Universidade do Minho
Escola de Engenharia

Pedro de Barbosa Mendonça Diogo

A Complete Internet of Things Solution for Real-Time Web Monitoring

Outubro de 2014

Pedro de Barbosa Mendonça Diogo A Complete Internet of Things Solution for Real-Time Web Monitoring UMinho | 2014

Universidade do Minho
Escola de Engenharia

Pedro de Barbosa Mendonça Diogo

A Complete Internet of Things Solution for Real-Time Web Monitoring

Dissertação de Mestrado
Ciclo de Estudos Integrados Conducentes ao
Grau de Mestre em Engenharia de Comunicações

Trabalho efetuado sob a orientação do
Professor Doutor Nuno Vasco Lopes
Professor Doutor Luis Paulo Reis

Outubro de 2014

Anexo 3

DECLARAÇÃO

Nome Pedro de Barbosa Mendonça Diogo

Endereço electrónico: Pedrodiogoum@gmail.com

Telefone: 912490912

Número do Bilhete de Identidade: 13610029

Título dissertação □/tese □ A Complete Internet of Things Solution for Real-Time Web Monitoring

Orientador(es): Nuno Vasco Lopes

Luis Paulo Reis

Ano de conclusão: 2014

Designação do Mestrado ou do Ramo de Conhecimento do Doutoramento: Mestrado Integrado em Engenharia de Comunicações

Nos exemplares das teses de doutoramento ou de mestrado ou de outros trabalhos entregues para prestação de provas públicas nas universidades ou outros estabelecimentos de ensino, e dos quais é obrigatoriamente enviado um exemplar para depósito legal na Biblioteca Nacional e, pelo menos outro para a biblioteca da universidade respectiva, deve constar uma das seguintes declarações:

1. É AUTORIZADA A REPRODUÇÃO INTEGRAL DESTA TESE/TRABALHO APENAS PARA EFEITOS DE INVESTIGAÇÃO, MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;

Universidade do Minho, ___/___/_____

Assinatura: _____

Terminado o trabalho, é tempo de refletir sobre os contributos e apoios de colegas, amigos e familiares.

Assim sendo, começo por agradecer a orientação do Professor Doutor Nuno Vasco Lopes e Professor Doutor Luis Paulo Reis: obrigado pelo apoio dado ao longo do trabalho; pela flexibilidade e confiança em permitir ajustar, sempre que achasse preciso, o tema e todo o trabalho envolvente; e pela oportunidade de aprendizagem e crescimento profissional e pessoal. Foi um prazer trabalhar e discutir os avanços nesta área.

Aos meus Pais, aos quais simplesmente não consigo agradecer o suficiente por esta oportunidade. Só eles sabem o quanto tudo isto significou para mim, e quanto aprecio tamanha oferenda. E, por fim mas não em último, à minha namorada, Ana Pereira, pela imensurável paciência e todo o grande apoio que foi dado.

Abstract

The Internet of Things (IoT) is the new Internet. It does not just connect people, it connects everything and everyone. It can be seen as a new network where objects, people and services talk to one another, providing valuable information. This information and their relationship with the most variable objects brings everything together, providing us, humans, ultimately, a better quality of life. When we possess the ability to sense and process different types of information from the world around us, complex intelligent and autonomous systems can be developed to work together for a better end. However, to achieve such a thing, there is a lot of work to be done: standards and architectures are needed so that all these entities know how to exchange information, with each other, in an open way. As the IoT is a complex organism composed of, in a simplified manner, wireless sensor networks (), network-located services and applications, an architecture that fits all solutions is difficult to find.

Therefore, this dissertation presents the related work done in this field, the typical current IoT systems described in the literature, and how such a system can be deployed using state-of-the-art open-source tools compliant to the most relevant standards. The developed system breaks down the current paradigm of vertical solutions, by adopting latest standards and protocols specifically designed for IoT. To demonstrate what kind of solutions can be build using this system architecture, a real-time fitness web monitoring testbed was set. The solution is composed of a low-cost node reporting, in real-time, the monitored heart rate to a Web page, while also possessing the ability to share this data with any other interested entity. To demonstrate the potential of IoT and how the adoption of proper standards and adequate protocols allow for the development of complex and richer applications, a complete solution was designed. With this solution, it was possible to demonstrate how one can share the sensed data, in a standardized way, to other potential interested parties (doctors or nursing facilities, for instance) and how such standardized architectures can be tweaked to meet specific requirements for special scenarios, like the real-time web monitoring here developed. To validate this solution, an experiment was set up, demonstrating that there is, indeed, a need to use adequate protocols for this type of monitoring.

Ultimately, this work showcases what kind of impact the IoT will have in the health-sector, breaking the current paradigm of silo solutions, making way for new, smarter, scalable and interoperable solutions, when adopting this technology in a proper way.

Resumo

A Internet das Coisas é o futuro da Internet. Não vai ligar apenas pessoas, mas sim tudo e todos. Pode ser visto como uma nova rede onde objetos, serviços e pessoas se interligam, fornecendo informação relevante. A coordenação inteligente do resultado desta informação é o que oferecerá qualidade de vida. Com a capacidade de monitorizar diferentes aspetos do dia-a-dia, processar diferentes tipos de informação, retirar conhecimento relevante de todo este processo e definir ações para determinados conhecimento, novos sistemas autonómicos, complexos e inteligentes podem ser criados que, juntos, trabalham para um propósito maior. Contudo, para que tudo isto seja possível, há ainda trabalho a fazer. Para coordenar e orientar todo este fluxo de informação, é necessária a criação de normas, protocolos apropriados e arquiteturas comuns, para que as diferentes entidades envolvidas saibam trocar a informação de forma tipificada. Além disso, uma arquitetura apenas pode não chegar, visto que a Internet das Coisas abrange diferentes áreas como redes de sensores sem fios (RSSF) e serviços a nível de rede (cloud) em diferentes sectores como, por exemplo, saúde, logísticas e aplicações domésticas, uma arquitetura apenas pode não ser a mais indicada para todas as soluções.

Neste sentido, esta dissertação apresenta, inicialmente, o trabalho relacionado nesta área, os tipos de sistemas que são naturalmente encontrados na literatura, e quais os problemas presentes nestas soluções auto-apelidadas de Internet das Coisas. Como forma de solução, é apresentado como estes podem agora ser desenvolvidos, usando as mais recentes normas e arquiteturas, bem como as ferramentas grátis e de código aberto que são compatíveis com essas mesmas normas. Seguindo esta lógica, é apresentado o sistema desenvolvido que, de uma forma geral, quebra o paradigma atual de criação de soluções verticais, adotando as mesmas normas e protocolos mencionados anteriormente, e que foram concebidos especificamente para a Internet das Coisas. Tendo em conta que esta tecnologia terá um grande impacto na área da saúde, foi desenvolvida uma aplicação de teste. O sistema é composto por um nó, de baixo custo, que monitoriza o batimento cardíaco e envia os dados para uma entidade que os disponibiliza, tanto em tempo real, via Web, como a quaisquer outras entidades que estejam interessadas nestes dados. Esta solução tem como objetivo demonstrar o potencial máximo que está inerente à Internet das Coisas: soluções mais ubíquas, abertas e escaláveis que, conjuntamente, possibilitam a criação de serviços e aplicações mais ricas e inteligentes. Visto que a solução desenvolvida tem que ser capaz de oferecer uma monitorização Web em tempo-real dos dados recolhidos, foi desenvolvido um mecanismo especifico e mais adequado para a aplicação em causa. Ainda, tendo em conta que o conjunto de protocolos utilizados não estão incorporados na norma adotada para o sistema Internet das Coisas, foi feita uma experiência onde é demonstrada a necessidade destas normas incorporarem protocolos adequados para as diferentes soluções tipicamente encontradas em sistemas que recorrem a esta tecnologia.

O objetivo desta dissertação é, portanto, demonstrar os problemas tipicamente encontrados nos atuais sistemas Internet das Coisas, como estes podem ser atualmente resolvidos utilizando as normas corretas e os protocolos adequados à solução em si, e o potencial que esta tecnologia apresenta, caso os sistemas sejam devidamente desenhados.

Contents

Contents **xi**

List of Figures **xv**

List of Tables **xix**

Nomenclature xxi

1 Introduction **1**

1.1 Motivation 1

1.2 Objectives 1

1.3 Document Structure 2

2 IoT Technology and Protocols **3**

2.1 Physical, Data and Network layers 4

2.1.1 IEEE 802.15.4 4

2.1.2 6LoWPAN 4

2.2 Transport and Application layers 6

2.2.1 CoAP 7

2.2.2 MQTT 15

2.2.3 MQTT-SN 17

2.2.4 Comparison . . . 17
2.3 Conclusion . . . 18

3 State of the Art **21**

3.1 ETSI M2M . . . 21
3.1.1 Architecture . . . 22
3.1.2 OneM2M . . . 25
3.2 OMA Lightweight M2M . . . 28
3.2.1 Overall High Level Functionalities . . . 28
3.2.2 Architecture . . . 28
3.2.3 Object Model . . . 30
3.3 IoT Reference Model and Guidelines . . . 31
3.3.1 IoT-A . . . 31
3.3.2 IPSO Alliance . . . 33
3.4 Comparison . . . 34
3.5 Literature Review . . . 35
3.6 Conclusion . . . 40

4 IoT Solution for Real Time Fitness Web Monitoring **43**

4.1 System Architecture . . . 43
4.2 Testbed Solution . . . 49
4.3 Solution Overview . . . 50
4.4 Strengths . . . 52
4.5 Conclusion . . . 56

5 System Implementation, Configuration and Validation **59**

5.1 Implementation Overview . . . 59

5.2 Software Configuration and Deployment 62
5.3 Hardware Configuration 73
5.3.1 Arduino 73
5.3.2 Raspberry Pi 74
5.4 Proof of Concept 76
5.4.1 Real-Time Monitoring Validation 80
5.5 Conclusion 84

6 Conclusion and Future Work **87**

Bibliography **91**

List of Figures

2.1 A 6LoWPAN example [16] . . . 5

2.2 6LoWPAN routing model “Route-Over” [16] . . . 7

2.3 [19] CoAP Message Format . . . 8

2.4 Lookup of a CoAP Resource Directory . . . 9

2.5 HTTP-CoAP proxying mechanism. [27] . . . 10

2.6 CoAP - Proxying and caching. [27] . . . 11

2.7 First interaction with an HTTP-CoAP proxy - listing CoAP resources to test. . . . 11

2.8 Second interaction with an HTTP-CoAP proxy - GET request. . . . 12

2.9 Final interaction with an HTTP-CoAP proxy - response of previous GET request. . . . 13

2.10 CoAP’s Observation feature example [29] . . . 14

2.11 MQTT: Pub/Sub protocol[34] . . . 15

2.12 MQTT’s wildcard support example . . . 16

2.13 MQTT-SN Architecture [34] . . . 18

3.1 ETSI’s M2M High Level System Architecture [44] . . . 23

3.2 SCL’s standardized resource structure tree[45] . . . 24

3.3 Simple use of SCL resources to exchange data (adapted from [48]) . . . 25

3.4 oneM2M Functional Architecture [59] . . . 26

3.5 oneM2M - Common Service Functions residing in CSE [59] . . . 27
3.6 oneM2M - Primitives [61] . . . 27
3.7 OMA Lightweight M2M Architecture [62] . . . 29
3.8 LWM2M Object Model [63] . . . 30
3.9 IoT Architectural Reference Model building blocks . . . 32
3.10 Representation of the System Architecture defined in [72] . . . 36
3.11 Architecture overview of [76] . . . 37
3.12 Monitoring system architecture of [79] . . . 39
3.13 Simplified data flow of [79] . . . 39

4.1 IoT System Architecture Proposal . . . 44
4.2 Current M2M/IoT Silos . . . 45
4.3 IoT System Architecture Proposal - Trusted Applications . . . 47
4.4 IoT System Architecture Proposal - Mobility . . . 48
4.5 Solution's Architecture . . . 51
4.6 Solution´s Protocol Stack . . . 51
4.7 Solution's architecture after ETSI M2M adoption. . . . 53
4.8 Node.js's event loop [93] . . . 55

5.1 Raspberry Pi's software modules overview . . . 60
5.2 6lbr's Smart Bridge operation mode [114] . . . 66
5.3 Gateway Interworking Proxy (GIP) Fluxogram . . . 68
5.4 Arduino - GIP Registration . . . 69
5.5 GIP - Update a new Application's Content Instance . . . 70
5.6 Real-Time Monitoring via Web Sockets . . . 72
5.7 Arduino nodes (left: temperature sensor; right: pulse sensor) . . . 75

5.8 Raspberry Pi with NooliberryT 75

5.9 6lbr's Web page displaying the Arduino node (under Routes) 77

5.10 6lbr's Web page displaying the Arduino node (under Routes) 78

5.11 Multicasted RA message at Border Router 78

5.12 Special multicast NS message sent for DAD purpose 79

5.13 Solicited NA message as a reply to the previously received NS 79

5.14 Arduino node registering application via CoAP PUT message 79

5.15 GIP registering new application and updating content 80

5.16 Applications registered at GSCL 81

5.17 Delay difference between MQTT and standard ETSI M2M subscription mechanism 83

5.18 ETSI M2M standard subscription method vs MQTT 85

List of Tables

3.1 Example of a LWM2M Object defined by a 3rd Party SDO and registered at OMNA. 34

5.1 GIP - Registration of a new Application 69

5.2 Traffic comparison when using ETSI M2M standard subscription method and MQTT 82

Nomenclature

6LoWPAN	IPv6 over Low power Wireless Personal Area Networks
ACK	Acknowledgement
ACL	Access Control List
AE	Application Entity
API	Applicaiton Programming Interface
ARM	Architectural Reference Model
BLE	Bluetooth Low Energy
BPM	Beats per Minute
CoAP	Constrained Application Protocol
CRUD	Create, Update, Retrieve and Delete
CSEs	Common Services Entities
CSF	Common Services Functions
DA	Device Application
DAD	Duplicate Address Detection
DIP	Device Interworking Proxy
DNS	Domain Name System
DODAG	Destination-Oriented Directed Acyclic Graph

DTLS	Datagram Transport Layer Security
ETSI	European Telecommunications Standards Institute
FP7	Seventh Framework Programme
GIP	Gateway Interworking Proxy
GPRS	General Packet Radio Service
GSCL	Gateway Service Capability Layer
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IoT	Internet of Things
IoT-A	Internet of Things Architecture
IP	Internet Protocol
iPSO	Internet Protocol for Smart Objects Alliance
ISPs	Internet Service Providers
LWM2M	Lightweight M2M
M2M	Machine to Machine
MAC	Media Access Control
MQTT	Message Queue Telemetry Transport
NA	Network Application
NACK	Negative-Acknowledge
NAT	Network Address Translation
ND	Neighbor Discover
NIP	Network Interworking Proxy
NSCL	Network Service Capability Layer

NTP	Network Time Protocol
OMA	Open Mobile Alliance
OMNA	OMA's Naming Authority
OTA	Over-the-Air
PIO	Prefix Information Option
QoS	Quality of Service
RA	Router Advertisement
RIO	Route Information Option
ROLL	Routing over Low Power and Lossy Networks
RPL	Ripple
RS	Router Solicitation
SAA	Stateless Address Auto-Configuration
SCL	Service Capability Layer
SDO	Standards Development Organization
SOTA	State-of-the-Art
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
VPN	Virtual Private Network
VPS	Virtual Private Server

WPAN	Wireless Personal Area Network
WSN	Wireless Sensor Network
XML	Extensible Markup Language

Chapter 1

Introduction

1.1 Motivation

There is no denying that the Internet of Things is evolving and it will generate billions of dollars in the upcoming years. Gartner, the world's leading information technology research and advisory company, said, in December 2013, that the IoT will grow to 26 billion units in 2020, resulting in 1.9$ trillion in global economic value-add through sales into diverse end markets [1]. Similarly, Cisco said that it will create, from 2013 to 2022, a 14.4$ Trillion of value at stake for companies and industry [2]. Since we are talking about connecting everything to the Internet, there is an unimaginable amount of business opportunities involved. Sectors like industry, logistics and health are a few of the major ones embracing IoT.

Currently, IoT suffers from fragmentation, interoperability and scalability issues. There are protocols and standards defined specifically for IoT, but they are not being adopted by every IoT system designer, which leads to, to put it simply, interoperability issues. For IoT to grow to its full potential, there is yet work to be done in this field. Currently, there are standards developed by, for instance, IETF, OMA and ETSI that will help overcome this issue, but it is still a work in progress, and its wide adoption is still uncertain. IoT is not just about connecting things to the internet - it is expected for those things and other complex services to work together, for a specific end, autonomically.

1.2 Objectives

This Masters Thesis' main goals are to demonstrate the most problematic issues around M2M and IoT systems/applications and how they can be avoided using latest architectural standards and protocols specifically tailored for IoT. This will demon-

strate how smarter, richer and more powerful applications could be born if M2M/IoT system designers adopted such features.

As e-health is one of the biggest areas where IoT will have a major impact on [3], a testbed on this field was designed. Specifically, a use case of mobile real-time monitoring of vital signs (heart rate), over the Web and by a Personal Trainer, using protocols and frameworks specifically designed for IoT. This testbed was chosen to prove how one specific implementation of this IoT System may serve other systems, where applications make use of the open standard and protocols - in other words, how different IoT Systems could interoperate and make way for smarter applications that provide, ultimately, better quality of live. The work also takes into account the cost of such system (as low cost as possible), openness (use of open and free source code only) and the incorporation of tools that are compliant to current and/or future final versions of the specifications developed by entities such as ETSI. This last case offers future-proof capabilities to the developed system.

1.3 Document Structure

In order to achieve the aforementioned claims, the document is structured in a way to ease the understanding of such objectives. Chapter 2 starts by introducing the most common technology and protocols present in typical IoT Systems, ranging from the physical layer up to the application layer. It is important to understand the complexity of such protocols and the resulting proliferation of work in this field, due to the technology's inherent complexity. For the IoT to work, proper protocols must be defined at each layer to seamlessly work together, as one IoT System is not simply composed of one protocol from one layer only. Continuing the same line of thought, chapter 3 introduces the most recent IoT/M2M architectural standards and guidelines which are much needed for correctly designing a real IoT System. This is still a work in progress, but the chapter already mentions the most promising ones and those which are already finalized. They form the first step to achieve a more robust Internet of Things. After this state-of-the-art analysis, the same chapter showcases the typical IoT Systems found in the literature, illustrating how the commonly found bad IoT System designing practices restricts its potential to fully grow to the desired level. Taking all this into account, Chapter 4 details what is the ideal Internet of Things and how such a thing can be achieved. It starts by introducing an high-level view of the desired Internet of Things and how it can be developed using the state-of-the-art architectural standards and protocols presented in chapter 2 and 3; with this previously mentioned guidelines, it is next detailed the implemented system and how it will make use of these state-of-the-art standards and protocols to operate in compliance to what was previously described as the desired IoT. Finally, chapter 5 details the implementation of such solution: which hardware and software was used and how it all works together as a complete IoT System. The last chapter concludes with an analysis of the whole work and how the solution could be improved, at hardware and software level, to get as close as possible to the previously mentioned desired IoT System

Chapter 2

IoT Technology and Protocols

As IoT as a concept started to emerge, commercial and industrial applications started to adopt different technologies and protocols all across their solutions. The literature, presented at section III, is also an evidence of this statement. Most of these IoT solutions adopts a specific architecture where the common sensing nodes transmit data to a gateway, which, itself and only itself, is then connected to the internet. There is no real end-to-end connectivity between the different nodes and the internet, as they do not support any means to communicate with the Internet world. In general, these systems possess limitations in terms of interconnectivity, reachability, and scalability.

However, major entities such as IETF and IEEE acknowledged this problem and started to work on standards and protocols specifically designed for IoT solutions. With their efforts, it is now possible for nodes to be directly connected to the internet while providing device management capabilities, standard interfaces for M2M and direct communication with regular web services and applications. All of this combined solves most of the common problems with present solutions: reachability, interoperability, scalability and security. The following sections will detail the on going efforts of different Organizations, Alliances and Task Forces on building protocols and standards to support IoT and M2M solutions.

When deploying an IoT application, one protocol is usually chosen, given the fact that some have advantages over others. However, they are chosen accordingly to the specific application in mind and parameters like messaging pattern, network capabilities, security, etc.). Most of the systems usually use either CoAP, MQTT, MQTT-SN or XMPP as protocols for application data, as Cisco acknowledges [4], and, as an example, they are already being used by commercially successful applications like Skynet [5], Pinoccio [6] and mbed (former Sensinode) [7]. At a networking level, 6LoWPAN [8], RPL [9], and ZigBee IP are the most widely used in commercial applications (Thingsquare [10], for example) and in the literature itself, as the next sections denotes. Besides these protocols and standards, there are also relevant and much needed frameworks and architectures combining all entities and services present in typical IoT and M2M systems to provide interoperability all

across the domains and between one another. Capabilities like device management, bootstrapping, abstract semantics and standard interfaces are now needed for current IoT and M2M systems adopting the previously mentioned protocols. ETSI's M2M architecture, oneM2M and Open Mobile Alliance's (OMA) Lightweight M2M are also detailed next.

The next section will then detail the above mentioned technology, protocols and standards providing a full picture of the common stack typically adopted by IoT systems - from physical to the application layer.

2.1 Physical, Data and Network layers

2.1.1 IEEE 802.15.4

IEEE 802.15.4 [11] is a standard which defines low power wireless embedded radio communications at 2.4 GHz (Global) and sub-GHz: 915 MHz (North America) and 868 MHz (Europe). 915 MHz and 2.4 GHz frequency bands are internationally unlicensed ones (ISM - Industrial, Scientific and Medical radio bands), while 868 MHz is defined for non-specific short range devices. As a result of this frequency difference, variable data rates of 20, 40 and 250 Kbps are possible to achieve. It was designed to be used in lossy and low power networks (LLN) where nodes must sleep most of the time to achieve multi month or multi year battery life. These typical constrained network often present unstable radio environment and the physical layer packet size is very limited (~100 bytes) [12]. IEEE 802.15.4's physical payload size has a maximum of 127 bytes, with 72-116 bytes of actual size after framing, addressing and optional security (128-bit AES encryption) at link layer.

2.1.2 6LoWPAN

IETF's Working Group 6LoWPAN started to develop, in 2005, a solution to bring IPv6 to IEEE 802.15.4. Two years after, in 2007, two initial RFCs (RFC 4919 and RFC 4944) were approved stating the problem [13] and specifying the format [14]. However, these two were later updated by RFC 6282 [15] and RFC 6775 [8] which state the two main functions of 6LoWPAN: compression format for IPv6 Datagrams over different IEEE 802.15.4-based networks (6LoWPAN was, initially, designed for 802.15.4 and its specific features [16]) and neighbor discovery optimization.

This standard adaptation layer allows small low power devices to connect to the whole Internet, providing interoperability between low power devices and existing IP devices. As IPv4 addresses are becoming scarce, IPv6 was the natural way to go. With it, current small islands of wireless devices can be transformed into stub networks of the Internet where IP packets can be sent from or destined to. When bringing IPv6 connectivity to such devices, criteria like limited energy power, resources and lossy links had to be kept in mind. 6LowPAN WG decided to work on IPv6 standard and leverage existing standards, rather than creating something completely new, thus, enabling the use of standard socket API (UDP), ICMP, and

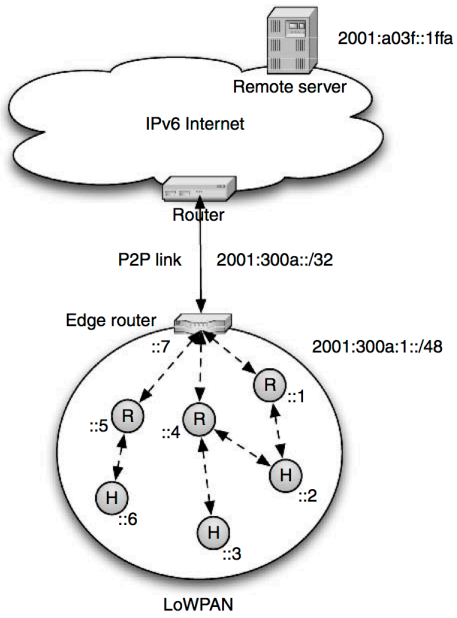

Figure 2.1: A 6LoWPAN example [16]

direct end-to-end IPv6 routing, which, in turn, allows for a better integration with the whole Internet and the development of new applications.

One of the major problems the Working Group encountered, was related to IPv6 typical frame size. IPv6 MTU alone is 1280 bytes long which mean that some times, a frame of around 400 or 500 bytes will have to be fragmented to fit in 802.15.4's 127 byte frame. Fragmentation, albeit not very efficient, was made possible to allow backward compatibility with the rest of the Internet. Also, because of this 127 byte frame limitation, compression must be done. With IEEE 802.15.4 MAC header (up to 25 bytes with no security, or 46 bytes with AES-128), 6LoWPAN has roughly only 100 bytes available to work with; and if IPv6 (40 bytes) and UDP (8 bytes) headers were used in there, the resulting payload space would be 53 bytes only. 6LoWPAN header compression transforms these 48 bytes down to just 6 bytes, in its best scenario, and to about 11-12 bytes in worst case scenario [17]. With this compression, the payload is now 108 bytes long, which is enough for IoT protocols like CoAP, MQTT-SN, RTP or custom UDP protocols. As 6LoWPAN was designed for typical Wireless Sensor Networks, mesh routing is also supported.

IP addressing in 6LoWPAN is very identical to any IPv6 network. They are automatically formed from LoWPAN's prefix and link-layer address of the wireless interface (EUI-64). A typical 128 bytes IPv6 address is composed of a 64 bytes prefix and a 64 bytes interface identification (IID). In 6LoWPAN the IID part of the IPv6 address is generated trough the link-layer address of the wireless interface and the prefix is obtained from Neighbor Discovery's (ND) Router Advertiser (RA) messages as in a typical IPv6 network. The use of link-layer EUI-64 allows for what is called a Stateless Address Configuration. This known

link-layer addresses alongside an also known prefix is what allows for a high header compression ratio [16]. That is why it is called Stateless Address Configuration - there is no need to discuss and agree on what is going to be compressed, the Edge Router just compresses the headers and all nodes understand it [17]. With this, IPv6 address can be omitted most of the time because every node knows their particular participation on the LoWPAN and the hierarchical IPv6 address [16]. Figure 2.1 taken from [16] represents a typical 6LoWPAN.

RPL

IETF has also defined a routing protocol for this kind of network and is called RPL (Ripple), which is detailed next.

Forwarding and Routing can be performed at different layers: Layer 2 ("Mesh-Under") and Layer 3 ("Route-Over"). Since IP routing is domain-centric, 6LoWPAN can adopt one routing algorithm while the domain above the Edge Router adopts its own. IPv6 and routing are different things, meaning that IPv6 is open to any algorithm. Route-Over means that IP routing always involves IP layer as noted in Figure 2.2. IETF has created a working group called ROLL (Routing over Low Power and Lossy Networks)[18] to design a new routing algorithm that takes these network characteristics in mind. Knowing that this algorithm would be used in M2M typical communications where data flows in a standard way (from nodes to the internet, and vice-versa), they have designed RPL [9]. RPL algorithm has a Proactive distance-vector approach. It is proactive because it creates a routing table before it is actually needed. With RPL each host only knows how to route to the Router above, which, in turn, knows how to route to the Edge Router; similarly, and following M2M typical data flow, the Edge Router knows only how to route in an downstream way. In these type of LoWPANs there is no need to maintain routing information about everything and everyone and that is why it is so efficient. Also important about it is that it supports duplicable backup routes - this means that if one host can not communicate to one of its linked Routers (Router's failure, or any other reason), it can use the backup link to route the data to another Router. This actually happens in real time and is much needed when the LoWPAN has more than one Edge Router (which is possible, to extend its range, just like in WiFi) [16].

Mesh-under happens at Layer 2 and is actually invisible to 6LowPAN. This means that it uses 64-bit EUI-64 or 16 bits short address from Link Layer.

2.2 Transport and Application layers

With 6LoWPAN layer, efficient routing algorithm and the RF technology to transmit data, all that is missing now is a proper transport and application protocol to be used in IoT. At the transport layer, TCP is not commonly used in these low power and lossy networks due to its complexity and lack of efficient dealing of wireless loss and packet error rate; for this matter, UDP is the de-facto transport protocol used with 6LoWPAN. At the application layer, there are several options. Given the fact that

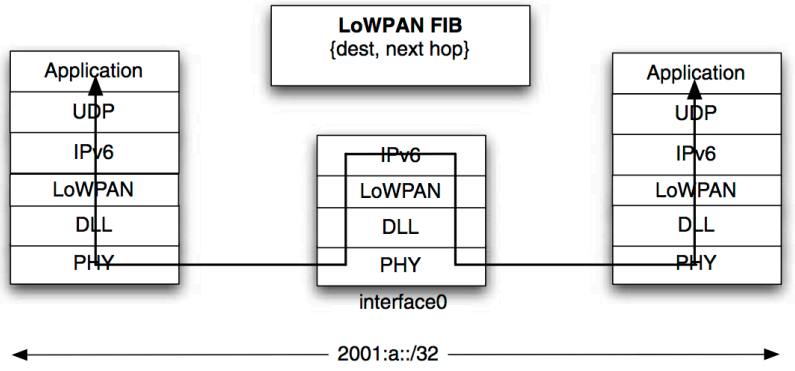

Figure 2.2: 6LoWPAN routing model "Route-Over" [16]

TCP should be avoid, the most efficient ones currently being used in a constrained device are the ones who support UDP, like: RTP, SIP, MQTT-SN, SNMP and CoAP, being the last three the most widely employed [16]. HTTP and FTP are too heavy for these devices and networks. The main UDP-based protocols are analyzed further and, to summarize, a comparison chart is presented, between those that are more IoT-oriented.

2.2.1 CoAP

CoAP (Constrained Application Protocol [19]) was developed by IETF Working Group called CoRE (Constrained RESTful Environments [20]) as an answer to the lack of Web Services capabilities present in this constrained environment. With CoAP, it is possible to embed a Web Service on small low power devices, enabling interaction with the whole Web due to its REST-architecture nature, resource discovery mechanism and HTTP binding. It is a simple protocol that implements HTTP's REST model, in a small 4-byte header, while avoiding most of its complex features. Furthermore, it was designed having sleeping nodes in mind, providing security, thanks to DTLS, multicast support, built-in resource discovery, reliability, and a new observe feature. REST model, enabling a notification architecture [21]. This is a very important protocol as it enables a many-to-many communication paradigm, via interaction with other services available on the web. Although it is often used with UDP as the underlying protocol, it is also possible to bind to SMS or TCP. This makes it easy to adopt Device Managing capabilities over cellular network like OMA's Lightweight M2M protocol [22], or LWM2M, which is also here analyzed.

