Hello→fragment_offset←Extract_Field(index, offset-1);
6  DTLS_Hshk_Server_Hello→fragment_length←Extract_Field(index, offset-1);
7  DTLS_Hshk_Server_Hello→version←Extract_Field(index, offset);
8  DTLS_Hshk_Server_Hello→random_epoch_time←Extract_Field(index, offset);
9  DTLS_Hshk_Server_Hello→session_id_length←Extract_Field(index, offset);
10 DTLS_Hshk_Server_Hello→ciphersuite←Extract_Field(index, offset);
11 Size←Calculate_Extraction_Size(packet,index);
12 DTLS_Hshk_Server_Hello→compression←Extract_Field(index, offset-Size);
13 Size←Calculate_Extraction_Size(packet,index);
14 DTLS_Hshk_Server_Hello→Extension←Extract_Field(index, offset-Size);
15 MMTExtract_Finsh_Plugin
```

Algorithm 12: MMT-Extract DTLS Handshake plugins (Hello Done, Hello Request, Hello Verify Request).

```
input : packet, offset, User_Args
output :
struct {
    uint8_t Handshake_Type ;
    uint32_t length ;
    uint16_t msg_seg ;
    uint32_t fragment_offset, fragment_length ;
} DTLS_Hshk_Hello_Request_Record;
struct {
    uint8_t Handshake_Type ;
    uint32_t length ;
    uint16_t msg_seg ;
    uint32_t fragment_offset, fragment_length ;
} DTLS_Hshk_Hello_Done_Record;
struct {
    uint8_t Handshake_Type ;
    uint32_t length ;
    uint16_t msg_seg ;
    uint32_t fragment_offset, fragment_length ;
    uint16_t Version ;
    uint8_t cookie[];
} DTLS_Hshk_Hello_Verify_Request_Record;
DTLS_Hello_Request_Plugin : DTLS_Hello_Done_Plugin :
DTLS_Hello_Verify_Request_Plugin:
1  index ←MMTExtract_Calculate_Index_Plugin(offset) ;
2  DTLS_Hshk_Hello_XXXX→Handshake_Type←Extract_Field(index, offset);
3  DTLS_Hshk_Hello_XXXX→length←Extract_Field(index, offset-1);
4  DTLS_Hshk_Hello_XXXX→msg_seg←Extract_Field(index, offset);
5  DTLS_Hshk_Hello_XXXX→fragment_offset←Extract_Field(index, offset-1);
6  DTLS_Hshk_Hello_XXXX→fragment_length←Extract_Field(index, offset-1);
7  DTLS_Hshk_Hello_Verify_Request→version←Extract_Field(index, offset);
8  Size←Calculate_Extraction_Size(packet,index);
9  DTLS_Hshk_Hello_Verify_Request→cookie←Extract_Field(index, offset-Size);
10 MMTExtract_Finsh_Plugin ;
```

size. And the rest because made extensive use of extensions. Followed by them is the DTLS Finished record with the content already hidden to us. The details for `HelloVerifyRequest` plugin are shown in Algorithm 12.

Algorithm 13: MMT-Extract DTLS Handshake plugins (Certificate, CertificateRequest, CertificateVerify).

```
input : packet, offset, User_Args
output :
struct {
    uint8_t Handshake_Type ;
    uint32_t length ;
    uint16_t msg_seg ;
    uint32_t fragment_offset, fragment_length ;
    uint8_t extension[];
} DTLS_Hshk_Certificate_Record;
struct {
    uint8_t Handshake_Type ;
    uint32_t length ;
    uint16_t msg_seg ;
    uint32_t fragment_offset, fragment_length ;
    uint8_t extension[];
} DTLS_Hshk_Certificate_Verify_Record;
struct {
    uint8_t Handshake_Type ;
    uint32_t length ;
    uint16_t msg_seg ;
    uint32_t fragment_offset, fragment_length ;
    uint8_t extension[];
} DTLS_Hshk_Certificate_Request_Record;
DTLS_Certificate_Plugin : DTLS_Certificate_Verify_Plugin :
DTLS_Certificate_Request_Plugin:
1 index ←MMTExtract_Calculate_Index_Plugin(offset) ;
2 DTLS_Hshk_Certificate_XXXX→Handshake_Type←Extract_Field(index, offset);
3 DTLS_Hshk_Certificate_XXXX→length←Extract_Field(index, offset-1);
4 DTLS_Hshk_Certificate_XXXX→msg_seg←Extract_Field(index, offset);
5 DTLS_Hshk_Certificate_XXXX→fragment_offset←Extract_Field(index, offset-1);
6 DTLS_Hshk_Certificate_XXXX→fragment_length←Extract_Field(index, offset-1);
7 Size←Calculate_Extraction_Size(packet,index);
8 DTLS_Hshk_Certificate_XXXX→Extension←Extract_Field(index, offset-Size);
9 MMTExtract_Finsh_Plugin ;
```

The plugins related to the certificates are `Certificate`, `CertificateRequest` and `CertificateVerify` are shown in Algorithm 13. All of them make use of extensions and are strongly linked to the cipher suites selected. Therefore, any deep inspection is better to be done by MMT-Security. This is also true for plugins `ServerKeyExchange`, `ClientKeyExchange` that are shown in Algorithm 14.

Algorithm 14: MMT-Extract DTLS Handshake plugins (ServerKeyExchange, ClientKeyExchange).

```
input : packet, offset, User_Args
output :
struct {
    uint8_t Handshake_Type ;
    uint32_t length ;
    uint16_t msg_seg ;
    uint32_t fragment_offset, fragment_length ;
    uint8_t extension[];
} DTLS_Hshk_Server_Key_Exchange;
struct {
    uint8_t Handshake_Type ;
    uint32_t length ;
    uint16_t msg_seg ;
    uint32_t fragment_offset, fragment_length ;
    uint8_t extension[];
} DTLS_Hshk_Client_Key_Exchange;
DTLS_Server_Key_Exchange_Plugin : ;
DTLS_Client_Key_Exchange_Verify_Plugin:
1 index ←MMTExtract_Calculate_Index_Plugin(offset) ;
2 DTLS_Hshk_XXXX_Key_Exchange→Handshake_Type←Extract_Field(index, offset);
3 DTLS_Hshk_XXXX_Key_Exchange→length←Extract_Field(index, offset-1);
4 DTLS_Hshk_XXXX_Key_Exchange→msg_seg←Extract_Field(index, offset);
5 DTLS_Hshk_XXXX_Key_Exchange→fragment_offset←Extract_Field(index, offset-1);
6 DTLS_Hshk_XXXX_Key_Exchange→fragment_length←Extract_Field(index, offset-1);
7 Size←Calculate_Extraction_Size(packet,index);
8 DTLS_Hshk_XXXX_Key_Exchange→Extension←Extract_Field(index, offset-Size);
9 MMTExtract_Finsh_Plugin ;
```

Extensions are currently being extracted together as a single property. MMT-Security requires to extract each one of them. This is because they are very flexible in type, order and size. Thus, detecting the end of an extension and the beginning of another requires to be moving inside the previous part of the DTLS record. This becomes even more complex if the jumbo datagrams are considered. Therefore, once the compression is extracted, all the remaining bytes of the DTLS record are extracted as part of the extensions.

Finally, the `Server Hello Done`, and `Hello Request` are the most simple plugins as they are not expected to have a body. Although the implementation of TinyDTLS does a wrongly addition to the body, it is still ignored by our plugin. Both of them are shown in Algorithm 12.

4.4 Security rules for monitoring the M2M communication

MMT-Security analyses and correlates network and application events to detect operational and security incidents. For each occurrence of a security property [122], MMT-Security allows detecting whether a security property was respected or violated; or, an abnormal behavior (e.g.,

attack) was detected or not.

MMT-Security intended to formally define occurrence of events that denote a security rule to be respected or violated; or the detection of an event of an attack or vulnerability taking place, referred as attack rule. The Figure 4.4 shows how the probe interacts with the MMT-Security and MMT-Extract. The Security engine will analyze the events captured by the Event Processing engine, receiving only the needed data from it. It will use the user specified MMT-Properties and send notifications to the probe when a property is detected or not. The register attributes that MMT-Security has access are all the structures defined in the previous algorithms of Section 4.2.

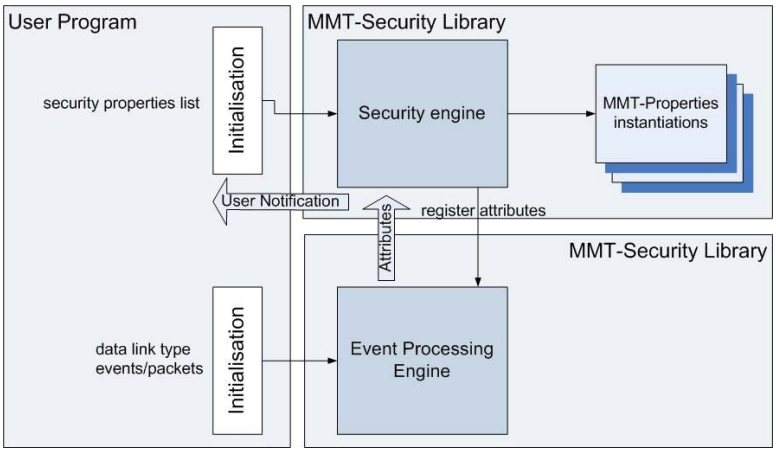


Figure 4.4: MMT_Security Library

As stated previously, the MMT-Security properties rely on LTL temporal logic and are written using XML syntax. Each one of those properties is defined inside of a XML field with the tag of `property`; each of them can be defined as a "Security rule" or "attack". Each property will have leaf fields with the tag of `events`. The first event is identified as the "left branch" or "context", while the second event is the "right branch" or "trigger" as shown in Figure 4.5.

The Context is the definition of the event to monitor that can be a security checked or an identified atack; meanwhile the trigger is the event that indicates if the context is valid. Possible interpretations for the context and trigger in relation to the events are the following:

- If the context is verified and the trigger is not, a non-respect instance is detected.
  - Security rule - The context of the rule has been found but not the trigger, therefore the security rule has been violated.
  - Attack rule - The context of an in-line-attack has occurred but the trace was attack free.
- If the context and the trigger are verified, a respect instance is detected.
  - Security rule - The security rule has been respected.
  - Attack rule - The behavioral of an attack has been detected.

The context and trigger can be formed by multiple events. When this happens XML fields called `operators` are used. They still need to follow the structure of the left and right branch of Figure 4.5. In any case, the context can occur at the same time, after or before the trigger.

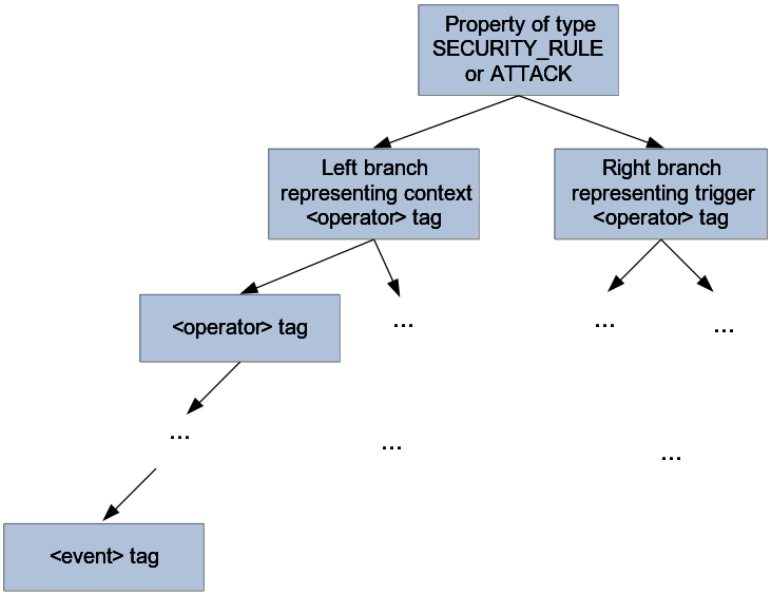


Figure 4.5: MMT_Security property structure

ETSI Threat Number (ET#)	Security rules
(ET8) Discover Keys by Eavesdropping on Communications Between Entities.	SR1, SR2
(ET10) Provisioning of non-Legitimate Keys.	SR5
(ET15) General Eavesdropping on M2M Service-Layer Messaging Between Entities.	SR3, SR4
(ET16) Alteration of M2M Service-Layer Messaging Between Entities.	SR3, SR4, SR5

Table 4.1: MMT Security Rules for the monitoring of mitigation of threats.

A quick example of an attack rule can be considered with the denominated ARP poisoning attack. The attack consists on rogue nodes answering an ARP request with fake information; usually its purpose is to misdirect the traffic to specific nodes. The context of the monitoring is any ARP request that occurs in the medium, and the trigger is the event where two or more ARP replies appear with different MAC addresses in a specific range of time.

In the previous example, the context of the rule is a simple packet: the ARP request. Meanwhile, the trigger is a series of events: the first ARP reply, which is expected, and one or more messages with different MAC addresses than the first one in a specific range of time. As general rule, MMT can handle complex events for the event and the trigger. Also, the events are not strictly in order: the trigger could happen before, after or at the same time than the context.

This example is extended in the Appendix C.

The rules are written in Boolean logic with a notation of LTL, but when required to do a complex analysis it is possible to use the embedded functions.

In Section 3.4 an analysis of threats and risks for M2M communication was performed and its conclusions was presented in Table 3.2. For this section, the objective is to translate the countermeasures to MMT-Security rules. This is reflected in Table 4.1 with the five security rules identified.

4.4.1 Security Rule 1 (SR1)

```
<property value="THEN" property_id="1" type_property="SECURITY_RULE"
          description="(SR1) Whitelist of Cipher suites" >

  <operator value="THEN" delay_max="10"
         description="Analysis of cipher-suites (client and server)">

       <event value="COMPUTE" event_id="1"
       description="Detecting a Client Hello (new DTLS session)"
       boolean_expression="(
       (PLUGIN_DTLS_HNDSHK_CLIENT_HELLO.length &gt; 0 ) &amp;&amp;
       (#dtls_cookie_size(PLUGIN_DTLS_HNDSHK_CLIENT_HELLO.cookie) &lt; 1 ) )"
       />

    <event value="COMPUTE" event_id="2"
       description="Detecting a Server Hello for a specific DTLS Session"
       boolean_expression="(
       (PLUGIN_DTLS_HNDSHK_SERVER_HELLO.length &gt; 0 ) &amp;&amp;
       (IEEE802154DATA.src == IEEE802154DATA.dst.1) &amp;&amp;
       (IEEE802154DATA.dst == IEEE802154DATA.src.1) &amp;&amp;
       (PLUGIN_PROTO_NHC_44.src == PLUGIN_PROTO_NHC_44.dst.1) &amp;&amp;
       (PLUGIN_PROTO_NHC_44.dst == PLUGIN_PROTO_NHC_44.src.1) )"
       />
  </operator>

  <event value="COMPUTE" event_id="3"
       description="Validating the cipher suites selected"
       boolean_expression="(
       #dtlsf_ciphers_valid(PLUGIN_DTLS_HNDSHK_CLIENT_HELLO.ciphersuites.1,
       PLUGIN_DTLS_HNDSHK_SERVER_HELLO.ciphersuite.2) == 1)"
       />

</property>
```

Figure 4.6: MMT Security Rule #1

The mitigation of the threat ET8, and in a minor degree of ET15 and ET16, is handled by DTLS with the cipher suites. Therefore, monitoring an adequate cipher suite selection is required. As discussed in Section 3.2.4.1, TinyDTLS currently supports only two cipher suites. A signal of alert would be that a node attempts to establish a session using different cipher suites. In Table 4.1 the monitoring of this mitigation is referred as SR1.

SR1 is identified in the Table 4.1 as a mitigation of the threat ET8. It consists in the monitoring of the cipher suites offered by the client and the one selected by the server. Because of the use of TinyDTLs, the only acceptable cipher suites are: `TLS_PSK_WITH_AES_128_CCM_8` and `TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8`.

The context is based on two events: I) the detection of a DTLS Client Hello record. And II) the detection of a DTLS Server Hello record for the same DTLS session. Both events must occur inside of the range of lifetime of the DTLS session. If this time is superior, the records are probably from different DTLS sessions between the same nodes.

The trigger is the field of `Ciphersuites` inside of the two records from the context. This

```
<property value="THEN" property_id="2" type_property="SECURITY_RULE"
       description="(SR2) Cookies size (Client and Server)" >

  <!-- NOTE: The HelloVerifyRequest is who define the cookie size. -->
    <event value="COMPUTE" event_id="1"
        description="Detecting an HelloVerifyRequest"
        boolean_expression="(PLUGIN_DTLS_HNDSHK_HELLO_VERIFY_REQUEST.length &gt; 0 )"
    />

  <event value="COMPUTE" event_id="3"
        description="Detecting if the cookies are of the correct size"
        boolean_expression="(
        #dtls_cookie_size(PLUGIN_DTLS_HNDSHK_HELLO_VERIFY_REQUEST.cookie.1) == 1 )"
  />

</property>
```

Figure 4.7: MMT Security Rule #2

is basically a whitelist checked by means of an embedded function. This could be optimized to only check the server message; however, there is a risk of not detecting rogues nodes or wrongly configured nodes running in the client side.

Figure 4.6 is a partial capture of the implementation of our rule. The context is composed by the two events inside the operators (lines 4 to 23). The embedded function `dtls_cookie_size` (line 11) extracts the dynamic field of the cookies and confirms its length. This helps us to identify the second `Client Hello` of the DTLS session. The embedded function `dtlsf_ciphers_valid` (line 28) extracts the dynamic fields of `cipher suites` of both DTLS record. The lines 20 and 21 are just a portion of the Boolean expression. They are only capturing traffic using the compression of NHC for UDP where the port addresses are reduced to nibbles. In reality, this rules embark the other three possible values.

4.4.2 Security Rule 2 (SR2)

The cookies are another factor to mitigate the ET8 threat, as they are intended to resist any replay attacks against DTLS. Therefore, an adequate cookie size should be in consideration, the monitoring of this mitigation is identified as the SR2.

SR2 is identified in the Table 4.1 as a mitigation of the threat ET8. It consists on the monitoring of the size of the cookie used by the client and the server. They are used for the first time in the DTLS Hello Verify Request record and for the second time in the DTLS Client Hello record. For this work, the default size of TinyDTLs is used, which is about 16 bytes.

The context is based on the detection of a DTLS Hello Verify Request record. The trigger is the cookie inside of said DTLS record and it is handled by means of embedded functions. Figure 4.7 is a partial capture of the implementation of our rule. The embedded function `dtls_cookie_size` (line 7) extracts the dynamic field of the cookies and confirms its length. Although it is not visible in the code, the accepted value for the cookie size is 16 bytes.

4.4.3 Security Rule 3 (SR3)

To mitigate the threats ET15 and ET16, all the communication between the sensors and other devices (the 6BR and the clients) must be by means of a previously established DTLS channel.