Methods, Transactions messages and URI

Just like HTTP, GET, POST, PUT and DELETE methods are also present in CoAP. Fig2.3 represents the message format. The Code field can represent methods (GET=1, POST=2, PUT=3, DELETE=4) or response codes, which are a subset of adapted

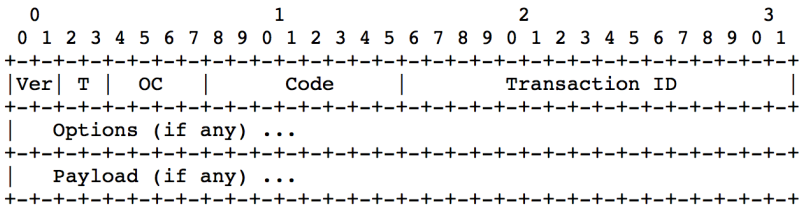

Figure 2.3: [19] CoAP Message Format

HTTP status codes and CoAP-specific.

It is possible to have synchronous messaging, when using Confirmable messages (“T = 0” field - Transaction messages). When sending confirmable requests, the server either piggy-backs the response with an Acknowledgement (ACK) or responds with a Reset (RST) if the confirmable has indeed arrived or if it was received but with missing context, respectively. Confirmable requests allows also for automatic re-transmission of the same request (same message ID), if a timeout, which is associated with every confirmable request, occurs. If the message is non-confirmable, then there is obviously no ACK reply from the server.

As it is an embedded web transfer protocol, it also provides support for URI as part of its REST mode. This URI support is very important because, with it, it is possible for the Web Server to provide a resource discovery mechanism which applications can interact to. The URI scheme is as follows for normal CoAP and CoAP with security enabled via DTLS [19].

"coap:" "//" host [":" port] path-abempty ["?" query]
or
"coaps:" "//" host [":" port] path-abempty ["?" query]

Resource Discovery

With CoAP's built-in Resource Discovery, it is possible to access a standard well-known URI to list all the available resources on that Web Service and their respective URIs. CoRE Working Group has also defined a Link Forma, called CoRE Link Format [23], for these URI to be presented, but it is not a mandatory representation. However, they possess the advantage of returning a link-header format compatible with HTTP, which eases the interaction with typical web applications through web linking. The standard URI to list all the registered resources's links is /.well-known/core. After the GET request, it is returned, along with each URI, the resource type, content type, and if it is actually observable or not. Interacting with this Resource Directory is pretty straightforward: nodes register their resources with a simple POST with a link (like CoRE's Link Format, for example); they can refresh it with PUT and delete their entries with DELETE. The lookup can be made, as said

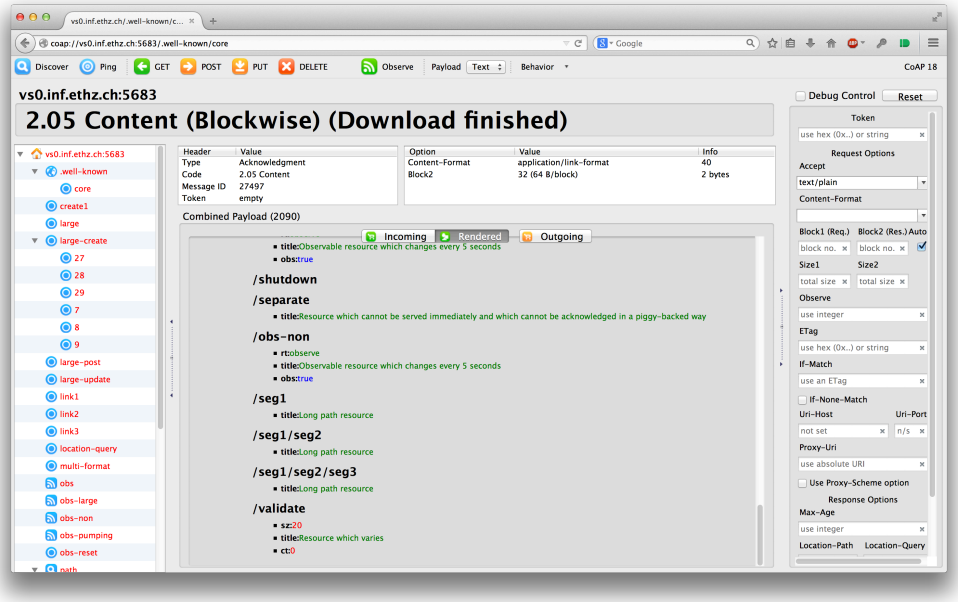

Figure 2.4: Lookup of a CoAP Resource Directory

before, with a GET request using COAP or by full fledged web applications using HTTP.

In IoT and M2M systems where devices must communicate autonomously with one another, this discovery mechanism is a must because, with it, devices do not need to know before hand how to reach the resource present in another device (actuator, for example), they can just lookup the Resource Directory and, with the use of known semantics present in each URI (Resource Type, and Interface Description, for example) interact with the returned one(s) it intends to. It is also possible to parse attributes in the query to filter the exact resource it is looking for. All of this without human intervention.

Fig 2.4 depicts a lookup on a public CoAP test server provided by Research Group for Distributed Systems at ETH Zurich [24]. This demonstration was possible using Firefox's extension called Copper (Cu), a CoAP protocol handler [25]. The GET request to the standard well-known URI returned a list of resources preset on that CoAP's server Resource Directory. Each of these returned URIs also provide its resource and content type and the observable flag.

It is possible to get semantics on these returned links, like "rt=ipso:dev-ser", indicating that the specific resource (/dev/ser, for instance) adopted the Application Framework designed by IPSO [26]. This is very important for interoperability purposes and is properly detailed in the dedicated "IPSO Alliance" section. LWM2M [22], as another example, is a very powerful protocol which defines data semantics to be used with CoAP, where resources are managed as standardized objects. As LWM2M is primarily a simple Device Management protocol that can be used with CoAP, it is also detailed further, given its importance.

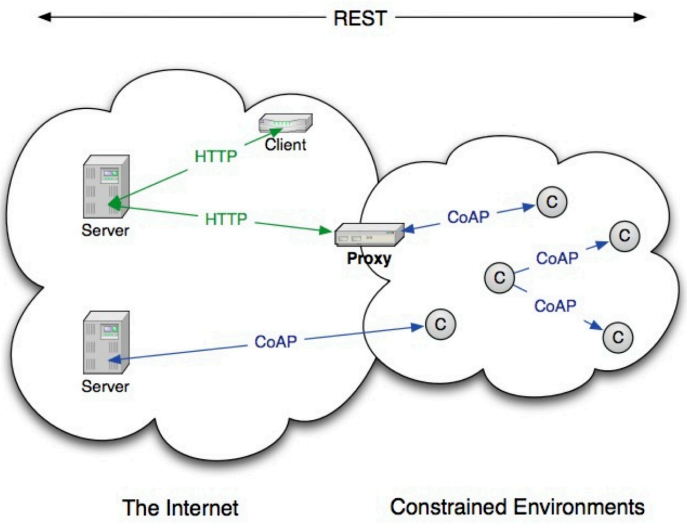

Figure 2.5: HTTP-CoAP proxying mechanism. [27]

HTTP and CoAP Proxying

It is possible to proxy to and from HTTP, as seen in Fig 2.5 This allows for interconnectivity all the way from the device domain to the network domain, using a proxy either at the network level or gateway's, as represented. Fig 2.6 represents this proxying example together with caching mechanism. This is an important feature for constrained environments, as it is easy to think of a network application constantly querying a constrained device - with this caching feature, the application does not need to implement CoAP itself since there is a proxy on the gateway and, secondly, because it saves the device's battery and network bandwidth by having only one interaction for each certain amount of time (CoAP's max-age option).

By deploying an HTTP-CoAP proxy, HTTP clients are able to interact with the CoAP server. This is possible by appending a coap or coaps URI in the HTTP request. Once the proxy entity receives this HTTP GET request, it forwards its builtin CoAP's URI GET request method to the CoAP server and responds to the original HTTP client. Likewise, the same mechanism is applied in a CoAP-HTTP proxy.

Fig 2.7, 2.8 and 2.9 showcase an example of an HTTP-CoAP interaction using Firefox as an HTTP client. The web page used to test this functionality was the publicly available http://coap.me [28] - Carsten Bormann's, one of the authors of CoAP's RFC, debug website. The first Fig 2.7, shows the initial interaction with the website. This webpage has a built-in HTTP-CoAP proxy on its side, and, as Fig 2.8 denotes, the previously clicked button shown in Fig 2.7 has now returned a number of links we can interact to - in other words, resources available and their URI. Once we send an HTTP GET request with CoAP's URI inside, http://coap.me/coap/://coap.me/test (which the link pointed out by the arrow), we get the HTTP 200 OK response code from the proxy and the resulting Payload response from CoAP's GET request, as shown in Fig 2.9.

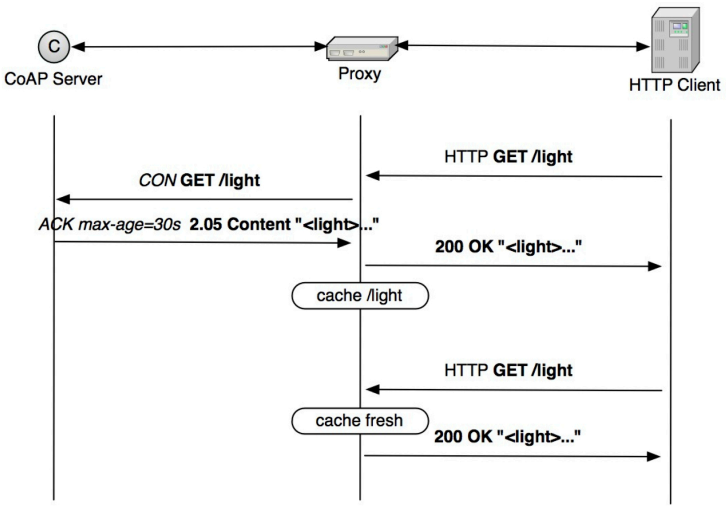

Figure 2.6: CoAP - Proxying and caching. [27]

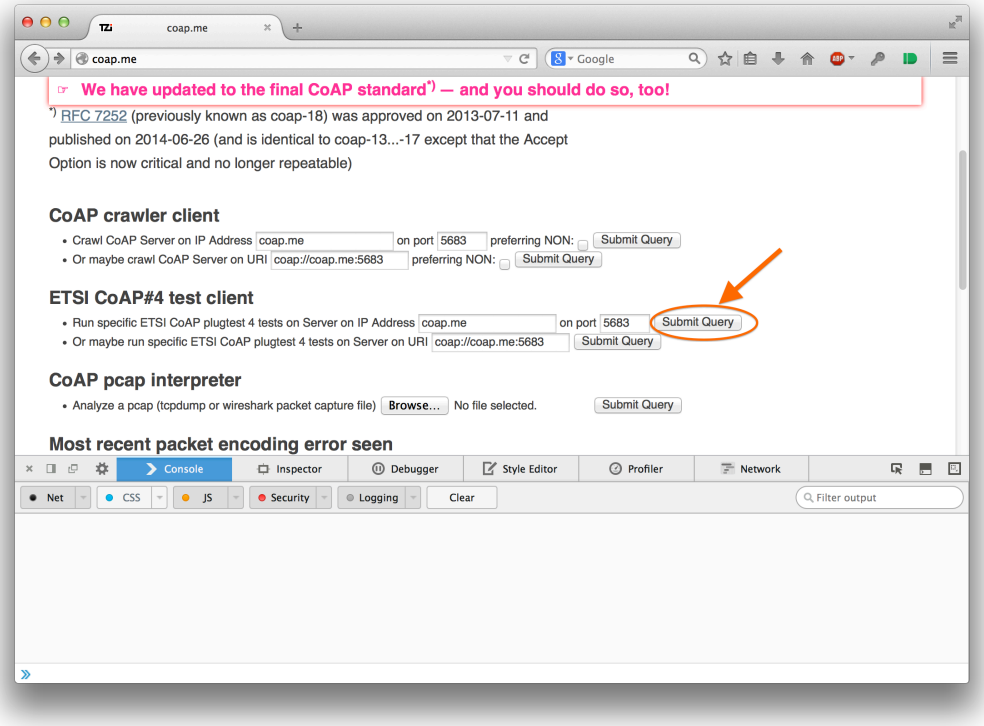

Figure 2.7: First interaction with an HTTP-CoAP proxy - listing CoAP resources to test.

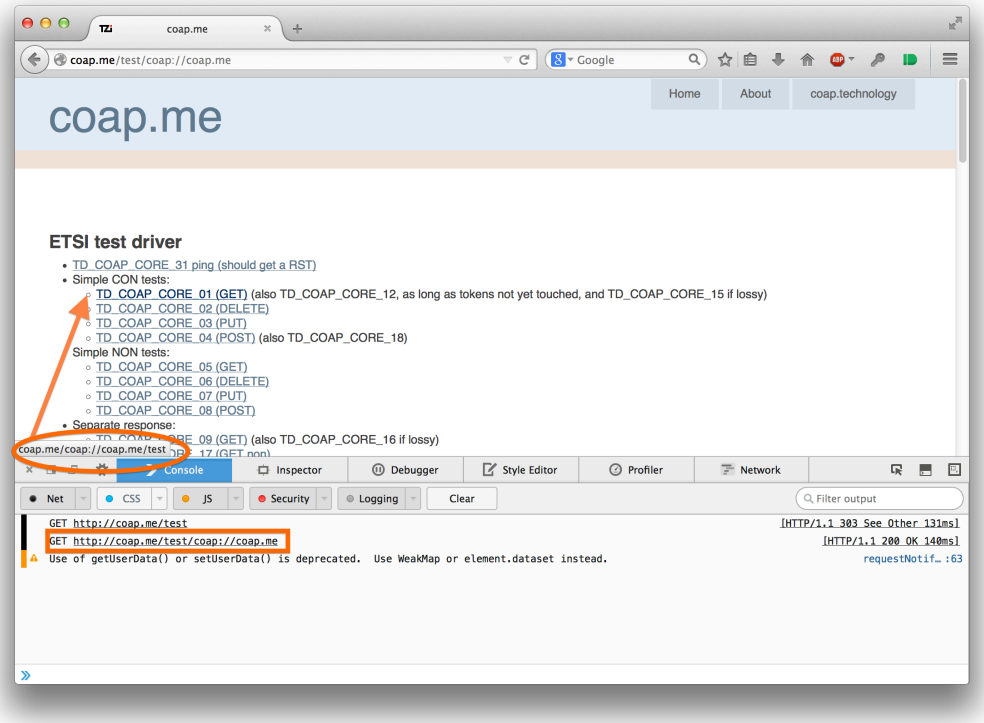

Figure 2.8: Second interaction with an HTTP-CoAP proxy - GET request.

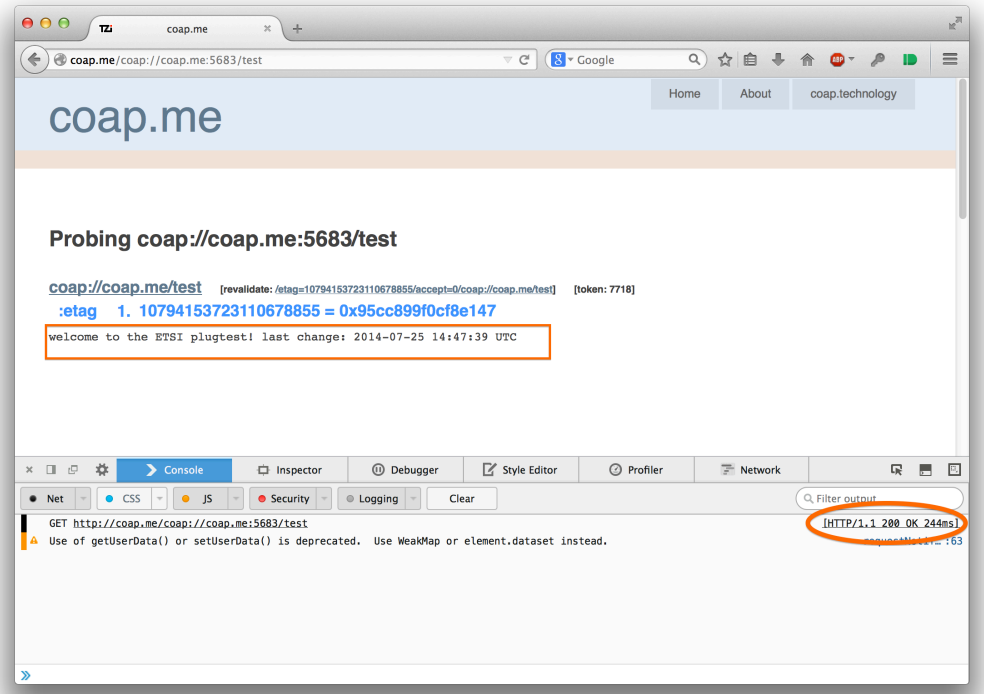

Figure 2.9: Final interaction with an HTTP-CoAP proxy - response of previous GET request.

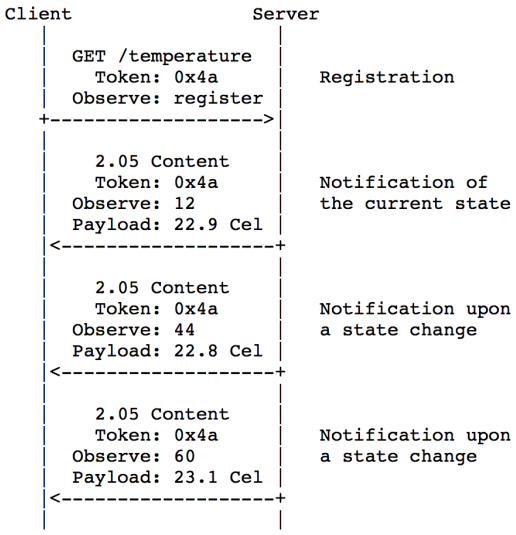

Figure 2.10: CoAP's Observation feature example [29]

Observation Feature

It is also possible to establish a subscription communication using CoAP's observation feature. Simply put, this means that once the client makes a GET request with the observer option set, the server will deliver every new representation as the state of the resource changes. The client must set a token code for the first GET request and the server will send notifications with this same token for each update message. The client can observe different resources at a time.

Fig 2.10 depicts this feature. The first time the client sends a GET with observe ('obs' option set to 0 means Register), the Server adds it to the list of interested, and will send out notifications, with the same originally defined by the client token, together with a sequence number for re-ordering purposes on the Client side. The Server interprets the Client as subscribed as long as it ACKs and will remove it from the list if it either deregisters ('obs' option set to 1), rejects a notification or the transmission times out after several transmission attempts. [29]

Block Transfer

Block Transfer is a CoAP feature designed to support transmission of large data. In these typical environments, sometimes there is a need to transfer large information like a complete log file or a new firmware as an update. With HTTP, TCP handles this by breaking down the data into multiple packets and controlling the information flow. As CoAP does not use TCP, fragmentation should be avoided for efficiency reasons , and so this new mechanism was introduced[30].

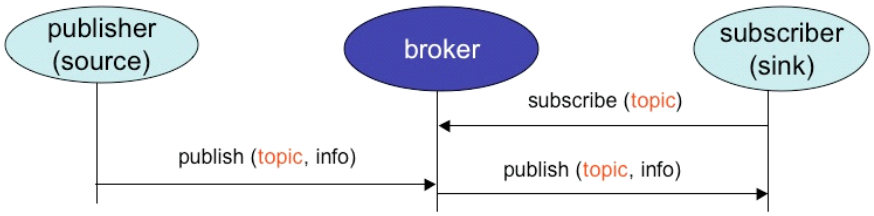

Figure 2.11: MQTT: Pub/Sub protocol[34]

Simply put, one client can get a large resource representation by a constant interaction with the server. The server is stateless and will keep on responding to the client with new blocks of information without a connection setup.

2.2.2 MQTT

MQTT (Messaging Queue Telemetry Transport) is a lightweight network protocol for publish/subscribe type of messaging. It was originally designed in the late 1990s with simplicity in mind [31] and so the final specification is very simple and small (42 pages vs. 112 found at CoAP's) [32]. It is now backed by IBM and is currently undergoing standardization at OASIS [33].

MQTT allows for messages to be publish to one central Broker (server) and then to many other entities (clients), if they have subscribed to the same particular topic, as depicted in Fig. 2.11. When an application publishes messages to topic "A", for example, all other applications, who had previously subscribed to that same topic, will immediately receive the same message. Its verbs are just connect, subscribe, publish and disconnect and, with this pattern, it is possible to efficiently distribute data from one to one and one to many in a simplified mechanism.

Topics are what differentiate which data clients want to receive. With it, it is possible to define semantics like "thermometer", "pump", for example. They define to who shall the Broker forward the message to and its powerful wildcard support makes it a very flexible and smart routing mechanism. Wildcards define the level of hierarchy: "+" means single-level, "#" means multi-level. Fig. 2.12 illustrates this mechanism. A client subscribed to "/sensors/health/+" would receive messages from topics "/sensors/health/hearth_rate/" and "/sensors/health/glucose"; similarly, a subscription to "/sensors/health/#" would subscribe to all nodes below this level, which are "/sensor/health/heart_rate/", "/sensor/health/glucose/", "/sensor/health/glucose/patient_1" and "/sensor/health/glucose/patient_2". It is also possible to use both in the same subscription, making it very efficient to subscribe to numerous different topics with one request only.

Its rich yet simple incorporated QOS allows for three different levels of messages delivery:

- Level 0 ("At most once"). This means no application-level quality of service. It relies solely on the underlying TCP/IP

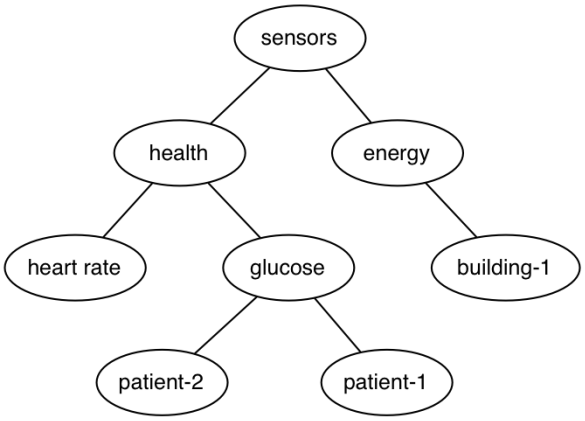

Figure 2.12: MQTT's wildcard support example

network layer and its best efforts mechanisms. Because there is no implementation of data delivery validation (ACK/NACK), there is no way to determine if messages actually got delivered. Should be used by typical WSN nodes which either stream frequent messages or its content is not vital.

- Level 1 ("At least once"). Ensures that messages arrive, but duplicates may occur. The duplicates messages can be discarded by analyzing the message ID.
- Level 2 ("Exactly once"). Ensures that messages arrive exactly once. No duplicates occur at the expense of bigger overhead. Needed for critical scenarios such as flight control where duplicate or lost messages can not occur.

Protocol version 3.1, latest one as of this writing, now features authentication username/password mechanism at the API level, which is sent via the initial CONNECT message. However, as this data is transferred over as clear text, network encryption (like SSL/TLS tunneling, or a VPN with IPSec support) should be used. Other mechanisms allow also for encryption, although this is not part of the specification. Brokers like mosquitto [35], one of the most widely used, supports encryption based on both certificate and pre-shared-key and even authorization via ACL (Access Control List), which controls client access to topics.

As it was designed for low-bandwidth and high latency networks (satellite and GPRS), it possesses interesting features for IoT systems and its constrained devices such as a small code footprint, low overhead, bandwidth efficiency and QoS (needed for this typically unreliable domain) mechanisms. This features are much needed in IoT for two reasons: first, at the device/gateway domain area, it allows devices to sleep and wake up to simply message the central Broker (detailed further), using any content format as it is data agnostic, with very little complexity (2 bytes header, the smallest packet size) while at the same time providing QoS; secondly, at the network domain, it allows for efficient one-to-many distribution of data with assured delivering. However, it may not always be the best suited protocol, given its dependency on TCP. TCP,

when compared to UDP, is a more complex and richer protocol given its incorporated re-ordering and packet retransmission mechanisms which, for typical constrained nodes, translates, ultimately, into greater unwanted battery consumption.

2.2.3 MQTT-SN

Because MQTT protocol started to be quite popular due to its simple pub/sub messaging pattern, there was a need to leverage its simplicity in lower-end constrained devices, which, normally, do not possess the TCP/IP stack (Zigbee, for example) and are battery-powered with very limited computational resources. For this matter, MQTT-SN (SN standing for Sensor Networks), formerly known as MQTT-S, was created. It was designed to be used in typical 802.15.4-based networks which provide only a packet size of 128 bytes long (not accounting for headers) and very low bandwidth of 250 kbit/s. However, due to its low dependencies, it can be adopted in every network, as long as it provides bi-directional data transfer between any node and a particular gateway. [36]

Its overall architecture differs only from the original MQTT in a sense that it adds a MQTT-SN Gateway between the client and the MQTT Broker. Other fundamental different features which were introduced to address these devices' characteristics were: shorter topic ids of only two bytes long instead of long strings (support for small messages length), and a new "keep-alive" procedure which allows a device to sleep while it is destined messages are buffered at the server/gateway and delivered later when it wakes up. More importantly, MQTT-SN depends solely on UDP, instead of TCP. Fig. 2.13 represents MQTT-SN's architecture. Besides the aforementioned MQTT-SN Gateway, there can also be a MQTT-Forwarder. The Gateway is in charge of translating MQTT-SN protocol into MQTT from one domain (Zigbee, for instance) to any other one that supports the TCP/IP stack where the MQTT Broker is located. The Forwarder does not translate one protocol to another but it forwards the message from one network to the other. As it supports multiple Gateway/Forwarders, the protocol helps to mitigate typical loss of messages (due to network's inherit low bandwidth and high link failure rates) by offering load sharing and redundancy services.

2.2.4 Comparison

As CoAP follows the HTTP approach, it is a document transfer protocol. The most important difference is that it was designed for constrained devices, adopting UDP instead of TCP and packet creation simplicity. However, it still follows a client/server model which translates into an one-to-one type of data flow. It does have a new "observe" approach, but it is not as efficient as a real pub/sub protocol like MQTT[-SN]. Unlike MQTT[-SN], CoAP's Web Service oriented architecture allows for a seamless integration in the Web and interoperability with common RESTful Web Services by making use of simple proxies. On the other hand, MQTT-SN simple approach allows for an efficient one-to-many communication at the expense of not being directly integrated, as easily, into RESTful Web Services. Furthermore, MQTT[-SN] is payload agnostic

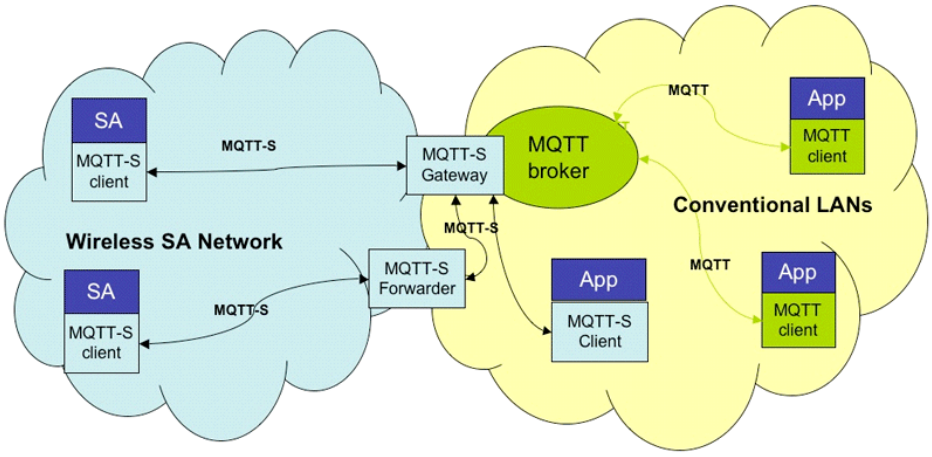

Figure 2.13: MQTT-SN Architecture [34]

and, being so, each application must know before hand how to process the representation; CoAP, in the other hand, allows for content negotiation. Also, CoAP's Resource Discovery allows for a typical sensor to act more as a server than as a client. This added complexity allows for much simpler end-to-end direct interaction with the sensor, but one may also encounter NAT problems. As MQTT makes use of TCP, it is connected-oriented TCP connection to the server results in non-existent NAT problems; however, once again, it may only be used in devices with TCP/IP support.

To conclude, it is important to understand that each protocol serves its own purposes. Both have advantages and disadvantages and its adoption depends solely on the particular scenario/application - it is best-suited messaging pattern, need for resource discovery and Web integration, etc.. Not every IoT/M2M system will use the same protocol - in some cases, it may even be best to use both. However, these are the ones mostly used [4] and so, IoT/M2M System Architecture SDOs (Standards Developing Organizations) must take this into consideration, providing, for instance, intrinsic protocol-binding for greater flexibility and scalability.

2.3 Conclusion

This chapter has showed how entities like IETF are working on specific protocols for IoT, adapting some of the existing technologies to the typical constrained environments most "things" will reside on. As there is a need to bring IP connectivity to these "things", protocols like 6LoWPAN, Zigbee IP [37] and even the next generation of Bluetooth Smart [38, 39] are proof that there is a need to adapt the typical low-energy RF technology to IP. At the application level, new lightweight protocols are also needed and some have been created specifically for IoT, like CoAP and MQTT-SN. The Internet of Things is still in its

infancy, but these new emerging protocols and standards will surely help bringing it to the next level. It is, however, a slow process as it is a new complex technology comprising different layers of the OSI model. IETF, for instance, has developed both a new adaptation layer (6LoWPAN) on top of 802.15.4 (physical layer), RPL (network layer) and CoAP (application layer).

Chapter 3

State of the Art

This chapter presents the on going efforts of major entities to develop a standard architecture for M2M and IoT systems. In a nutshell, all of these standards are working for the same end: to deploy an end-to-end architecture, providing an horizontal solution where services can serve multiple vertical applications, while offering interoperability between them, independently of the underlying network and technology. From an industry standpoint this simply means that it will enable a faster and reliable deployment of a complete interoperable M2M solution. It is ETSI's goal to re-use and integrate already existing standards and technology. The term M2M is often used, while IoT is not, but they all serve the same purpose - all standards have M2M in its name, but the specification acknowledges that it is also designed for IoT, as M2M is at IoT's core. Fundamentally, IoT differentiate itself from M2M simply because it has implicitly associated features that may not be present in M2M solutions, like context-aware (self adaptation to the environment), collaboration (inter-exchange of smart object's data) and cognition (autonomous systems making sense of data, producing intelligent decisions), which, in others words, means no Human intervention. The next sections will then detail the latest standards and proposals for a common IoT System Architecture.

3.1 ETSI M2M

ETSI TC (Technical Committee) M2M was established in January 2009 [40] and it follows an open approach, by integrating and cooperating with different organizations and by publicly publishing latest drafts and final TR (Technical Requirements) and TS (Technical Specifications) [41]. It is organized into 5 WG (Working Group):

1. Use Cases & Requirements;

2. Architecture & Network Interworking;
3. Protocols & Interfaces;
4. Security Aspects;
5. Management Aspects.

ETSI TC M2M acknowledges that M2M solutions will serve different applications such as e-health, smart metering, smart grid, automotive and connected consumers. Because of that, studies were made and specifications have been released. These are Technical Reports (TR) often detailing different use cases which are later addressed as part of specifications in Technical Specifications (TS). Particularly important for IoT is TR 101 584 [42] where a "Study on Semantic support for M2M Data" was conducted. In this Technical Report, a study on the "benefit, feasibility and potential requirements for the support of semantic information on application related M2M Resources in the M2M system" was made. It does not support the idea of defining semantics to be included in the standard, but instead studies the mechanisms needed to create and handle such information. Use cases are presented and most of them related to e-health, demonstrating the benefits for the whole M2M ecosystem. It was also investigated discovery mechanisms to be used within the system architecture, taking into account how other solutions (like CoAP's) could be used within ETSI's M2M architecture. Such mechanism could enable, among others, feature like "simplified configuration of M2M applications and more intelligent adaptation to changing situations". This abstraction level would provide the information - much needed by smart applications - to create knowledge out of it and correlate events between other applications and/or IoT/M2M systems.

Current vendors and organizations have their own vertical solution with proprietary functionalities, transport and network protocols and communications infrastructure. This paradigm leads to more expensive and complex solutions, larger time to market and lack of intercommunication with other M2M systems. To fight all this, an horizontal solution with a common set of service functions and open middleware platform is being developed by ETSI TC M2M, which is network and application-agnostic, but aware [43]. It is based on existing standard and technologies (GPRS, WiMAX, WLAN, UWB, Zigbee, 6LowPAN, CoAP, HTTP, for example) and it provides a generic set of capabilities for M2M services and a framework for developing services independently of the underlying network. With it, it is possible to break down current silos.

3.1.1 Architecture

To summarize ETSI's M2M System Architecture, one can break it down into three main area domains: M2M Device Domain, M2M Network Domain and M2M Application Domain. Across these domains are three standard Reference Points connecting them, by means of a set of Service Capabilities. Figure 3.1 depicts the high level architecture. The Service Capabilities are present in each SCL (Service Capabilities Layer) - DSCL, GSCL and NSCL (Device, Gateway and Network, respectively). A

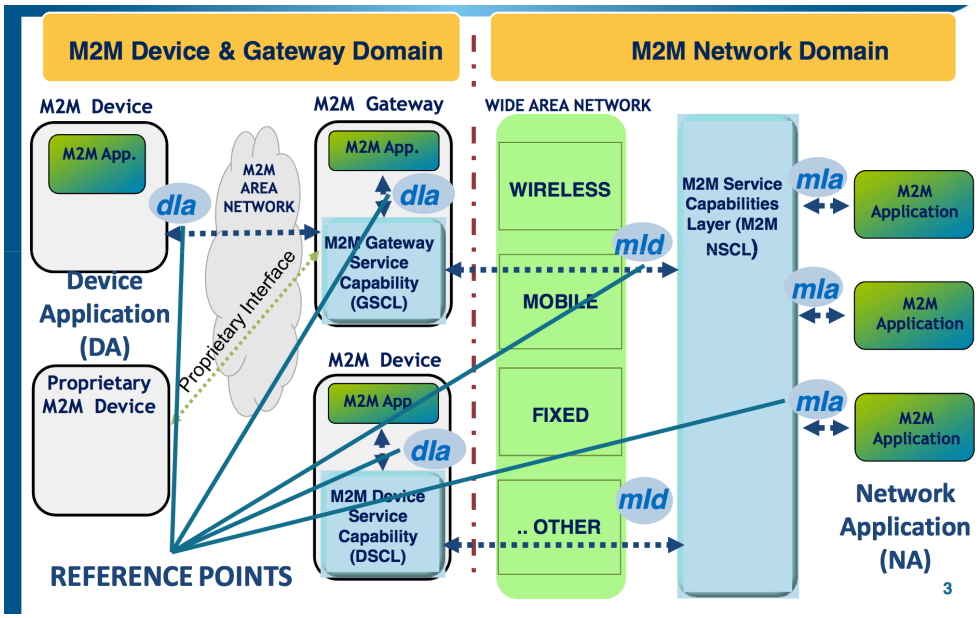

Figure 3.1: ETSI's M2M High Level System Architecture [44]

RESTful architecture style was adopted, where information is represented by resources. The SCL holds this standardized resource hierarchical tree structure (Fig 3.2). The information present on each resource can then be exchanged between the different SCL over the Reference Points (dIa, mId and mIa), via standardized procedures. As it follows a RESTful architecture, it is possible to manipulate the information using CRUD methods (Create, Read, Update, Delete) in a Client-Server model.

These SCL allow for management of information related to applications and devices like registration, access rights, security and authentication, data-transfer, subscription/notification and group management [46] [47]. All of this is achieved by RESTFul operations using HTTP or CoAP, as CoAP binding is also part of the standard, over the standardized resource tree. As a simple example of how this RESTful Architecture works across the SCLs, a diagram is shown 3.3 denoting a typical scenario where a sleepy Device (D) sends data to a Network Application (NA). First, the Device writes data to the DSCL through NSCL (UPDATE verb); then, the NSCL notifies the NA of that resource change (notify means either a response of a RETRIEVE or an UPDATE, if a polling or the asynchronous mechanism is used, respectively); at last, the NA reads the new resource change. For the sake of simplicity, the illustrated diagram represents only the events flow, hiding all methods, protocols and responses used in real applications.

The different Reference Points mIa, mId, dIa, and mIm have their own purpose. They all, fundamentally, offer the same mechanisms, but at different domains. dIa offers a generic and extendable mechanism for Device Application (DA) and Gateway Application (GA) to interact with DSCL (Device Service Capability Layer) or GSCL (Gateway Service Capability Layer). This allows the applications to register in the SCL, request read/write permissions on the NSCL, GSCL or DSCL, subscribe

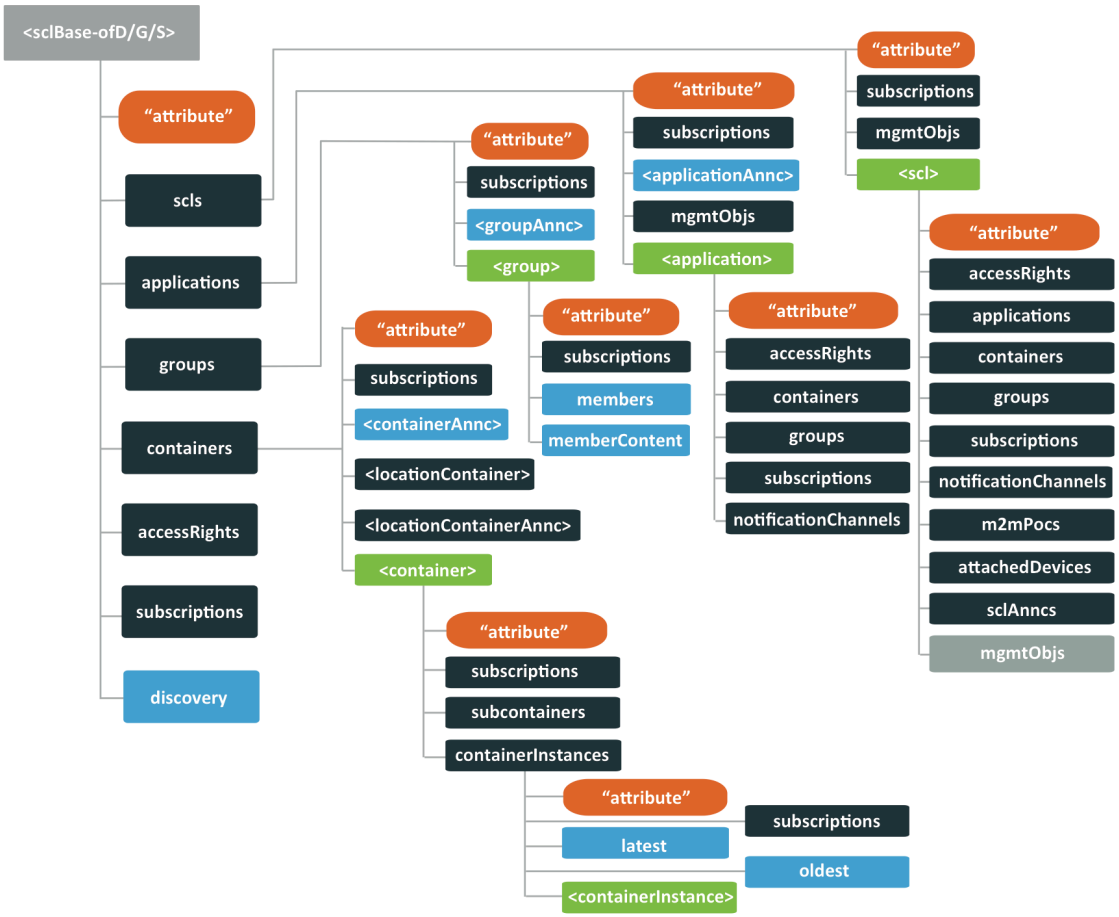

Figure 3.2: SCL's standardized resource structure tree[45]

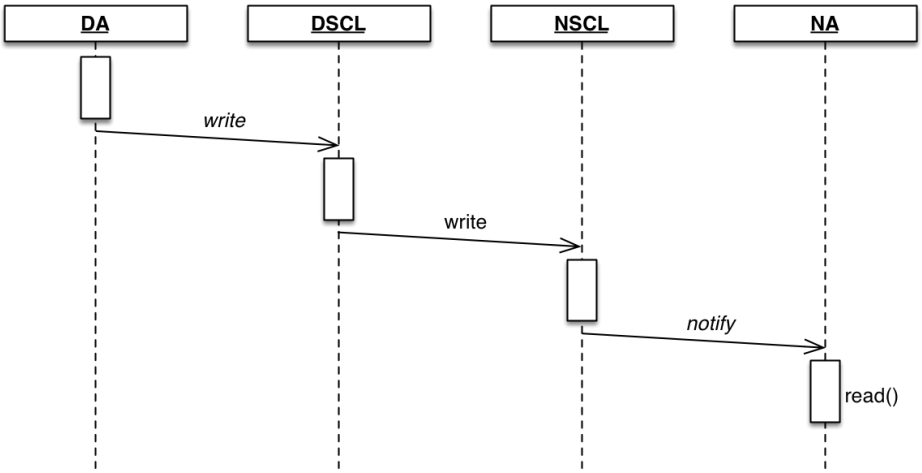

Figure 3.3: Simple use of SCL resources to exchange data (adapted from [48])

and notify to/of specific events and request managing capabilities of groups. mIa functionality is identical to that of dIa, but the interaction is between the Network Application (NA) and the NSCL (Network Service Capability Layer) and it may be used to to request device management actions like firmware update. Likewise, mId follows the same pattern, but over IP connectivity between DSCL/GSCL to NSCL backed by security features. The last Reference Point defined in the Functional Architecture [48], mIm, is a special one used only for inter-domain across different M2M Service Provider; its capabilities differ from mIm when there is no NSCL registration - if this happens, the offered capabilities are some of those offered by mIm like request to read/write information in the NSCL, GSCL or DSCL across two different M2M Service Providers, if properly authorized to do so. The previously mentioned device management capability is possible because of BBF TR-069 [49] and OMA DM [50] compatibility and integration with the standard [51].

3.1.2 OneM2M

ETSI was not the only SDO (Standards Development Organization) working on a M2M architecture - others had their own solution as well, which would lead to interoperability problems and a slow development of the global M2M market. To avoid any duplication of work between SDOs involved in the same domain, and to agree on a truly global standard M2M architecture, oneM2M was born. In January 2012, seven SDOs - ARIB, ATIS, CCSA, ETSI, TIA, TTA, and TTC -, have agreed on a Global Initiative for M2M Standardization [52]. As they have agreed on an open collaboration with other interested organizations and parties, many others have joined in to contribute in different levels of functionality - OMA, responsible for LWM2M, IPSO Alliance and Continua Health Alliance, as Partners Type 2, which were mentioned in this work, are some examples. Six months later, in July 2012, the oneM2M partnership project was established having ETSI TC M2M work

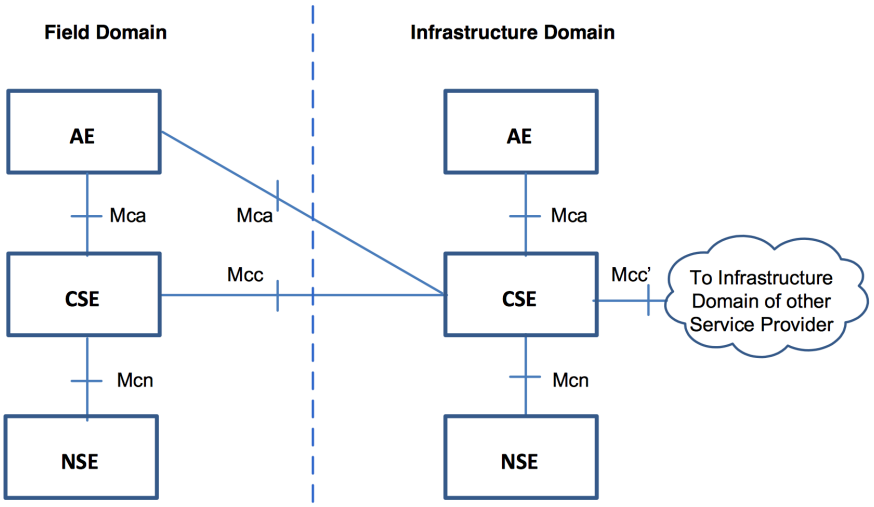

Figure 3.4: oneM2M Functional Architecture [59]

being transferred over to form its basis. Fundamentally, oneM2M Partnership Project aims to establish a cooperation "in the production of globally applicable, access-independent M2M Service Layer specifications, including Technical Specifications and Technical Reports related to M2M Solutions" [53].

OneM2M 's ambition to release specific Technical Specifications on different transport layer protocol bindings (like CoAP, HTTP, MQTT and, possibly, XMPP) combined with the inclusion of device management, abstraction layers and semantics view make it a very compelling uniform Standard. The Candidate Release [54] was available for public commenting on August 2014 which already specifies the functional architecture, CoAP and HTTP protocol bindings [55, 56] and mappings to device-management related protocols like OMA DM, OMA LWM2M and BBF TR-069 [57, 58].

Its functional architecture is depicted in Fig. 3.4 and one can easily spot its resemblance to ETSI M2M. Like ETSI M2M, the resources described in the architecture can all be interacted via a RESTful API and its primitives, the service layer messages sent over Mca and Mcc reference points, are then mapped to the appropriate transport layer protocols like CoAP, HTTP and MQTT [60]. Bellow these transport layer protocol bindings is the underlying network to which oneM2M will also define a set of bindings to. This last feature will be possible via NSE (Network Services Entity) which provides services of the underlying network to the CSEs (Common Services Entity), like device management, location and device triggering. The services provided by CSE are called CSF (Common Services Functions) and these are the key services functions with which AE (Application Entity) and CSEs interact with via the appropriate reference points. These are detailed in Fig. 3.5 . The interaction between AE and CSE is made by primitives, as illustrated in Fig. 3.6. Each CRUD (Create, Update, Retrieve and Delete) operation is mapped to one or more primitives which is then further mapped to transport layer protocols like HTTP, CoAP or MQTT.

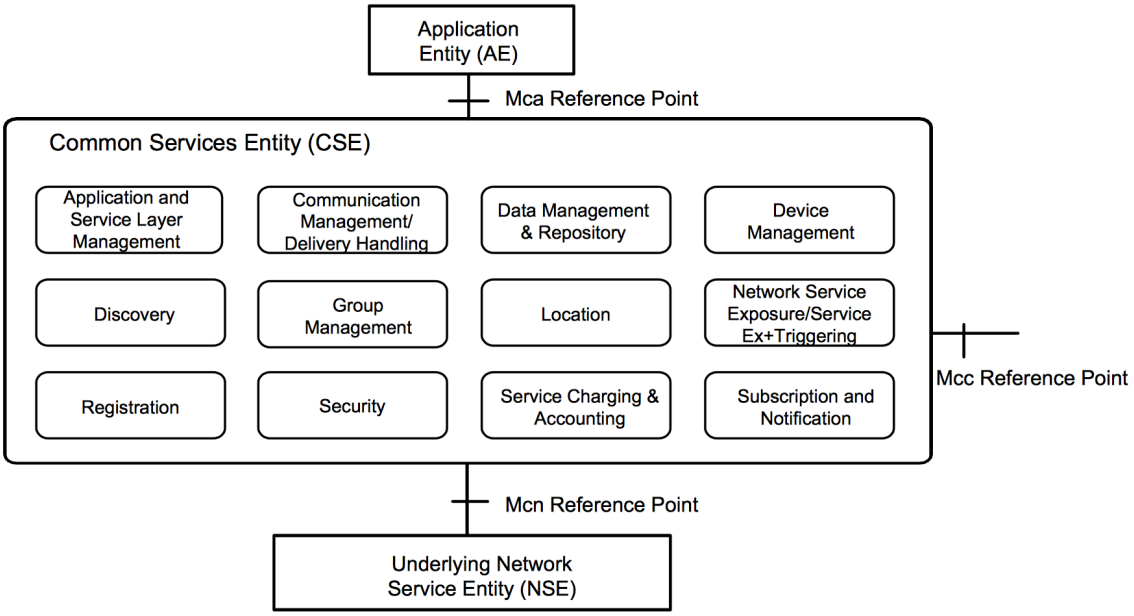

Figure 3.5: oneM2M - Common Service Functions residing in CSE [59]

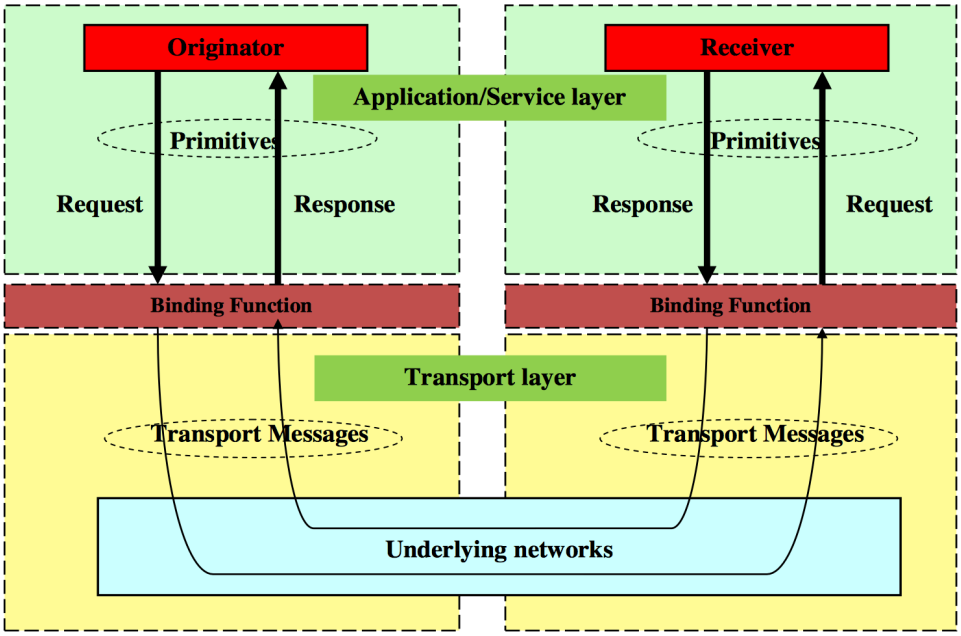

Figure 3.6: oneM2M - Primitives [61]

3.2 OMA Lightweight M2M

This new standard (Technical Specification Candidate Version 1.0, released in December 2013 [22]) is a fresh new approach to Device Management and was designed having IoT in mind. It is different from OMA DM standard, which is mainly used in Cellular devices, in a sense that it is suitable for any kind of device as long as it uses IP (Cellular, WiFi, 6LowPAN, as examples) and was designed for constrained devices and environments. It is an unique solution that enables both Management and Application data. In essence, OMA Lightweight M2M (LWM2M) is built upon CoAP and DTLS protocol with bindings to UDP and SMS (for Cellular Device management) and offers an extensible Object and Resource model for application semantics which can be published to OMA public registry. This use of known application semantics provides interoperability and abstraction between different IoT/M2M systems.

3.2.1 Overall High Level Functionalities

The Device Management functionalities are as follows:

- Bootstrapping: automatically connect the device to the right server using key management.
- Device Configuration: change parameters of the device and network settings.
- Firmware Update: Over-the-Air (OTA) software updates (to overcome latest security problems, apply patches, enhancements, etc.)
- Fault Management: automatic error reporting from the device and ability to query it. For debugging purposes such as network unreachability and misconfigured applications and/or services.

And, as it is also a solution for Application's Data, it also allows for:

- Configuration & Control: in-app configuration of settings (control commands to define new Application's parameters)
- Reporting: Notification mechanism to alert for new sensor values, alarms and events.

3.2.2 Architecture

Its architecture is simpler than ETSI's and oneM2M's. Fig 3.7 illustrates its components and relationships between them.

With LWM2M's architecture, there is a small LWM2M Client library in a constrained device which uses the standardized interfaces to manage the built-in Objects created by the Device Manufacture and/or the Application Developer. Managing

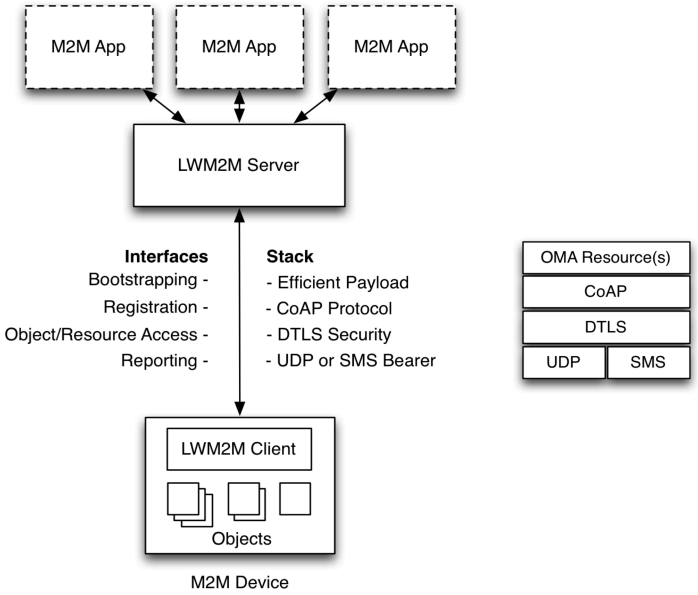

Figure 3.7: OMA Lightweight M2M Architecture [62]

these known objects between the Client and the multiple M2M Applications, as long as they have authorization to do so, is, therefore, very straightforward, allowing for greater interoperability and abstraction when making use of standard Objects (detailed further). The LWM2M server works as an intermediate between the M2M Applications and the Devices itself - it can command the Device to Read, Write, Execute, Create or Delete Objects and its resources (also detailed further). All of these commands are sent to the Device via four interfaces:

- Bootstrapping: it is possible to do pre-provisioning out of the SIM card or flash memory, or initiate the configuration mechanism between Client and Server by making use of initial shared keys, server configuration and ACL (Access Control List), for example.

- Registration: the LWM2M Client can alert the LWM2M Server of its existence and capabilities, by means of Resource Directory (another IETF standard).

- Management & Service Enablement: used when the Server wants to send operations and commands to the Client. Allows for Device Management and Service Enablement over the previously announced, by the Client, Objects and Resources. The Client responds via this interface as well.

- Information Reporting: allows for periodically report of resource information by the LWM2M Client in case of triggered events and/or alarms. The notification mechanism is possible due to CoAP's observation feature.

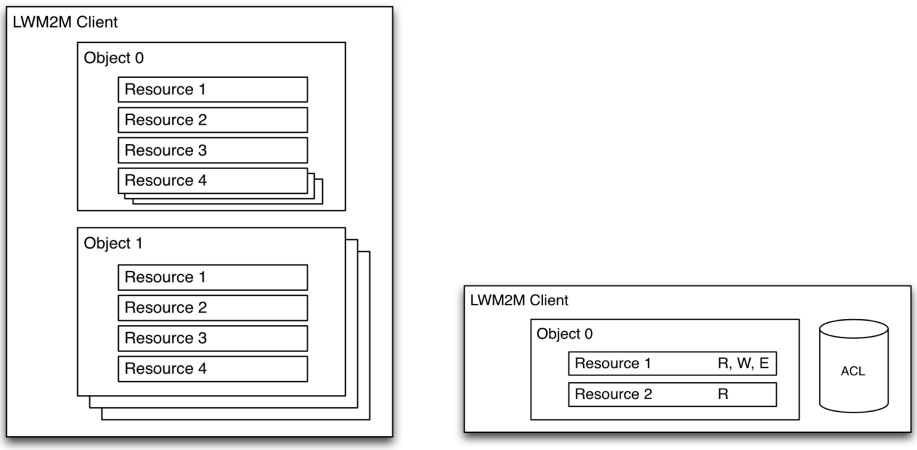

Figure 3.8: LWM2M Object Model [63]

The stack is very small as well, as it uses simple protocols and technologies. CoAP, DTLS and UDP are all lightweight enough, efficient and secure to be used in constrained environments and SMS binding is optional but a must for Cellular use. DTLS is used between LWM2M Server and LWM2M Client to secure all the information exchanged between these two endpoints.

3.2.3 Object Model

As stated before, LWM2M structure information resides on Objects and their Resources. These work as an URI formatted as

/{Object ID}/{Object Instance}/{Resource ID}

Hierarchically, an Object may contain several Resources, which, in turn, may have multiple Instances. Each Object as associated an ACL Object, defining which restrictions are imposed to that particular Object (managed by the Server). As it makes use of CoAP, the three known verbs GET, PUT, POST are also present on each Resource as Read Write and Execute operations respectively. Like UNIX files, all sorts of combinations are possible: some might allow the reading operation along writing or simply execution operation, etc., as showcased on Fig 3.8

To access an URI, the machine needs only to GET, POST or PUT a small and easy to parse URI as small as, for example, 2/0/1. Objects and Resources are identified by a 16-bit integer, while an Instance needs only an 8-bit Integer. The resource's payload can be represented as pain text and as binary TLV or JSON for Objects or Resources arrays.

One important aspect of this model is that its cooperation with other M2M/IoT systems is very simple, since the definition and registration of Objects is open for third parties (Enterprises or SDOs). OMA itself has defined 8 built-in Objects to be used

in LWM2M . These can be defined and registered with OMA's Naming Authority (OMNA), by simply filling out the Lightweight Object form available on OMA's website [62]. This form allows for an Object definition: object name, description, possibility to have multiple instances, list of and resources. OMA Working Groups, third party organizations (like the IPSO Alliance which is detailed in the next chapter) and other Enterprises can register these Objects. This definition and registration of known objects is crucial for interoperability purposes, since anyone can access its definition online and consume it appropriately in their own application and/or system.

3.3 IoT Reference Model and Guidelines

3.3.1 IoT-A

IoT-Ais a FP7 program [64] whose main goal is to develop an Architectural Reference Model for the Internet of Things. The project started in 2010 and finished three years later, in 2013, but its resulting work has passed on to IoT Forum. The authors of such project have recognized the current IoT implementations as not really Internet of Things, but rather an Intranet of Things, given the nonexistent of an holistic approach to implement such systems. The latest version of their Deliverable about the "architectural reference model for the IoT" [65], which dates July 2013, the authors have justified the need of an Architectural Reference Model (ARM) for IoT to support both a common understanding of the IoT domain (Reference Model) and a set of frameworks and guidelines for businesses who want to create their own compliant IoT solution (Reference Architecture). The conjunction of this Reference Model, the Reference Architecture and a set of Guidelines are what constitutes the ARM. For the authors, "the interoperability of solutions at the communication level, as well as at the service level, has to be ensured across various platforms" so that it is possible to achieve their envisioned IoT.

In order to achieve the ARM which will serve as a the matrix for the final concrete architecture, the project specifies a logical methodology. It is important to thoroughly study existing architectures and solutions (State of the Art - SOTA) which will enable the establishment of, first, a Reference Model and then a Reference Architecture, as can be seen in Fig. 3.9. First, it is important to know that the authors have defined such a Reference Model and a Reference Architecture having in mind OASIS's definition of such. Before analyzing both, one must understand a Reference Model as a "minimal set of underlying concepts, axioms and relationships within a particular problem domain, and is independent of specific standards, technologies, implementations, or the concrete details" [66]and a Reference Architecture as "an architectural design pattern that indicates how an abstract set of mechanisms and relationships realizes a predetermined set of requirements. It captures the essence of the architecture of a collection of systems. The main purpose of a reference architecture is to provide guidance for the development of architectures. One or more reference architectures may be derived from a common reference model, to address different purposes/usages to which the reference model may be targeted." [67]

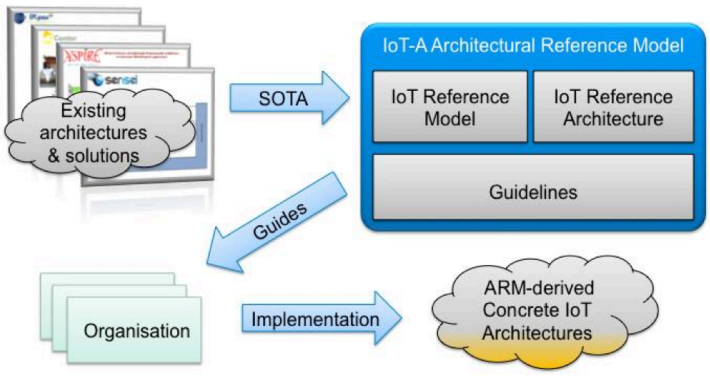

Figure 3.9: IoT Architectural Reference Model building blocks

The ARM defined by the project is not, in itself, an IoT Architecture - it is a generic architecting framework which can derive a concrete IoT Architecture. This process of achieving a concrete IoT Architecture depends solely on the architect, and the resulting analysis of the project's guidelines, perspectives and models.

As the Reference Model provides the highest abstraction level for the definition of an ARM, it includes a set of models related to the IoT domain:

- IoT Domain Model - top-level description;
- IoT Information Model - how IoT knowledge is going to be modeled;
- IoT Communication Model - to better understand specifics about communication between many heterogeneous IoT devices and the Internet as a whole.

In the other hand, the Reference Architecture, is the one responsible for building compliant IoT architectures. It does not provide concrete application architectures, but rather a set of abstract mechanisms which businesses and implementors of the final concrete IoT Architecture must take into account (Views and Perspectives). Finally, the last block of the ARM, as represented in Fig. 3.9, is the "Guidelines" which is an extensive work done by the project (available in [65]). It offers recommendations and discusses how the previously given Models, Views and Perspectives can derive in a concrete Architecture. This are, fundamentally, Design Choices which are most important to businesses and overall IoT Systems implementors. Design Choices like real-time access to information, security features, semantics, etc, are some examples of architectural choices which forms instances from the Reference Architecture.