```
<property value="THEN" property_id="3" type_property="SECURITY_RULE"
       description="(SR3) Valid (compressed) port">

  <operator value="OR"
           description="Detecting any of the possible NHC headers" >

    <event value="COMPUTE" event_id="1" ... /> <!-- NHC FF -->
    <event value="COMPUTE" event_id="3" ... /> <!-- NHC 8F -->
    <event value="COMPUTE" event_id="4" ... /> <!-- NHC F8 -->
    <event value="COMPUTE" event_id="2"
          description="NHC (Src and Dst) (16, 8 or 4 bits)"
          boolean_expression="(PLUGIN_PROTO_NHC_44.src &gt; 0)"
    />
  </operator>

  <operator value="AND"
    description="Detecting NHC Headers and DTLS records " >
      <operator value="OR"
          description="Detecting the correct port (16, 8 or 4 bits)" >

        <event value="COMPUTE" event_id="6" ... /> <!-- NHC FF -->
        <event value="COMPUTE" event_id="7" ... /> <!-- NHC 8F -->
        <event value="COMPUTE" event_id="8" ... /> <!-- NHC F8 -->

        <event value="COMPUTE" event_id="9"
        description="NHC (Src) 4-bits (Dst) 4-bits"
        boolean_expression="(
              (PLUGIN_PROTO_NHC_44.src.3 == 2)
              (PLUGIN_PROTO_NHC_44.dst.3 == 2) )"
        />
      </operator>

      <event value="COMPUTE" event_id="11"
          description="This UDP message must be only DTLS records"
          boolean_expression="(PLUGIN_DTLS_RECORD.length &gt; 0)"
      />
  </operator>

</property>
```

Figure 4.8: MMT Security Rule #3

```
<property value="THEN" property_id="4" delay_min="0" delay_max="25"
          type_property="SECURITY_RULE"
          description="(SR4) Unique DTLS session for Application Data"
          >

    <event value="COMPUTE" event_id="1"
           description="DTLS record CHANGECIPHERSPEC (0x14) detected."
           boolean_expression="(
           PLUGIN_DTLS_RECORD.content_type == 20
          )
           "
     />

   <event value="COMPUTE" event_id="2"
           description="There was a Client Hello in the near past (10 seg)"
           boolean_expression="(
            (PLUGIN_DTLS_RECORD.content_type == 23) &amp;&amp;
            ...
            (IEEE802154DATA.dst == IEEE802154DATA.dst.1) &amp;&amp;
            (IEEE802154DATA.src == IEEE802154DATA.src.1)
           )"
     />
```

Figure 4.9: MMT Security Rule #4

Any communication in the upper layer that goes in plain text should trigger an alert. This security rule is referred as the SR3.

It is possible to have the following criteria for monitoring this rule: To identify the protocols expected to be used by the sensors by means of the port addresses. Anything else should trigger an alert, even other protocols that are using the same port addresses. For this, is required the DPI property of MMT.

The default port for DTLS is 20220. However, if the NHC compression of UDP is used, this port address is outside the range of compression discussed in Section 3.2.2.1. Therefore, for this work the port address used will be 61618 (0xF0B2), which can be used for compressing the addresses from sixteen to eight or four bits.

For this rule, the context is based on any packet that carries an NHC header. Meanwhile, the trigger is composed of two events: I) Analyzing that the port address used is 61618 for the field destiny or source. And II) that the protocol used is identified as DTLS.

Figure 4.8 is a partial capture of the implementation of our rule where the context and the trigger inside their respective operators try to catch any of the possible sizes for the port addresses using NHC for UDP. In this capture, only the full event for the nibble sizes is displayed: In the context are the lines 7 to 9, in the trigger are the lines 21 to 23 inside of the second operator tag (line 18). The last event (line 33) permits to avoid any false negative in the rule. This rule is written in this way for detecting an untypical configuration of the nodes (and probably rogue nodes) although it is currently omitting the standard UDP header without NHC.

4.4.4 Security Rule 4 (SR4)

To reinforce the previous mitigation measures, the lifetime of a DTLS session should be considered; for each DTLS session a unique pair of symmetric keys is valid, and it should renew the session each time new data is ready. Still debatable, is the frequency of the session reset. The monitoring to verify that the DTLS sessions are renewed inside an adequate frame time is referred as the SR4. The SR4 is identified in the Table 4.1 as a mitigation of the threat ET15 and ET16.

The frequency to renew the DTLS session in any given scenario with constrained resources is still debatable. As a proof of concept, in this work it is considered that for each new data to be sent between a client and a server, a full DTLS handshake process must take place before. The DTLS session is completed when the couple of DTLS Finished records are sent. However, TinyDTLS omits those records, thus forcing another approach. This will be with the DTLS ChangeCipherSpec record, which forces the nodes to renew the DTLS session (or to create the first one). Accordingly, the context is the presence of a DTLS ChangeCipherSpec record while the trigger is the occurrence of a previous DTLS Hello Client record that happened inside a valid range of time.

The Figure 4.9 is a partial capture of the implementation of our rule. As already stated, this is a proof of concept and it is strongly linked to the frequency of the DTLS session renewal. The session can be identified as it involves a socket address (IPv6 and port addresses must be the same) between the client and server. To confirm the socket address, the rule requires to confirm the UDP port, which can be in any of the four plugins related to NHC (similar to Security Rule 3). As a consequence of the available space, this is left outside of the capture (line 18).

4.4.5 Security Rule 5 (SR5)

The ET16 is particularly complex to mitigate. DTLS can generate a pair of symmetric keys for each DTLS session. However, they way in which the asymmetric keys are distributed in the sensors is a key factor. One alternative is to load the keys to the nodes at the moment of their physical implementation, yet, this also comes with its own risks and challenges. In contrast, if the nodes use an online option to update the keys, then this process must be under a strict control and the monitoring is to be referred as SR5. The SR5 is left outside of the scope of this dissertation due to its high complexity, however it is an important work to do in the near future.

4.5 Conclusion

In Section 4.1 our monitoring tool, MMT, is introduced. MMT is highly versatile and powerful over standard networks. As discussed in Section 4.2, MMT can be divided into three modules with two of them being highly relevant for this dissertation: MMT-Extract and MMT-Security. With the former, it was possible to adapt MMT for 802.15.4/6LoWPAN networks. The latest lets us to configure the monitoring for the threats and the previously risk identified.

In Section 4.3 the details and characteristics of the development of MMT-Extract plugins are disclosed. The Figure 4.10 shows all the plugins developed for this dissertation. Those plugins are still in beta with the following states for each TCP/IP layer:

- Network access layer (0802.15.4) - Almost completely implemented. With exception of the MAC command and beacon frames. Also, the AES-128 secure mode was not supported.

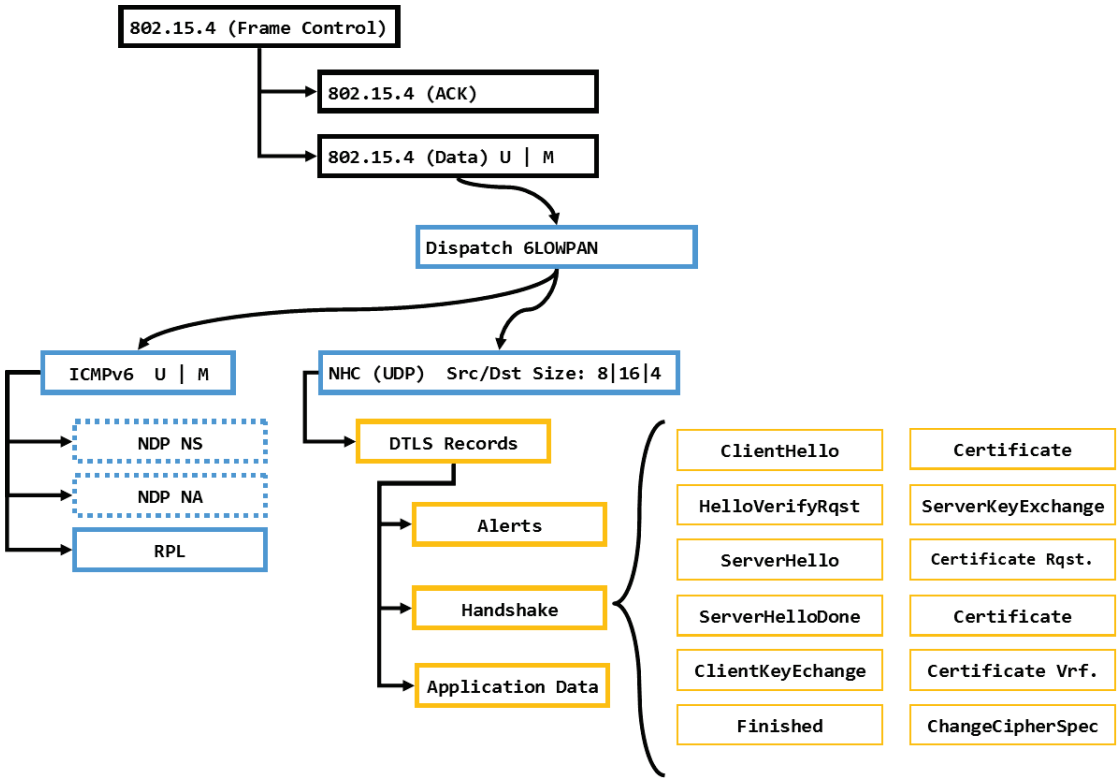


Figure 4.10: Summary of plugins developed for MMT.

- Network layer (6LoWPAN) - This layer is the most complex and currently only the IPHC dispatch is supported. The DPI at this layer is also supported with FRAG1 and FRAGN dispatches.
- Transport layer (NHC) - Fully supported.
- Application layer (DTLS/CoAP) - The support for the tinyDTLS stack is fully integrated. However, any support for other stacks that make use of jumbo datagrams and fragmentation over this layer requires further work. Because CoAP messages are hidden to the monitoring tool, CoAP is not yet supported.

All the security rules for the module MMT-Security are discussed in Section 4.4. Those rules are in relation to the threats and their countermeasures discussed in Section 3.4.1. The fifth security rule has been left out of the current work of this dissertation as it is very problematic.

The adaptation of MMT to make it able to monitor 6LowPAN networks is probably the most important contribution of this dissertation, even if the emphasis has always been given to the monitoring of the upper layers. Due all the special restrictions of those constrained networks, direct approaches were not possible.

Finally, the next step will be the tests in real time of the tool with the testbed.

Chapter 5

Experimental set-up and result analysis

Contents

5.1 Introduction

The purpose of this chapter is the analysis of the results from our experiment to validate our adaptation of MMT for 6LoWPAN and the implementation of a testbed with TinyDTLS and RIOT OS. Therefore, the testing will be divided into two categories in two different sections: Section 5.2 where all the components are tested separately: the behavior of TinyDTLs over RIOT OS and Contiki OS, the performance of MMT for 6LoWPAN, and the detection of the security rules by separated. And, in Section 5.3 are executed a series of experiments in the FIT-IoT lab testbed with MMT for 6LoWPAN for monitoring real M2M traffic over scenarios with 2, 5, 10, 20, 50 and 100 sensors.

At the moment of writing the present dissertation, MMT for 6LowPAN is unable to capture traffic in real time. This is because there is not a native support for the IEEE 802.15.4 standard over Linux[1]. Therefore, all our tests are based over previously recorded traffic in pcap file

formats. Appendix D provides a list of files generated for this dissertation, including instructions for retrieving them from an online repository.

The content of this chapter was extensively employed on the elaboration of the work presented [***under review***] in the International Journal of Computers and Applications.

5.2 Experimental testing

This section has as purpose to test all the different components separately. As previously discussed in Section 3.5.3, the stack for TinyDTLS was adapted to work for RIOT. To our best knowledge, this is the first adaptation made for both components. The tests for TinyDTLS are less inclined over power performance as they are to the measurement of the memory use comparing all the works discussed in Section 2.5.

MMT for 6LoWPAN is also evaluated to verify that it is able to operate correctly in this environment. The series of experiments for 6LoWPAN concerns the analysis of the previously recorded traffic. The tests involve sending different type of traffic, including malicious traffic.

Finally, in this section, the term "motes" and nodes are used for referring to the different hardware platforms (boards) for the sensors.

5.2.1 TinyDTLS over RIOT OS

The first series of tests for TinyDTLS is on the native mote for RIOT. These tests have as purpose to validate the correct implementation of TinyDTLS. The native mote emulates a hardware platform at the API level. Yet, it is only a Linux process referred as RIOT instances which can be connected between themselves by means of TAP interfaces.

TinyDTLS 0.8.2 supports the cipher suites `TLS_PSK_WITH_AES_128_CCM_8` (mode PreShared-Key or PSK) and
`TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8` (mode RawPublicKey or RPK). However, TinyDTLS servers are always to select the latter over the former as it is stronger. Therefore, the first series of tests will be to execute two RIOT instances which only support PSK. Figure 5.1 shows the output of the RIOT instance for the client. Line 5 shows the manual command to begin the DTLS session. Line 6 is an output message of the client indicating the supported cipher suites (in this case only those related to the PSK mode). The six DBG-Client outputs are the messages received by the server: four of them correspond to the DTLS handshake process, a fifth one is a DTLS record of Hello Request and the sixth is the echo of the data sent. The line 29 is the output that comes when the DTLS session is being completed. However, this involves the sending of a DTLS-Alert record not displayed in this output.

Figure 5.2 shows a portion of the traffic captured in the tap interfaces of the previous test and it is available in the file `RIOT-instances.pcap`. The RIOT instances run over Ethernet and IPv6. Both occurrences of the DTLS record "`Encrypted DTLS handshake message`" are the DTLS *Hello Request* record, but as TinyDTLS is adding a body to the record Wireshark is unable to identify them as records. Also worth to note in this capture, is that the DTLS Finished record is not present in the communication, which is also provoked by TinyDTLS.

The second test consists on testing the mode RWP. The client and the server are compiled supporting only the cipher suite `TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8`. Figure 5.3 is a portion of the server output of this testing. The line 4 is the manual command for starting the server.

[1]Experimental support for IEEE 802.15.4 and 6LoPWAN in the Linux Kernel is available at http://lxr.free-electrons.com/source/net/ieee802154/

```
main(): This is RIOT! (Version: 2017.01-devel-94-g7ba48-Dell-Thesis_Tests)
RIOT (Tiny)DTLS testing implementation
All up, running the shell now

> dtlsc fe80::8421:dfff:febb:66f0 "Temperature: 10C"
Client support PSK
DBG-Client: Debug ON
DBG-Client: Msg received from
        Addr Src: fe80::8421:dfff:febb:66f0
        Current Peer: fe80::8421:dfff:febb:66f0
DBG-Client: Msg received from
        Addr Src: fe80::8421:dfff:febb:66f0
        Current Peer: fe80::8421:dfff:febb:66f0
DBG-Client: Msg received from
        Addr Src: fe80::8421:dfff:febb:66f0
        Current Peer: fe80::8421:dfff:febb:66f0
DBG-Client: Msg received from
        Addr Src: fe80::8421:dfff:febb:66f0
        Current Peer: fe80::8421:dfff:febb:66f0
DBG-Client: Msg received from
        Addr Src: fe80::8421:dfff:febb:66f0
        Current Peer: fe80::8421:dfff:febb:66f0
DBG-Client: Msg received from
        Addr Src: fe80::8421:dfff:febb:66f0
        Current Peer: fe80::8421:dfff:febb:66f0