To summarize, it is the combination of the large SOTA analysis (and its resulting set of possible functionalities, mechanisms and protocols) with a set of Design Choices that offers IoT System Architects the set of protocols, functional components

and architectural options needed to build the concrete IoT System. Thus, the resulting ARM eventually derives a large set of concrete architectures. Moreover, according to the authors, the IoT ARM "provides best practices to the organizations so that they can create compliant IoT architectures in different application domains." where interoperability is therefor ensured. It is important to notice that the ARM itself cannot guarantee interoperability between any two concrete architectures. Concrete architectures deriving from the same requirement set and Design Choices may not be interoperable. However, this building process allows for an easier bridging between the two systems, as both of them have been derived from similar architectural choices.

The ARM itself represents an extensive work as a result of three years of R&D, and given the complexity of its latest version (version 3, released in July 2013 [65] is a 482 pages long document), and for this matter, any further detailing would be out of scope. IOT Forum acknowledges that the ARM now needs a set of re-usable profiles [68]. These profiles will take ARM one step forward to achieve broad adoption as these profiles are expected to be adopted by IoT system architects. With this future work, the ARM will evolve from a generic architecting framework to a set of pre-build concrete architectures, taking away much of the work previously needed by actual architects.

3.3.2 IPSO Alliance

The IPSO Alliance main objective is to educate, document and support the use of IP for the IoT, while defining the "appropriate protocols, architecture and data definitions for Smart Objects so that engineers and product builders will have access to the necessary tools for "how to build the IoT RIGHT"" [69]. It is a much needed Alliance for IoT, giving their efforts on the use of semantics for the whole IoT. These semantics help to represent, in a standard way, common resources like temperature, light control and power control, as examples.

IPSO has published a guideline [26], which should not be recognized as a standard, on how to define Smart Objects using a RESTful design for common M2M Applications such as Home Automation and Building Automation. This defines how a Smart Object can represent its available resources and how to interact with other Smart Objects and backend services, using the defined set of REST interfaces. The goal here is to show vendors simple guidelines on using IP and Web-based technology to develop and rapidly test interoperability among devices and services. Following these guidelines assures compatibility with the previously mentioned OMA LWM2M. As stated before, third party SDOs can define and register Objects on the OMNA - and that is what IPSO has done. OMNA LWM2M Object & Resource Directory [62] lists the resources already defined by the IPSO Alliance. For example, Table 3.1 describes the "Luminosity Sensor" Object registered by the Alliance with an Object ID (3301) provided by OMNA.

In September 2014, the Alliance has released a guideline called "Smart Objects Starter Pack 1.0" [70] which list and details 18 IPSO Smart Objects based on OMA LWM2M's object model specification, like the one represented in 3.1. This type of

ID	URN	Name	Multiple Instances	Description
3301	urn:oma:lw m2m:ext:3301	Luminosity Sensor	Yes	This IPSO object should be used over a luminosity sensor to report a remote luminosity measurement. It also provides resources for minimum/maximum measured values and the minimum/maximum range that can be measured by the luminosity sensor. The unit used here is Lux (ucum:lx).

Table 3.1: Example of a LWM2M Object defined by a 3rd Party SDO and registered at OMNA.

work represents IPSO Alliance's effort on supporting an IoT where Smart Objects can easily integrate the current Web. Their support on specific technologies and protocols (6LoWPAN, UDP, CoAP and LWM2M) justifies the ease of a simple IoT/M2M system architecture where common Objects can easily be mapped between constrained devices (6LoWPAN and LWM2M) and the whole Web (meaning mobile applications and services) through HTTP and IPv6 by making use of the appropriate Content-Type and access methods (HTTP protocol binding is also defined in LWM2M).

3.4 Comparison

Each standard has its own particularities, and, as it happens with transport layer protocols, each serves a purpose. One standard does not fit all, but a combination might be a good solution. As IoT-A is not really a functional architecture, but an architecting framework, not much can be said at this point; however, this might change in the future as IoT Forum will continue to work its defined ARM so that a a set of re-usable ARM-profiles can be built, to be immediately adopted by IoT Architects. In a much more advanced phase, is ETSI's M2M architecture. It is flexible enough to be widely adopted by either IoT/M2M architects and Application/Services Developers. Its REST API combined with the decoupling of different Services Layer allow for rich applications to be build, while inter-exchanging information and application data with each other, extending their functionality. However, it was not really designed to be supported in small and constrained devices. For a device to be fully compliant with ETSI M2M, its application (DA - Device Application) must support the correct HTTP or CoAP (if built-in bindings or Interworking Proxies are supported) methods to specific URIs with complex XML data.

For real constrained devices, the OMA DM included in ETSI M2M functional architecture might be too complex for such devices. The Remote Entity Management (REM) Service Capability must be supported at both Device/Gateway level and Network level. As the OMA DM standard was initially conceived for not so constrained devices and environments, its natively inclusion in ETSI M2M might be obsolete for real IoT systems. ETSI has laid the foundations for a standard M2M architectural

framework; however, as IoT differs from M2M, its standard must support other suitable for IoT protocols.

Contrary to OMA DM, OMA Lightweight M2M's (LWM2M) architecture seems to be more suitable for real IoT systems as some key features are specifically tailored for IoT. It was designed having real constrained devices in mind with the much needed Device Management functions, while adopting open IETF standards designed for typical IoT scenarios (CoAP, DTLS, Resource Directory, UDP and SMS bindings). Although its architecture is not as complex and rich as ETSI M2M, it offers some advantages over the latter. Its interfaces offer much needed functionalities to IoT systems, like: bootstrapping (which allows for a device to automatically connect securely to the correct server), on-the-fly application and devices re-configuration (proper adaptation of the device to the network), security (over UDP and, very important, firmware update to patch security issues). These combined with application data (send and retrieve data using CoAP and UDP/SMS) makes it a very compelling option for an IoT system architect. Also very important is OMA's Naming Authority (OMNA) which allow for simple known semantics to be registered through its extensible model object - like the ones defined by the IPSO Alliance. As it is also very light, most constrained devices need only a small library to support the whole protocol - device manager and application data.

Finally, OneM2M seems to tackle both ETSI M2M's and LWM2M's cons by combining the two. It is a more modern standard that takes ETSI M2M as its basis while offering binding to most commonly used IoT protocols and tools like MQTT and Web Socket (still a work in progress, as evidences found in [60] and [71] suggest so). Its LWM2M integration makes OneM2M the most complete standard ready to target both M2M and IoT solutions supporting all functionalities and services needed by both while, at the same time, providing a concrete way for Application and Services Developers to interact with. This is what is needed in IoT to make way for richer, smarter and interoperable Applications across every domain.

3.5 Literature Review

As most of the protocols and standards described earlier are fairly recent, they have not been very referenced in the literature when deploying IoT systems. The concept itself is present in such systems, but only a small part of them are using the specifically-designed protocols such as 6LoWPAN, CoAP, MQTT and architectures and frameworks like ETSI's M2M Architecture, oneM2M or LWM2M. For this matter, three different recent articles that are either related to the proposed solution or announced as an IoT solution, are detailed and compared.

A recent work was published on a "Middle East Conference on Biomedical Engineering (MECBME)" in February 2014, presenting an Ubiquitous Healthcare System [72] as a prototype for the authors' future system. This system makes use of 6LowPAN between the nodes and the Edge Router (ER) and a custom proprietary application on the Base Station (PC) to receive the transmitted UDP packets to then make it available on the internet using either SensorMonkey and an UbuntuOne

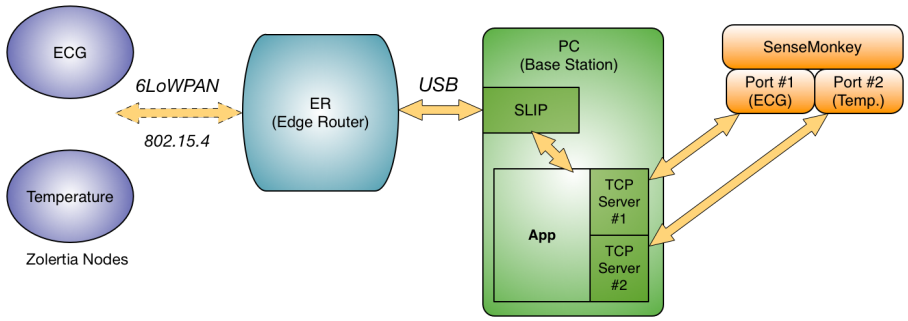

Figure 3.10: Representation of the System Architecture defined in [72]

Server, as represented in Fig 3.10. The Base Station has a running C program that listens for UDP Packets in the interface provided by the SLIP (Serial Line Interface Protocol) utility. This C program is multi-threaded and, being so, it is composed of two TCP Servers who are responsible for sending the previously obtained data to SenseMonkey's platform. This platform, once an account is created, provides a way of registering the sensors and subscribe to them by inserting the Base Station's IP and the dedicated TCP ports.

When analyzing the typical and the already-defined protocols and standards to be used in IoT, [72] makes use of only one of them: 6LoWPAN. The data transmitted via 6LoWPAN relies solely on UDP and makes no use of standard application layer protocols like CoAP or MQTT[-SN]. This accounts for interoperability issues. Further more, the real time streaming process relies solely on non-standard applications (SensorMonkey and UbuntuOne Server). If another application (as the author's Remote Monitoring) wanted to access this streamed data, they would also have to interact with SensorMonkey's website using the original ID and key to first request a connection and a subscription of the streaming data - SensorMonkey provides a JavaScript client which can be used to interact with its service. This is not a practical solution and makes it very hard to autonomically interact with other IoT Applications. The other adopted method relies on a synchronization of the sensed data, using UbuntuOne cloud service, once the Base Station writes new data into temperature's and ECG's files. After having the data stored into the respective files, a graphical Matlab client reads the data and plots it.

Although there is no precise information on the hardware used, the authors do claim that the sensor nodes made use of Zolertia motes (MSP430 + CC2420). As the ER (Edge Router), as descried by the authors, is the responsible node for converting IPv6 packets to 802.15.4 packets and vice-versa, and it forwards the received packets via USB to the Base Station, it is assumed that the hardware used was either Z1 Starter Platform or Zolertia Gateway. These components, when adding the full-fledged PC as the system's Base Station, makes up for an expensive solution. Zolertia's online store price tags the components as:

- Z1 Platform (sensing nodes): 95€ [73]

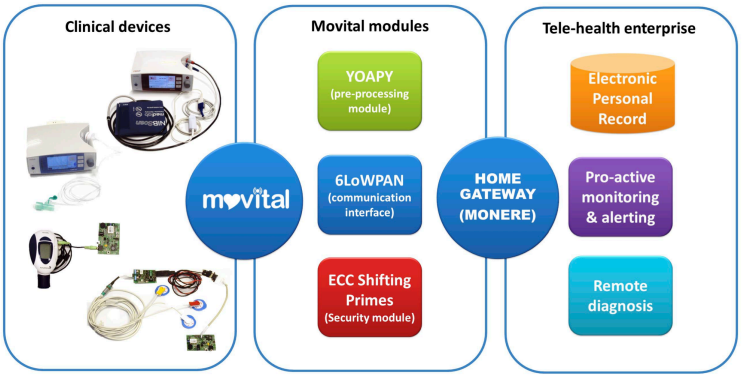

Figure 3.11: Architecture overview of [76]

- Z1 Gateway: 353€ [74]
- Z1 Starter Platform: 253€ [75]

This, however, assumes the price for one item only - the price per item is lower when buying in larger quantities. Assuming the authors used Z1 Starter Platform instead of the Gateway (100€ less), the total cost for the solution, excluding the PC, was 443€.

Making use of known semantics and registered web objects or resources is just as important as having a standard M2M/IoT System Architecture. A prime example of such claim is present in [76]. The authors of this work, published in 2012, have designed two key components which allow for acquisition of knowledge from typical clinical devices and it is transfer and integration into known tele-health's information system. One of the most important aspects of the authors' solution was the inclusion and interpretation of medical data, acquired from normal clinical devices, into the information system. For this, they had to enrich the functionalities of common medical devices which were not certified by Continua Alliance [77] and, thus, do not provided the much needed semantic interoperability. Continua Alliance offers a set of guidelines manufactures must follow to get their devices certified. This certification makes interoperability possible between other certified devices.

The architecture overview of this system is depicted in Fig 3.11. It is composed of two vital components: Movital and Monere. The first, Movital, is an hardware card developed to bring 6LoWPAN technology to medical devices, by adapting its basic communications like USB, RS232 and IrDA. This allows for internet communication and interaction of the collected data with other entities of the architecture. Monere, is located at the patient's house and is the gateway between sensors and patient monitors and the external information system. It allows for remote management, monitoring and repair of devices. Connecting these two entities is a data model called YOAPY which was designed to efficiently pre-process and aggregate the acquired raw data from the medical devices, so they can transferred with no overload and optimized payload size.

When carefully analyzing this solution - which is presented as an IoT one - a major issue comes to mind: lack of standardized semantics and representation of typical medical data in current IoT/M2M Architectures (ETSI's and OMA's). This work focus on non Continua Alliance compliant devices, but from the M2M/IoT technology point of view, this is something worth studying too. As the author's work dates 2012, lack of standard IoT/M2M Architecture implementation is something that must be negligible, since no architecture standard had been specified as of its publication. However, this serves as an example of how such systems will, in the future, be able to adopt such standardized Architectures, which will ease the integration and communication of the collected data with other applications and services. For instance, using LWM2M, it would be possible to register (at OMA's Object and Resource Registry Naming Authority - OMNA) the type of collected data from a specific device as a known object. If they have done so, further re-use of this collected data would be much simpler, as developers and vendors would already have the necessary tools and knowledge to interpret such data to enrich its other devices, services and/or applications. This applies only to devices which are not compliant to Continua Alliance's standard. Such devices are already interoperable with other certified ones, and thus, would only require Continua Alliance to register their objects at OMNA for further re-use of the data over the internet using protocols such as CoAP, as defined in Chapter 2. In fact, as the time of this writing, OMA officially acknowledged the importance of mobile healthcare in the standards industry and has announced that the next OMA Meeting (taking place August 25 to 28, 2014) will be co-located with Continua Health Alliance [78].

IoT can also make way to smarter WSNs which can stream data in real-time, allowing others (any kind of consuming entity) to monitor this data in real-time using the Web. As introduced before, this is actually part of the solution described further in Part IV and implemented and detailed in Part V of this Master Thesis. The authors of [79], a research article published in July 2013, have designed such a system which, fundamentally, makes use of a combination of 6LoWPAN and Zigbee networks streaming data to the Web via Web Sockets. It is an interesting work given its analysis of different technologies and methods to achieve a scalable, fast and efficient way of real-time web monitoring of sensed data. Its core functionality is somewhat similar to that of the solution here proposed and implemented, so, before digging into its pros and cons, it should be carefully analyzed. The system has also its own storage solution, but the analysis and discussion here presented concentrates solely on its effort to provide real-time Web monitoring of the sensed data.

The whole monitoring system architecture is depicted in Fig. 3.12. The end-nodes sense data and transmit back to the Gateway using the standard protocol (6LoWPAN or ZigBee); the Gateway has its own script which acquires this transferred data and forwards back to the WebSocket Server, via ethernet, using the WebSocket Protocol; and, the final component - WebSocket Server -, makes use of Node.js's module called Socket.IO to push the data into the client's browser which, in turn, renders the graphic using Chart.js. This is the overall functionality of the monitoring system and the data flow is best illustrated in Fig. 3.13

This solution is an interesting one as it provides a good real-time web monitoring solution. The authors have reported a

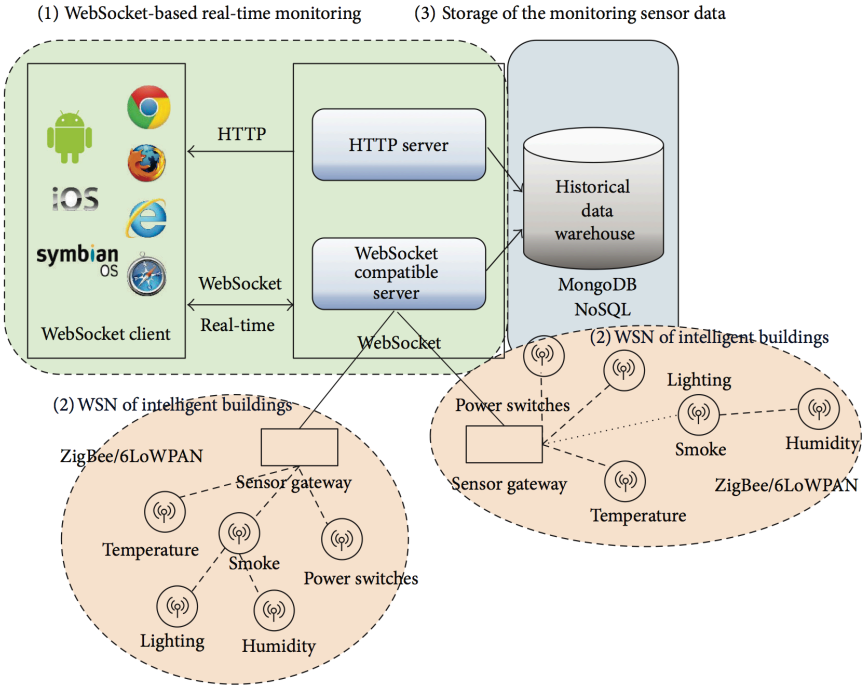

Figure 3.12: Monitoring system architecture of [79]

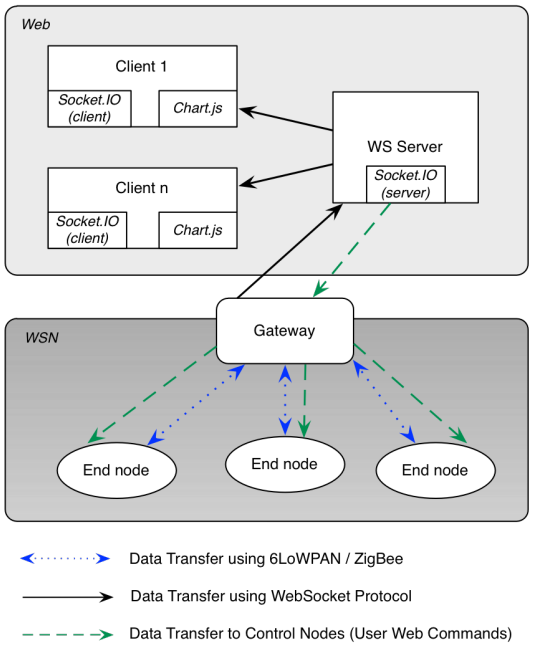

Figure 3.13: Simplified data flow of [79]

comparison between different technologies that could have been used to achieve the same goal: HTTP Polling, Sockets in ActiveX and Flash Sockets. They have compared their impact on latency, capability to lead with concurrency, and CPU utilization. As a result of this analysis, Web Socket turned out to be the most consistent option given that it outperformed any other option in every test.

To finalize, all that is left to say is that this solution proved to be suited for scenarios where real-time monitoring of a WSN is needed; however, as an IoT solution it lacks core functionalities. It is not totally clear if the authors wanted to achieve an IoT solution, but they do claim that "Our motivation of this paper is trying to address these two limitations to design a WebSocket-based real-time monitoring system for remote intelligent buildings. We extend the Internet of Things paradigms to support more scalable and interoperable and monitor the sensors for intelligent buildings in real-time and bidirectional manner." - as they mention their efforts in achieving a "more scalable and interoperable" IoT, it should be treated as so. As such, it lacks integration of common IoT protocols (CoAP, for example, as 6LoWPAN is included and, thus, also UDP) and a standard IoT architecture. Furthermore, it is unclear if the Gateway really is a 6LoWPAN Border Router and if the 6LoWPAN nodes make use of ND (Neighbor Discovery) to achieve global internet connectivity. Also, there is no way to determine the solution cost, as there is no mention of the used hardware, except for the machine used for experimenting.

3.6 Conclusion

The three different systems described before are prime examples of common deployed IoT solutions found in the literature. Although most of them are now starting to support 6LoWPAN, other protocols and system architectures are still to be adopted. This is also, obviously, due to the fact that these standards are fairly recent and some of them do not even have a final specification. It is also very rare to find real end-to-end IP connectivity in these typical WSN: although they do support 6LoWPAN, there is no mention of direct interaction with the web or any other IP application. Furthermore, they all fundamentally work in a silo manner, with nonexistence interoperability. Although it might suit the authors' intended solution, it lacks a concrete unique standard which would enable their solution to scale properly while also being efficient and flexible by making use of both a standard IoT/M2M standard architecture and protocols well tailored for this IoT solutions, as detailed in the previous chapter.

The architectural standards and frameworks detailed here (ETSI M2M, oneM2M, LWM2M, IoT-A and IPSO Alliance) are crucial for IoT to grow as intended. With these, one can design a complete solution that is both efficient (with intrinsic proper protocol bindings, when present), flexible (by means of expandable horizontal service layers), secure (using pre-provisioned keys for bootstrapping, like LWM2M, and privacy by making use of DTLS) and interoperable by means of semantics and adequate IoT application protocols (standard representation of common objects, such as those defined by IPSO Alliance, to be used in LWM2M through CoAP).

With all this in mind, the IoT System proposed next, and detailed further, takes these issues into account and uses as much as open-source IoT-designed protocols, architectures and tools as possible. This allows for a future-proof solution, while tackling the common scalability and interoperability concerns. It is also a low cost solution with a total cost of 150€ for the whole system: 6LoWPAN node (74,9€) , hear rate sensor (22,95€) and 6LoWPAN gateway and network server (74,95€). The components are listed and detailed further in the next chapter.

Chapter 4

IoT Solution for Real Time Fitness Web Monitoring

4.1 System Architecture

In order to achieve a more generic design that could potentially improve interoperability between other IoT Systems, a proposal is here presented, describing, first, how the implemented solution (described later in Chapters 5 & 6) could make use of state of the art standards and architectures to provide, ultimately, real interconnectivity and interoperability with other IoT/M2M Systems, making it a full-fledged real IoT System. The goal of this System Architecture is to showcase how IoT/M2M Systems should be deployed from now on, as there are new standards and frameworks that should always be adopted. It was designed with abstraction in mind, hiding much of what is represented with current IoT protocols and standards. Its foundation is based on middleware abstraction layers, ETSI M2M Architecture, LWM2M and smart objects defined by the iPSO Alliance.

IoT should be open and all things should be able to communicate with each other, even if indirectly - one device, or "thing", should be able to alert other "thing", which is part of a different IoT System, that something occurred. With this, many autonomous and intelligent systems could be born to provide an even better quality of life, since there would be smarter and broader coordination of data and their corresponding action triggers. An example would be the IoT System deployed on an elderly disabled person's smart house: if the system detected a major fall, it could immediately alert that person's family and even trigger an alert on a nursing facility's IoT System, so that they could take immediate action by sending appropriate help to the correct location. These different IoT Systems would be independent and would not be designed with this interactivity

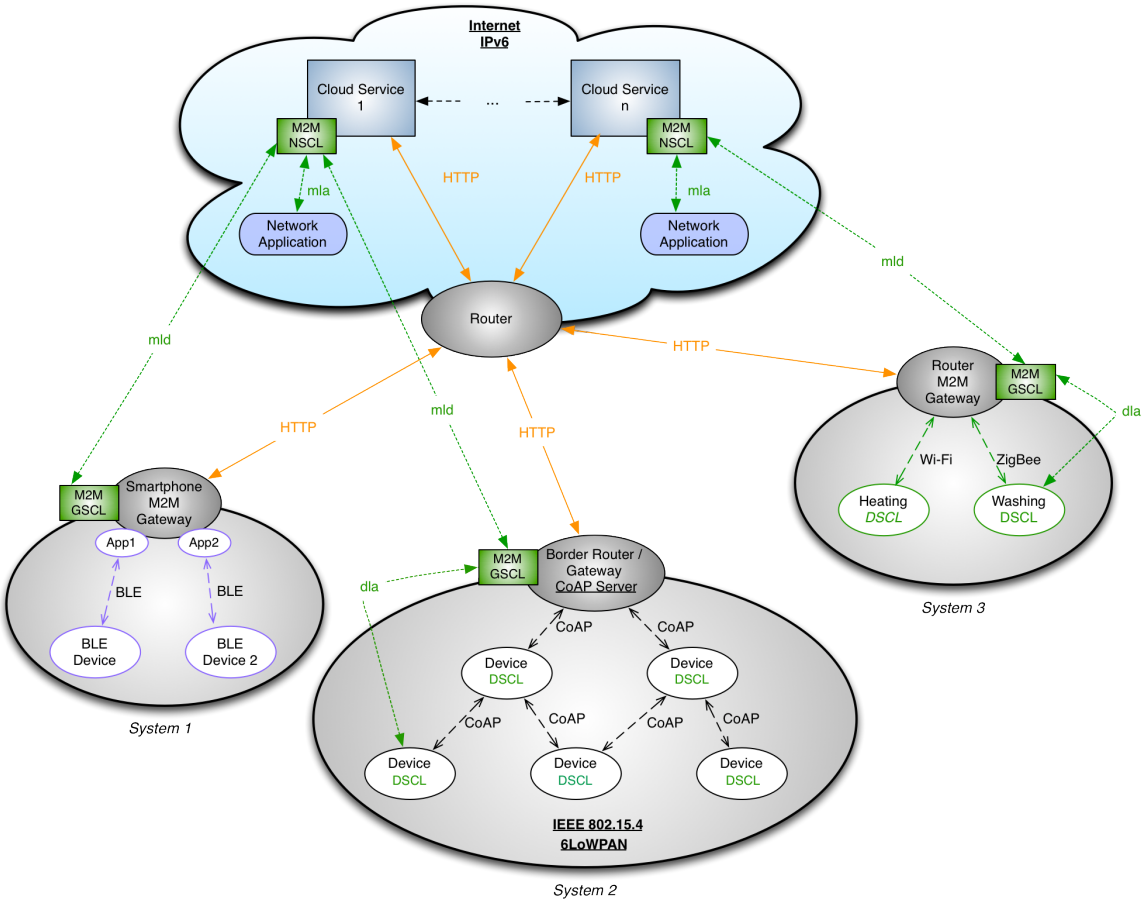

Figure 4.1: IoT System Architecture Proposal

in mind - in the future, already deployed IoT Systems will not communicate with other IoT Systems, because they were not designed to do so nor have the needed capabilities. This is why IoT needs a standard System Architecture, so that new systems can, in the future, talk with others, creating more intelligent, useful and efficient applications.

With this in mind, Figure4.1 illustrates the layers, at different levels, needed for all of this, and how different systems could interact with each other.

The added flexibility comes from the use of ETSI's M2M standard interfaces (mIa, dIa and mId) and Service Capabilities Layers at different domains: device's, gateway's and network's. With ETSI M2M specification as the core of the proposed architecture, different IoT Systems would be ready to interact with each other, even if they do not use the same protocol at the device domain. This way, a system using 802.15.4 to transmit MQTT-SN messages, would also be able to send data to another Gateway from another system, independently of its protocols in use. To put in other words, an MQTT node would

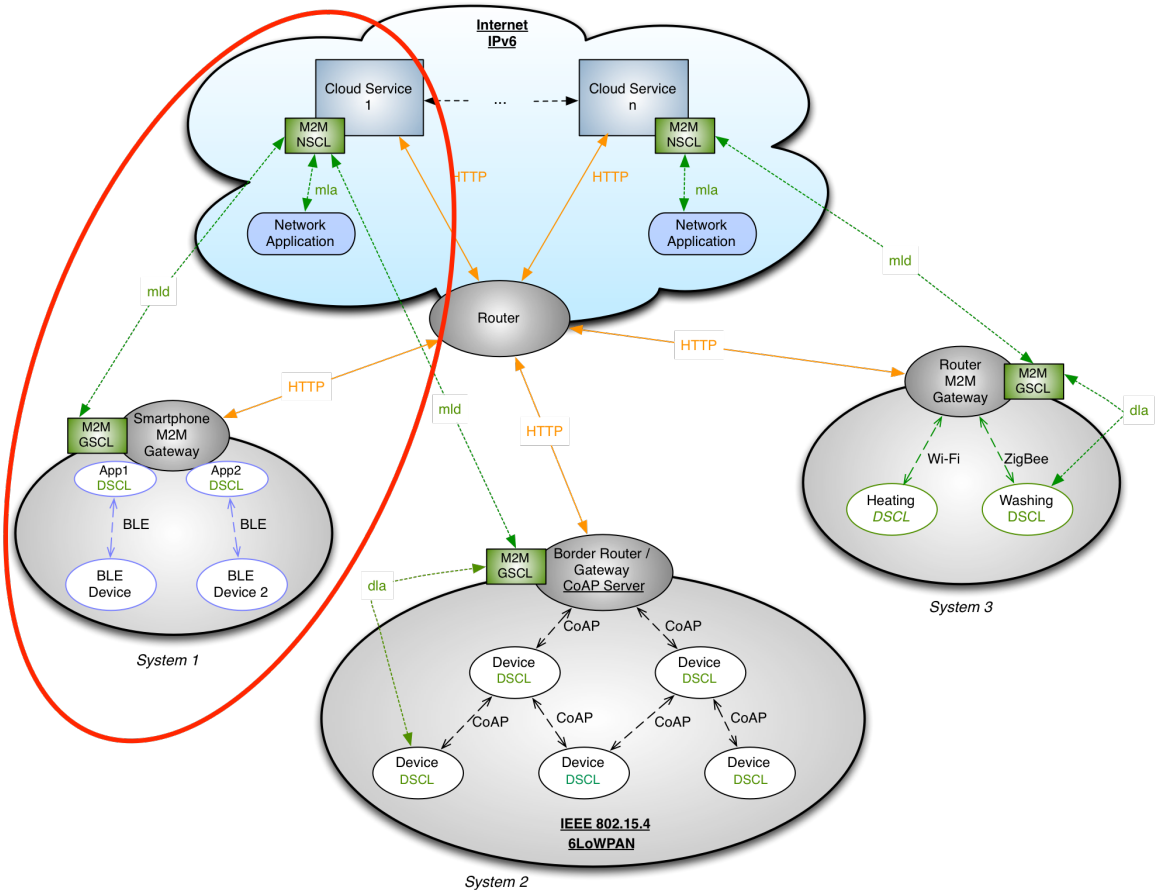

Figure 4.2: Current M2M/IoT Silos

talk to a different CoAP node in a different system, while also having both a different radio link (802.15.4 and Bluetooth, for example).

With a proper M2M/IoT architecture, a specific device, which was initially designed to work in a silo manner (Fig 4.2), could then interact with a different Network Application (NA) and/or Gateway, providing both a different service to the user and/or an uninterrupted service, if the mobile device connected to a different Gateway, at a different location. This is something ETSI TC M2M has also realized, when analyzing TS 102 689 [80], which refers to M2M Service Requirements. This Technical Specification describes two particular requirements that are much needed for IoT Systems. Although not mandatory for all systems, they were described because of the work reported in previous TR (Technical Requirements), where different use cases for different areas were presented, as mentioned in Chapter 3. To satisfy much of those use cases, the two particular requirements were: Trusted Applications and Mobility. Trusted Applications means that the M2M Core could

handle requests for trusted M2M Applications by providing a streamlined authentication procedure - the M2M System would then support Trusted Applications, as they were pre-validated by the M2M Core. The Mobility requirement simply means that if the underlying network supports such mechanism and roaming, the M2M System should be able to make use of such mechanisms. Using Fig 4.1 as a reference, it is easier to showcase these two functionalities: Trusted Applications (Fig 4.3) and Mobility (Fig 4.4). Combining the two of them with LWM2M's pre-provisioning and auto-configuration of devices, would result in a truly ubiquitous IoT System where connectivity, reachability and intelligent coordination would always be possible, which could then make way for smarter Network Applications and new overall services at a personal (end-user) and industry level. With LWM2M's bootstrapping capabilities, any kind of device would be assured of continuous communications if moved from its original domain area or IoT System. In other words, future Bluetooth mobile devices with IP connectivity (Bluetooth Smart [81, 38]) would be able to auto-connect to other Gateways of different IoT Systems, and, thus, ensure continuous Internet connectivity. The seamless integration with current Internet services would be possible trough a well documented RESTful API, support for known semantics and various protocol bindings. For instance, having Fig. 4.1 as a reference, Network Applications from System 1 could discover devices present at System 2, through ETSI M2M discovery mechanism (or even CoAP's, if supported by the node itself), consume the data with a protocol of its choice (some protocols are more suited to specific application/use cases then others) then properly interact with it by making use of known semantics (like IPSO's).

Having this in mind, it is easy to conclude that one M2M/IoT standardized architecture like ETSI M2M might not be enough for all solutions. A combination of different ones with proper bindings to different protocols, use of known semantics and a simple, yet well documented, RESTful API with which one can easily interact with and integrate into the exiting Web, is what is currently needed to achieve the so called Internet of Things.

Next, the lower-level architecture of a developed IoT System is detailed (all the building blocks of it) and, in Chapter 7, the details of such practical implementation is reported. The IoT System described next follows the architectural guidelines mentioned before, using as much both free and open source code, tools and frameworks as possible.

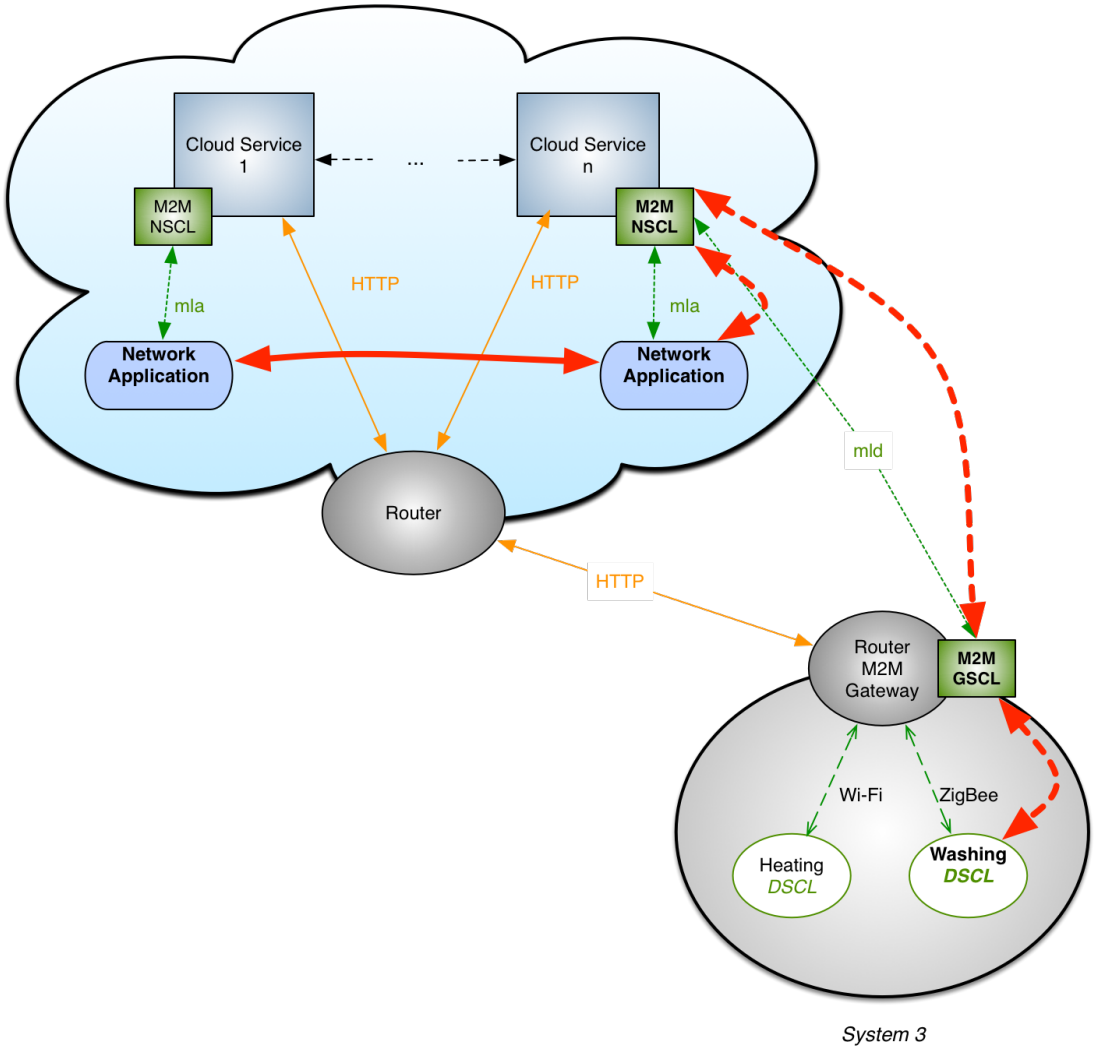

Figure 4.3: IoT System Architecture Proposal - Trusted Applications

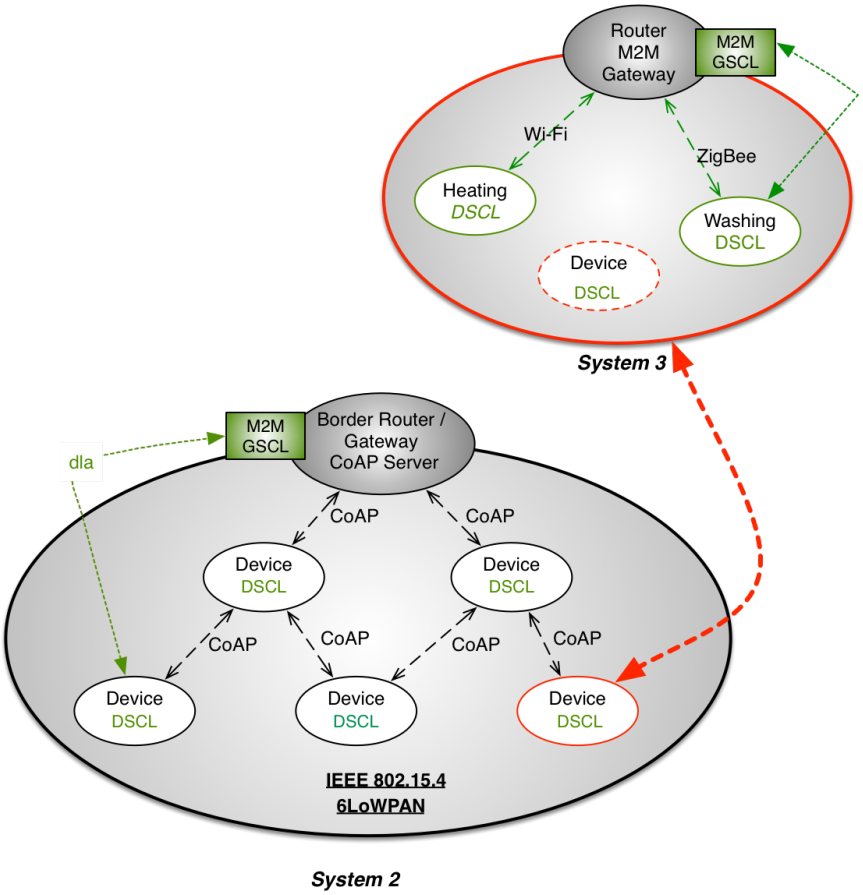

Figure 4.4: IoT System Architecture Proposal - Mobility

4.2 Testbed Solution

As a mean to showcase the power of deploying an IoT System with a proper standard and set of protocols, a low cost heart-rate monitoring solution was set. The testbed set allows the wearer to stream the data wirelessly to the internet, using a battery-powered wireless device that is , which is then made accessible to other interested parties. One entity (doctor, physician or any other entity) has real-time access to this data, via web browser, while the other entity is simply alerted of such alterations. The first simulates the need to get the data with little latency as possible, while the second simulates any other entity (nursing facility or hospital's own IT system) which is interested on this data and does not need to access it as fast as the first one. Moreover, this second entity might need to simply store the aggregated data for historical purposes.

From the technological point of view, there are a set of aspects that must be considered for this type of scenario:

- Devices being used: as low cost as possible, while being powerful enough to perform the necessary tasks.

- Networking technology: the wearer is a moving node, so there is a need to make sure the linking is reliable while being low power at the same time.

- IPv6 connectivity: the interest in this case is the use of Internet of Things to its full potential. Bringing this object to the internet is needed trough a unique global IPv6 address.

- Transport protocol: as mentioned before, there are a few protocols for IoT. The most suited for the specific domain and practical application should always be chosen.

- Interoperability, Scalability, QoS & Security - as with almost every system, these issues have to be dealt with. Interoperability in IoT plays a big role, so there is a need to make sure the developed solution is compliant to most recent IoT standards, making it also future-proof. The network itself must also deal with scalability - it must be able to deal with a larger amount of nodes, properly handling the traffic load by means of adoption of efficient protocols and load balancing capabilities. QoS & Security are of major importance also - we need to make sure the network is reliable and that we deliver data as fast as possible (real-time monitoring), while protecting user's privacy since we are dealing with personal physiological data.

Next, the System Architecture is presented with all of these issues thought out. All of its parts are carefully explained over the following sections.

4.3 Solution Overview

After considering all of the above mentioned, the system presented in Figure4.5 was designed. Simply put, the system is composed of a WPAN and Network Services. The WPAN is where all the data is sensed to then be available to Network Services, where Applications make use of it. Arduino nodes will make use of CoAP PUT messages to update the resource - the heart rate monitor, in this case. This data is transmitted over IEEE 802.15.4 RF link using 6LoWPAN as the adaptation layer for IPv6 connectivity. The Raspberry Pi works as a 6LoWPAN Border Router which will bridge the connection to and from IPv6 and 6LoWPAN, allowing the Arduino nodes to be completely reachable over the Internet. Once the Gateway collects this data, it must deliver it immediately to the real-time subscriber and properly store it, so others can also fetch it.

To make it immediately available for real-time monitoring, a special mechanism must be developed. For this, the Gateway must bridge the incoming CoAP PUT messages, sent by the Arduino node, to MQTT. This protocol was chosen, because of its efficiency and publish/subscribe type of messaging. More over, with MQTT, it was possible to stream the data directly to the Web Applications, by making use of Web Sockets and a MQTT Javascript client library. As the solution should not work in a silo manner, this data is also properly forwarded to the services specified by ETSI M2M standard (detailed further).

The overall look of the system is depicted in Fig. 4.5with much of the underlying features hidden, for simplicity sake at this stage. The next chapters will go into much more detail covering the unlisted features and components while also explaining the reason why such protocols, detailed in Fig. 4.6, were chosen.

Hardware

This system uses Arduino as sensor nodes and Raspberry Pi as a Gateway. The reason for using Arduino is simple: it is a great open source prototyping platform with big software support, cheap, and expandable by means of shields. For this reason, the sensor node is composed of:

- Arduino MEGA [82];
- Wireless Proto Shield [83];
- XBee 802.15.4 Series 1 [84];
- Pulse Sensor [85]

With this configuration, it is possible to hold 256KB of Flash (needed for the libraries), sense heart rate data and communicate over IEEE 802.15.4 RF.

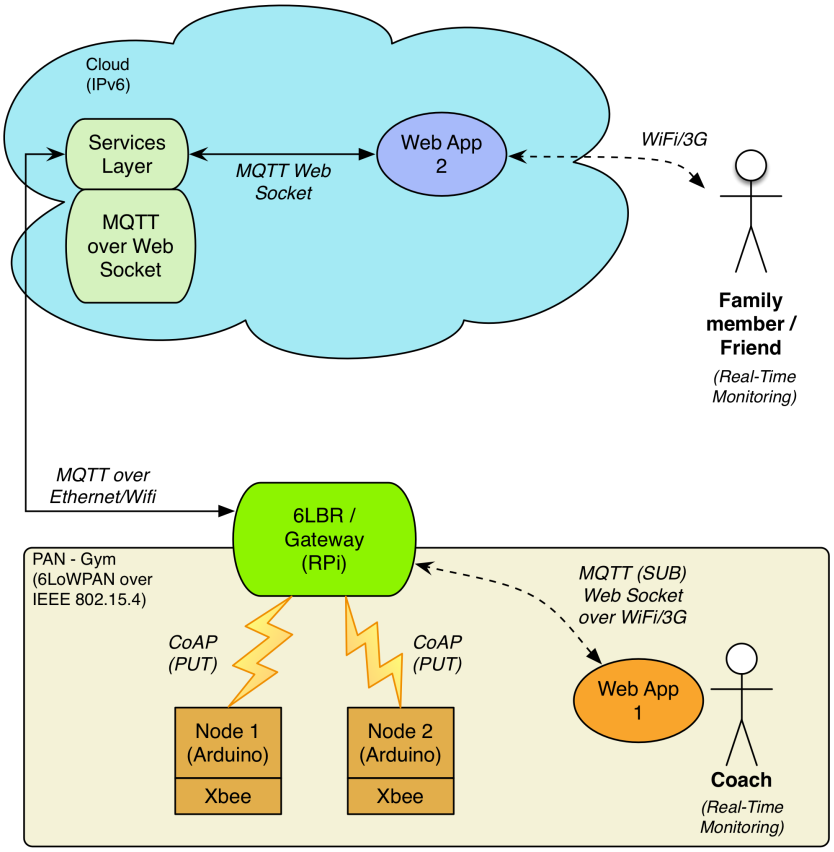

Figure 4.5: Solution's Architecture

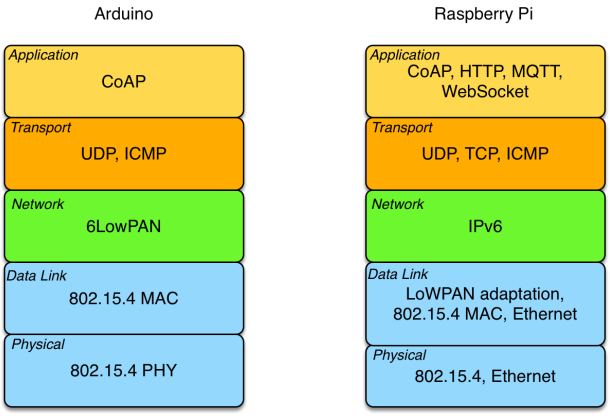

Figure 4.6: Solution´s Protocol Stack

The Platform chosen to work as a Gateway and Border Router of the PAN was the Raspberry Pi. It is a powerful, yet very efficient, Linux Computer capable of working as a 6LowPAN Border Router (to route between the 6LoWPAN and IPv6 network). This means that it is flexible enough to install any software needed to support the whole system, which, ultimately, provides a way to tweak and enhance its QoS and security if such software modules were developed. The 6LowPAN Border Router / Gateway is then composed of:

- Raspberry Pi model B [86]with Raspbian OS;
- NooliberryT card by NooliTIC to provide interconnectivity between IEEE 802.15.4 RF link and Ethernet [87].

4.4 Strengths

This solution is a fresh approach when it comes to IoT system designing. It breaks away the current silo communications manner, by adopting latest IoT open standards and protocols. As these are rather new - specially the IoT/M2M architectures - there is not yet many tools and frameworks to be included. However, there are a few and can be used as a base layer to provide interoperability, scalability, QoS and security across all domains. For this, ETSI M2M was adopted for the whole solution with extended capabilities to support real-time monitoring.

Adopting ETSI M2M Architecture

While the solution presented may work for the specific testbed, it does not qualify as a full-fledge IoT System. To achieve such a thing, a standard architecture must be adopted. As a result of the analysis done in Chapter 3, OMA's LWM2M standard combined with ETSI M2M (future oneM2M) would be the best option; however, at the moment of this writing, there is a lack of Arduino Client libraries and, being so, ETSI M2M was chosen with a specially developed Interworking Proxy at the Gateway (GIP) to support the Arduino nodes. It was not chosen because there are Client libraries for Arduino, but because it is possible to implement a real ETSI M2M architecture while also having non-ETSI-compliant devices (Arduino). This is what ETSI defines as a 'd' device: "non-ETSI-compliant devices ('d') that connects to SCL using the xIP Capability (GIP, NIP, DIP)" [48]. As a result, a GIP (Gateway Interworking Proxy) was designed, as an application, to provide interworking between the Arduino and the GSCL, via reference point dIa.

To make the solution compliant to ETSI M2M architecture, the OM2M - the Open Source platform for M2M communication [88]- was adopted, alongside a Node.js module called Ponte [89] to help build up the GIP. With OM2M, it was possible to deploy both a GSCL and a NSCL - all there was left to do was to connect them all together with the previous solution, using ETSI M2M reference points and all the procedures defined by the standard.

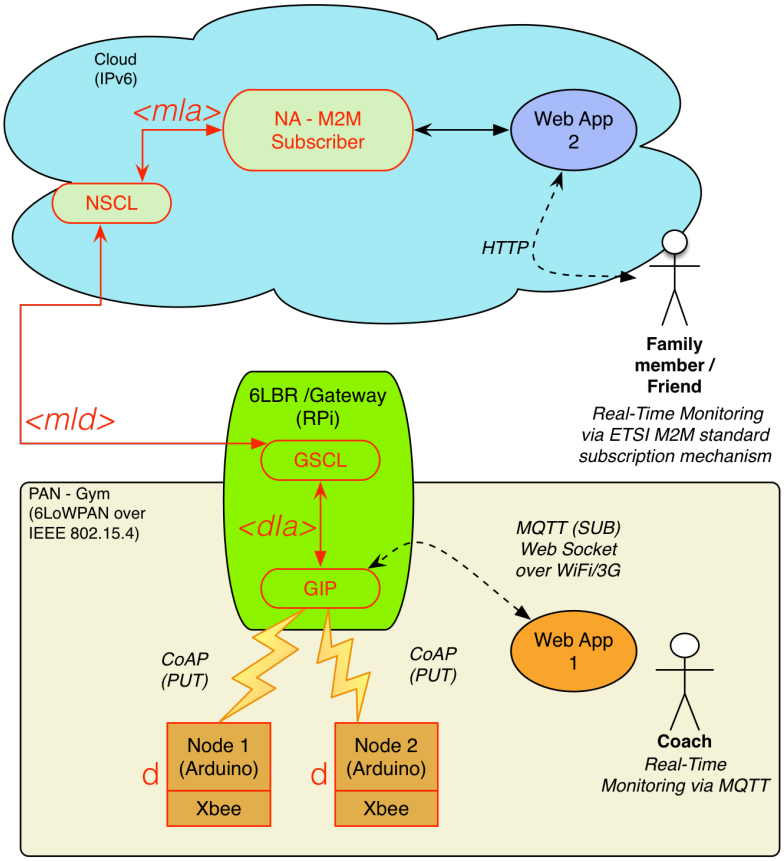

Figure 4.7: Solution's architecture after ETSI M2M adoption.

As a result of this adoption, the previous architecture was modified to what Fig 4.7 depicts. This image showcase the different xSCL, reference points and also the NA, the Network Application responsible to subscribe to the resource (Heart Rate Monitor application), the developed GIP (Gateway Interworking Proxy) and the real-time monitoring solution. This last special requirement (real-time monitoring) is properly detailed in the following section.

Nonetheless, this is not how the actual real representation of the deployed system is. As only one Raspberry Pi was used, the network-domain modules are also present at the Gateway. This is simply a matter of resources availability (the NSCL should be hosted on another server or Raspberry Pi at a remote network domain). However, the system is designed to work with other machines as well, since the communication between the modules is not localhost-restricted, but configured to communicate over a specific IP and Port.

Node.js & Real-Time Web Monitoring Support

Having the system in compliance to ETSI M2M standard is a good starting point, however, that standard, as it is, lacks some functionalities required for this particular solution. Specifically, there is no real-time monitoring support. ETSI M2M does, in fact, support synchronous (long polling) and asynchronous subscriptions to new events, but those mechanisms rely solely on HTTP POST messages for every new update. The synchronous type of subscription is not a real push notification mechanism, as the client needs to keep polling the server for new data, which is not optimal for real-time monitoring. On the other hand, the asynchronous mechanism does support instant notifications to a specific listening application. This last mechanism was adopted for the developed NA, which is a Network Application simulating other interested parties in the same sensed data. It is detailing and implementation is described in the next chapter. As there are specific protocols designed that work best for this type of messaging pattern (publish/subscribe, or PUB/SUB), like MQTT, this protocol was used to enable real-time web monitoring, when coupled with Web Sockets technology.

To support this real-time web monitoring, Node.js modules were adopted. Node.js is a "platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications." [90] Its event-driven non-blocking I/O model make it ideal for this kind of IO-bound applications, as stated by the authors: "Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices." [90] In a simplified manner, Node.js manages to do just that by taking away the normal waiting time for I/O tasks to complete and replacing it by small CPU activity, by means of an event loop and a thread pool. The Node.js event loop is a single-thread application, however, the I/O is delegated to a thread pool which is maintained by the OS. The event loop keeps grabbing code from the event queue and execute it, while there is no callbacks from previously stacked IO tasks. Once the previously I/O task is completed, the callback will be picked up by the event loop and process it right away. This mechanism is represented in Fig. 4.8. It is a little bit different from what is typically found in other language programming models, but it serves its purpose (fast and scalable IO-bound applications) very well, as it abstracts the low level complex event loop and all the necessary OS's callbacks in a very simplified way, as a Javascript programming language. It is possible to do so, due to its bindings to the OS and the fundamental dependencies on libraries supporting thread pool (libeio [91]) and event loop (libev [92]). These low level bindings and dependencies are written in C++ and C code while the upper node standard library allows for developers to leverage this non-blocking I/O programming model by means of Javascript's callbacks and anonymous functions. For this matter, the whole GIP and its built-in real-time web monitoring support was developed as single-thread Node.js application. By making use of the appropriate modules, it was easy to provide support for MQTT and Web Socket communication, under the GIP. Further detailing on this subject can be found in the next chapter.

Extending ETSI M2M to support bindings for this type of communication protocols greatly enhances its overall ability to support different IoT applications, as each of them have its own pre-requisites. The OneM2M partnership is already working on this, as they have too realized the importance of proper built-in protocol bindings [60]. This mechanism could also be

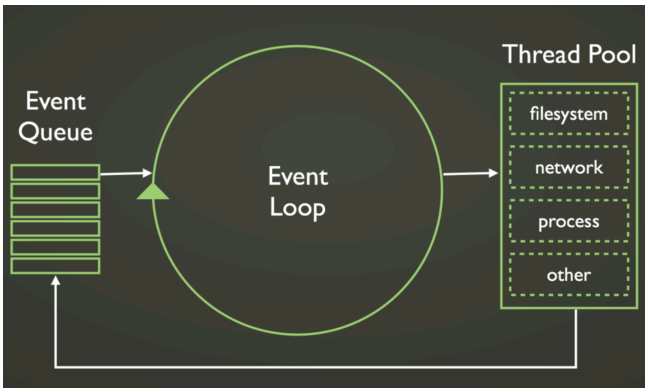

Figure 4.8: Node.js's event loop [93]

incorporated onto the OM2M platform, as it supports the development of external plugins; however, due to the real-time nature of the application at hand, an external application developed entirely on Node.js supporting the Interworking Proxy (needed for the Arduino and its CoAP to work with ETSI M2M, as explained before) and MQTT plus WebSocket support was thought to be the best solution. These separate applications (the NA making use of ETSI M2M standardized subscription method and the Web application making use of MQTT and Web Sockets) allowed for setting up an experiment where different metrics were measured, comparing both latency the traffic load differences. The results can be found in the next chapter and they support the need for IoT/M2M standards to built-in different protocol bindings.

Bluetooth Low Energy Comparison

Bluetooth Low Energy (BLE) technology is well suited for IoT and typically used in the fields of healthcare and sports [94]. It was introduced when Bluetooth Core Specification 4.0 came out, in June 2010 [95]. It really is Low Energy when compared to IEEE 802.15.4 and ZigBee [96, 81]and, for this reason, a comparison of its use in the proposed solution is necessary.

As advantages, BLE has a theoretically larger Data Rate of 1Mbps when compared to 250Kbps of 6LoWPAN over 802.15.4 - however, the application throughput is only 270Kbps [95]and authors from [97]found it to actually be only 236,7Kbps; it is a low cost technology when compared to 802.15.4 - almost 3 times cheaper [96]; and, as stated before, it consumes less energy. Nonetheless, it has disadvantages: network topology limitation; larger implementation size and no IPv6 support. It currently supports only one single-hop star network topology [97] and the standard specifies that one node can only belong to one piconet at a time [96].

Analyzing the aforementioned pros and cons of Bluetooth Low Energy, IEEE 802.15.4 is still preferable for this specific scenario. This is true for different reasons:

1. Larger data rate of BLE is not of importance in our scenario as CoAP PUT messages are small enough.

2. Network topology limitation makes this scenario nearly impossible to achieve. For the particular scenario, it would be very challenging to transport data throughout the network, given the fact that slave nodes are mobile and the master is fixed. If the Slave node (wearer) got too far from the Master (Border Router), communication would be impossible, since it could not rely on other slave nodes. The only solution would be to add other Master nodes, thus creating different piconets, allowing nodes to jump between them if multi-hopping was needed. However, this would result in much more complex, costly and unreliable networking.

3. No IPv6 support. Sensing nodes must be directly connected to the internet for network testing and to allow for truly global connectivity. Although there has been some attempt to bring IPv6 support to BLE [39], this solution is not a standard one and there are no details on whether it is energy efficient to do so or not. Bluetooth SIG has been doing some official work on this field, though [38] [39].

One major advantage would be the direct transmission of sensed data to the Coach's tablet/smartphone, since most of this type of devices now support BLE technology. However, if this was to be implemented, it would be more complex to route this traffic to the Cloud and make use of the pub/sub messaging, meaning that no other Web Application could consume this data. Also, the Coach's would always have to be in the required range of BLE's transmission range to allow for such thing, which is not always the case in this scenario, since the wearer is scattered and not in a fixed position.

4.5 Conclusion

In contrast to what can be commonly found in the literature, where IoT systems are deployed, this chapter showcased how such systems can actually be developed using real, specific designed and proper Internet of Things standard architectures and protocols. It denotes the importance of expanding the most complete M2M/IoT standard to date, ETSI M2M, and how its conjunction with others (LWM2M and iPSO Alliance's smart objects) will ultimately help to achieve the so desired Internet of Things. The IoT, as it is, will not reach the potential that is commonly thought of, due to its interoperability issues, scalability problems and lack of proper protocols to handle the overall needed communication patterns and security. The System Architecture defined for IoT, in the beginning of this chapter, illustrates how it should actually work and how it is becoming possible to achieve such a thing with the evolution of the latest standards described in the previous chapter.

The high-level architecture of the developed IoT solution is also documented, emphasizing the importance of adopting IoT-specific standards to ease the interoperability of different systems and the importance of incorporating different protocol bindings to satisfy the particular requirements of different applications. In this case, the ETSI M2M standard was adopted

which greatly facilitates the development of other applications that consume the same data in a standard and well documented manner, by means of a RESTful API. As an example of such applications, the NA was implemented. The developed GIP (Gateway Interworking Proxy) also denotes how important it is to develop special proxies for legacy devices, as they are not compliant with latest standards. Also important to notice is the added support for real-time web monitoring support, as it demonstrates the need for IoT/M2M standards to incorporate proper protocol bindings or, at least, be flexible to allow the development of such.

The next section will detail all the hardware used, software modules and communication paradigm between those modules which forms the final solution.

Chapter 5

System Implementation, Configuration and Validation

5.1 Implementation Overview

As the solution is composed of different software modules, each of them will be detailed separately. Fig 5.1 showcase the software grouping at the Raspberry Pi for easier understanding.

As not all ISPs support IPv6 natively, a tunneling system had to be used. This was also required for this system as both a static IPv6 and a prefix are needed by the RPi. With it, reachability is possible (due to the static IP) and configuration of 6LoWPAN nodes' IP is also possible using the assigned prefix (by means of both gogoc [98]and radvd [99] software). The tunnel broker used was Freenet6 [100] and an account was registered so that a static IPv6 and a /56 prefix could be assigned to the gogoc. With gogoc, radvd and 6lbr properly configured, the whole system (Raspberry Pi and Arduinos) was ready to work as an IPv6 network with internet-wide reachability.

To facilitate the decoupling of the different modules, their meaning and relationship, one should understand how the gateway works in a sequence manner:

1. Establish an IPv6 tunnel to get a static IPv6 and a /54 prefix to then send Router Advertisement messages to its connected 6LoWPAN nodes:

   (a) gogoc configured to work as a Router and to connect to Freenet6's broker using the previously-obtained registered account details. gogoc (gogoClient) is a software whose author is gogo6 - the same entity who offers the

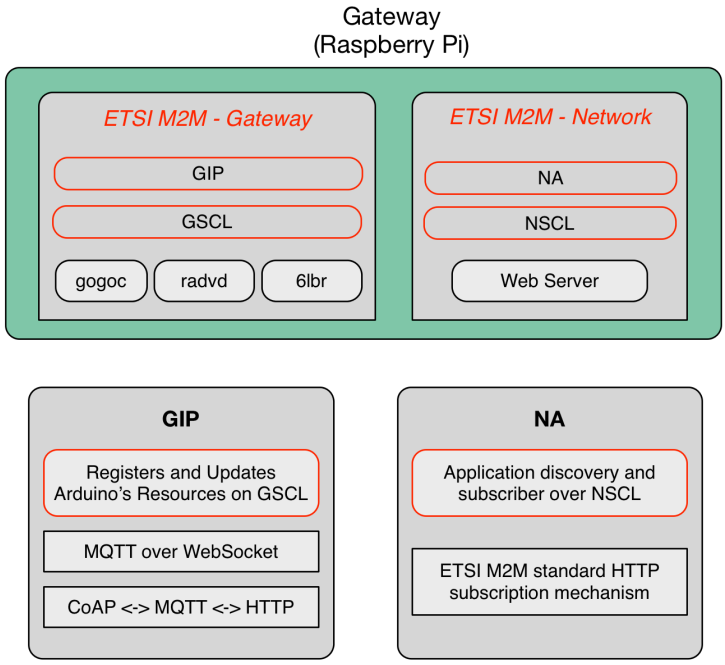

Figure 5.1: Raspberry Pi's software modules overview

Freenet6 tunnel broker -, which implements TSP (Tunnel Setup Protocol). In this solution, gogoc was configured to establish a v6udpv4 tunnel which means IPv6 encapsulated in UDP over IPv4 and is best suited for nodes (Router in this case) which are located behind an IPv4 NAT.