 Echo received: Temperature: 10C

DTLS-Client: DTLS session finished
```

Figure 5.1: TinyDTLS test with the PSK mode as a RIOT instance

No.	Source	Destination	Protocol	Info
55	fe80::20d5:45ff:fe53:f89	ff02::1	ICMPv6	Router Advertisement from 22:d5
56	fe80::20d5:45ff:fe53:f89	fe80::8421:dfff:febb:66f0	DTLSv1.2	Client Hello
57	fe80::8421:dfff:febb:66f0	fe80::20d5:45ff:fe53:f89	DTLSv1.2	Hello Verify Request
58	fe80::20d5:45ff:fe53:f89	fe80::8421:dfff:febb:66f0	DTLSv1.2	Client Hello
59	fe80::8421:dfff:febb:66f0	fe80::20d5:45ff:fe53:f89	DTLSv1.2	Server Hello
60	fe80::8421:dfff:febb:66f0	fe80::20d5:45ff:fe53:f89	DTLSv1.2	Server Hello Done
61	fe80::20d5:45ff:fe53:f89	fe80::8421:dfff:febb:66f0	DTLSv1.2	Client Key Exchange
62	fe80::20d5:45ff:fe53:f89	fe80::8421:dfff:febb:66f0	DTLSv1.2	Change Cipher Spec
63	fe80::20d5:45ff:fe53:f89	fe80::8421:dfff:febb:66f0	ICMPv6	Neighbor Solicitation for fe80:
64	fe80::8421:dfff:febb:66f0	fe80::20d5:45ff:fe53:f89	ICMPv6	Neighbor Advertisement fe80::84
65	fe80::20d5:45ff:fe53:f89	fe80::8421:dfff:febb:66f0	DTLSv1.2	Encrypted Handshake Message
66	fe80::8421:dfff:febb:66f0	fe80::20d5:45ff:fe53:f89	ICMPv6	Neighbor Solicitation for fe80:
67	fe80::20d5:45ff:fe53:f89	fe80::8421:dfff:febb:66f0	ICMPv6	Neighbor Advertisement fe80::20
68	fe80::8421:dfff:febb:66f0	fe80::20d5:45ff:fe53:f89	DTLSv1.2	Change Cipher Spec
69	fe80::8421:dfff:febb:66f0	fe80::20d5:45ff:fe53:f89	DTLSv1.2	Encrypted Handshake Message
70	fe80::20d5:45ff:fe53:f89	fe80::8421:dfff:febb:66f0	DTLSv1.2	Application Data
71	fe80::8421:dfff:febb:66f0	fe80::20d5:45ff:fe53:f89	DTLSv1.2	Application Data
72	fe80::20d5:45ff:fe53:f89	fe80::8421:dfff:febb:66f0	DTLSv1.2	Encrypted Alert
73	fe80::8421:dfff:febb:66f0	fe80::20d5:45ff:fe53:f89	DTLSv1.2	Encrypted Alert
74	fe80::8421:dfff:febb:66f0	ff02::1	ICMPv6	Router Advertisement from 86:21

Figure 5.2: A capture of DTLS session with the mode PSK and RPK (`RIOT-instances.pcap`).

```
main(): This is RIOT! (Version: 2017.01-devel-94-g7ba48-Dell-Thesis_Tests)
RIOT (Tiny)DTLS testing implementation
All up, running the shell now
> dtlss start
dtlss start
Server support ECC
DBG-Server On
Success: started DTLS server on port 61618
> DBG-Server: Record Rcvd!
DBG-Server: Sending record to peer
DBG-Server: Record Rcvd!
DBG-Server: Sending record to peer
DBG-Server: Sending record to peer
DBG-Server: Sending record to peer
DBG-Server: Sending record to peer
DBG-Server: Sending record to peer
DBG-Server: Record Rcvd!
DBG-Server: Record Rcvd!
DBG-Server: Record Rcvd!
DBG-Server: Record Rcvd!
DBG-Server: Record Rcvd!
DBG-Server: Sending record to peer
DBG-Server: Sending record to peer
DBG-Server: Record Rcvd!

DBG-Server: Data from Client: ---Temperature: 10C---  Sending echo..
DBG-Server: Sending record to peer
DBG-Server: Record Rcvd!
Nov 17 20:00:37 ALRT 0 invalidate peer
DBG-Server: Sending record to peer
Nov 17 20:00:37 WARN received alert, peer has been invalidated
```

Figure 5.3: TinyDTLS test with the RPK mode as a RIOT instance

Requirement	PSK and RPK		PSK only		RPK only	
	RAM	ROM	RAM	ROM	RAM	ROM
TinyDTLS (Core)	8	20245	8	16496	8	18913
TinyDTLS (Crypto)	180	12550	180	11790	180	12476
TinyDTLS (Debug)	64	713	64	713	64	713
Client side	65	2094	65	1863	65	1705
Server side	2076	1956	2076	1729	2076	1658
Main program	64	221	64	221	64	221
RIOT Kernel	21359	83133	21343	80428	21287	83086

Table 5.1: Requirement in memory (RAM/ROM) for our DTLS application over RIOT OS.

Line 6 is an output indicating that only the RPK mode is supported. Lines from 9 to 23 are DTLS handshake flights, including the records related to the certificate. Line 24 is the reception of the data sent by the client, which is shown in line 26; this data is processed (displayed for this demo). Line 28 is the reception of the DTLS Alert record sent by the client to finish the DTLS session. Once the server receives this message, it starts to release its resources and sends its own DTLS Alert record to the old client in line 30.

The integration of TinyDTLS to RIOT OS is working well. However, it is important to emphasize that up to this point the tests do not reflect a real implementation. Here, the nodes have more than one single megabyte of RAM. The buffer for the messages reception can store messages for an almost unlimited amount of time, while still accepting new messages. Also, 6LoWPAN is not being used. Therefore, the performance can fall greatly once real constrained motes are used. The RPK mode has been criticized in the past because of its heavy use of resources as discussed in Section 2.5.

5.2.2 FIT IoT Laboratories

As already discussed in Section 3.5.4, the FIT IoT Lab is an open source platform composed of hundreds of sensors and robots, all distributed in separate facilities. The hardware platform selected for the sensors is the M3 mote (64 KB RAM and 256 KB of FLASH). This type of mote is C2 and can be found in the Table 1.1; this type can also include an external memory of 1 megabyte and other utilities for measuring power and sniffing the traffic between the sensors.

Because of the high amount of resources available to the M3 motes, both cipher suites supported by TinyDTLS can be compiled to measure the required memory; they can also be compiled separately. These measurement of the memory is done with the use of the GNU tools `objdump` and `size`. Table 5.1 shows the amount of ROM (FLASH) memory required for the following components:

- TinyDTLS (Core) - The main package of TinyDTLS without including libraries related to encryption.
- TinyDTLS (Encryption) - The libraries used by TinyDTLs for handling the encryption (AES, SHA2 and ECC).
- TinyDTLS (Debug) - This is optional and can be ignored if the performance of the mote is affected, or if there is not enough space.
- Client - This is the amount of memory required for running the client process of TinyDTLS.

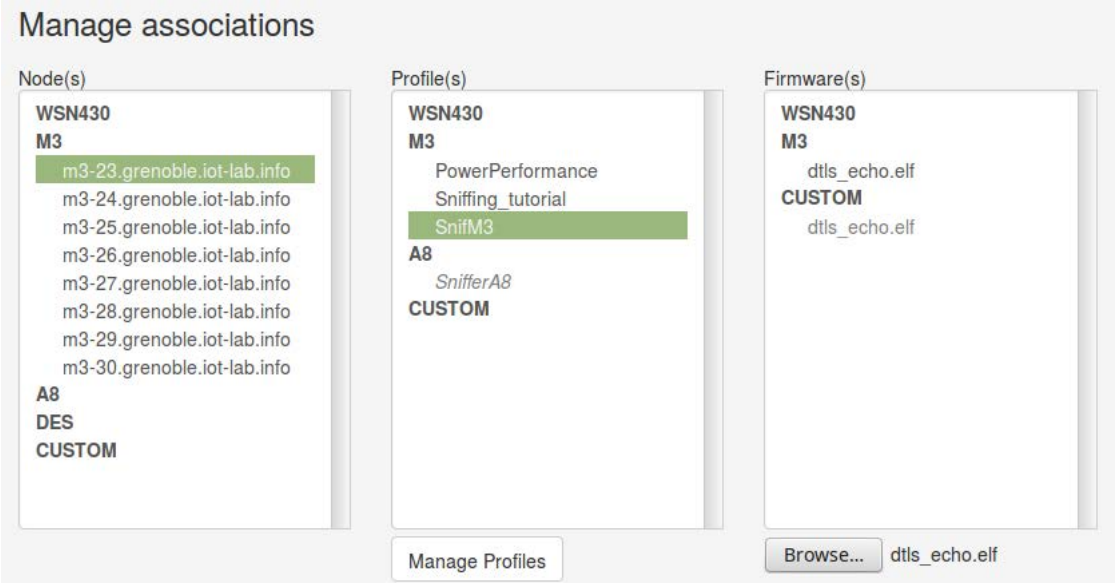


Figure 5.4: Web interface of the FIT IoT Lab.

- Server - This is the amount of memory required for running the server process of TinyDTLS.
- Main - This is the shell command that invokes the instance of the DTLS client and/or server

The sizes reflected in the client and server are highly dynamic as both of them have some additional debugging options. Additionally, PSK requires some extra space for its operations. Therefore, the sizes displayed are just a quick reference. The values for the core of TinyDTLS are of great interest, but they are also affected by the number of DTLS sessions expected to be handled by the nodes (2 sessions). The size of the rest of the program, the "kernel" with the 6LoWPAN stack, is not considered. In relation to the values for RAM shown in Table 5.1, the server requires more amount because it is listening to the request of the dynamic clients. The clients have also a dynamic size of the data to transmit, as shown in the line 5 of Figure 5.1.

The configuration of the testbed, for selecting and loading the firmware to the sensors, is done by means of a web interface. Also, it is possible to configure and interact directly with the sensors by means of a secure tunnel over SSH. Two of the main benefits of using the FIT IoT Lab are the hardware integrated in the sensors for capturing traffic, and the measuring power performance on the sensors as shown in Figure 5.4.

Almost all the recorded traffic utilized for this dissertation was generated with this testbed. Only in some exceptional cases is generated using the Cooja simulator and Linux devices. Two tools extensively used for all the tests are `sniffer_aggregator` and `serial_aggregator`. The former grants the use of one or more sensors in promiscuous mode, in parallel to the firmware loaded. The latter tool handles the configuration of multiple sensors at the same time.

The testbed generates real traffic from the sensors. Yet, because of the high number of sensors and the time shared nature of the testbed, it is also possible to capture traffic not related to our sensors. To try to minimize these occurrences, in all our tests we tried to select a cluster of sensors distant from other clusters. Another factor is that the logical topology formed by the sensors are not under our control. Therefore, for all the tests a snap made with the

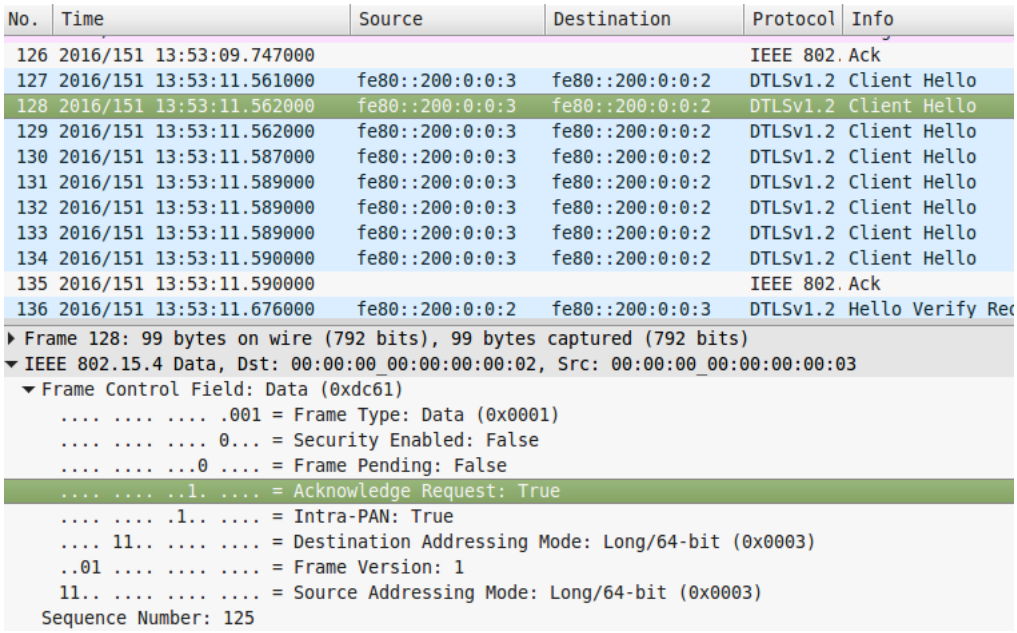


Figure 5.5: Multiple transmission of a single packet in a small frame of time due to the 802.15.5 ACK request.

Foren6 tool will be generated to show the formed topology, which can be replicated using the recorded traffics listed in the Appendix D.

There is one problem detected at the moment of our testing: If the sensors are handling the fragmentation of any packet and at the same time, are receiving 802.15.4 frames requesting an ACK, there is chance that the sensors will not respond, provoking that the peer floods the network with re-transmissions. This issue is reflected in the file *FIT-ACK-Conflicts01.pcap* and in Figure 5.5. This behavior was mitigated with an artificial timer between each transmission in the sensors. However, the sensors using the RPK mode still present issues that interfere with the DTLS handshake process, but as the current ECC library will be removed in the next TinyDTLS version, no more efforts were done in that regard.

5.2.3 MMT over 802.15.4/6LoWPAN

In Section 3.2all the work for adapting MMT to 6LowPAN to the traffic generated by WSN 802.15.4/6LoWPAN and using DTLS was discussed. All the features of those protocols are discussed in more detail in the Section 4.3. For testing the protocol we generated a simulated traffic with Cooja, in the early steps of the development, as well as real traffic with FIT IoT Lab.

Figure 5.6 is a partial capture of MMT processing the file *FIT-Fields.pcap*, which only contains five packets: one (ICMPv6) RPL message, one (ICMPv6) ND Router Solicitation message, one DTLS record Hello Client (fragmented in two parts) and one DTLS record Hello Verify Request record. The report of the MMT security rules is out of this capture. Each packet is processed by MMT identifying each field for each protocol, although it only displays generic information. The lines 2 and 1 are only identifying the last layer of ICMPv6 as that protocol is not yet supported.

Our adaptation of MMT's DPI property is tested by assembling the packets 4 and 5 that are

```
P. 1:~~802.15.4 (U)~~ ~~6LoWPAN IPHC~~ ~~In-line IPv6~~ ~~ ICMPv6 (U) ~~
P. 2:~~802.15.4 (M)~~ ~~6LoWPAN IPHC~~ ~~In-line IPv6~~ ~~ ICMPv6 (M8) ~~
P. 3:~~802.15.4 (U)~~ ~~6LoWPAN FRAG1~~
P. 4:~~802.15.4 (U)~~ ~~6LoWPAN FRAGN~~
00000000 61 DC 7A 23 00 16 A7 D5 46 33 48 32 36 72 8F DA
00000010 46 33 48 32 36 7E 33 F3 32 2F A6 16 FE FD 00 00
00000020 00 00 00 00 00 00 00 54 01 00 00 48 00 00 00 00
00000030 00 00 00 48 FE FD 00 00 00 00 D8 FE AB D9 48 29
00000040 93 72 D3 81 2D 1C CE 03 E6 D5 8C FB 3C EA 16 AD
00000050 0D 75 0B 57 75 1A 00 00 00 04 C0 AE C0 A8 01 00
00000060 00 1A 00 13 00 02 01 02 00 14 00 02 01 02 00 0A
00000070 00 04 00 02 00 17 00 0B 00 02 01 00 2F 87

P. 5:(*)~~802.15.4 (U)~~ ~~6LoWPAN IPHC~~ ~~NHC (UDP 44)~~ ~~DTLS Handshake (0x01)~~
P. 6:~~802.15.4 (U)~~ ~~6LoWPAN IPHC~~ ~~NHC (UDP 44)~~ ~~DTLS Handshake (0x03)~~
```

Figure 5.6: Partial output of MMT for 6LoWPAN *FIT-Fields.pcap*.