(b) make use of radvd to work with the special configuration file previously generated by gogoc. radvd (Router Advertisement Daemon) is an open-source software which sends RA (Router Advertisement) messages via the attached interface; these RA messages are sent when the Arduino nodes multicast RS (Router Solicitation) messages which allow for these nodes to proceed with the stateless auto-configuration and, thus, get their own global IPv6 address.

2. Start 6lbr and start working as the 6LoWPAN Border Router of the WPAN. This is the module needed to make the Gateway (Raspberry Pi) work as a 6LoWPAN Border Router. It provides seven different modes of operation, each having its own particular purpose. Smart Bridge Mode was the chosen one, which seamlessly interconnects an IPv6 network to a RPL-based WSN. In this mode, the integrated ND Proxy will configure its WSN-connected nodes. The proxy translates source and destination MAC addresses so that any standard IPv6 node, on the Ethernet side, can connect to the IPv6 node on the 802.15.4 WPAN side and vice-versa. This mode of operation is detailed further, in its dedicated section. As 6lbr can handle prefix in auto-configuration mode, it will use radvd's RA messages to configure its own global IPv6 address using that prefix and will then propagate it on the WSN side. This is not the

typical solution, though, as the RA should be sent by a full-fledge IPv6-capable Router - since there is none in this system, the combination of the local gogoc and radvd were most needed.

3. Start GSCL and NSCL. These two Java modules running on top of an OSGi Equinox are part of the oM2M eclipse project [88]. They are compliant to latest ETSI M2M specification and are key elements to enable an ETSI M2M Architecture on this solution. GSCL will interact with both GIP and NSCL. When both start, the GSCL automatically connects to NSCL and registers its resources.

4. Start GIP (Gateway Interworking Proxy) module. This module was developed using Node.js which is ideal for fast Real-Time applications as it scales very well and thrives when used in I/O bound type of applications. With its bridging capabilities over CoAP, HTPP and MQTT, it was designed two serve two purposes:

   (a) Assists the Arduino non-ETSI-compliant nodes interaction with GSCL. By analyzing the CoAP PUT messages, it understands whether the Arduino node wants to register (at boot) or update its resource (new PUT message) in the NSCL. Once determined the desired action, it starts the standardized way of interacting with GSCL (HTTP POSTs with XML representation).

   (b) Allows for real-time monitoring of the sensed data (CoAP PUTs sent by the Arduino nodes) by means of a CoAP to MQTT translation and use of MQTT over WebSockets. This way, it is possible to offer real-time access to sensed data, as it is much faster than the standardized way of direct interaction with NSCL.

5. Start NA (Network Application). This application is responsible for discovering all applications registered in the GSCL and subscribe to them. . This method complies to ETSI M2M specification and it is the correct method to get every new update sent by the Arduino to the GSCL.

6. Start the Web Server. The Web Server is needed for the real-time web monitoring. With this static page, an MQTT connection to the GIP is established using Web Sockets to communicate new incoming heart rate data as soon as it is collected by the GIP.

From the Arduino nodes end, they fundamentally serve one purpose: sense data and report it back to the GIP. However, before they loop on that state, there are three phases they go through first:

1. Initialize Xbee and IPv6 modules;

2. Multicast RS (Router Solicitation) and wait for RA (Router Advertisements);

   (a) Auto-configure its own global IPv6 using the /56 prefix sent by the Border Router;

This is the overall interaction and of the whole system. Further reporting on installation, configuration and deployment of such modules is detailed in the next chapters.

5.2 Software Configuration and Deployment

gogoc

This module is responsible for making the Raspbery Pi to work as an IPv6 router with its static IPv6 and /56 prefix. gogoClient, the software which automatically configures an IPv6 in IPv4 UDP TSP tunnel as specified in RFC 2893 [101], and is crucial for this solution to bring both IPV6 and 6LoWPAN capabilities to the whole network. The gogoc's version 1.2 source code was downloaded from gogo6 website [98] and compiled in Raspbian (Raspberry Pi's OS which is based on Debian and optimized for the ARM architecture found on Raspberry Pi boards [102]). The commands needed to do so were the typical make and make install, specifically "make && make installdir=/usr/local/gogoc". It did not, however, worked at the first time, as there were missing dependencies. These errors were fixed by installing the reported missing ones using "sudo apt-get install <missing package>".

Once correctly installed, it was only needed to configure it and tweak the included UNIX bash script responsible of handling all the network-related configuration. The included configuration file, located at /usr/local/bin/gogoc/gogoc.conf, was used as a template for the new one which was named gogoc_router.conf. The modifications needed and/or added, were:

```
# username and password registered at gogo6's website to make use
#of static IPv6 and /56 prefix
userid=pedrodiogoamsterdam
passwd=jbrsL46KjYci
# the tunneling service
server=amsterdam.freenet6.net
# let gogoc choose the most secure method
auth_method=any
# needed to handle prefix propagation
host_type=router
# this is the interface 6lbr software is bridged with
if_prefix=br0
# IPv6 in IPv4 UDP.
# Needed for NAT traversal as the Raspberry Pi is bellow an IPv4 NAT
tunnel_mode=v6udpv4
# maximum logging for debugging purposes
```

```
log_file=3

log_filename=/var/log/gogoc/gogoc.log
```

The comments (after the '#' character) are self explanatory, except for the "host_type" parameter. By setting it to "router", gogoc automatically configures a proper configuration file for radvd to use. This is detailed next.

radvd

Router Advertisement Daemon (radvd) is a Linux and BSD application needed for hosts which need to work as an IPv6 router. It works as specified in RFC 2461 (Neighbor Discovery for IP Version 6 (IPv6) [103]) and thus sends periodic RA (Router Advertisement messages) or whenever the Arduino nodes send their RS (Router Solicitation) messages when establishing their networking at boot (IPv6 stateless auto-configuration). This daemon is launched by gogoc's bash script after the tunnel is successfully created. Once the tunnel is created, this is radvd's generated configuration file:

```
##### rtadvd.conf made by gogoCLIENT ####
interface br0
{
AdvSendAdvert on;
AdvLinkMTU 1280;
prefix 2001:05c0:1508:8600::/56
 {
 AdvOnLink on;
 AdvAutonomous on;
 };
};
```

This software was installed using "sudo apt-get install" with no problems. The only encountered problem was with the previously mentioned gogoc's bash script when starting radvd. That default bash script started radvd with the command

```
exec $rtadvd -u radvd -C $rtadvdconfigfile
```

which was failing since there is no UNIX user called radvd. After modifying it to

```
exec $rtadvd -C $rtadvdconfigfile -l /var/log/rtadvd_gogoc.log
```

it worked straight away. As one can understand from the configuration file represented above, it is advertising on interface br0 which is the interface 6lbr is bridged with. This is crucial for 6lbr to work properly and, therefor, the whole 6LoWPAN.

Dynamic DNS

A free dynamic dns provider was used for simplicity's sake. The Border Router has its own global unique IP assigned via the IPv6 tunneling broker, but IPv6 addressing is complex and not easy to remember. For this matter, an account at DUIADNS was created [104] and the hostname "rpi.duia.us" was set.

6lbr

This software is a "A deployment-ready 6LoWPAN Border Router solution based on Contiki" ([105]) maintained by CETIC (Centre d'Excellence en Technologies de l'Information et de la Communication) [106]. It was designed to work with some open source embedded hardware platforms like the RaspberryPi when using the appropriate hardware for 802.15.4 connectivity (NooliTIC's Nooliberry T, as mentioned before).

To install 6lbr, the appropriate wiki page of its Github's page was used as a guide [107]. Starting with needed dependencies, the following were installed:

```
apt-get install libncurses5-dev bridge-utils
```

After that, it was installed from sources using the stable master branch:

```
git clone https://github.com/cetic/6lbr
cd 6lbr
git submodule update –init –recursive
make native
make plugins
make tools
sudo make install
```

Once properly installed, it was time to make it work with Nooliberry T. Again, the official Github wiki page of NooliberryT was used. The line

```
console=ttyAMA0,115200 kgdboc=ttyAMA0,115200
```

and

```
T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

were removed and commented out from /boot/cmdline.txt and /etc/inittab respectively. This was needed since Raspberry Pi's UART (serial port) appears to Raspbian as a device under /dev/ttyAMA0.

After all this, 6lbr itself could finally be configured as represented. As mentioned before, there are seven different modes of operation once can chose and, for this system, Smart Bridge, which is detailed next, was chosen.

```
#This file contains a default configuration for Raspberry PI platform using
```

```
#a Nooliberry SLIP Radio
#The full list of parameters and their meaning can be found in 6lbr.conf.example
#MODE=ROUTER
MODE=SMART-BRIDGE
#MODE=RPL-RELAY
#MODE=FULL-TRANSPARENT-BRIDGE
#MODE=NDP-ROUTER
#MODE=6LR
#MODE=RPL-ROOT
RAW_ETH=0
BRIDGE=1
DEV_BRIDGE=br0
DEV_TAP=tap0
DEV_ETH=eth0
RAW_ETH_FCS=0
CREATE_BRIDGE=0 #do not create at launch.
#taken from 6lbr's wiki/faq.
#solves the internet connectivity issues
DEV_RADIO=/dev/ttyAMA0 #NooliberryT
BAUDRATE=115200
LOG_LEVEL=5 #INFO and above only
NVM=/etc/6lbr/nvm.dat
LIB_6LBR=/usr/lib/6lbr
WWW_6LBR=/usr/lib/6lbr/www
BIN_6LBR=/usr/lib/6lbr/bin
ETC_6LBR=/etc/6lbr
```

Smart Bridge mode

It is important to note that this mode, depicted in Fig 5.2, allows for a seamless interconnection between a common IPv6 and a 6LoWPAN by adopting a mechanism proposed and defined by its authors [108]. Fundamentally this is possible due to a special implementation of a ND Proxy. The current 6LoWPAN and RPL RFCs do not fully address the interconnection of a 6LoWPAN to the Internet, except by routing. Because of this limitation, a ND Proxy concept has been defined in RFC 4389 [109] but is not applicable to RPL. Further work [110] on this matter was done, defining a NDP adaptation to 6LoWPAN-ND (called 6LP-GW) and was proposed for RPL [111]. The Smart Bridge mode found on 6lbr software differs from this previous Proxy mechanism in a sense that it covers the 6LoWPAN route-over scenario and its interaction with RPL, by fully bridging an IPv6 network with a 6LoWPAN/RPL network [112]. It then uses the information received trough NDP to configure the RPL network; specifically the PIO (Prefix Information Option), contained in the RA messages sent by radvd, is extracted and used to configure the DODAG (Destination-Oriented Directed Acyclic Graph - RPL's tree-like topology) prefix and the RIO (Route

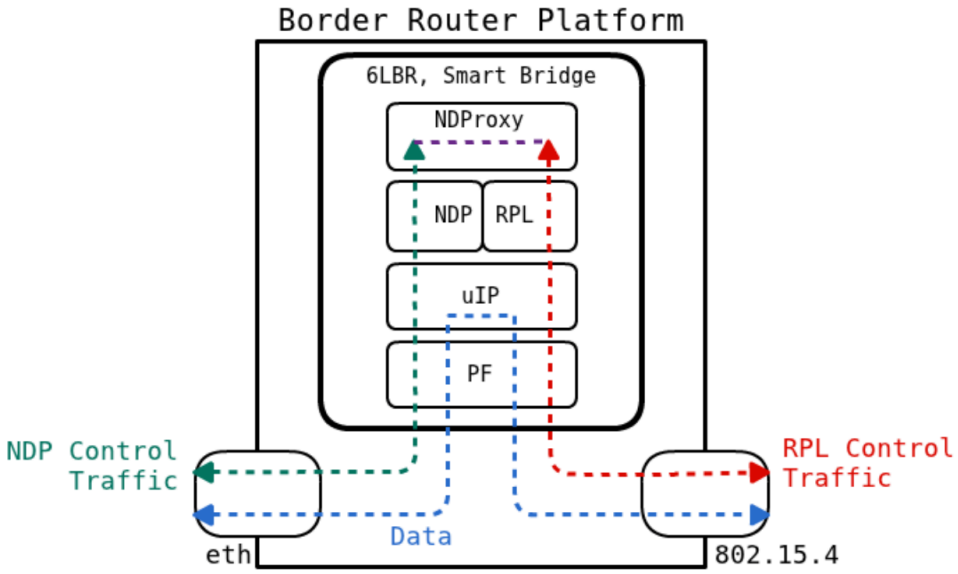

Figure 5.2: 6lbr's Smart Bridge operation mode [114]

Information Option) is also extracted and stored in 6lbr's routing table [112][113].

Gateway and Network Service Capability Layer (G/NSCL)

Both GSCL and NSCL modules were provided by OM2M - the open-source implementation of ETSI's M2M standard [88]. OM2M is an open-source platform for M2M interoperability compliant to ETSI M2M standard. It makes use of an open RESTful API with open interfaces which allow the development of services and applications independently of the underlying network, as per ETSI M2M's architecture. As this standard is very complex and its functionality depends heavily on abstraction horizontal layers, the OM2M platform was designed as a modular architecture running on top of an OSGi layer, which facilitates the development of Interworking Proxy Units and protocol and technology bindings. This is important as some systems might need different bindings to different technologies, like Zigbee IP, Bluetooth low energy, or Thread, as examples.

There was no need to compile it as two different bash executables (GSCL and NSCL) were already made available for Linux machines and, that being, were ready to be used under Raspbian OS. The only thing needed to install was Java 1.7, since OM2M was developed under Java on top of an OSGi Equinox runtime. As a default configuration, OM2M offers an authorization mechanism based on base64-encoded username and password. The default "admin/admin" was set, so every new interaction with both of these entities must include this authentication header, properly encoded to base64.

Gateway Interworking Proxy (GIP)

The Gateway Interworking Proxy, is the application that communicates, via reference point dIa, with the GSCL and interprets the messages sent by the d device - the non-ETSI compliant device. It was designed entirely using Node.js, which, given its characteristics, provides a fast, light and efficient way of dealing with these real-time data intensive applications.

Fig 5.3 depicts its overall functionality and it is the best way to understand how it works. Every HTTP interaction with the GSCL is done via POST messages with base64-encoded authentication headers and a body with XML data structured as needed per specific POST. Every time the Arduino boots, it sends a CoAP PUT message with the URI '/r/d/98EB/hrm' and the GIP automatically recognizes this as a registration request, because of the 'd' character. This is not specified by ETSI - it was a protocol needed to implement, due to the Arduino not being compliant with the specification. So, once the GIP recognizes this incoming CoAP message, it triggers a series of five HTTP POST messages, as specified by ETSI and which can bee seen in Fig. 5.3. The 98EB part of the CoAP's URI sent by the Arduino represents the last hexadecimal representation of its global IPv6 address and is used to identify the Application at the GSCL. Fig 5.4 depicts this scenario and the following Table 5.1 represents the further five HTTP POSTS issued by the GIP.

Once it is registered, the following messages sent by the Arduino (Fig 5.5) result in a single HTTP POST where the Content Instance is updated, like the last row of Tab 5.1

Network Application (NA)

The NA was developed to demonstrate how future Applications can be created leveraging the work of M2M and IoT standards, using simple RESTful applications. This is a prime example of how the IoT will evolve in the future. Currently, there are different wearables capable of sensing physiological data in a non-intrusive way. This data is somewhat constrained to the user only, via a proprietary application (the wearable's companion app). In the future, by adopting IoT open standards, like the previously discussed ones, this proprietary data (which the user owns) will be open to third parties entities like doctors, physicians and coaches, for instance. This Application here described is an example of such new relationships the IoT offers.

The developed NA interacts solely with the NSCL, as ETSI M2M defines that each application and/or device should communicate only to or via the SCL hosted at its domain level. For this matter, the developed NA application interacts with NSCL to first discover what applications are registered at the GSCL, so it can then subscribe to their updates. All the commands sent to the NSCL are passed to the GSCL, which replies back in the same order. Once the NA discovers the applications, it creates an asynchronous subscription at the GSCL. This way the NA will be notified, with a single HTTP POST message, every time there is a new instance of that same resource. For the developed system, this means that every time the Arduino reports new data, the NA will also be notified.

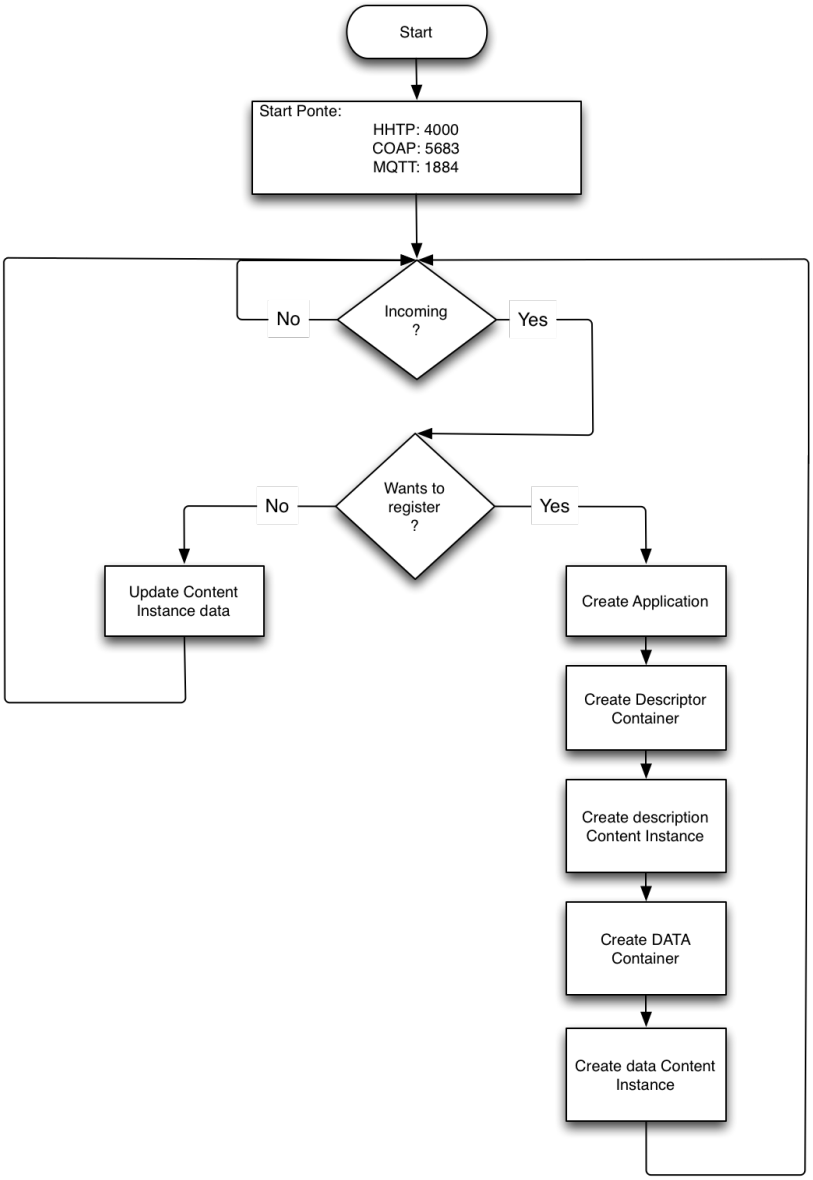

Figure 5.3: Gateway Interworking Proxy (GIP) Fluxogram

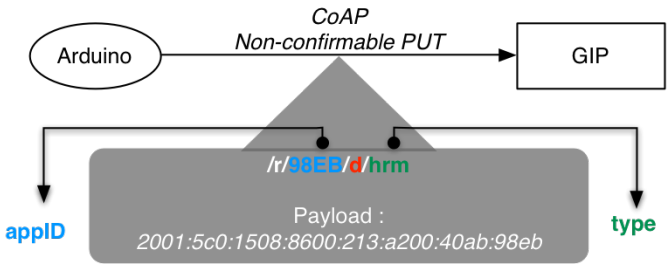

Figure 5.4: Arduino - GIP Registration

Description	URI	Payload
Create Application	http://localhost:3000/ om2m/gscl/applications	<om2m:application xmlns:om2m =”http://uri.etsi.org/m2m” appId=”98EB”> <om2m:searchStrings> <om2m:searchString>Type/sensor </om2m:searchString> <om2m:searchString>Category/hrm </om2m:searchString> <om2m:searchString> IPv6/ 2001:5c0:1508:8600:213:a200:40ab:98eb </om2m:searchString> </om2m:searchStrings> </om2m:application>
Create Descriptor Container	http://localhost:3000/ om2m/gscl/applications/98EB/ containers	<om2m:container xmlns:om2m=” http://uri.etsi.org/m2m” om2m:id=”98EB”> </om2m:container>
Create description Content Instance	http://localhost:3000/ om2m/gscl/applications/98EB/ containers/ Heart_Rate_Monitor/ contentInstances	<obj> <str name=”type” val=”HRM Sensor”/> <str name=”location” val=”Home”/> <str name=”appId” val=”98EB”/> <op name=”getValue” href=”gscl/applications/’98EB’/ containers/DATA/contentInstances/ latest/content” in=”obix:Nil” out=”obix:Nil” is=”retrieve”/> </obj>
Create DATA Container	http://localhost:3000/ om2m/gscl/applications/98EB/ conta	<om2m:container xmlns:om2m=”http://uri.etsi.org/ m2m” om2m:id=”DATA”> </om2m:container>
Create data Content Instance	http://localhost:3000/ om2m/gscl/ applications/ 98EB/containers/ DATA/contentInstances	<obj> <str name=”appId” val=”98EB”/> <str name=”category” val=”hrm”/> <int name=”data” val=”0”/> <int name=”unit” val=”BPM”/> </obj>

Table 5.1: GIP - Registration of a new Application

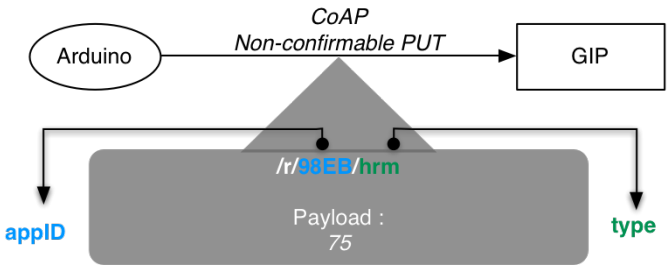

Figure 5.5: GIP - Update a new Application's Content Instance

This application was built entirely in Node.js making use of different modules: ponte [89], request [115], xmldoc [116] and base64 [117]. To create a subscription, the application will send the HTTP POST to the URI

'http://"nsclip":"nsclport"/om2m/gscl/applications/"appID"/containers/DATA/contentInstances/subscriptions'

with the following body:

```
<om2m:subscription xmlns:om2m="http://uri.etsi.org/m2m">
 <om2m:filterCriteria>
 <ifMatch>content</ifMatch>
 </om2m:filterCriteria>
 <om2m:contact>http://"naip":"naport"/resources/monitor</om2m:contact>
</om2m:subscription>
```

as with every interaction with NSCL or GSCL, the header includes the authorization "admin:admin" encoded to base64. Since we are using ponte, the HTTP service is already listening at the specified "naip" and "naport", and, being so, every new incoming notification will be targeted to this service.

It serves a specific purpose only, but it was designed to be future proof and to be integrated into the Web, once the application makes use of the express module [118]. With this, it would be possible to use the Web page as a UI to interact with the whole IoT system (NSCL, GSCL, future devices, etc.).

Real-Time Web Monitoring

Given the fact that there is a need to achieve real-time web monitoring, the best way to do so is to get the data directly from the GIP.

Currently, the standard specifies that, in order to subscribe for new data, an IP and listening Port must be passed on to the GSCL which will, in turn, send an HTTP POST to the specified location as soon as the application reported new data (like the previously mentioned NA). At first sight, this does not appear to be a practical solution, since there are proper protocols for this kind of messaging pattern. One should expect that, over time, once there are thousands of Device Applications

(DA) and subscriptions, it would be difficult to handle and scale such a large number of notifications. Furthermore, in the future, there will be a need to monitor sensors and/or devices in real-time, so a better mechanism should be adopted. This asynchronous HTTP reporting mechanism is not ideal for this solution and it would not scale as well as a proper Pub/Sub protocol would. For this matter, a combination of MQTT-binding and Web Sockets was developed under the GIP. Besides taking away the complexity of the HTTP notification mechanism, the Web Application communicates directly with the GIP, taking away the processing time between this one and the GSCL.

On the Server end, the GIP is leveraging the ponte module (detailed before) and its ability to seamlessly bridge from CoAP to MQTT. As this module is embedded in the GIP, it was also configured to make use of its capabilities to work as a MQTT broker using the dependent mosca [119] and its built-in Web Socket mechanism through mows [120]. All in all, ponte is a powerful tool that wraps a plethora of other modules (and their modules, obviously) to provide a fast, event-based mechanism to bridge HTTP, CoAP and MQTT services and messages. Once properly configured, the GIP was ready to both automatically bridge from CoAP to MQTT as it was possible to connect and subscribe to its MQTT broker via Web Socket. The client end needed only to use the appropriate Javascript library which allows the use of WebSockets to connect to the broker, which was part of the Eclipse's Paho open-source project [121][122].

This Web page was designed to solely demonstrate how real-time monitoring web-based monitoring solutions can be built, using standard and appropriate IoT protocols and technology. By leveraging Node.js capabilities to scale and offer fast real-time I/O bound applications, CoAP, Web Sockets and MQTT, a future-proof solution was deployed using only already available open source libraries and tools. The Web page was designed to work for this particular scenario only. There is no way to choose which broker we want to connect and which topics we wish to subscribe to - all of this is hard-coded for the particular solution. Fig. 5.6 showcases the Web page developed. The graph is updated as soon as new data comes in through the Web Socket (in form of a MQTT Publish message).

Chapter 6 details the results obtained when comparing this custom solution to that of what is currently offered by ETSI M2M and the OM2M platform.

Arduino

As stated in the previous chapter, the Arduino device, as a typical 6LoWPAN node, has one function only: to sense and send data (CoAP PUT) back to the gateway. However, before looping into that state, it must configure itself to work as a 6LoWPAN node. To achieve such thing, the μIPv6Stack library [123] and part of its counterpart, ρIPv6Stack, [124] were used. ρIPv6Stack differs from μIPv6Stack in that it possess a simplified version of RPL and, as the authors say but which could not be confirmed, reduced CoAP capabilities. With the first one, μIPv6Stack, the authors claimed that there was CoAP support, but there was not a single file under its libraries' folder that indicated so. Because of that, the CoAP Engine

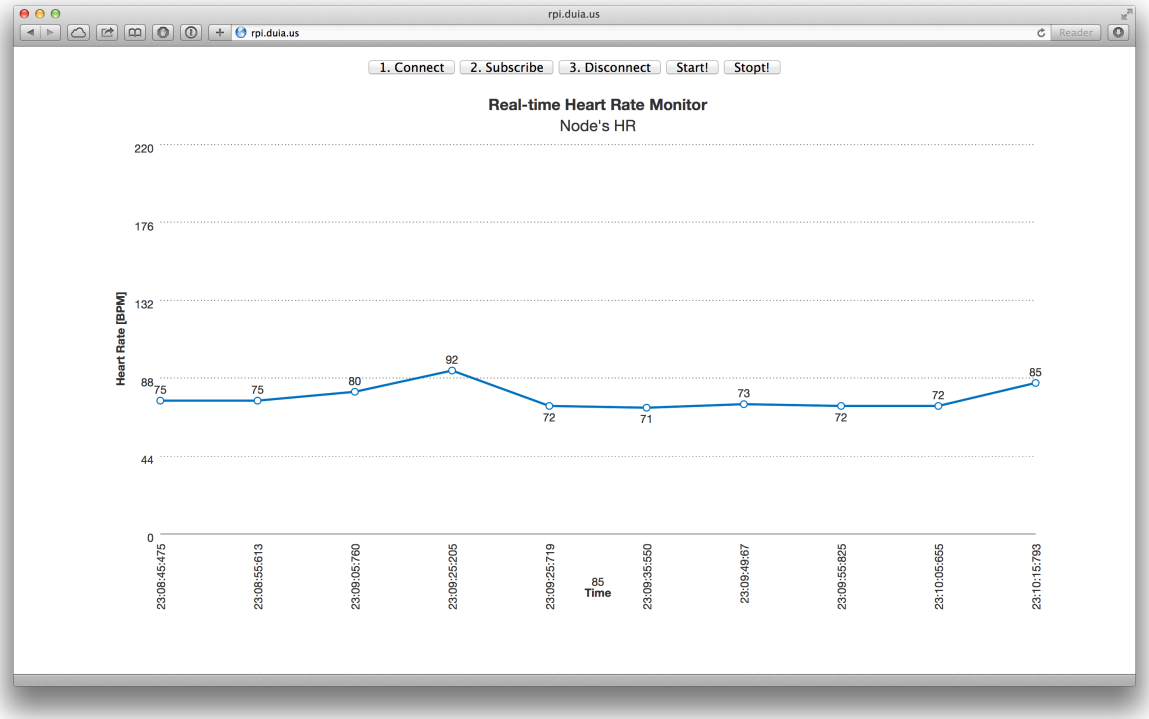

Figure 5.6: Real-Time Monitoring via Web Sockets

found on the ρIPv6Stack library was adopted and modified to work under ρIPv6Stack. However, this called CoAP Engine library serves only one purpose: to send CoAP Non-Confirmable PUT messages. As this solution does not require further functionality besides this sensing and reporting, the library was enough. Few modifications to the source code was needed. As the called CoAP Engine was designed to work in combination with ρIPv6Stack, that link had to be broken. This library was then re-configured to work with the μIPv6Stack library.

Once the device boots, it starts the Xbee module, IP stack and UDP. If everything is ok, it starts broadcasting RS (Router Solicitation). Once it is has a properly configured global unique IPv6 (detailed further, under Chapter 6) it will report this address in its first registration (described under the GIP section). After the registration, it loops and sends periodically the sensed data according to its configuration. The unique .ino file works for both heart rate sensor and temperature sensor - one must only change the initial defined constants to match the installed sensors.

5.3 Hardware Configuration

5.3.1 Arduino

The Arduino board is an open-source electronics platform most suited for fast prototyping purposes. It is a very interesting platform given its low cost features, extended capabilities via shields and the its large supportive community. It was used in this system as a 6LoWPAN node with limited CoAP capabilities. The chosen Arduino was the MEGA 2560 as it is capable of holding 256KB of RAM which, for prototyping purposes, was ideal since that much capacity could be needed to implement new libraries.

In essence, Fig. 5.7 represents how the final set up Arduino looks like with the assembled Wireless Proto Shield, Xbee S1 module and sensors. The Xbee module implements IEEE 802.15.4 networking protocol and was designed for high-throughput applications requiring low latency and predictable communication timing. It is ideal for the solution as it provides Arduino IEEE 802.15.4 RF communication capabilities trough the Wireless Proto Shield. Once installed, it had to be configured. To do this, one of the methods described in the Github wiki page of the imported IPv6 Stack library (detailed in the following chapter), was adopted [123]. As there was no way to connect the Xbee module to neither the Raspberry Pi nor the Macbook (the computer used to develop the solution), the interaction was made by connecting it to the Arduino and then communicate with this one via Serial Interface using CoolTerm [125]. The Arduino code used for this communication was the “BareMinimum” which is part of the examples contained in the Arduino IDE [126]. When connected, it was possible to set Xbee parameters, by sending specific commands, as the wiki [123] suggested:

ID - PAN ID 0xABCD (Contiki's default)
MM - MAC Mode 2 - 802.15.4 WITH ACKS

AP - API Enable 2 - API ENABLED W/PPP

MY - 16-bit Source Address 0xFFFF

DH - Destination Address High 0x00
DL - Destination Address Low 0xFFFF

After configuring the Xbee module, all that was left was to connect the heart rate sensor to the board. This was an easy process since the sensor used was the Pulse Sensor which needs only software code to filter the gathered data into usable heart rate data (BPM - beats per minute). This software code comes with the sensor itself and is licensed under Creative Commons [127]. The connection to the Arduino board is very simple as there is only three wires. This can be seen in the aforementioned Fig. 5.7 and are connected as follows:

- Purple wire (Acquires the signal): A0;
- Red wire (VCC +3 or +5v): 5v;
- Black wire: GND

To showcase how one NA could interact with a NSCL to list the available applications, so it could then retrieve their latest data instance, another Arduino board was used with the same hardware as described before, but with a temperature sensor instead of Pulse Sensor. The temperature sensor used was TMP 36 [128]. It was connected to board's pins A0, 5v and GND and the formula converting the reading voltage (minivolts from A0) to Celsius was:

```
voltage = A0 * .004882814;

temperature = (voltage - .5) * 100;
```

obtained from [129]. This second Arduino node is illustrated on the left side of Fig. 5.7.

5.3.2 Raspberry Pi

The chosen Raspberry Pi used as the Gateway/Border Router of the solution was Model B. No external hardware was connected to it, except for the 802.15.4 platform by NooliTIC [87] which transforms the already low cost Gateway into a low cost 6LoWPAN Border Router . This platform, NooliberryT, is based on CC2538 and is compatible with 6lbr. It has a SLIP (Serial Line Internet Protocol) radio application, taken from Contiki OS 2.7, with which 6lbr talks to over Raspberry Pi's UART interface. Besides connecting it to the Raspberry Pi board (Fig. 5.8), there was nothing else needed to do besides configuring 6lbr to communicate properly with this platform. This was detailed previously (6lbr).

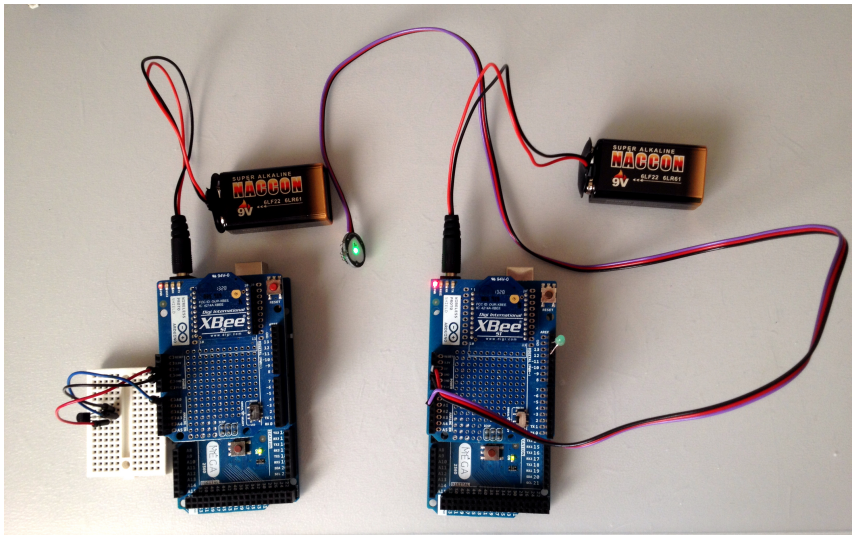

Figure 5.7: Arduino nodes (left: temperature sensor; right: pulse sensor)

Figure 5.8: Raspberry Pi with NooliberryT

5.4 Proof of Concept

As the system is composed of different interactions with different blocks, it is easier to understand and see it in action by means of simple print screens. The following Figures will demonstrate the following in a sequential manner:

1. Arduino searching, at startup, for the Border Router (RS - Router Solicitation messages)
2. RA (Router Advertisement) reply via Border Router
3. Arduino's stateless address auto-configuration (SAA) and registration at GSCL
4. Arduino reporting sensed data (CoAP PUT messages)
5. GIP translating and forwarding CoAP messages to HTTP, following ETSI M2M standard
6. Applications registered at GSCL
7. NA subscription on the previously mentioned applications
8. Real-time Web-based monitoring

Starting with Arduino, it first starts by initializing the whole IPv6 and UDP stack and, in case of success, it multicasts the RS message. As noted by the specification, nodes should not multicast or broadcast these RS messages; however, the library used [123] did so. As the 6lbr module, on the Border Router, is periodically multicasting RA message (as demonstrated in Fig. 5.11 taken from Wireshark), the Arduino will use this message to add the Border Router as the default router. As this message includes a prefix, it starts the SAA process (this sent prefix is part of the gogoc and radvd combination as described in the previous Chapter (this prefix can be seen in the previously mentioned RA message 5.11). As part of the SAA process, the node must check if its identifier (taken from Xbee S1's MAC address) is truly unique. This mechanism is called DAD (Duplicate Address Detection) and it is performed by Neighbor Solicitation with a special multicast address composed of ff02::1::ff and the last 24 bits of the IPv6 address as seen in Fig. 5.12. As a reply, a Neighbor Advertisement is sent as a solicited message (Fig. 5.13). Once all of this process is set, it is possible to find this 6LoWPAN node under 6lbr's client web page, as Fig. 5.10 illustrates.

At this stage, the Arduino is ready to interact with the GSCL. First, the registration process of the node can be seen in Fig. 5.15 where the GIP is receiving CoAP messages, translating and properly forwarding them to the GSCL, as per ETSI M2M standard. This is the mechanism explained before in Chapter 5, illustrated by Figs. 5.1 5.5 and which can be seen in Fig. 5.14, as captured by Wireshark.

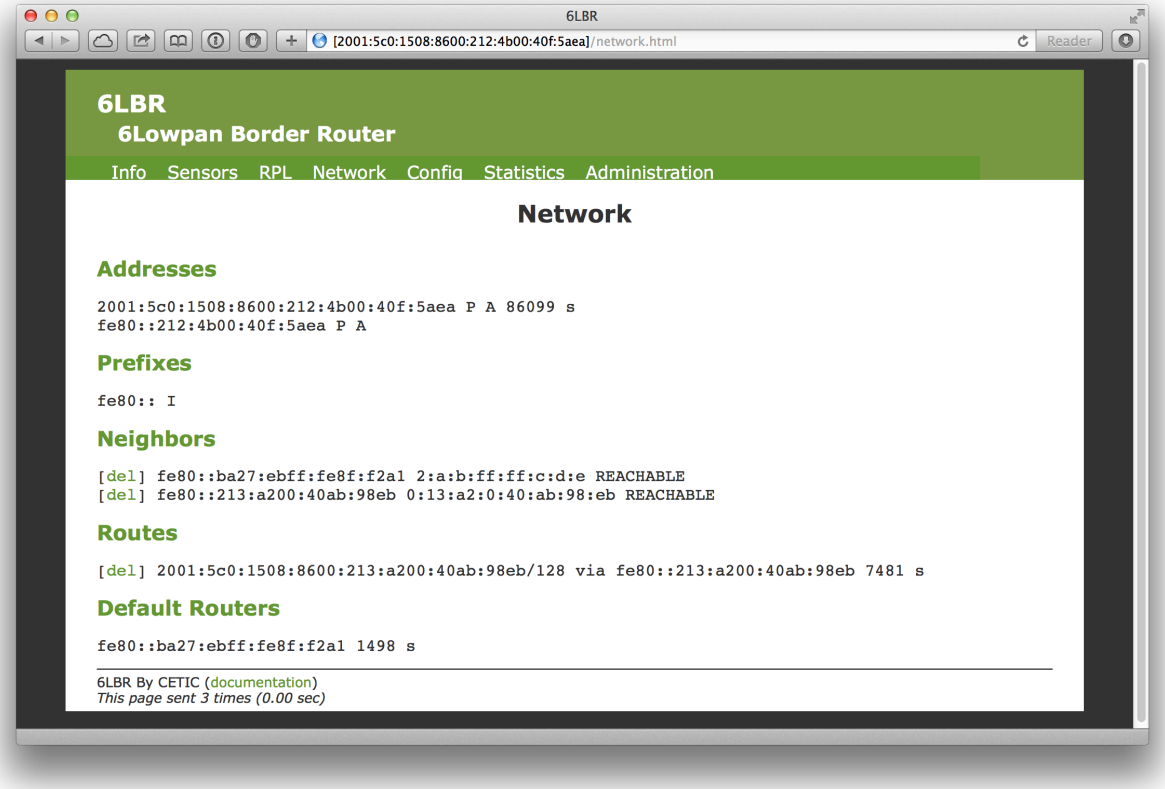

Figure 5.9: 6lbr's Web page displaying the Arduino node (under Routes)

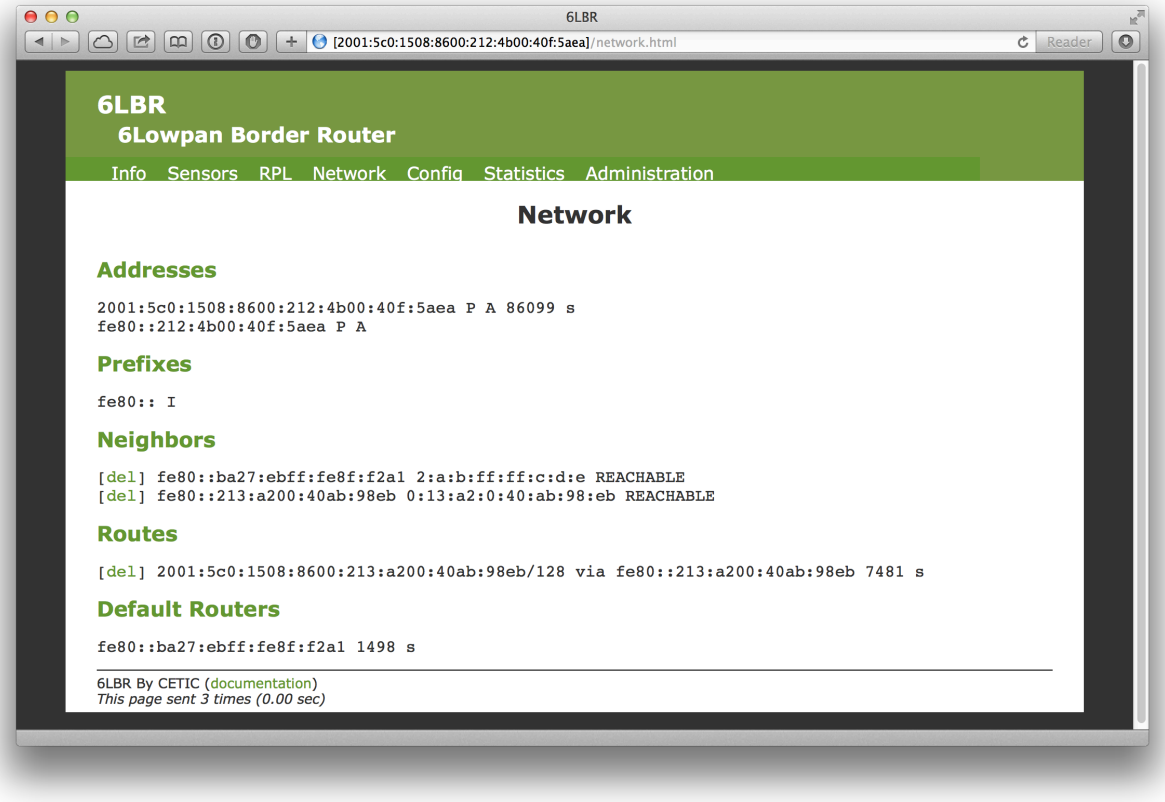

Figure 5.10: 6lbr's Web page displaying the Arduino node (under Routes)

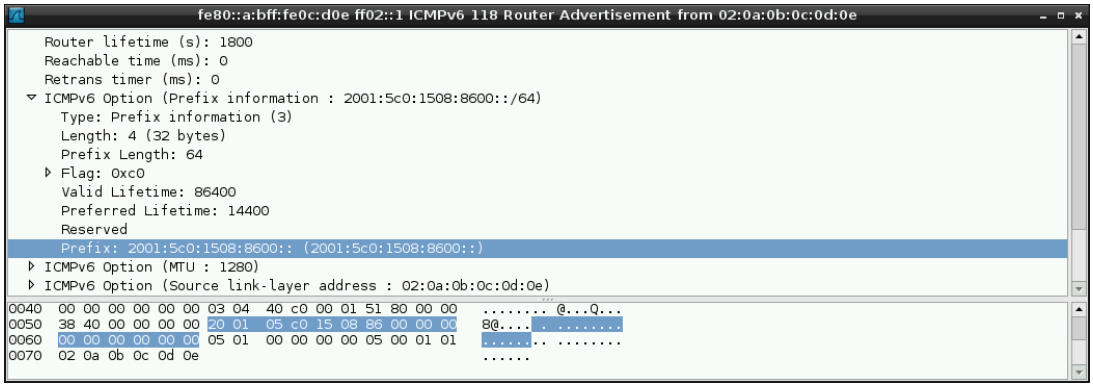

Figure 5.11: Multicasted RA message at Border Router

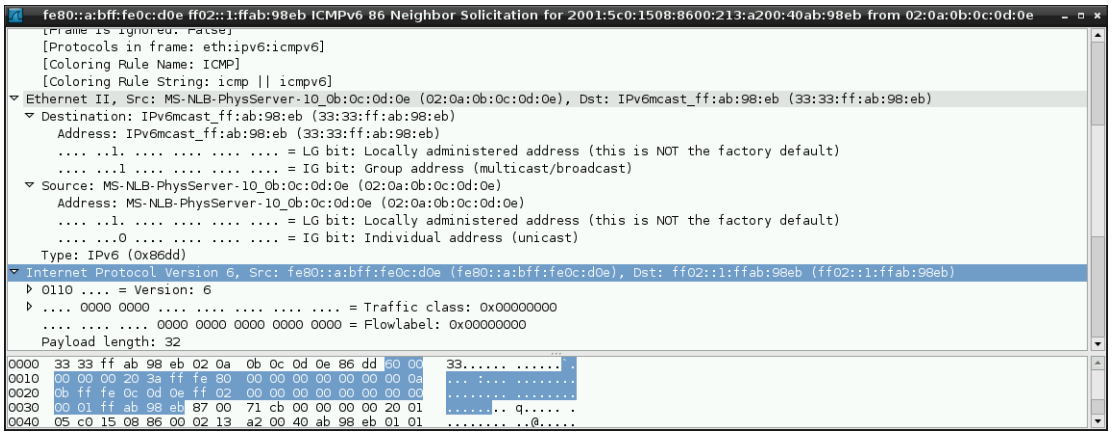

Figure 5.12: Special multicast NS message sent for DAD purpose

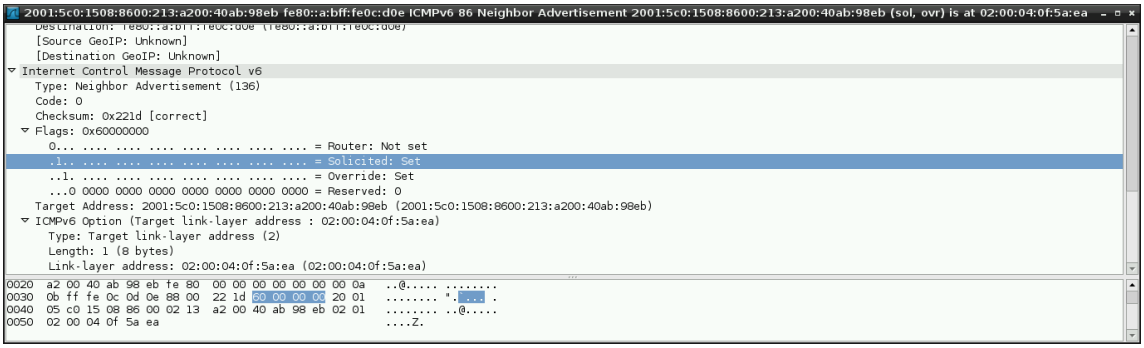

Figure 5.13: Solicited NA message as a reply to the previously received NS

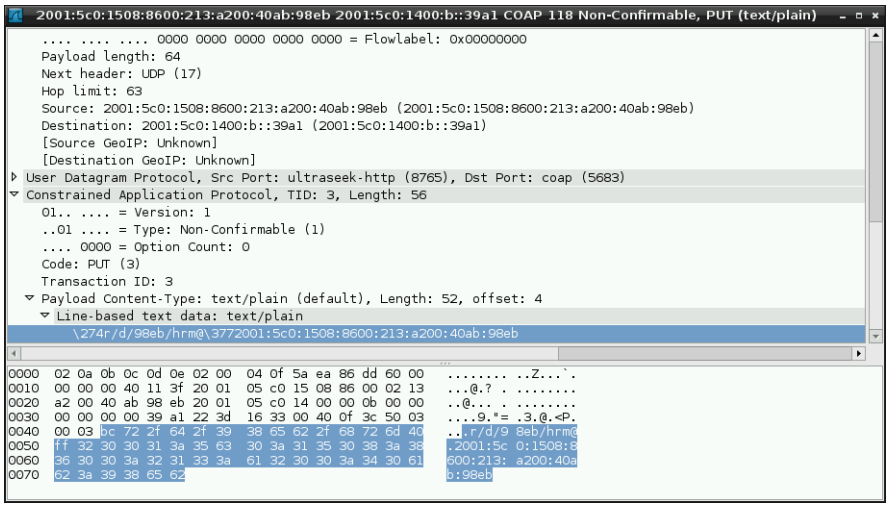

Figure 5.14: Arduino node registering application via CoAP PUT message

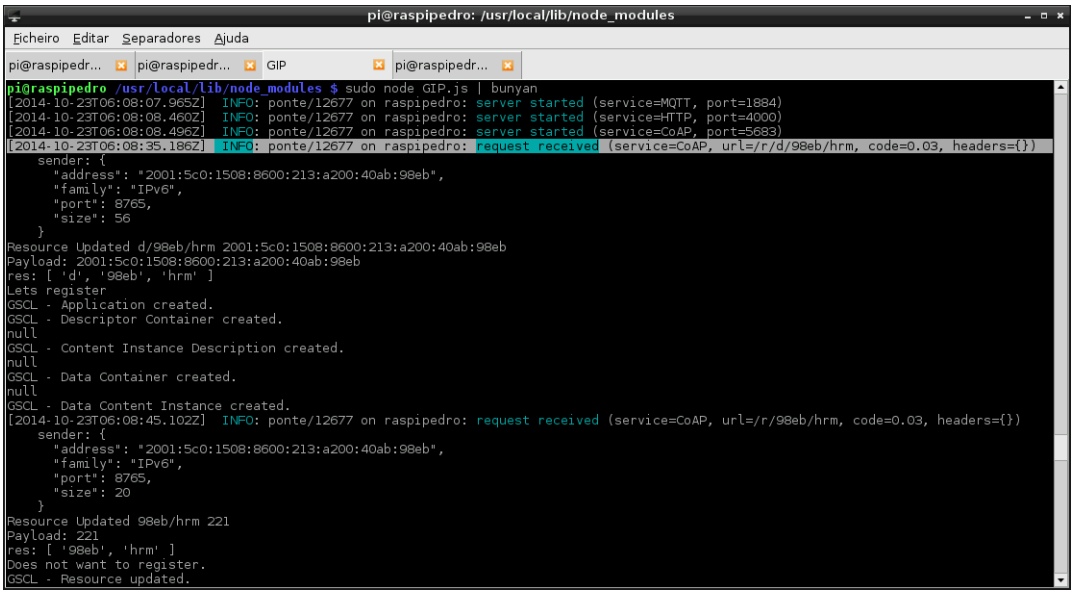

Figure 5.15: GIP registering new application and updating content

Once the GIP is done with the registration process, the registered applications can now be found in the GSCL, as seen in Fig 5.16. This was taken by crawling the REST Client web application provided by the GSCL itself, but they could also be found using the discovery mechanism . The discovery is possible by targeting a specific URI (GET http://rpi.duia.us:3001/om2m/gscl/discovery) which would provide a list of all resources present at the GSCL. This is actually what the developed NA does: it crawls the resources, parses all registered applications and then asks the NSCL to create a subscription for those applications found. For each application, the NSCL will report new data to the specific contact point (IP and Port) indicated by the NA itself. Once it is properly created, every new data sensed by the nodes sense will be automatically reported to this NA. As per specified by ETSI M2M standard, the NA does not actually create a subscription in the NSCL, but it uses it as a relay to contact the GSCL. The applications and subscriptions are only present at the GSCL. Having this standard subscription, using ETSI M2M standard, it is now possible to study this approach and see how it performs when compared to a proper publish/subscribe messaging protocol like MQTT which was used for the real-time Web monitoring.

5.4.1 Real-Time Monitoring Validation

In this section, it is shown the practical difference when adopting current ETSI M2M standard subscription mechanism and MQTT's, as detailed in Chapter 5, under the "Real-Time Web Monitoring" section. For this, it was tested both the traffic and latency found in each case. As a normal NSCL would be part of an IoT/M2M Service Provider, located at the network domain, changes in the previously mentioned solution (where both GSCL and NSCL were functioning at the domain/gateway

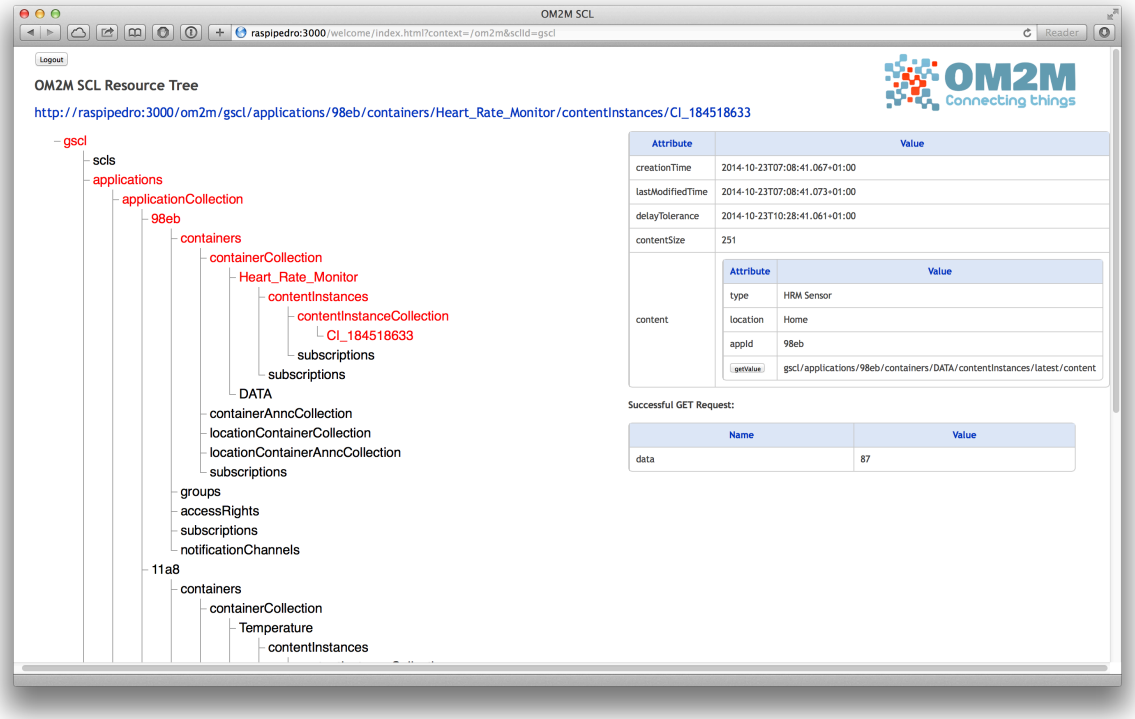

Figure 5.16: Applications registered at GSCL

	HTTP (bytes)	MQTT (bytes)
BR -> NA	715	15
NA -> BR	395	0
Total per message	1110	15
Total session	33300	511
Diff. Factor	65,17x	1x

Table 5.2: Traffic comparison when using ETSI M2M standard subscription method and MQTT

domain) had to be done. To approximate the experiment with a real life properly deployed IoT solution, both the NSCL and NA were installed on a VPS (Virtual Private Server) located in the USA and the MQTT client on a Macbook Air located in a domestic Spanish network. Although the NA is in the same computer as the NSCL, the transmitted data is coming directly from the Border Router's GSCL which is located in a domestic Portuguese network.

With only one subscription in the whole system (the one used for the test), it was already possible to notice a considerable difference of 788ms average. This was expected the standard ETSI M2M subscription mechanism depends heavily on new TCP sessions for every new notification with a complex XML payload. This experiment, detailed in Fig. 5.17, lasted for 37 minutes with the Arduino node reporting new data every 10 seconds. For best results, the different machines' clocks were synchronized with the same server, using NTP (Network Time Protocol). As one of the two machines used was a Macbook Air, the chosen NTP server for both machines was "time.apple.com". It is important to notice that during this experiment, there was only one active subscription registered at the GSCL. It is expected that each IoT system is composed of thousands of devices, applications and subscriptions. The graph shows the delay noticed on each message and the accumulated for the whole experience. In the end the accumulated delay was of 2m 52s and 700ms. Also important to notice is the 0m 7s 647ms delay occurred at the 66th message and various others in the range of almost 2 seconds. It clearly demonstrates how inadequate it currently is for real-time solutions.

One could argue that this difference could be debatable given the nature of Node.js event-driven, non-blocking IO model. That is no certainty that it first bridges to MQTT rather than HTTP. However, it is expected for the first to happen because the MQTT client (Web page) is automatically subscribed to the same CoAP's resource, whereas in the case of HTTP bridging, the application needs first to decode the incoming CoAP URI to then forward to an external HTTP service. This is why the experiment lasted for 37 minutes, with a total of 219 updates. Independently of the result - which bridging takes place first -, the average difference was still 788ms, being the MQTT solution faster. This could be analyzed with further deeper debugging.

The next experiment tested how much data was actually transmitted during the whole session and for each new notification. Making use of Wireshark, it was possible to conclude that there is a big difference between the standardized ETSI M2M solution and that of MQTT's. For starters, the standard subscription initializes a new TCP connection every time it wants to report data - in real-time monitoring, it is expected that new data comes in very frequently and, thus, the session should

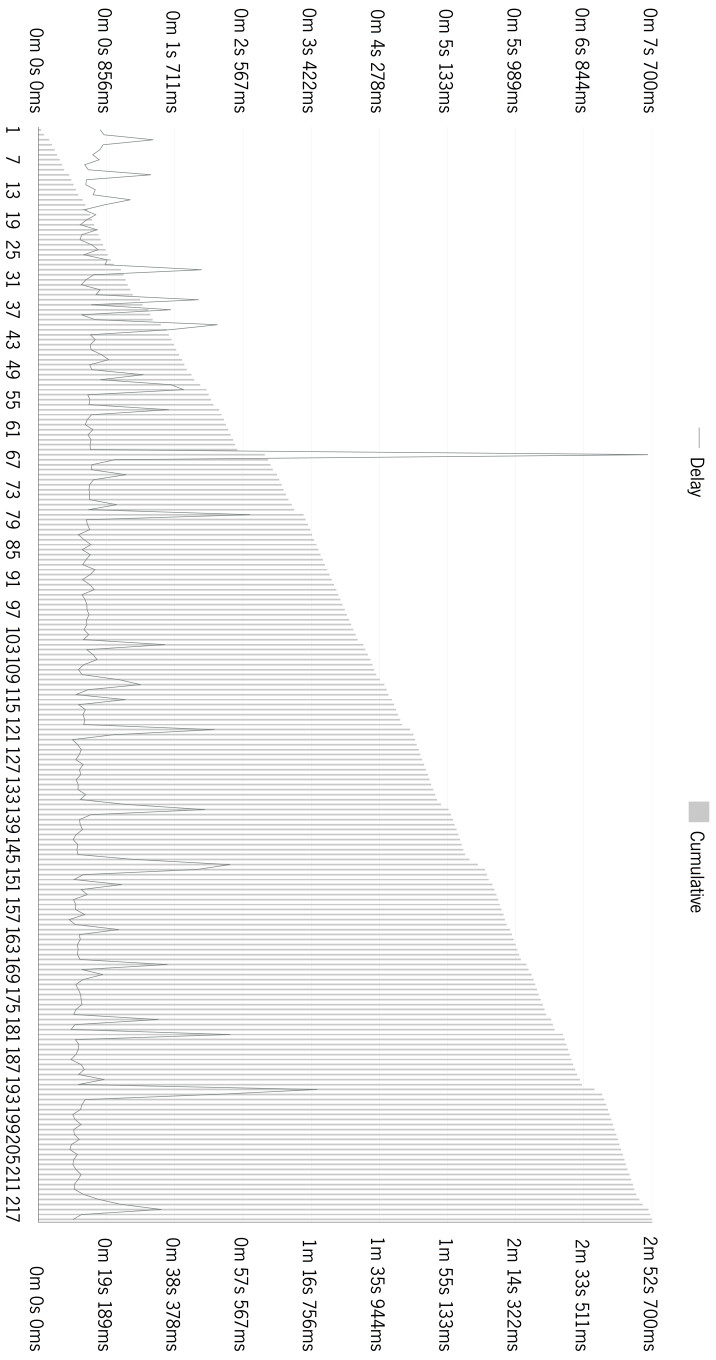

Figure 5.17: Delay difference between MQTT and standard ETSI M2M subscription mechanism

be open all the time, for efficiency reasons. Also, as there is no way to tweak the QoS, the NA automatically replies with another HTTP POST for every notification. MQTT takes all of this complexity and inefficiency away. First, it keeps a single TCP connection open during the entire subscription and, secondly, it is possible not to wait for ACK messages, by setting the level of QoS to 0 when subscribing to the specific topic. Again, for real-time monitoring, there is no need to acknowledge the reception of every single notification. If, however, one feels the necessity of such, a simple adjustment of the QoS parameter would assure so. Tab. 5.2 summarizes the findings, when using Wireshark. As detailed in the table, the difference is quite big: 65 times bigger when compared to MQTT protocol with QoS parameter set to 0. The reason for this is that, as per RFC 2616 [130], for every POST message received, a response code of 200 (OK) or 204 (No Content) should be sent whenever the resource can not be identified by the URI. As the NA was developed merely to showcase how one can interact with ETSI M2M architecture, the 204 reply code is sent automatically by Ponte, with a total of 395 bytes. Had this reply been completely ignored, the overall generated traffic would still be roughly 42 times greater than that of MQTT's. Fig 5.18 illustrates what is actually happening during each notification, when making use of ETSI M2M's standard subscription mechanism and MQTT's, respectively. The NA subscription model could be more efficient if OM2M's mechanism had support for persistent connection. As Wireshark has demonstrated there a new TCP connection is established whenever there is new data to be reported.

This experiment shows how important it is for IoT standard architectures to have this kind of built-in protocol binding and why MQTT is such a good protocol for IoT. As Cisco estimates that 50 billion devices and objects will be connected to the Internet by 2020 [131] this traffic efficiency is much needed. MQTT offers even more flexibility and scalability capacity, has documented before in Chapter 2. All of these factors are crucial to provide real-time monitoring.

5.5 Conclusion

This chapter has demonstrated how it is possible to design a complete IoT solution, using adequate protocols while being compliant to the latest and most complete standardized IoT/M2M architecture to date: ETSI M2M. This ensures that the designed solution is future proof, enables interoperability with other IoT solutions (when adopting the same standard) and is flexible enough to deploy a proper solution to the most anticipated use-cases for IoT and its applicable sectors (smart cities, smart meter reading, logistics and healthcare, as examples). The developed real-time web monitoring is an example of this flexibility to adapt to the specific requirements of specific use cases. In this case of real-time heart rate monitoring, a proper protocol had to be incorporated to deliver the data as fast and efficient as possible without compromising the compliance to the ETSI M2M standard. The importance of having this compliance to a standard architecture was also showcased here by means of a Network Application (NA). This NA simulates a possible application developed by an entity like a Doctor, Physician or even a service as part of the already existing IT solution of a nursing facility, for instance.

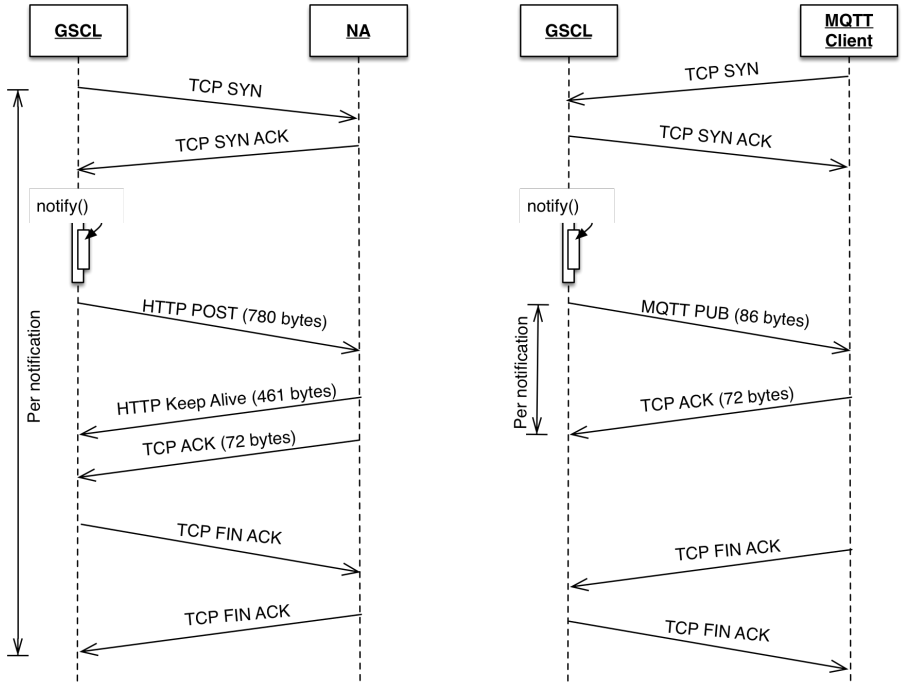

Figure 5.18: ETSI M2M standard subscription method vs MQTT

All in all, this whole solution demonstrates why it is important to adopt standard architectures: IoT can only evolve to its full potential when there is an easy way to interexchange relevant data to specific entities, creating new synergies across the entire public and private sectors, making way for smarter and richer services and applications.

Chapter 6

Conclusion and Future Work

This dissertation has showed: the potential of IoT; the problems that it is currently facing; and how to mitigate these problems, using horizontal architectures and IoT protocols. As the IoT is composed of different protocols at each layer of the OSI model, it is a vast and complex technology which covers all different communication domains. This work has demonstrated the most widely used ones, and those specifically designed for IoT. As it will have a major impact in the health sector, a testbed was set to demonstrate how real IoT systems can currently be deployed using open standards and open-source tools. With this testbed implementation and validation, one can conclude that an IoT architecture should be as generic as possible, meaning that it should be designed to cover the most different use cases typically found in IoT and M2M. The real-time monitoring web-based solution, found in this testbed, is a prime example of how such flexibility is needed. If this solution had not been well adapted to meet the real-time requirement, the IoT architecture adopted (ETSI M2M) would not be able to deliver such a service. This is something of utmost importance and that is something the next generation of IoT standards (oneM2M, specifically) will tackle, by combining even more technologies into the standard (like device management via LWM2M, for example) and protocol bindings like MQTT. Having this standard architecture with different horizontal layers offering fundamental functionalities, while being network agnostic, is a major breakthrough and will allow the IoT to fully grow up to its potential. Interacting with the services via a standardized RESTful API will make way for new Web and mobile based applications to grow and directly interact with the already well established Internet. In the end, this will, ultimately, result in smarter, fully aware IoT applications which combine different data sources and mechanisms to trigger events in other IoT systems/applications, offering the end-user a better quality of life with simplified, smart and autonomous real-life actions.

The system deployed fulfilled the pre-determined requirements using state-of-the-art open source tools compliant with the most complete IoT standard - ETSI M2M. This allowed for a development of an open solution completely future proof with the

next iteration of the standard: oneM2M . This proof of concept demonstrated that it is possible to use low cost embedded devices that are not compliant to the standard itself, by developing internal Interworking Proxies which are, in fact, praised by the standard. Besides, it was possible to notice something really important for IoT: a combination of a real-time web monitoring solution and storage of this aggregated data. The first by combining a MQTT binding at the Interworking Proxy and the latter by interacting with the standard using the documented RESTful API - Network Application (NA), specifically, which discovers new applications and subscribes to its updates. The results have shown a difference of 788ms and a traffic increase of, at least, 42 times fold, which fundaments the need to use proper protocols and messaging patterns

Future Work

Although the implemented solution should have covered al bullets mentioned before, under the “Solution Overview” section of chapter 4, not all of it was possible.

For instance, there is no guaranteed privacy at the device domain area. This is an issue for all constrained devices and the Arduino is not an exception. Although the model used had 8KB of SRAM, the minimal code size needed for a modified version of the already optimized DTLS library (tinyDTLS [132]) is 9812 bytes [133]. This makes it hard for constrained devices to use both CoAP and DTLS/TLS and that is something the IETF Working Group, called LWIG (Light-Weight Implementation Guidance [134]), is actually working on. However, this could be temporarily mitigated by implementing relic-toolkit [135] which is fast and its code footprint is small enough to fit in the used Arduino Mega as per [136] findings - the final Arduino code developed for this solution could still hold 3342 bytes of code which would be enough to hold the needed 2804 bytes, if the NIST K163 algorithm was chosen [136]. Besides this encryption at the PAN level, the OM2M tool should also be able to encrypt the exchanged data. Currently, it supports only a basic “username/password” type of authentication mechanisms encoded to base64. As both GSCL and NSCL work at the network layer (and, thus, not a constrained environment), they already possess the ability to provide security over the Internet, if cryptographic protocols like TLS (Transport Layer Security) were adopted.

Although the chosen prototyping board (Arduino) was capable of doing the necessary minimum for the whole solution to work properly, it has demonstrated not to be fully capable of adopting typical IoT technology. It was possible to configure real IPv6 routing over a 802.15.4 radio link, however, it was not possible to communicate directly with it from any other IPv6 endpoint. Tests were made by simply sending IPv6 UDP packets, but the used library could not recognize the UDP datagram, although it did recognized the incoming IP packet. Further research on the adopted Arduino library would have to be done to understand what was actually preventing this. Had this been possible, the GSCL could have been re-configured to support a mechanism called Application Point of Contact (aPoC). This aPoC is an attribute of the registered application in the SCL and once the “aPoCPaths” were configured, it would then be possible to re-target incoming HTTP POST messages

aimed at the SCL directly to the Arduino using the CoAP protocol. If such real IPv6 end-to-end bi-directional communication was possible, this GSCL's aPoC feature would allow for real direct orders to reach the nodes (like a start/stop button on a web page to start/stop the reporting mechanism on the node). The CoAP library was also very limited. In essence, it only supported the transmission of non-confirmable PUT messages. However, that feature was indeed the only needed one for the whole IoT solution to work as intended.