```
Packet 5
 UDP: (Full) src 61618 - (FULL) dst 61619
DTLS (0x16) - version: 0xFEFD, epoch: 0, length 31 Seq number: 0x000000000000
 DTLS Handshake type: 3 (HVR) length: 19 mesage sequence: 0
       Fragment offset: 0 Fragment length: 19 Version: FEFD
       Cookie length: 16 Cookie: 10 F4 E8 D6 92 EF 93 44 1D 25 F1 18 EC 3B A0 CC 24

...

Packet 23
 UDP: (4 bits) src 2 - (4 bits) dst 3
DTLS (0x16) - version: 0xFEFD, epoch: 0, length 31 Seq number: 0x000000000000
 DTLS Handshake type: 3 (HVR) length: 19 mesage sequence: 0
       Fragment offset: 0 Fragment length: 19 Version: FEFD
       Cookie length: 16 Cookie: 10 A1 58 EC 8A 55 76 C6 D6 EF 81 5D 94 A5 9F 5F 97
```

Figure 5.7: Partial output of MMT for 6LoWPAN with UDP compression *FIT-Ports-Tests.pcap*.

present in lines 3 and 4. In this first approach, as soon as the last FRAGN of one fragmented message is processed, it will be inserted to MMT. Lines 5 to 12 shown the original packet. Currently, MMT is forced to add a fake 802.15.4 frame around the packet for returning it to MMT-Extract. Line 14 is the new assembled packet already processed. If this output is compared against the display of Wireshark, there will be a difference of one packet because MMT is currently identifying both the assembled packet and the last FRAGN as two different packets. All the details of the fragmentation are discussed in Section 4.3.2.1.

In the work presented at APCC 2016 [115], MMT for 6LoWPAN performance was measured with different file traces, including malicious packets for the RPL protocol, malformed packets and highly fragmented packets. This traffic was also generated in the FIT IoT Lab, but using Contiki 2.7 instead of RIOT OS. The file *FIT-Contiki300.pcap* contains one portion of that traffic. This traffic is inside of a tunnel in the communication. Yet, MMT for 6LoWPAN can easily begin to process the packets ignoring the tunnel. One of the main conclusions of that work was that the MMT processing rate is around 420 Mbps. Finally, this measurement was limited to extract the network access and the network layers.

There is one last factor to consider at the moment of using the rules: The UDP port

	# Total	# Completed
2 nodes	1	1
5 nodes	7	4
10 nodes	16	6
20 nodes	52	28
50 nodes	46	20
100 nodes	43	21

Table 5.2: Number of DTLS sessions started.

compression. Contiki OS requires an explicit request for compression while RIOT OS relies on the use of masks (akin to the IPv4 subnet mask); other implementations could fall between those two. For MMT Security Rules where the required ports required must be considered. Our TinyDTLS implementation made use of the port 61618 (0xF0B2). However with this, it might be possible that any of the four plugins related to NHC UDP can retrieve the following values: 61618 (0xF0B2), 178 (0xB2) or 2. The Figure 5.7 has a partial output of the file *FIT-Ports-Tests.pcap* with the processed DTLS record Hello Verify Request. Both DTLS records are recovered by different NHC plugins.

5.3 Testing in real environments

This section is composed of the experiments that involve the use of all the MMT Security Rules discussed in Section 4.4 over real scenarios of 2, 5, 10, 20, 50 and 100 sensors which are deployed over the IOT FIT-Lab. The topology of each scenario is displayed using the foren6 tool. The following is a reminder of the five MMT Security Rules:

1. Whitelist of cipher suites.
2. Validation of the cookie size.
3. Validation of the (compressed) port.
4. Unique DTLS session for transmission of new data.

The traffic generated by the sensors consists basically in ICMPv6 traffic (most of it related to the routing protocol) and manually triggered DTLS communication events between the motes. The data transmitted are small chains of text, no longer than fifteen bytes which are echoed by the receptor to the emitter. After this, the DTLS sessions are terminated. With the purpose of testing the MMT Security Rule #1 the use of the RPK mode will be marked as non wished. Additionally, the sensors are not always able to finish their sessions. There are two identified reasons of those fails: 1) The sensors are having issues at the moment of processing the certificates and handling the reception of new messages; when this happens the nodes are required to restart. 2) Because a DTLS session can be re-established after a certain time, the registries related to them are not erased until the internal timer expires. This situation can provoke that a DTLS session for a third sensor becomes interrupted. In the scenario with 10 motes, the DTLS sessions were trying to be too close in time. On the contrary, in the scenario with 5 and 20 motes, between each pair of sessions the gap of time was higher than 10 seconds and consequently, the number of successful connections is higher on those two scenarios.

The tool `sniffer_aggregator -r` provided by FIT-LAB records all the communication of the motes. For each experiment, around 45 minutes of traffic are recorded, which in turn are

	SR1		SR2		SR3		SR4	
	Respected	Violated	Respected	Violated	Respected	Violated	Respected	Violated
2 nodes	1	0	1	0	14	0	2	0
5 nodes	4	4	8	0	85	27	8	4
10 nodes	7	3	15	0	134	0	11	3
20 nodes	36	6	47	0	534	0	50	18
50 nodes	30	7	41	0	464	0	41	12
100 nodes	23	10	38	0	428	1	40	8

Table 5.3: Summary of the MMT security rules results.

processed by MMT. In the Table 5.2 is shown the total number of DTLS sessions triggered manually together with the number of successful connections. Meanwhile, Table 5.3 shows the security rules analyzed by MMT (Section 4.4).

A main difference between those experiments and the discussed in Section 2.5 is the fact that the communication under analysis is from sensor to sensor, without any external clients. This is because the purpose is not a power performance, but the testing of the monitoring tool inside the network. For this first approach, monitoring requests made from sensor to sensor are enough.

5.3.1 Scenario with 2 sensors

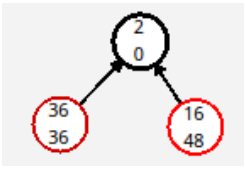
The recorded traffic is available in the file *FIT-2nodes.pcap*. The topology generated is shown in Figure 5.8a. There are 2 nodes that operate with DTLS and a third node working as the 6BR which also sniffs the traffic.

The two sensors support only the PSK mode. The communication between the sensors only happens once to test the rules. The SR1 displays one positive instance of the rule because the context is the occurrence of the DTLS Client Hello and Server Hello records. Meanwhile the trigger is the selected cipher suite for both of them. The only SR2 positive instance detected is because only the second DTLS Client Hello is utilized for the context. The SR3 shows 14 occurrences as this is the sum of all the DTLS records sent in the six flights of the DTLS handshake process, plus the data sent between the client and server and the termination of the DTLS session. The SR4 shows 2 positive instances detected, one for each DTLS ChangeCipherSpec record transmitted after the reception of a DTLS Client Hello, including a lifetime timer.

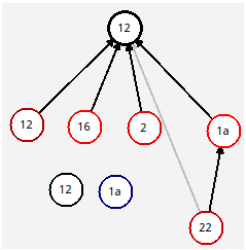
Also, note that the number of positive occurrences for a similar communication may vary with the current implementation of our rules. The SR2 and SR3 could have a higher number of occurrences if the DTLS records are forced to be re-transmitted. The SR1 is also affected in this situation, yet, because it is composed of two events in the same context as a timer, the number of occurrences can vary without relation to the other two rules.

5.3.2 Scenario with 5 sensors

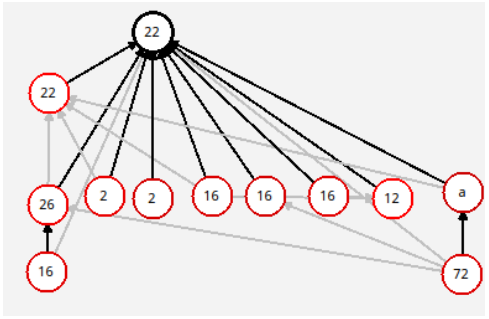
The recorded traffic is available in the file *FIT-5nodes.pcap*. The topology generated is shown in Figure 5.8b. There are 5 nodes that operate with DTLS and a sixth node that takes the place of the 6BR and the sniffer mode. However, there are two extra nodes which were not configured by us in the testbed. A deeper inspection in the recorded packets indicates that the nodes are only communicating between them. Therefore, it is assumed that our sniffer node was too close to those rogue nodes and was able to capture theirs packets. This unexpected situation had an impact over SR3.



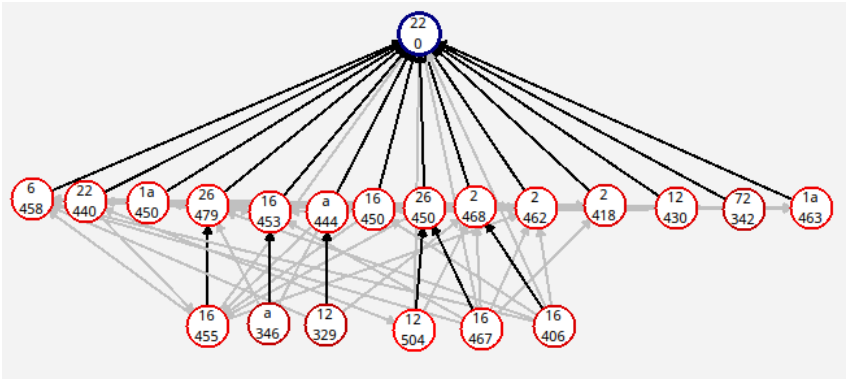
(a) 2 nodes and 6BR (`FIT-2nodes.pcap`).



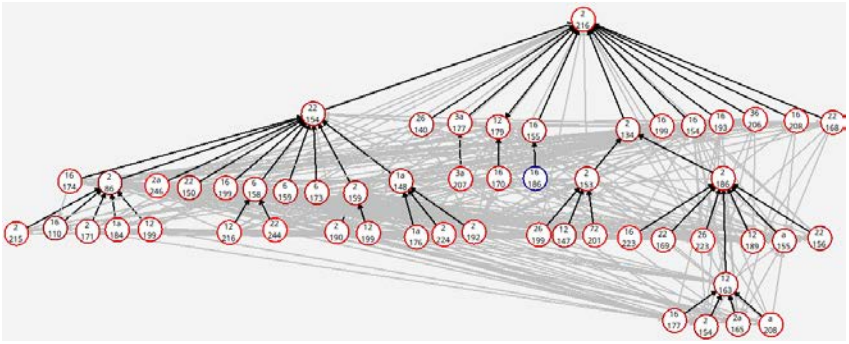
(b) 5 nodes and 6BR (`FIT-5nodes.pcap`).



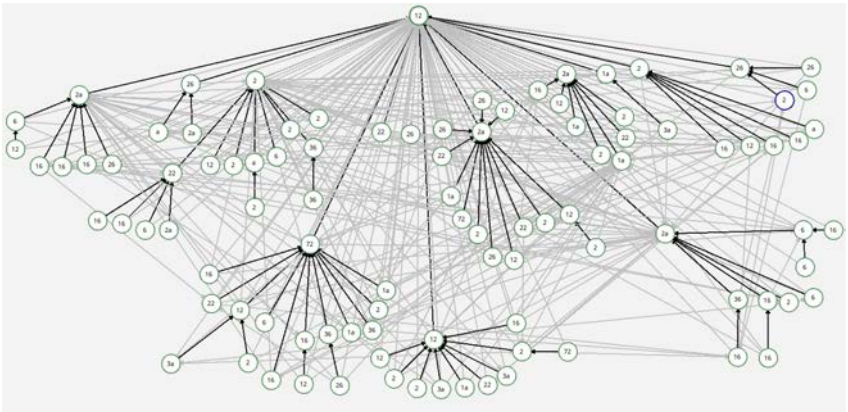
(c) 10 nodes and 6BR (`FIT-10nodes.pcap`).



(d) 20 nodes and 6BR (`FIT-20nodes.pcap`).



(e) 50 nodes and 6BR (`FIT-50nodes.pcap`).



(f) 100 nodes and 6BR (`FIT-100nodes.pcap`).

Figure 5.8: Topologies used for testing MMT-Security Rules.

The presence of rogue nodes is due to the open nature of the wireless channel. This occurrence is not unexpected on real implementations, particularly on environments with a high density of nodes. Therefore, detecting the presence of those nodes is significant and positive. Those two non-malignant nodes were not adverted until MMT for 6LoWPAN was executed, and it was after this that a more careful analysis over the recorded traffic was made. Although this inspection was manual, it is possible to automate it with MMT and this should be considered for a future work.

In this specific scenario, the 6BR was able to sniff all the traffic generated between the nodes as the proximity of the six nodes was enough for capturing the traffic. The gray line of the fifth node in Figure 5.8b confirms that the 6BR was reached individually by all the sensors. This approach is similar to install MMT over the 6BR node.

This scenario has eight DTLS sessions between the sensors, four of the PSK mode and four of the RPK. The SR1 and SR2 are in concordance with this amount of established sessions. However, the SR4 detected twelve occurrences: eight respected and four violated. After a deep analysis on the traffic recorded, it is discovered that the SR4 is detecting four couples of a DTLS ChangeChipherSpec record, one couple from each DTLS session. Still, there were 4 occasions where clients using the RPK mode were unable to finish the session. All the violation occurrences for SR3 are because the rogue nodes were using NHC UDP without compression.

5.3.3 Scenario with 10 sensors

The recorded traffic is available in the file *FIT-10nodes.pcap*. The topology generated is shown in Figure 5.8c. There are 10 nodes that operate with DTLS, one node operating as a 6BR, plus an eleventh node working as our sniffer node. The last node is identified in Figure 5.8c as the solitary node in the upper left corner. The idea behind this node was to test the feasibility of placing potentially MMT nodes (or NIDS) in other points of the WSN. However, the results are not conclusive as the node was able to sniff the 6BR router. This is a limitation of the use of the FIT IoT-Lab: the physical location of the nodes is not under our control.

In this scenario fourteen DTLS sessions supporting PSK were started with only six of them being established successfully. For the RPK mode, two DTLS sessions tried to be established, both without success. A deeper analysis of the recorded traffic shows that one or more DTLS servers rejected many of the DTLS sessions. This happened because the DTLS servers are only able to attend two simultaneous requests at anytime and, in some occurrences the server did not release the resources after the client ended the session as the internal timer didn't expire. There was a case where one DTLS client had to be rebooted in the middle of a DTLS session. The DTLS sessions with the RPK mode were aborted by the clients.

The SR1 reflects 3 instances where the context was verified but not triggered (and thus violated). However, there were only 2 DTLS sessions using the RPK mode. The third instance is due to the presence of a DTLS Client Hello retransmitted from a first attempt. The current implementation of the context for SR1 looks first for a DTLS Client Hello, and then for a DTLS Server Hello inside of a valid range of time. This duplicated record is inside that time, provoking a false positive for this security rule. Although the timer for SR1 was modified, it was not possible to find a better value that did not arise also the number of false negatives. Finally, the SR2 and SR3 are in concordance with the amount of DTLS records. The SR4 is affected by the issue detected in the previous scenario with 5 sensors.

5.3.4 Scenario with 20 sensors

The recorded traffic is provided in the file *FIT-20nodes.pcap*. The topology generated is shown in Figure 5.8d. There are 20 nodes that operate with DTLS and a 21sh node that is the 6BR which is used also for sniffing the traffic.

This test was more focused on the measurement of the sensor's resistance to overload with the DTLS sessions. A total of 53 DTLS sessions with the PSK mode were initialized, together with only 2 DTLS sessions with the RPK mode. At least, 24 of the DTLS sessions using PSK were executed from two different DTLS clients to a single DTLS server at the same time. However, 26 of these sessions failed to be fully established, mostly because the server was unable to handle its resources or because the timer for releasing the resources for each DTLS session was not expiring quickly enough. This is reflected with the results of the SR1, where only 42 DTLS sessions were identified. Two of the violated instances are the original 2 DTLS sessions for PSK. The other 4 are false positive due to the retransmission, similar to the observed in the previous experiment.

The retransmission also affected the high number of occurrences for the SR2 and SR3. The SR4 has a total of 68 occurrences with 54 corresponding to the original occurrence of the DTLS ChangeChipherSpec record. Although 2 of them were outside of the expected time. The remaining occurrences were caused by the retransmission of the DTLS ChangeChipherSpec record which put them outside of the valid lifetime, compared with the previously transmitted DTLS Hello Client records.

As a final comment about this experiment, probably the most relevant fact of this test is the identification of memory issues in the nodes which must be addressed.

5.3.5 Scenario with 50 sensors

The recorded traffic is provided in the file *FIT-50nodes.pcap*. The topology generated is shown in Figure 5.8e. There are 53 nodes that operate with DTLS and another node that is the 6BR which is used also for sniffing the traffic.

This test is different to the previous scenarios because the sensors are getting a friendlier firmware in relation to the use of RAM memory. The optional packet of TinyDTLS debug (See Table 5.1) is removed. Additionally, some RIOT modules are not loaded with the sensors[2].

There are 41 DTLS sessions initialized with the PSK mode and 5 for the RPK mode. For the PSK mode, 20 sessions were completed, where five couples of them were executed simultaneously. In relation to the unsuccessful DTLS sessions, some of the DTLS sessions never got an answer from the servers as a result from a memory exhaustion from previous DTLs sessions established, and only four of them had the nodes affected by a memory overflow in the middle of the handshake.

The nodes were divided in three groups: One group supporting only the PSK mode, a second group with support only for the RPK mode and a third group with support for both modes. The last group was more prone to have issues handling memory. Also, every time that more than one DTLS client contacted a DTLS server, they did it simultaneously, instead of starting a new negotiation at different times, which is more realistic in real implementations. In general, we concluded that the majority of issues detected in the scenario with 20 nodes were related only to the sensors performance and not to MMT.

[2] All the changes done to the client and server are already included in the repository.

The SR1 detected the five occurrences with the RPK mode, plus two false negative due to the retransmission of the DTLS Hello Client records inside the valid lifetime. The SR2 detected 41 occurrences which include both modes as the RPK mode is being is rejected by the clients after the server sends its certificate. The SR3 does not present any erroneous data. Finally, there are 53 DTLS ChangeCipherSpec records captured, however, the violated occurrences were due to problems with the nodes (memory overflow) or because of the presence of repeated messages.

5.3.6 Scenario with 100 sensors

The recorded traffic is provided in the file *FIT-100nodes.pcap*. The topology generated is shown in Figure 5.8f. There are 99 nodes that operate with DTLS and a 100th node that is the 6BR which is used also for sniffing the traffic. This test also included the light version for the client and servers. And it was also divided into three groups.

The DTLS sessions were established using as a client the nodes farther away from the 6BR, while the servers were neighbors to the 6BR (all of them have direct connection to the 6BR). For this scenario were generated 34 DTLS sessions with the PSK mode and 9 with the RPK mode. For the former, 21 sessions were successful, and only four of them were done simultaneously to a couple of servers. There were two occurrences where the server did not finish the DTLS session after receiving the client data.

The behavior of SR1 is in agreement with the number of DTLS sessions. The violated occurrence is a retransmitted DTLS Client Hello packet with the RPK mode. The difference between the occurrences detected by SR2 and the total of sessions detected are those requests made by clients to servers with saturated buffers which did not answer any message.

The SR3 detected an unusual error in the communication: The server answered with a corrupted payload for the UDP header, due to memory issues at the moment of processing the answer. This problem occurred two times and the servers are the same that did not finish the DTLS session. However, MMT was able to discern partially one of them as part of the DTLS protocol while the second record was beyond recognition.

In total 48 DTLS ChangeChiperSpec records were generated, including the two instances where only the client was able to send its data. Four of them are repeated messages. Therefore the SR4 is still affected by the issues identified in the previous scenarios.

5.4 Conclusions

The SR1 for the 5 motes scenario detects properly the DTLS ClientHello and ServerHello records. However, for the other scenarios not all of those records end with a fully established DTLS session. The SR2 has as a strong link to the SR1 because both of them monitor the same records, although they watch different fields. Their results differ only when the DTLS session negotiation is stopped before the DTLS Server Hello record is sent.

The final purpose of the SR3 is to validate that the traffic at the upper layer is composed only of DTLS records. In the case of the scenario with 5 motes, there were two rogue motes transmitting non-compressed UDP traffic; those were partially captured by MMT, leaving untracked only those that rely on a configuration not yet supported by our plugins.

The results thrown by the SR4 can identify the moment when the motes update their symmetric keys, by sending the DTLS ChangeCipherSpec record. The numbers in the first three scenarios double the number of successful connections because the rule takes separately the discover of each couple of DTLS ChangeCipherSpec records. This in turn is echoed by the

servers to the clients. This rule is impacted when the sensors are forced to resend packets due to the congestion in the network, which is the case of the scenario with 20 sensors. The cases where the rule was violated was because the lifetime expected was inferior to the real time. Therefore, there are two important conclusions: 1) The rule requires to follow the stream to try to identify the beginning and the end of a data transmission. 2) With the current configuration for the rules, it is not possible to guarantee the standard lifetime expected by DTLS in bigger scenarios, maybe because of the congestion in the network or to the resources of the motes.

The objective of all the scenarios proposed in this chapter was to measure the viability to guarantee a native monitoring over 6LowPAN. Almost all the present issues in Table 3 were due to problems in the performance of the sensors. This is a reminder that the challenge in order to secure M2M over WSN is not an easy one; yet, it is already feasible. It is also important to note that the tests were done in the most challenging approach to the nodes: The DTLS sessions are finished immediately, the sensors are able to establish more than one single DTLS session, they are forced to treat the sessions in peaks of traffic, etc.

Chapter 6

Conclusions and perspectives

Contents

6.1 Conclusions

Through this dissertation, we have presented and validated MMT for 6LoWPAN as a solution for monitoring M2M communication over WSN based on the architectures of IEEE 802.15.4 and 6LoWPAN and using as upper layer standard protocols recognized by different entities, such as the IETF and ETSI. More specifically the protocols of UDP, DTLS and CoAP. The standardization is required to reduce and to mitigate the bleak scenario where millions of objects are used to generate massive DDoS attacks. And even with them, the limited amount of available memory in the sensors make the use of well known secure protocols hard to achieve. Yet, they are feasible.

As discussed in Section 2.2 and Section 2.5, there have been many efforts dedicated to establish the type of WSN able to operate as the "Internet of Things" where many of them are not even compatible with the Internet (IPv4 or IPv6). Other works have focused only in validating the feasibility of using some type of protection in one or more of the TCP/IP layers (PANA, IPsec, DTLS) and even then their analysis are restricted to simple and controlled environments where only some parameters are measured. The last is done without detecting, or describing the issues that we detected through all the Section 5.2 (more than two sensors in the testbed, the client and/or server sides tested, the exhaustion of memory, impact on the lifetime of the DTLS session, etc.). There is a third kind of work that focuses on being able to monitor the WSN, which is indeed hard to achieve due to the wireless nature and mesh topology of those networks.

The works related to the monitoring over WSN can be divided into three categories: 1) Those focused only on detecting issues in the stage of implementation. 2) Those focused on detecting issues in the routing protocol to determine the state of the network (Foren6, PAD, SVELTE). 3) Those that are focused on detecting any type of aberrant traffic and not only routing attacks (LHSFW, LOMOM, Suricata for 6LoWPAN). Of those last IDS, probably the

work of adapting Suricata for 802.15.4/6LowPAN [71] is the most close to us. However, we are able to distinguish from them because of the following reasons:

1. MMT can monitor the traffic over 802.15.4/6LoWPAN without translating to IPv6.
2. As discussed in our works presented at AINA 2016 [120] and APPC 2016 [115], MMT is able to do more than Suricata. It can work with more flexible rules and detect a greater majority of new types of attacks without relying exclusively on the signatures.

The benefit of a native monitoring over 6LoWPAN, one that is able to analyze the original packet without translations, for monitoring only the upper layers may not be apparent. However, once the lowest layers are considered for the monitoring it becomes crucial. In the work with Suricata it is unclear what happens when a translation to IPv6 fails due to strange values on the FRAG1, FRAGN or IPHC headers, values that could be occurring because a fuzz[1]attack is taking place. MMT is able to detect those type of attacks. Particularly, the dispatches for fragmentation seem to be particularly vulnerable to those type of attack due to their paradigm with the offset and final size of the packets (Section 3.2.2.2).

As already stated in Section 5.4, our current MMT Security Rules are too simple and prone to generate false positives and/or negatives. Yet, those first proposed rules have been a proof of concept. MMT is a robust tool that allows us to generate a more robust set of rules able to follow strictly the flow of a DTLS session for each couple of nodes and determine if the correct mitigation for the M2M communication was done.

Finally, we want to enumerate briefly our contributions:

1. The official integration of the package TinyDTLS for the open source RIOT operating system.
2. To our best knowledge, the first implementation of an IDS with native support for IEEE 802.15.4 and 6LowPAN.
3. Security rules to validate the compliance with the mitigation identified by the ETSI IT M2M.
4. A testbed of up to 100 sensors to validate the previous point as feasible over WSN.

6.2 Future work

We have validated the feasibility of using MMT for monitoring 6LoWPAN networks, and also in the design of WSN with security as a base. Still, there are issues that were not feasible to handle on this dissertation. They can be divided into three big categories: 1) Improving the plugins for 802.15.4/6LoWPAN, 2) Improving the MMT Security Rules, and 3) Testing the feasibility for a secure WSN as well as identifying the challenges related to performance.

Improving the MMT-Extract plugins. The work presented in this dissertation was the first step for obtaining a native monitoring tool over 802.15.5/6LoWPAN. Yet, the plugin is currently in a beta state and the following considerations are required:

1. The IEEE 802.15.4 support requires to be finished. The beacon mode and the headers with the security were not included for this dissertation.

[1]Fuzzer attacks are those that send "legitimate" traffic with random values in one or more fields of the message

2. 6LoWPAN also needs to be finished. However, as discussed in Section 3.2, 6LowPAN is highly complex. And, with the original architecture for developing plugins for MMT, it would be necessary to develop more than 100 plugins only to cover all the possible header sizes for IPHC. To reduce this unnecessary complexity it is possible to use the previous IEEE 802.15.4 plugins for inspecting the following bytes in the packet to determine if it is IPHC (UDP, non-NHC, NHC with UDP, etc.), FRAG1, FRAGN, raw IPv6 or other type of dispatches. And of course, the support for other types of dispatches not discussed in this dissertation is also required.

3. The fragmentation over 6LoWPAN can be benefited from a discussion. For this proof-of-concept, the packets are stored and calculated when they are ready to assemble, and this is not an easy task. Yet, another alternative is to use two buffers for handling the fragmentation: one with the real packets transmitted over network, as it is done now, and another one with the translation of the IPv6 packet. However, the impact of the resources overhead of resources can be bigger than the benefits. The fake 802.15.4 frame for the new assembled packet should be removed also, as its advantages are not worth the extra overhead that currently brings.

4. The DTLS plugins also require to be finished. They can be used for 6LowPAN and standard IP networks (IPv4 or IPv6). But the plugins must be able to reassemble the DTLS records if they are fragmented, as well as to detect multiple records in a single message.

Expanding the MMT-Security rules. As discussed in Section 5.4, the MMT Security Rules are currently a proof of concept. The basics discussed in Section 3.4.1 are correct, however the implementation of the rules can be improved as follows:

1. SR1 and SR4 can be greatly benefited if MMT is able to track each DTLS Session flow. This is possible to do using the embedded functions.

2. Reactive actions are added to all the rules for executing proper actions, once one or more instances are violated.

3. The SR5 discussed in Section 3 must also be implemented. It was left outside this dissertation due to a constraint in the time.

Testing the implementation with the sensors. Due to time constraints, the methods discussed in this dissertation were tested only in the communication between sensors with DTLS sessions. However, the proposal for a secure design for WSN is shown in Figure 3.1. In this design, external clients are not expected to interact directly with the sensors, but instead only by means of a central server, which can work as a HTTP/CoAP proxy. This has the advantage of guaranteeing that only legitimate request reaches the sensors from the exterior of the 6LoWPAN subnetwork. Future works based on this proposed design can be the following:

1. Integrating TinyDTLS to nanocoap library of RIOT OS.

2. Implementation of the HTTP/CoAP proxy.

3. Performance tests with TinyDTLS/nanocoap implemented over the sensors, with a similar scenario than the used in Section 5.3 and with an emphasis on resource performance over power consumption.

4. Discussion about additional MMT nodes inside the WSN able to capture different portions of the wSN traffic and using a different channel than the sensors.

Finally, the following scenarios should be considered for future discussions:

1. Implementing a pseudo NAT over the 6BR with the purpose of using the 6BR's local addresses inside the WSN. This allows to elide the destination IPv6 addresses. And consequently, to use only 3 bytes for the IPv6 header instead of the 19 that are required for an external IPv6 address, recovering enough space for the DTLS overhead in the communication.

2. The implementation of a proxy for DTLS between the WSN and the exterior. Similar to the work proposed by Abeele et al. [85] (Section 2.5.3).

Appendices

Appendix A

ETSI TC M2M Threats and countermeasures

This annex contains the list of threats identified by the ETSI Technical Committee M2M [11].

A.1 Threats

The Table A.1 has the list of threats specific to the M2M service layer and its interfaces. The Table A.2 has the list of threats affecting the M2M functional requirements.

A.2 Countermeasures

The following is the description of the countermeasures (CM#) identified in tables A.1 and A.2.

- CM1: M2M long-term service keys (other than public keys) are stored in a Secured Environment [i.7] (whose tamper-resistance can be certified) which renders it unfeasible for the attacker to discover the value of keys by logical or physical means.
- CM2: Access Network credentials from which M2M long-term service keys are derived or bootstrapped are stored in a Secured Environment which renders it infeasible for the attacker to discover the value of the credentials.
- CM3: The Secured Environment will not reveal the value of stored keys, even to a management system or to an authorised representative of the M2M Core Operator, such as a System Administrator.
- CM4: The execution of Sensitive Functions (e.g. the derivation of further keys from long-term M2M service-layer keys) never causes long-term service keys to be exposed outside of the Secured Environments in which they are stored.
- CM5: The derivation or bootstrapping of M2M long-term service keys from Access Network credentials never causes the former or the latter to be exposed outside of the Secured Environments in which they are stored, or in which the derivation or bootstrapping processes take place.
- CM6: The Secured Environment containing the M2M long-term service keys is bound to the M2M Device or M2M Gateway, using logical and/or physical means.

ETSI Threat Number (ET#)	Likelihood	Impact	Risk	Countermeasures
(ET1) Discovery of Long-Term Service-Layer Keys Stored in M2M Devices or M2M Gateways	Moderate	Serious	Major	CM1. CM2, CM4, CM6
(ET2) Deletion of Long-Term Service-Layer Keys Stored in M2M Devices or M2M Gateways	Moderate	Serious	Major	CM8
(ET3) Replacement of Long-Term Service-Layer Keys Stored in M2M Devices or M2M Gateways	Moderate	Serious	Major	CM8
(ET4) Discovery of Long-Term Service-Layer Keys Stored in the SCs of the M2M Core	Severe	Enterprise	Critical	CM1, CM3, CM4, CM6
(ET5) Deletion of Long-Term Service-Layer Keys Stored in the SCs of an M2M Core	Severe	National	Critical	CM8
(ET6) Discovery of Long-Term Service-Layer Keys Stored in MSBF or MAS	Substantial	Enterprise	Critical	CM1, CM3, CM4, CM6
(ET7) Deletion of Long-Term Service-Layer Keys Stored in the MSBF/MAS	Substantial	Serious	Major	CM8
(ET8) Discover Keys by Eavesdropping on Communications Between Entities.	Substantial	High	Critical	CM9, CM10, CM11, CM12, CM13, CM14, CM15
(ET9) Modification of Data Stored in the M2M Service Capabilities	Severe	National	Critical	CM8, CM16, CM17, CM18
(ET10) Provisioning of non-Legitimate Keys	Severe	High	Critical	CM9, CM10, CM11
(ET11) Unauthorised or Corrupted Application and Service-Layer Software in M2M Devices/Gateways	Substantial	National	Critical	CM19, CM20
(ET12) Subverting the M2M Device/Gateway Integrity-Checking Procedures	Substantial	National	Critical	CM16, CM21, CM23
(ET13) Unauthorised or Corrupted Software in M2M Core	Severe	National	Critical	CM19, CM20
(ET14) Subverting the Integrity-Checking Procedures in the M2M Core	Severe	National	Critical	CM16 , CM22, CM24
(ET15) General Eavesdropping on M2M Service-Layer Messaging Between Entities.	Severe	High	Major	CM13, CM25
(ET16) Alteration of M2M Service-Layer Messaging Between Entities.	Substantial	High	Major	CM9, CM10, CM13, CM25
(ET17) Replay of M2M Service-Layer Messaging Between Entities.	Severe	High	Major	CM26
(ET18) Breach of Privacy due to Inter-Application Communications	Severe	Serious	Major	CM27
(ET19) Breach of Privacy due to Attacks on M2M Device/Gateway Service Capabilities	Severe	Serious	Major	CM8

Table A.1: Threats of Category 1 defined by the ETSI.

ETSI Threat Number (ET#)	Likelihood	Impact	Risk	Countermeasures
(ET20) Discovery of M2M long-term service-layer keysfrom knowledge of access-network keys	Substantial	Serious	Major	CM2, CM5. CM29
(ET21) Transfer of Module Containing Access-Network keys and/or M2M long-term keys to a different terminal/Device/Gateway	Substantial	Serious	Major	CM6, CM7, CM28, CM29

Table A.2: Threats of Category 2 defined by the ETSI.

- CM7: The Secured Environment containing the AN keys is bound to the specific M2M Device or M2M Gateway, using physical and/or logical means.
- CM8: Access to and/or the modification of stored Sensitive Data requires strong (i.e. cryptographic) authentication of the accessing/modifying party, followed by authorisation.
- CM9: A security association is established between the communicating entities, which provides for mutual authentication and confidentiality.
- CM10: The security association between communicating entities uses protocols which are proven to resist man-in-the-middle attacks.
- CM11: M2M service-layer keys in a provisioning message are encrypted for confidentiality, independently of any confidentiality provided by the messaging protocol.
- CM12: During provisioning of M2M service-layer keys, the protocol end-points for the encryption/decryption of those M2M service keys are Secured Environments.
- CM13: Communications whose security is anchored in M2M service-layer keys use session keys, i.e. keys with a limited lifetime which can be set by security policy. Session keys can be derived from M2M service-layer keys.
- CM14: Secured communications use only those cryptographic algorithms which are assessed as being fit for purpose, e.g. the length and randomness of cryptographic parameters is sufficient to resist a brute-force attack.
- CM15: Industry-accepted recommendations for the use of cryptographic algorithms in secured communications are followed.
- CM16: Stored Sensitive Data is integrity-protected, such that unauthorised modification can be detected.
- CM17: If the integrity-verification of stored data uses cryptographic keys (other than public keys), those keys are stored and used in a Secured Environment or Trusted Environment, according to where the measurement and verification processes take place.
- CM18: The integrity-verification of stored Sensitive Data takes place in a Secured Environment or a Trusted Environment.
- CM19: The integrity of executable functions can be verified.
- CM20: Policy-based action can be taken to prevent the use of functions which fail the integrity verification test.
- CM21: The measurement part of Integrity Validation of executables takes place in a Trusted Environment and the comparison with the RIVs takes place in a Secured Environment.
- CM22: The process of integrity-verification of executables in an M2M Core is protected against tampering.

- CM23: If the integrity-verification of executables uses cryptographic keys (other than public keys), those keys are stored and used in a Trusted Environment or Secured Environment, according to whether the keys are used in the measurement part or the comparison part of the Integrity Validation.
- CM24: If the integrity-verification of executables uses cryptographic keys (other than public keys), those keys are protected against discovery and against modification by an unauthorised entity.
- CM25: Communications between entities in the M2M system are protected by security associations which provide end-to-end confidentiality.
- CM26: The protocol includes functionality to detect if all or part of a message is an unauthorised repeat of an earlier message or part of a message.
- CM27: A framework is used by the SCL which provides methods for securely: assigning attributes to the resource container regarding an M2M Application's access rights; managing those attributes; enforcing the access rights.
- CM28: Means exist in the Access Network and/or M2M Core to prevent AN credentials from being used for purposes other than for connection of a Device/Gateway to its intended M2M service layer.
- CM29: Fraud management systems are deployed in the M2M Service Provider's Domain, which detect the use of duplicated M2M service keys and take appropriate action.

Appendix B

FIT IoT-Lab Hardware Information

B.1 WSN430 Open Node

The WSN430 open node is a WSN430 node based on a low power MSP430-based platform, with a fully functional ISM radio interface and a set of standard sensors. Concerning the radio, two versions are developed: version 1.3b presents an open 868 MHz radio interface while version 1.4 has an IEEE 802.15.4 radio interface at 2.4 GHz.

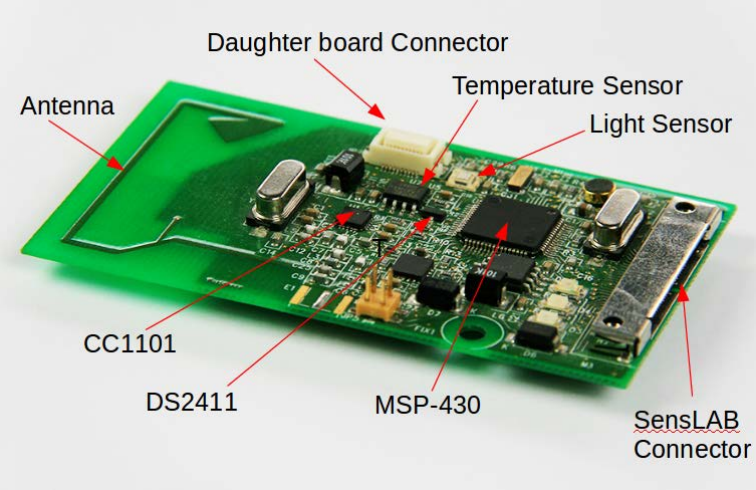


Figure B.1: WSN430 Open Node

In details, the main hardware components contained in the node are :

- Micro-controller MSP430 offering 48kbyte ROM, and 10kbyte RAM
- Sensors (Temperature, Sound, Ambient light)
- Radio: 868MHz radio interface for version WSN430 1.3b and 2.4GHz radio interface for version WSN430 1.4
- Serial Number: An EEPROM serial number is available thanks to a Maxim DS2411 chip, giving each node a unique identifier, readable by the MPS430 firmware over a 1-Wire interface.

- Flash memory: 1MByte
- Battery charger is controlled by a Microchip MCP73861 chip. It allows battery charge, and is supervised by MSP430 through 2 digital outputs
- Three LEDs (green, red, blue)

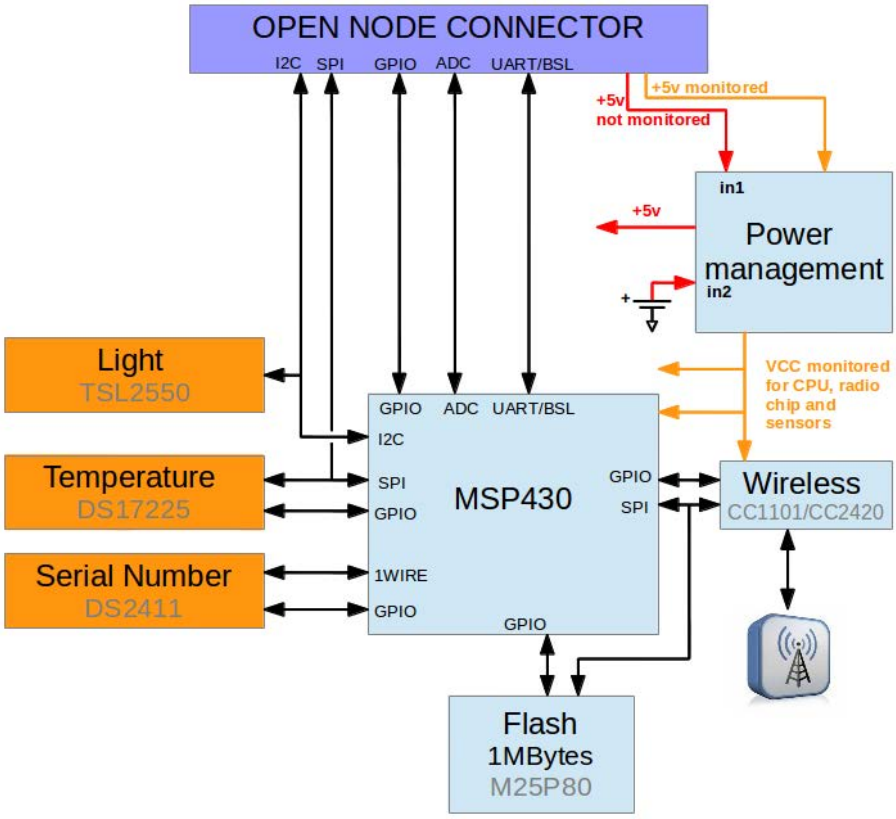


Figure B.2: WSN430 Open Node's hardware in detail

B.2 M3 Open Node

The M3 open node is based on a STM32 (ARM Cortex M3) micro-controller. Like the WSN node this next generation contains a set of sensors and a radio interface. Main evolutions are a more powerfull 32-bits processing, a new ATMEL radio interface in 2.4 Hz and more sensors.

In details, the main hardware components contained in the node are :

- ST2M32F103REY (72 MHz, 32bits, 64kB RAM)
- Radio interface 2.4 GHz AT86RF231
- Sensors
  - Light sensor ISL29020
  - Pressure sensor LPS331AP

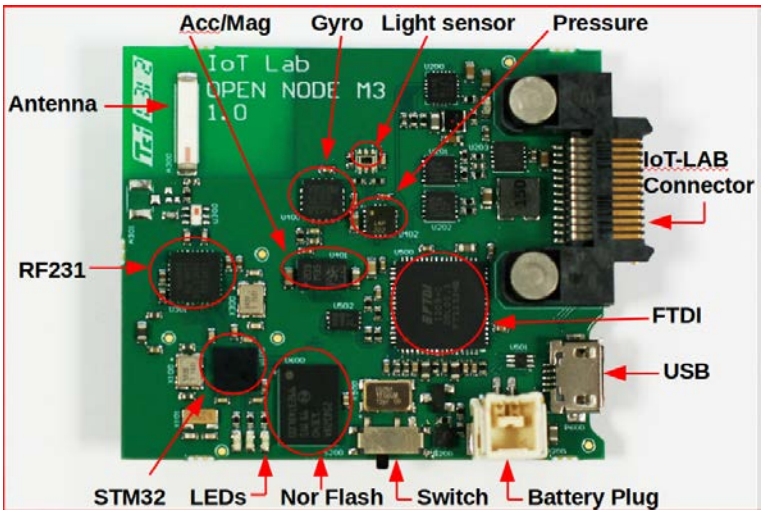


Figure B.3: M3 Open Node

  - Tri-axis accelerometer/magnetometer LSM303DLHC
  - Tri-axis gyrometer L3G4200D
- External Nor flash (128 Mbits) N25Q128A13E1240F
- Three LEDs (green, red, orange)
- 3,7 V LIPO battery (650 mAh)

B.3 A8 Open Node

The A8 open node is the most powerful IoT-LAB node and allows to run high-level OS like Linux. The main processor is a TI SITARA AM3505 (Arm Cortex A8) combined with a STM32 micro-controller and a radio interface. It enables to run applications used in advanced devices such as set-top box or smart-phone/tablet in order to concentrate sensors information coming from a wireless sensor network.

Main hardware components contained in this node are :

- A Variscite VAR-SOM-AM35 CPU which a high performance System On Module. It is a board of the shell based on a TI SITARA AM3505 (600 Mhz, 256 MB)
- A “co-microcontroller” based on ST2M32F103REY (72 MHz, 32bits, 64kB RAM) which controls :
  - Radio interface 2.4 GHz AT86RF231
  - Tri-axis accelerometer/magnetometer L3G4200D
  - Tri-axis gyrometer LSM303DLHC
- A USB device FTDI2232H to control UART and JTAG
- A GPS device MAXQ (optional)

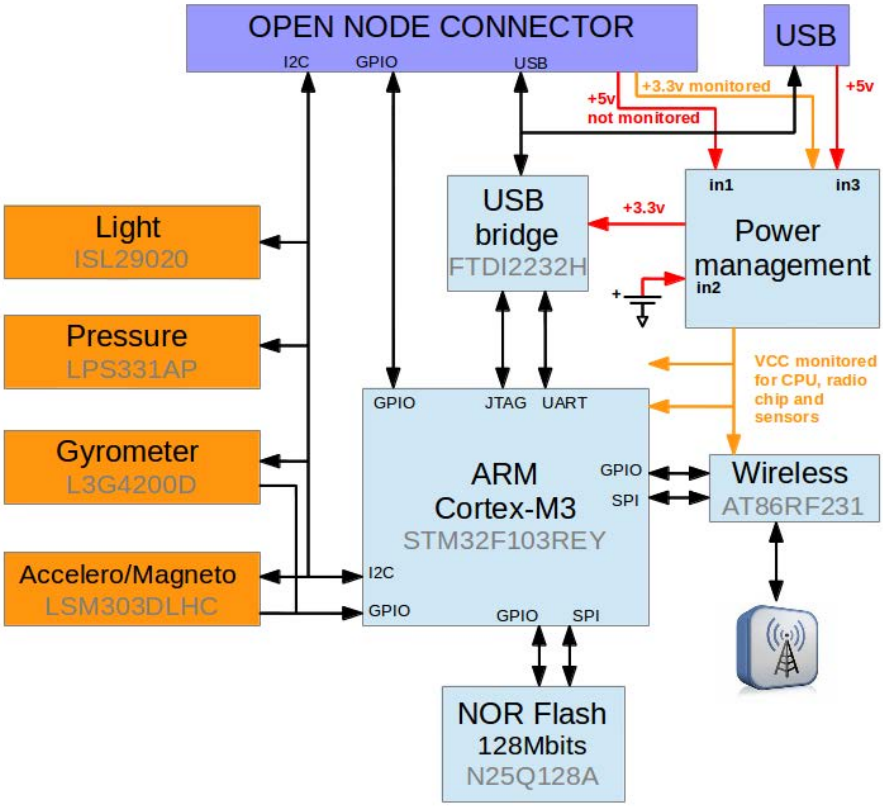


Figure B.4: M3 Open Node's hardware in detail

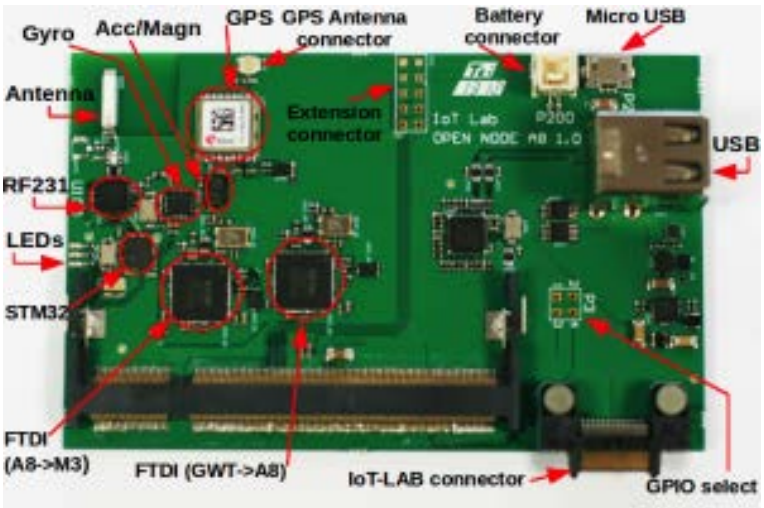


Figure B.5: A8 Open Node

- Three LEDs (green, red, orange)
- 3,7 V LIPO battery (600 mAh)

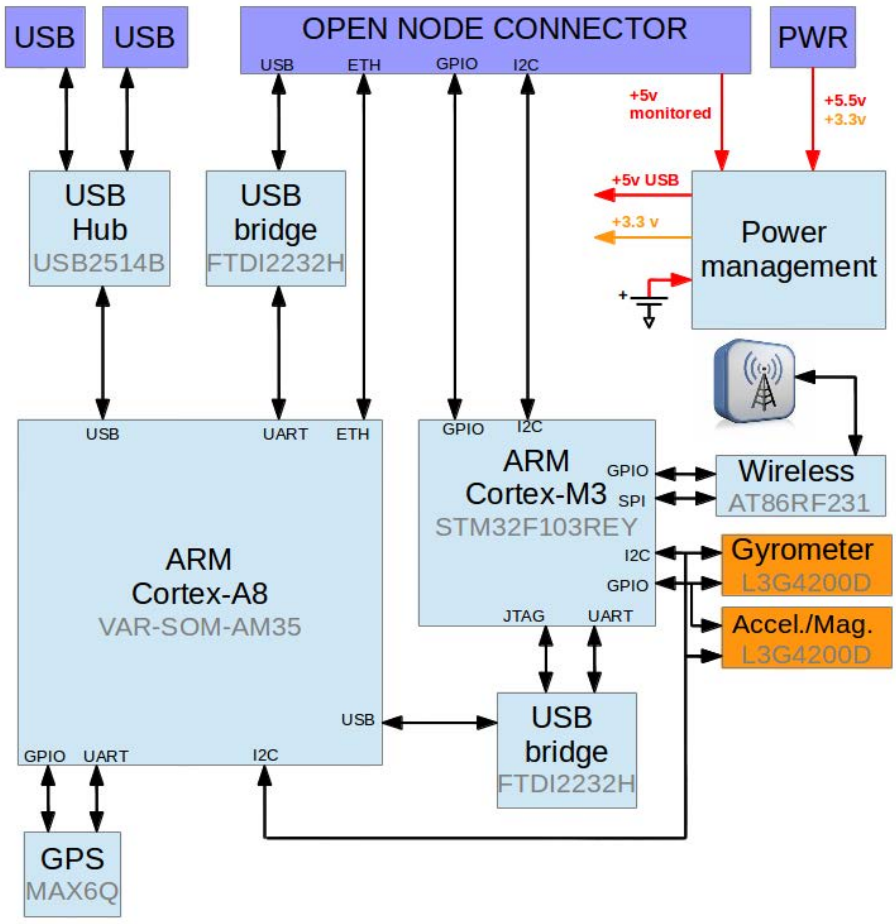


Figure B.6: A8 Open Node's hardware in detail

Appendix C

MMT Security rule example of ARP Poisoning

This annex contains the example given in the user guide for MMT Security [122].

Address Resolution Protocol (ARP) is a telecommunications protocol used for resolution of network layer addresses into link layer addresses, a critical function in multiple-access networks. ARP was defined by RFC 826 in 1982[1]. ARP spoofing[2](RFC5227[3]) is a technique used to attack a local-area network (LAN). ARP spoofing may allow an attacker to intercept data frames on a LAN, modify the traffic, or stop the traffic altogether. Figure C.1 represents an attack by a device using MAC address MAC3 (in red).

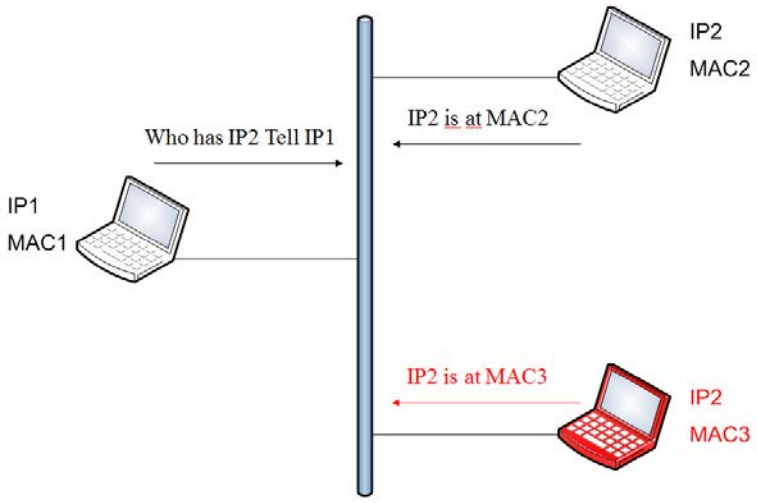


Figure C.1: ARP spoofing

Attack model: Following an ARP request (*who has*), a node receives more than one reply with different MAC addresses. This could mean that an IP duplication has been detected that

[1] https://en.wikipedia.org/wiki/Address_Resolution_Protocol
[2] https://en.wikipedia.org/wiki/ARP_spoofing
[3] http://www.networksorcery.com/enp/rfc/rfc5227.txt

could potentially be an ARP spoofing attack.

This attack behavior can be specified as follows:

$$(e_1; e_2)_{0,10} \;\; AFTER_{0,10} \;\; e_3$$

Where:

- **Event 1** (e_1): reception of ARP request message
- **Event 2** (e_2): reception of ARP reply message (src MAC2)
- **Event 3** (e_3): reception of ARP reply message with (scr MAC3) != (src MAC2)
- **AFTER$_{0,10}$** is an operator indicating that event e_3 occurs after the occurrence of the sequence of events e_1 ; e_2 with a time interval that represents the interval defined in RFC5227 between conflicting ARPs below which a host must not attempt to defend its address. This interval could be reduced or increased to detect spoofing attempts and avoid to many verdicts that are false positives.

This security attack is specified in pseudo-XML as follows:

Table C.1: Security property example

```
<property value="THEN" property_id="1" type_property="ATTACK" delay_max="2"
        description="(description of attack)">
   <operator value="THEN" delay_max="1" >
     <event value="COMPUTE" event_id="1"
           description="(description of event 1)"
           boolean_expression="(boolean expression allowing to detect event 1)"/>
     <event value="COMPUTE" event_id="2"
           description="(description of event 2)"
           boolean_expression="(boolean expression allowing to detect event 2))"/>
   </operator>
   <event value="COMPUTE" event_id="3"
         description="(description of event 3)"
         boolean_expression="(boolean expression allowing to detect event 3)"/>
</property>
```

For example; the following expression means that the event is valid if the packet received corresponds to the ARP protocol; the OPCODE is 2; SRC_PROTO is the same as SRC_PROTO of an event 1; and, SRC_HARD is the same as SRC_HARD of event 2. Where events 1 and 2 occurred before or will occur after depending on the order that the events should occur (defined by the time intervals specified).

```
boolean_expression="(((ARP.OPCODE == 2)&amp;&amp;
(ARP.SRC_PROTO == ARP.DST_PROTO.1))&amp;&amp;
(ARP.SRC_HARD == ARP.SRC_HARD.2))"
```

Appendix D

Compendium of files used for the experiment

The following is a list of the recorded traffic used for this dissertation.

- *FIT-2nodes.pcap* DTLS traffic generated with FIT IoT Lab and used in Section 5.3.1.
- *FIT-5nodes.pcap* DTLS traffic generated with FIT IoT Lab and used in Section 5.3.2.
- *FIT-10nodes.pcap* DTLS traffic generated with FIT IoT Lab and used in Section 5.3.3.
- *FIT-20nodes.pcap* DTLS traffic generated with FIT IoT Lab and used in Section 5.3.4.
- *FIT-50nodes.pcap* DTLS traffic generated with FIT IoT Lab and used in Section 5.3.5.
- *FIT-100no des.pcap* DTLS traffic generated with FIT IoT Lab and used in Section 5.3.6.
- *FIT-ACK-Conflicts01.pcap* Example of excessive retransmissions due to the IEEE 802.15.4 ACK request.
- *FIT-Fields.pcap* Testing a file for MMT's DPI property over fragmented messages.
- *FIT-Ports-Tests.pcap* Special testing files. RIOT OS was modified to suppress the port masks, avoiding the compression for valid port addresses.
- Cooja-Testing.pcap DTLS traffic generated with Contiki 3.0 and Cooja, affected by excessive retransmissions.
- *FIT-Contiki300.pcap* A file containing malicious attacks to the RPL protocol. Used for APCC [115].
- *FIT-ACK-Conflicts02.pcap* Additional samples of retransmissions, partially mitigated with an internal timer in the sensors.
- *FIT-Testing.pcap* This is the DTLS traffic generated for native RIOT OS interfaces. It makes use of IPv6 instead of 6LoWPAN.

All of these files are available online at https://zenodo.org/badge/latestdoi/75090070 .

Bibliography

[1] N. Salman, I. Rasool, and A. H. Kemp, "Overview of the ieee 802.15. 4 standards family for low rate wireless personal area networks," in *Wireless Communication Systems (ISWCS), 2010 7th International Symposium on*, pp. 701–705, IEEE, 2010.

[2] "Ieee standard for local and metropolitan area networks–part 15.4: Low-rate wireless personal area networks (lr-wpans) amendment 1: Mac sublayer," *IEEE Std 802.15.4e-2012 (Amendment to IEEE Std 802.15.4-2011)*, pp. 1–225, April 2012.

[3] S. Deering and R. Hinden, "Rfc 2460 - internet protocol, version 6 (ipv6) specification," tech. rep., IETF, december 1998.

[4] J. Hui and P. Thubert, "Rfc 6282 - compression format for ipv6 datagrams over ieee 802.15.4-based networks," tech. rep., IETF, September 2011. [accessed 11/11/2015].

[5] E. Rescorla and N. Modadugu, "Rfc 6347 - datagram transport layer security version 1.2," tech. rep., IETF, January 2012. [accessed 11/11/2015].

[6] Z. Shelby, K. Hartke, and C. Bormann, "Rfc 7252 - the constrained application protocol (coap)," tech. rep., IETF, June 2014. [accessed 11/11/2015].

[7] K. Hartke, "Rfc 7641 - observing resources in the constrained application protocol (coap)," tech. rep., IETF, September 2015. [accessed 11/09/2016].

[8] B. Wehbi, E. Montes de Oca, and M. Bourdelles, "Events-Based Security Monitoring Using MMT Tool," in *IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST), 2012*, pp. 860–863, April 2012.

[9] C. Bormann, M. Ersue, and A. Keranen, "Rfc 7228 - terminology for constrained-node networks," tech. rep., IETF, May 2014. [accessed 11/11/2015].

[10] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Rfc 4944 - transmission of ipv6 packets over ieee 802.15.4 networks," tech. rep., IETF, September 2007. [accessed 11/11/2015].

[11] "ETSI TR 103 167 V1.1.1 (2011-08) Machine-to-Machine Communications (M2M); Threat analysis and counter-measures to M2M service layer," tech. rep., ETSI (European Telecommunications Standards), 2011. accessed 11/06/2015.

[12] E. Baccelli, O. Hahm, M. Gunes, M. Wahlisch, and T. C. Schmidt, "Riot os: Towards an os for the internet of things," in *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*, pp. 79–80, IEEE, 2013.

[13] C. Pereira and A. Aguiar, "Towards Efficient Mobile M2M Communications: Survey and Open Challenges," *Sensors*, vol. 14, no. 10, pp. 19582–19608, 2014.

[14] H. A. Khattak, M. Ruta, and D. Sciascio, "CoAP-based Healthcare Sensor Networks : a survey," in *11th International Bhurban Conference on Applied Sciences & Technology (IBCAST)*, (Islamabad, Pakistan), pp. 499 – 503, IEEE, 2014.

[15] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, pp. 2787–2805, oct 2010.

[16] "Securing the internet of threats, issue two," January 2015. `https://www.theintelligenceofthings.com/article/securing-the-internet-of-threats/` [Online; Accessed 7/11/2016].

[17] B. Krebs, "Krebsonsecurity hit with record ddos," September 2016. `https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/` [Online; Accessed 21/09/2016].

[18] P. Paganini, "Ovh hosting hit by 1tbps ddos attack, the largest one ever seen," September 2016. `http://securityaffairs.co/wordpress/51640/cyber-crime/tbps-ddos-attack.html` [Online; Accessed 26/09/2016].

[19] S. Hilton, "Dyn analysis summary of friday october 21 attack," October 2016. `http://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/` [Online; Accessed 01/11/2016].

[20] A. Nixon, J. Costello, and Z. Wilkholm, "An after-action analysis of the mirai botnet attacks on dyn," October 2016. `https://www.flashpoint-intel.com/action-analysis-mirai-botnet-attacks-dyn/` [Online; Accessed 01/11/2016].

[21] J. P. Amaral, L. M. Oliveira, J. J. P. C. Rodrigues, G. Han, and L. Shu, "Policy and Network - based Intrusion Detection System for IPv6 - enabled Wireless Sensor Networks," in *Communications (ICC), 2014 IEEE International Conference on*, (Sydney, NSW), pp. 1796–1801, IEEE, 2014.

[22] J. Fried, P. Gaudry, N. Heninger, and E. Thomé, "A kilobit hidden SNFS discrete logarithm computation," *CoRR*, vol. abs/1610.02874, 2016.

[23] D. Minoli, *Building the internet of things with IPv6 and MIPv6: The evolving world of M2M communications*. John Wiley & Sons, 2013.

[24] R. A. Fuentes-Samaniego, A. R. Cavalli, J. A. Nolazco-Flores, and J. Baliosian, "A survey on wireless sensors networks security based on a layered approach," in *International Conference on Wired/Wireless Internet Communication*, pp. 77–93, Springer International Publishing, 2015.

[25] L. Catarinucci, R. Colella, and L. Tarricone, "Sensor data transmission through passive RFID tags to feed wireless sensor networks," pp. 1772–1775, IEEE, may 2010.

[26] D. Dressen, "Considerations for rfid technology selection," *Atmel Applications Journal*, vol. 3, pp. 45–47, 2004.

[27] A. Juels, "Rfid security and privacy: A research survey," *IEEE journal on selected areas in communications*, vol. 24, no. 2, pp. 381–394, 2006.

[28] L. Zhu and T.-S. Yum, "A critical survey and analysis of rfid anti-collision mechanisms," *Communications Magazine, IEEE*, vol. 49, pp. 214–221, May 2011.

[29] H. Ba, I. Demirkol, and W. Heinzelman, "Feasibility and benefits of passive rfid wake-up radios for wireless sensor networks," in *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, pp. 1–5, Dec 2010.

[30] S. F. Pileggi, C. E. Palau, and M. Esteve, "On the convergence between wireless sensor network and rfid: Industrial environment," in *8th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*, pp. 430–436, May 2010.

[31] B. Zhang, K. Hu, and Y. Zhu, "Network architecture and energy analysis of the integration of RFID and Wireless Sensor Network," pp. 1379–1382, IEEE, may 2010.

[32] S. Tozlu, "Feasibility of wi-fi enabled sensors for internet of things," in *2011 7th International Wireless Communications and Mobile Computing Conference*, pp. 291–296, July 2011.

[33] S. Tozlu and M. Senel, "Battery lifetime performance of Wi-Fi enabled sensors," in *2012 IEEE Consumer Communications and Networking Conference (CCNC)*, pp. 429–433, IEEE, jan 2012.

[34] M. Hulea, G. Mois, S. Folea, L. Miclea, and V. Biscu, "Wi-sensors: A low power wi-fi solution for temperature and humidity measurement," in *IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society*, pp. 4011–4015, Nov 2013.

[35] M. Desjardin, "Meet the new, low-power wi-fi standard," January 2016. `http://www.usatoday.com/story/tech/2016/01/05/meet-new-low-power-wi-fi-standard/78309526/` [Online; Accessed 7/11/2016].

[36] C. Gomez, J. Oller, and J. Paradells, "Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology," *Sensors*, vol. 12, pp. 11734–11753, aug 2012.

[37] H. Wang, M. Xi, J. Liu, and C. Chen, "Transmitting IPv6 Packets over Bluetooth Low Energy based on BlueZ," tech. rep., Nokia Research Cente, 2013.

[38] "Bluetooth 4.1 frequently asked questions," tech. rep., Bluetooth SIG, November 2013. `http://www.ineltek.com/wp-content/uploads/2015/09/20160330-Bluetooth_4.1_FAQ.pdf` [Online; accessed 08/04/2014].

[39] Z.-W. Alliance, "Z-wave." `http://z-wavealliance.org`, 2015. [Online; accessed 11/04/2014].

[40] C. Gomez and J. Paradells, "Wireless Home Automation Networks : A Survey of Architectures and Technologies," *IEEE Communications Magazine*, vol. 48, no. June, pp. 92–101, 2010.

[41] M. Knight, "Wireless security-how safe is z-wave?," *Computing & Control Engineering Journal*, vol. 17, no. 6, pp. 18–23, 2006.

[42] "Ant message protocol and usage," tech. rep., Dynastream, April 2014. `https://www.thisisant.com/developer/ant/ant-basics` [Online; accessed 08/04/2014].

[43] I. S. association, *IEEE Standard for Local and metropolitan area networks — Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)*. IEEE, 3 Park Avenue, New York, NY 10016-5997, USA, ieee std 802.15.4 2011 ed., September 2011. [accessed 28/03/2014].

[44] E. H. Callaway Jr, *Wireless sensor networks: architectures and protocols*. CRC press, 2003.

[45] R. Maley, "The new zigbee ip specification: Ipv6 control for low-power, low-cost devices," tech. rep., 2400 Camino Ramon, San Ramon, CA 94583, abril 2013. [Accessed 22/04/2013].

[46] H. C. Foundation, "HART what is hart?." `http://www.hartcomm.org/protocol/about/aboutprotocol_what.html`, note = [Online; Accessed: 03/04/2014], 2013.

[47] C. Alcaraz and J. Lopez, "A Security Analysis for Wireless Sensor Mesh Networks in Highly Critical Systems," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 40, pp. 419–428, jul 2010.

[48] X. Zhu, S. Han, A. Mok, D. Chen, and M. Nixon, "Hardware challenges and their resolution in advancing WirelessHART," pp. 416–421, IEEE, July 2011.

[49] J. Granjal, E. Monteiro, and J. Sa Silva, "Security for the Internet of Things : A Survey of Existing Protocols and Open Research Issues," *Communications Surveys & Tutorials*, vol. 17, no. 3, pp. 1294–1312, 2015.

[50] F. P. Rezha and S. Y. Shin, "Performance Analysis of ISA100.11a under Interference from an IEEE 802.11b Wireless Network," *IEEE Transactions on Industrial Informatics*, vol. 3203, no. c, pp. 1–1, 2014.

[51] A. Khurri, D. Kuptsov, and A. Gurtov, "On application of host identity protocol in wireless sensor networks," in *The 7th IEEE International Conference on Mobile Ad-hoc and Sensor Systems (IEEE MASS 2010)*, pp. 358–345, Nov 2010.

[52] J. Kim, J. Lee, H. K. Kang, D. S. Lim, C. S. Hong, and S. Lee, "An id/locator separation-based mobility management architecture for wsns," *IEEE Transactions on Mobile Computing*, vol. 13, no. 10, pp. 2240–2254, 2014.

[53] F. V. Meca, J. H. Ziegeldorf, P. M. Sanchez, O. G. Morchon, S. S. Kumar, and S. L. Keoh, "Hip security architecture for the ip-based internet of things," in *Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on*, pp. 1331–1336, IEEE, 2013.

[54] C. Hochleitner, C. Graf, D. Unger, and M. Tscheligi, "Making Devices Trustworthy : Security and Trust Feedback in the Internet of Things," in *Pervasive'12 Fourth International Workshop on Security and Privacy in Spontaneous Interaction and Mobile Phone Use*, (Newcastle, UK), 2012.

[55] W. Leister and T. Schulz, "Ideas for a Trust Indicator in the Internet of Things," in *SMART*, no. c, pp. 31–34, IARIA, 2012.

[56] D. Pietro, "Security and Trust Challenges in the Area of IoT," in *INNOSUMMIT*, 2012.

[57] A. Dunkels, "utrustit website," 2013. `www.utrustit.eu` [Online; accessed: 11/05/2014].

[58] L. Atzori, A. Iera, and G. Morabito, "From Smart Objects to Social Objects: The Next Evolutionary Step of the Internet of Things," *Communications Magazine*, vol. 52, no. January, pp. 97–105, 2014.

[59] M. Nitti, L. Atzori, and I. P. Cvijikj, "Network navigability in the social Internet of Things," in *2014 IEEE World Forum on Internet of Things (WF-IoT)*, (Seoul, Korea), pp. 405–410, IEEE, Mar. 2014.

[60] M. Nitti, R. Girau, L. Atzori, and S. Member, "Trustworthiness Management in the Social Internet of Things," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 26, no. 5, pp. 1253–1266, 2014.

[61] H. Nasiraee and J. B. Mohasefi, "A Novel Three Party Key Establishment Scheme in the Context of Internet-of-Things," in *Information Security and Cryptology (ISCISC)*, (Yazd), pp. 1–5, IEEE, 2013.

[62] S. Machara, S. Chabridon, and C. Taconet, "Trust-Based Context Contract Models for the Internet of Things," *2013 IEEE 10th International Conference on Ubiquitous Intelligence and Computing and 2013 IEEE 10th International Conference on Autonomic and Trusted Computing*, pp. 557–562, Dec. 2013.

[63] O. Garcia Morchon, S. Kumar, R. Hummen, and R. Struik, "Security Considerations in the IP-based Internet of Things draft-garcia-core-security-06," tech. rep., IETF.

[64] N. Vlajic and D. Stevanovic, "Performance Analysis of ZigBee-Based Wireless Sensor Networks with Path-Constrained Mobile Sink(s)," *2009 Third International Conference on Sensor Technologies and Applications*, pp. 61–68, june 2009.

[65] K. Zen, D. Habibi, S. Member, A. Rassau, and I. Ahmad, "Performance Evaluation of IEEE 802 . 15 . 4 for Mobile Sensor Networks," tech. rep., School of Engineering, Edith Cowan University, Joondalup, Australia, 2008.

[66] D. W. Carman, P. S. Kruus, and B. J. Matt, "Constraints and approaches for distributed sensor network security," Tech. Rep. 00-10, NAI LABS, september 2000.

[67] X. Chen, K. Makki, K. Yen, and N. Pissinou, "Sensor network security: a survey," *IEEE Communications Surveys & Tutorials*, vol. 11, no. 2, pp. 52–73, 2009.

[68] A. Wood and J. Stankovic, "Denial of service in sensor networks," *Computer*, vol. 35, pp. 54–62, Oct 2002.

[69] A. Mpitziopoulos, D. Gavalas, C. Konstantopoulos, and G. Pantziou, "A survey on jamming attacks and countermeasures in WSNs," *IEEE Communications Surveys & Tutorials*, vol. 11, no. 4, pp. 42–56, 2009.

[70] S. Anjali and M. Sharma, "Wireless Sensor Networks : Routing Protocols and Security Issues," in *5th ICCNT*, pp. 3–7, 2014.

[71] P. Kasinathan, C. Pastrone, and M. A. Spirito, "Denial-of-Service detection in 6LoWPAN based Internet of Things," in *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Network and Communications (WiMob)*, (Lyon, France), pp. 600–607, IEEE, 2013.

[72] A. Abduvaliyev, A.-S. K. Pathan, J. Zhou, R. Roman, and W.-C. Wong, "On the vital areas of intrusion detection systems in wireless sensor networks," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 3, pp. 1223–1237, 2013.

[73] K. Doddapaneni, E. Ever, O. Gemikonakli, L. Mostarda, and A. Navarra, "Effects of IDSs on the WSNs lifetime: Evidence of the need of new approaches," *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 907–912, 2012.

[74] Y. Liu, K. Liu, and M. Li, "Passive diagnosis for wireless sensor networks," *IEEE/ACM Transactions on Networking*, vol. 18, no. 4, pp. 1132–1144, 2010.

[75] S. Dawans and L. Deru, "Troubleshooting with foren6." `http://cetic.github.io/foren6/`, 2011. [Online; Accesed 7/01/2016].

[76] H. Sedjelmaci and S. M. Senouci, "A lightweight hybrid security framework for wireless sensor networks," in *2014 IEEE International Conference on Communications (ICC)*, pp. 3636–3641, June 2014.

[77] M. Z. A. Bhuiyan, G. Wang, J. Cao, and J. Wu, "Local monitoring and maintenance for operational wireless sensor networks," in *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 837–844, IEEE, 2013.

[78] S. Raza, L. Wallgren, and T. Voigt, "SVELTE: Real-time intrusion detection in the Internet of Things," *Ad Hoc Networks*, vol. 11, no. 8, pp. 2661–2674, 2013.

[79] J. Granjal, E. Monteiro, and J. S. Silva, "On the effectiveness of end-to-end security for internet-integrated sensing applications," in *Green Computing and Communications (GreenCom), 2012 IEEE International Conference on*, pp. 87–93, IEEE, 2012.

[80] J. Granjal, R. Silva, E. Monteiro, J. S. Silva, and F. Boavida, "Why is ipsec a viable option for wireless sensor networks," in *2008 5th IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, pp. 802–807, IEEE, 2008.

[81] S. Raza, S. Duquennoy, T. Chung, D. Yazar, T. Voigt, and U. Roedig, "Securing communication in 6lowpan with compressed ipsec," in *2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS)*, pp. 1–8, June 2011.

[82] S. Raza, H. Shafagh, K. Hewage, R. Hummen, and T. Voigt, "Lithe: Lightweight Secure CoAP for the Internet of Things," *IEEE Sensors Journal*, vol. 13, pp. 3711–3720, oct 2013.

[83] T. Kothmayr, C. Schmitt, W. Hu, M. Brunig, and G. Carle, "A DTLS based end-to-end security architecture for the Internet of Things with two-way authentication," in *Proceedings - Conference on Local Computer Networks, LCN*, (Clearwater, Florida), pp. 956–963, IEEE, 2012.

[84] M. Vučinić, B. Tourancheau, T. Watteyne, F. Rousseau, A. Duda, R. Guizzetti, and L. Damon, "Dtls performance in duty-cycled networks," in *Personal, Indoor, and Mobile Radio Communications (PIMRC), 2015 IEEE 26th Annual International Symposium on*, pp. 1333–1338, IEEE, 2015.

[85] F. Van den Abeele, T. Vandewinckele, J. Hoebeke, I. Moerman, and P. Demeester, "Secure communication in ip-based wireless sensor networks via a trusted gateway," in *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2015 IEEE Tenth International Conference on*, pp. 1–6, IEEE, 2015.

[86] R. A. Fuentes-Samaniego, A. R. Cavalli, and J. A. Nolazco-Fores, "An analysis of secure m2m communication in wsns using dtls," in *2016 IEEE 36th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pp. 78–83, IEEE, June 2016.

[87] R. A. Fuentes-Samaniego, "Descubrimiento y barrido de nodos remotos en ipv6," Master's thesis, Instituto Tecnológico y de Estudios Superiores de Monterrey (ITESM), Campus Monterrey, Av. Eugenio Garza Sada 2501 Sur, Tecnológico, 64849 Monterrey, N.L., Mexico, December 2013.

[88] L. Mokdad, A. Abdelli, and J. Ben-Othman, "Detection of greedy behavior in wsn using ieee 802.15 protocol," in *2014 IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems*, pp. 106–111, IEEE, 2014.

[89] M. Franceschinis, C. Pastrone, M. A. Spirito, and C. Borean, "On the performance of zigbee pro and zigbee ip in ieee 802.15. 4 networks.," in *WiMob*, pp. 83–88, 2013.

[90] D. Striccoli, G. Boggia, and L. A. Grieco, "A markov model for characterizing ieee 802.15. 4 mac layer in noisy environments," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 8, pp. 5133–5142, 2015.

[91] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 22–32, 2014.

[92] A. F. Skarmeta, J. L. Hernández-Ramos, and M. V. Moreno, "A decentralized approach for security and privacy challenges in the internet of things," in *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pp. 67–72, March 2014.

[93] C. Hennebert and J. D. Santos, "Security Protocols and Privacy Issues into 6LoWPAN Stack: A Synthesis," *IEEE Internet of Things Journal*, vol. 1, no. 5, pp. 384–398, 2014.

[94] V. Sharma, *Understanding Constrained Application Protocol*. EXILANT, 2014.

[95] P. Wouters, H. Tschofenig, J. Gilmore, S. Weiler, and T. Kivinen, "Rfc 7250 - using raw public keys in transport layer security (tls) and datagram transport layer security (dtls)," tech. rep., IETF, June 2014. [accessed 11/11/2015].

[96] D. Guinard, V. Trifa, F. Mattern, and E. Wilde, "From the internet of things to the web of things: Resource-oriented architecture and best practices," in *Architecting the Internet of Things*, pp. 97–129, Springer, 2011.

[97] O. Hersent, D. Boswarthick, and O. Elloumi, *The internet of things: Key applications and protocols*. John Wiley & Sons, 2011.

[98] "Efficient xml interchange (exi) format 1.0 (second edition)," tech. rep., W3C, february 2014. `http://www.w3.org/TR/exi/` [Online; accessed 10/05/2012].

[99] D. Caputo, L. Mainetti, L. Patrono, and A. Vilei, "Implementation of the exi schema on wireless sensor nodes using contiki," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*, pp. 770–774, IEEE, 2012.

[100] N. Gligori, I. Dejanovi, and S. Krco, "Performance Evaluation of Compact Binary XML Representation for Constrained Devices," in *Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on*, (Baercelona), pp. 1–5, IEEE, 2011.

[101] A. Castellani, M. Gheda, N. Bui, M. Rossi, and M. Zorzi, "Web services for the internet of things through coap and exi," in *Communications Workshops (ICC), 2011 IEEE International Conference on*, pp. 1–6, June 2011.

[102] P. Wehner, C. Piberger, and D. Göhringer, "Using json to manage communication between services in the internet of things," in *Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), 2014 9th International Symposium on*, pp. 1–4, May 2014.

[103] F. Pacini, F. A. Aderohunmu, P. Pagano, A. Azzara, M. Petracca, and S. Bocchino, "Performance Analysis of Data Serialization Formats in M2M Wireless Sensor Networks," *12th The European Conference on Wireless Sensor Networks*, pp. 7–8, FEBRUARY 2015.

[104] C. Bormann and P. Hoffman, "Rfc 7049 - concise binary object representation (cbor) layer security (dtls)," tech. rep., IETF, October 2013. [accessed 11/11/2015].

[105] Y. Ohba and S. Chasko, "Performance Evaluation on 6LoWPAN and PANA in IEEE 802.15.4g Mesh Networks," *journal of cyber security*, vol. 2, no. 3, pp. 329–350, 2014.

[106] P. Moreno Sanchez, R. Marin Lopez, and A. F. Gomez Skarmeta, "PANATIKI: a network access control implementation based on PANA for IoT devices.," *Sensors (Basel, Switzerland)*, vol. 13, pp. 14888–917, jan 2013.

[107] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges," *Ad Hoc Networks*, vol. 10, pp. 1497–1516, sep 2012.

[108] M. Castro, A. J. Jara, and A. F. Skarmeta, "Smart lighting solutions for smart cities," in *Advanced information networking and applications workshops (WAINA), 2013 27th international conference on*, pp. 1374–1379, IEEE, 2013.

[109] D. Boswarthick, O. Elloumi, and O. Hersent, *M2M communications: a systems approach*. John Wiley & Sons, 2012.

[110] D. Yang, Y. Xu, and M. Gidlund, "Coexistence of ieee802. 15.4 based networks: A survey," in *IECON 2010-36th Annual Conference on IEEE Industrial Electronics Society*, pp. 2107–2113, IEEE, 2010.

[111] A. Milinkovi, E. E. Belgrade, S. Milinkovi, C. Belgrade, L. Lazi, and I. T. Belgrade, "Choosing the right RTOS for IoT platform," *Professional Symposium INFOTEH®-JAHORINA*, vol. 14, no. Iii, pp. 504–509, 2015.

[112] T. Watteyne, X. Vilajosana, B. Kerkez, F. Chraim, K. Weekly, Q. Wang, S. Glaser, and K. Pister, "Openwsn: a standards-based low-power wireless development environment," *Transactions on Emerging Telecommunications Technologies*, vol. 23, no. 5, pp. 480–493, 2012.

[113] O. Bergmann, S. Gerdes, and C. Bormann, "Simple keys for simple smart objects," 2012.

[114] A. Dunkels, "The contikimac radio duty cycling protocol," tech. rep., 2011.

[115] V. H. La, R. Fuentes, and A. R. Cavalli, "A novel monitoring solution for 6lowpan-based wireless sensor networks," in *Communications (APCC), 2016 22nd Asia-Pacific Conference on*, pp. 230–237, IEEE, 2016.

[116] E. Fleury, N. Mitton, T. Noel, and C. Adjih, "FIT IoT-LAB: The Largest IoT Open Experimental Testbed," *ERCIM News*, p. 4, Apr. 2015.

[117] W. Mallouli, B. Wehbi, and E. M. de Oca, "Online Network Traffic Security Inspection Using MMT Tool," in *Systems Testing and Validation Workshop 2012*, pp. 23–31, 2012.

[118] J. Pokhrel, B. Wehbi, A. Morais, A. Cavalli, and E. Allilaire, "Estimation of QoE of video traffic using a fuzzy expert system," in *Consumer Communications and Networking Conference (CCNC), 2013 IEEE*, pp. 224–229, Jan 2013.

[119] F. B. Abreu, A. Morais, A. Cavalli, B. Wehbi, and E. Montes de Oca, "An Effective Attack Detection Approach in Wireless Mesh Networks," in *27th International Conference on Advanced Information Networking and Applications Workshops*, pp. 1450–1455, IEEE, mar 2013.

[120] V. H. La, R. Fuentes, and A. R. Cavalli, "Network monitoring using mmt: An application based on the user-agent field in http headers," in *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, pp. 147–154, IEEE, 2016.

[121] R. Fuentes, A. Cavalli, W. Mallouli, and J. Baliosian, "Monitoring-based validation of functional and performance aspects of a greedy ant colony optimization protocol," in *Network Operations and Management Symposium (LANOMS), 2015 Latin American*, pp. 69–72, IEEE, 2015.

[122] E. M. de Oca, "Mmt security user manual," tech. rep., Montimage, August 2013. [Accessed 02/03/2014].

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre: Monitorage des réseaux des capteurs sans fils (WSN) - Application a l'interopérabilité sécurisée

Mots clés: WSN; Internet des objets; surveillance; 802.15.4; 6LoWPAN; CoAPS;

Résumé: La formule "Internet of Things" a pris du sens à la fois au sein de la communauté public et de recherche. La raison principale est qu'en 2011, le nombre d'objets connectés à Internet surpassent le nombre d'humains en ligne, et il est attendu qu'en 2020, le nombre d'objets connectés dépassent les 20 billions. Etant donné la présence d'un grand nombre de plateformes hétérogènes qui composent l'IoT, notre intérêt se focalise sur les Reseaux de Capteurs (WSN), qui sont composés des petits dispositifs avec des contraintes de ressources (capacité de mémoire faible, processeur de faible puissance et faible puissance matérielle) qui collectent un ou plusieurs types de données. Presque toutes les recherches menées à ce jour reposent sur la standardisation de protocoles de communication, l'amélioration de la performance, l'optimisation de la consommation de ressources, etc.

La sécurité a été relégué au second plan, dû principalement au faibles ressources disponibles sur les capteurs. Cependant, les données collectées dans de multiples scénarios peuvent être très sensibles. Les données doivent être stockées de manière sûr et doivent être transmises de manière sûr d'un point à un autre. Le travail développé dans cette thèse définit les mécanismes permettant de garantir une communication sûr entre les capteurs. Et aussi fournissant des outils natifs pour le monitorage des communications, afin de valider ces mécanismes directement sur le réseaux.

Title: Wireless Sensors Networks (WSN) monitoring - Application to secure interoperability

Keywords: WSN; Internet of Things; surveillance; 802.15.4; 6LoWPAN; CoAPS;

Abstract: The denominated "Internet of Things" (IoT) has been getting relevance in both the public and research communities. The main reason is that on 2011, the number of "objects" connected to the Internet surpassed the number of humans online, and is expected that for 2020, the number of objects exceeds the amount of 20 billion. Because of the high number of heterogeneous platforms that composed the IoT, our interest is centered around the Wireless Sensors Networks (WSN), which are composed by small devices with constrained resources (small amount of memory, small power processor, and small power supply) that collect one or more type of data. Almost all the research conducted to date relies on standardizing the communication protocols, ameliorating the performance, optimizing the resource consumption, etc.

Security has been relegated to a second plane, due mainly to the low resources available on the sensors. However, the data collected in many scenarios can be highly sensitive. The data must be stored in a safe way and must be transmitted in a safe approach from the origin to the destiny. The work developed in this dissertation defines mechanisms to guarantee the safety of the communication between sensors. And, providing native tools for the monitoring of the communication, to validate these mechanisms directly on the network.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France