In order to achieve the ultimate optimal solution, there would have to be changes at both hardware and software level. As the Arduino has showed limited capabilities to work as a full fledge 6LoWPAN node, new more capable boards would have to be used. As there are many platforms proven to work properly as a 6LoWPAN node, by supporting Contiki OS [137], those could be a good upgrade. Also, by natively supporting Contiki OS, there would also be support to a complete CoAP implementation, as it is part of the OS. After having this basic layer, the whole solution could then be improved by making use of the much praised LWM2M standard. As mentioned before, in Chapter 3 and 4, this standard offers some great functionalities to IoT systems. As it works with CoAP, it would be possible to also bring this standard to both clients (new boards) and the Raspberry Pi, by porting the recent project called Wakaama (formerly known as liblwm2m) [138]. Also important, these boards would then be much more efficient and secure by supporting sleeping mode and privacy through DTLS.

With all these additions, the solution would be compliant to the next upcoming universal M2M/IoT standard: oneM2M. Heading towards this standard, which will provide bindings to MQTT and LWM2M, would allow for further development of the ultimate features mentioned in Chapter 4: trusted applications and mobility. At this point, a new framework supporting oneM2M and its needed bindings would be developed, or further improvements to the oM2M would be made. As this tool runs on top of an OSGi Equinox runtime, it would be possible to extend its functionalities by developing new plugins. Alternatively, the new oneM2M framework could be developed completely in Node.js to leverage its great capabilities to deal with I/O-bound applications and systems. Either way, security using TLS could also be developed to secure the whole communications at the network level.

As a final note, one can deduct from this work that there really is much work to be done in the Internet of Things field. Since this new emerging technology is composed of different other technology all across the OSI model, there is an inherent difficulty in bringing all of this together to define a solution that solves every problem and which is suited for every IoT System. Due to its own broad nature, the IoT is applicable at very different areas, each of them having their own prerequisites and singularities. This means that while there is a need to make all of these different systems cross-compatible and interoperable, there is also a demand to define specific mechanisms necessary to solve specific solutions. With this in mind, one can conclude that the most sophisticated standard IoT architecture is the one that presents greater flexibility to adapt to other technologies (and standards and protocols, for that matter), while abstracting this interworking and binding mechanisms. ETSI M2M laid the foundation for this horizontal abstraction layer, but it must be completed with the much needed proper

protocol bindings and different various interworking processes. OneM2M is already working on this and it might be the much needed common framework for the IoT. This cross-compatibility with different standards, where each serves its own purpose, is part of the needed capabilities to achieve the desired IoT, as demonstrated under chapter 4. The developed system takes all of this into account and showcases how future-proof it is and how it would be ready to interact with other IoT systems, inter-exchanging relevant data. At another level, accompanying the evolution of the IoT, this data would then become relevant knowledge, which would trigger appropriate actions on other different IoT Systems. This is what is possible to achieve with the so called Internet of Things, when its issues are sorted out and there is a consensus across the whole industry and literature.

Bibliography

[1] Gartner, “Gartner Press Release, STAMFORD, Conn.” 2013. [Online]. Available: http://www.gartner.com/newsroom/id/2636073

[2] J. Bradley, J. Barbier, and D. Handler, “Embracing the Internet of Everything To Capture Your Share of $ 14 . 4 Trillion,” 2013. [Online]. Available: http://www.cisco.com/web/about/ac79/docs/innov/IoE_Economy.pdf

[3] M. C. Domingo, “An overview of the Internet of Things for people with disabilities,” J. Netw. Comput. Appl., vol. 35, no. 2, pp. 584–596, 2012.

[4] P. Duffy and Cisco, “Beyond MQTT: A Cisco View on IoT Protocols,” 2013. [Online]. Available: http://blogs.cisco.com/ioe/beyond-mqtt-a-cisco-view-on-iot-protocols/

[5] Skynet.im, “Skynet.im homepage,” 2014. [Online]. Available: http://skynet.im/

[6] Eric and Pinocc.io, “A single protocol for the Internet of Things?” 2013. [Online]. Available: https://pinocc.io/blog/building-the-internet-of-things/a-single-protocol-for-the-internet-of-things/

[7] ARM Ltd, “ARM mbed,” 2014. [Online]. Available: http://mbed.org

[8] Internet Engineering Task Force (IETF) and 6lowpanWG, “RFC 6775 - Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs),” Tech. Rep., 2012. [Online]. Available: http://datatracker.ietf.org/doc/rfc6775/

[9] Internet Engineering Task Force (IETF) and roll WG, “RFC 6550 - RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks,” Tech. Rep., 2012. [Online]. Available: https://datatracker.ietf.org/doc/rfc6550/

[10] Thingsquare and B. Center, “Thingsquare.” [Online]. Available: http://www.thingsquare.com

[11] I. The Institute of Electrical and Electronics Engineers, “IEEE 802.15 WPAN Task Group 4 (TG4).” [Online]. Available: http://www.ieee802.org/15/pub/TG4.html

[12] C. Bormann, “Getting Started with IPv6 in Low Power Wireless Personal Area Networks (6LoWPAN),” 2011. [Online]. Available: http://www.iab.org/about/workshops/smartobjects/tutorial.html

[13] Internet Engineering Task Force (IETF) and 6lowpanWG, “RFC 4919 - IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs) Overview, Assumptions, Problem Statement, and Goals,” pp. 1–12, 2007. [Online]. Available: https://datatracker.ietf.org/doc/rfc4919/

[14] 6lowpanWG, “RFC 4944 - Transmission of IPv6 Packets over IEEE 802.15.4 Networks,” pp. 1–30, 2007. [Online]. Available: https://datatracker.ietf.org/doc/rfc4944/

[15] ——, “RFC 6282 - Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks,” pp. 1–55, 2011. [Online]. Available: http://datatracker.ietf.org/doc/rfc6282/

[16] Z. Shelby and C. Bormann, 6LoWPAN: The Wireless Embedded Internet. Wiley Publishing, 2010.

[17] Z. Shelby, “6LoWPAN Seminar Video,” Oulu, Finland, 2010. [Online]. Available: http://portal.sensinode.com/downloads/6LoWPAN-seminar.mp4

[18] Internet Engineering Task Force (IETF) and roll WG, “Routing Over Low power and Lossy networks (Active WG).” [Online]. Available: http://tools.ietf.org/wg/roll/

[19] Z. Shelby, K. Hartke, and C. Bormann, “RFC 7252 - The Constrained Application Protocol (CoAP),” Tech. Rep., 2014. [Online]. Available: http://www.rfc-editor.org/rfc/pdfrfc/rfc7252.txt.pdf

[20] C. R. E. (CoRE) and I. E. T. F. (IETF), “Constrained RESTful Environments (CoRE).” [Online]. Available: https://datatracker.ietf.org/wg/core/documents/

[21] C. Bormann, “CoAP: Scaling the Web to billions of nodes,” IoT Int. Forum Launch, Berlin, 2011. [Online]. Available: http://6lowpan.net/wp-content/uploads/2011/11/iot-forum-v0-sanitized.pdf

[22] Open Mobile Alliance, “Lightweight Machine to Machine Technical Specification Candidate Ver 1.0,” 2013. [Online]. Available: http://technical.openmobilealliance.org/Technical/technical-information/release-program/release-program-copyright-notice?rp=154&r_type=technical&fp=Technical%2FRelease_Program%2Fdocs%2FLightweightM2M%2FV1_0-20131210-C%2FOMA-TS-LightweightM2M-V1_0-20131210-C.pdf

[23] (IETF) Internet Engineering Task Force and Z. Shelby, “RFC 6690 - CoRE Link Format,” pp. 1–22, 2012. [Online]. Available: http://tools.ietf.org/html/rfc6690

[24] ETH, “Californium Interop Servers.” [Online]. Available: http://vs0.inf.ethz.ch/

[25] M. Kovatsch, “Copper (Cu) Firefox extension,” 2014. [Online]. Available: https://addons.mozilla.org/pt-PT/firefox/addon/copper-270430/

[26] IPSO Alliance and Z. Shelby, “The IPSO Application Framework (draft-ipso-app-framework-04),” pp. 1–19, 2012. [Online]. Available: http://www.ipso-alliance.org/technical-information/ipso-guidelines

[27] ARM and Z. Shelby, “ARM IoT Protocol. CoAP: The Web of Things Protocol.” 2014. [Online]. Available: http://www.slideshare.net/zdshelby/coap-tutorial

[28] C. Bormann and TZI, “coap.me,” 2014. [Online]. Available: www.coap.me

[29] (CoRE) Constrained RESTful Environments and K. Hartke, “Observing Resources in CoAP (draft-ietf-core-observe-14),” pp. 1–32, 2014. [Online]. Available: http://tools.ietf.org/html/draft-ietf-core-observe-14

[30] CoRE Working Group, C. BormannBormann, and Z. Shelby, “Blockwise transfers in CoAP (draft-ietf-core-block-15),” pp. 1–30, 2012. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-core-block/

[31] A. Piper and Mqtt.org, “MQTT - Design principles.” [Online]. Available: https://github.com/mqtt/mqtt.github.io/wiki/DesignPrinciples

[32] International Business Machines Corporation (IBM) and Eurotech, “MQTT V3.1 Protocol Specification,” pp. 1–42, 2010. [Online]. Available: http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/MQTT_V3.1_Protocol_Specific.pdf

[33] R. Coppen, R. Cohn, and G. Brown, “OASIS Message Queuing Telemetry Transport (MQTT) TC.” [Online]. Available: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt

[34] IBM, “MQTT-S publish/subscribe middleware for wireless sensor systems.” [Online]. Available: http://www.zurich.ibm.com/sys/energy/middleware.html

[35] Mosquitto.org, “Mosquitto - An Open Source MQTT v3.1/v3.1.1 Broker.” [Online]. Available: http://mosquitto.org/

[36] A. Stanford-Clark and H. L. Truong, “MQTT For Sensor Networks (MQTT-SN) Protocol Specification,” 2013. [Online]. Available: http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf

[37] Z. Alliance, “Zigbee ip and 920 ip specification overview,” 2014. [Online]. Available: http://www.zigbee.org/Specifications/ZigBeeIP/Overview.aspx

[38] Bluetooth SIG Inc., “Updated Bluetooth® 4.1 Extends the Foundation of Bluetooth Technology for the Internet of Things,” 2013. [Online]. Available: http://www.bluetooth.com/Pages/Press-Releases-Detail.aspx?ItemID=197

[39] J. Nieminen, T. Savolainen, M. Isomaki, and Z. Shelby, "Transmission of IPv6 Packets over BLUETOOTH Low Energy," pp. 1–16, 2013. [Online]. Available: http://tools.ietf.org/html/draft-ietf-6lowpan-btle-12#section-2.4

[40] ETSI, "Overview of ETSI TC M2M Activities," 2012. [Online]. Available: http://docbox.etsi.org/SmartM2M/open/Information/M2M_presentation.pdf

[41] ETSI TC M2M, "ETSI - Machine to Machine Communications Official Web Page," 2014. [Online]. Available: http://www.etsi.org/technologies-clusters/technologies/m2m

[42] ——, "ETSI TR 101 584 - Study on Semantic support for M2M Data," Tech. Rep., 2013. [Online]. Available: http://www.etsi.org/index.php/technologies-clusters/technologies/m2m

[43] C. Wang and InterDigital Inc., "M2M Service Architecture: Delivering M2M Services Over Heterogeneous Networks," IEEE Commun. Qual. Reliab. 2012 Int. Work., 2012. [Online]. Available: http://www.ieee-cqr.org/2012/May17/Session9/Chonggang_Wang_InterDigital.pdf

[44] B. Pareglio, "OVERVIEW OF ETSI M2M ARCHITECTURE," no. October, 2011. [Online]. Available: http://docbox.etsi.org/workshop/2011/201110_m2mworkshop/02_m2m_standard/m2mwg2_architecture_pareglio.pdf

[45] InterDigital Inc., "Standardized Machine-to-Machine and Internet of Things Service Delivery Platform (M2M / IoT SDP)," p. 20, 2013. [Online]. Available: http://www.interdigital.com/wp-content/uploads/2012/07/M2M-Platform-WhitePaper.pdf

[46] D. T. O. T. M. Boswarthick, "Status of Machine to Machine Standards work in TC M2M and oneM2M," 2012. [Online]. Available: http://www.etsi.org/plugtests/COAP2/Presentations/03_ETSI_M2M_oneM2M.pdf

[47] G. Lu, "Overview of ETSI M2M Release 1 Stage 3 - API and resource usage," pp. 1–16, 2011. [Online]. Available: http://docbox.etsi.org/workshop/2011/201110_m2mworkshop/02_m2m_standard/m2mwg3_api_andresource_usage_lu.pdf

[48] ETSI, "ETSI TS 102 690 - Machine-to-Machine communications (M2M): Functional architecture," pp. 1–332, 2013. [Online]. Available: http://www.etsi.org/deliver/etsi_ts/102600_102699/102690/02.01.01_60/ts_102690v020101p.pdf

[49] The Broadband Forum, "TR-181 Device Data Model for TR-069," p. 124, 2013. [Online]. Available: http://www.broadband-forum.org/technical/download/TR-181_Issue-2_Amendment-7.pdf

[50] OMA, "OMA Device Management V1.2." [Online]. Available: http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/dm-v1-2

[51] ETSI TC M2M, "ETSI TS 103 093 V2.1.1," pp. 1–19, 2013. [Online]. Available: http://www.etsi.org/deliver/etsi_ts/103000_103099/103093/02.01.01_60/ts_103093v020101p.pdf

[52] ETSI, "Major Standards Development Organizations Agree on a Global Initiative for M2M Standardization," 2012. [Online]. Available: http://www.etsi.org/news-events/news/381-news-release-17-january-2012

[53] J. Koss, "oneM2M - A Global Initiative for M2M Standardization," 2012. [Online]. Available: http://www.slideshare.net/zahidtg/onem2m-a-global-initiative-for-m2m-standardization

[54] OneM2M, "onem2m candidate release august 2014," August 2014. [Online]. Available: http://www.onem2m.org/technical/candidate-release-august-2014

[55] ——, "CoAP Protocol Binding Technical Specification," pp. 1–13, 2014. [Online]. Available: http://www.onem2m.org/candidate_release/TS-0008-CoAP_Protocol_Binding-V-2014-08.pdf

[56] ——, "HTTP Protocol Binding Technical Specification," pp. 1–14, 2014. [Online]. Available: http://www.onem2m.org/candidate_release/TS-0009-HTTP_Protocol_Binding_V-2014-08.pdf

[57] ——, "News release 11 April, 2014," 2014. [Online]. Available: http://www.onem2m.org/press/2014-0411TP10_Release_final2.pdf

[58] ——, "Management Enablement (OMA)," pp. 1–54, 2014. [Online]. Available: http://www.onem2m.org/candidate_release/TS-0005-Management_Enablement(OMA)-V-2014-08.pdf

[59] ——, "oneM2M Functional Architecture Baseline Draft," 2014. [Online]. Available: http://www.onem2m.org/candidate_release/TS-0001-oneM2M-Functional-Architecture-V-2014-08.pdf

[60] ——, "oneM2M - Developing MQTT Protocol Binding," 2013. [Online]. Available: ftp://ftp.onem2m.org/Meetings/TP/2013meetings/20131209_TP8_Miyazaki/oneM2M-TP-2013-0388-WI_for_MQTT_binding.DOC

[61] ——, "RFC 5139 - oneM2M Service Layer Protocol Core Specification," pp. 1–176, 2014. [Online]. Available: http://www.onem2m.org/candidate_release/TS-0004-CoreProtocol-V-2014-08.pdf

[62] Open Mobile Alliance, "OMNA Lightweight M2M (LWM2M) Object & Resource Registry." [Online]. Available: http://technical.openmobilealliance.org/Technical/technical-information/omna/lightweight-m2m-lwm2m-object-registry

[63] Sensinode and Z. Shelby, "OMA Lightweight M2M Tutorial," 2013.

[64] IoT-A, "Internet of Things Architecture." [Online]. Available: http://www.iot-a.eu/public

[65] ——, "Internet of Things – Architecture Deliverable D1.5 – Final architectural reference model for the IoT v3.0," no. 257521, 2013.

[66] C. M. MacKenzie, K. Laskey, P. F. Brown, R. Metz, and B. A. Hamilton, "Reference Model for Service Oriented," pp. 1–31, 2006. [Online]. Available: http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf

[67] IoT-A, "IoT-A terminology definitions." [Online]. Available: http://www.iot-a.eu/public/terminology

[68] IoT Forum, "IoT Forum - Architecture and Interoperability." [Online]. Available: http://iotforum.org/work-groups/architecture-and-interoperability-2/

[69] IPSO Alliance, "About IPSO - Vision and Mission." [Online]. Available: http://www.ipso-alliance.org/about/mission

[70] ——, "iPSO - SMART OBJECT GUIDELINES." [Online]. Available: http://www.ipso-alliance.org/smart-object-guidelines

[71] OneM2M, "OneM2M - WebSocket based Notification," 2014. [Online]. Available: ftp://ftp.onem2m.org/Meetings/PRO/20140407_PRO10.0_Berlin/PRO-2014-0160R01-WebSocket_based_Notification.DOC

[72] R. Tabish, A. M. Ghaleb, R. Hussein, F. Touati, A. Ben Mnaouer, L. Khriji, and M. F. a. Rasid, "A 3G/WiFi-enabled 6LoWPAN-based U-healthcare system for ubiquitous real-time monitoring and data logging," 2nd Middle East Conf. Biomed. Eng., pp. 277–280, Feb. 2014. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6783258

[73] Zolertia Shop, "Zolertia Shop - Z1 Platform." [Online]. Available: http://webshop.zolertia.com/product_info.php/cPath/23/products_id/32

[74] ——, "Zolertia Shop - Gateway." [Online]. Available: http://webshop.zolertia.com/product_info.php/products_id/46

[75] ——, "Zolertia Shop - Z1 Starter Platform." [Online]. Available: http://webshop.zolertia.com/product_info.php/products_id/49)

[76] A. J. Jara, M. a. Zamora, and A. F. Skarmeta, "Knowledge Acquisition and Management Architecture for Mobile and Personal Health Environments Based on the Internet of Things," 2012 IEEE 11th Int. Conf. Trust. Secur. Priv. Comput. Commun., pp. 1811–1818, Jun. 2012. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6296204

[77] Personal Connected Health Alliance, "Continua." [Online]. Available: http://www.continuaalliance.org

[78] OMA, "OMA 2014 Newsletter Volume 2," 2014. [Online]. Available: http://openmobilealliance.org/oma-2014-newsletter-volume-2/

[79] K. Ma and R. Sun, "Introducing WebSocket-Based Real-Time Monitoring System for Remote Intelligent Buildings," Int. J. Distrib. Sens. Networks, vol. 2013, pp. 1–10, 2013. [Online]. Available: http://www.hindawi.com/journals/ijdsn/2013/867693/

[80] ETSI TC M2M, "ETSI TS 102 689 - Machine-to-Machine communications (M2M); M2M service requirements," pp. 1–35, 2013. [Online]. Available: http://www.etsi.org/deliver/etsi_ts/102600_102699/102689/02.01.01_60/ts_102689v020101p.pdf

[81] M. Siekkinen, M. Hiienkari, J. K. Nurminen, and J. Nieminen, "How Low Energy is Bluetooth Low Energy ? Comparative Measurements with ZigBee / 802 . 15 . 4," pp. 232–237, 2012.

[82] Arduino, "Arduino Mega 2560," 2014. [Online]. Available: http://arduino.cc/en/Main/arduinoBoardMega2560

[83] ——, "Arduino - Wireless Proto Shield," 2014. [Online]. Available: http://arduino.cc/en/Main/ArduinoWirelessProtoShield

[84] Digi International Inc, "XBee® 802.15.4." [Online]. Available: http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/point-multipoint-rfmodules/xbee-series1-module

[85] PulseSensor, "PulseSensor." [Online]. Available: http://pulsesensor.myshopify.com/pages/code-and-guide

[86] Raspberry Pi Foundation, "Raspberry Pi Model B." [Online]. Available: http://www.raspberrypi.org/products/model-b/

[87] Noolitic, "Noolitic's Nooliberry." [Online]. Available: https://github.com/Noolitic/Nooliberry/wiki

[88] M. B. Alaya, Y. Banouar, T. Monteil, C. Chassot, and K. Drira, "OM2M: Extensible ETSI-compliant {M2M} Service Platform with Self-configuration Capability," Procedia Comput. Sci., vol. 32, no. 0, pp. 1079–1086, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050914007364

[89] M. Collina, "ponte - The Internet of Things Bridge for REST developers." [Online]. Available: https://www.npmjs.org/package/ponte

[90] I. Joyent. Node.js official website. [Online]. Available: http://nodejs.org

[91] "libeio's offical web page." [Online]. Available: http://software.schmorp.de/pkg/libeio.html

[92] "libev's official web page." [Online]. Available: http://software.schmorp.de/pkg/libev.html

[93] J. Kunkle. (2014) Node.js presentation. [Online]. Available: http://kunkle.org/nodejs-explained-pres/#/event-loop

[94] T. Rault, A. Bouabdallah, and Y. Challal, "Energy efficiency in wireless sensor networks: A top-down survey," Comput. Networks, vol. 67, pp. 104–122, Jul. 2014. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1389128614001418

[95] Bluetooth, “Specification of the Bluetooth System. Core Version 4.0.” vol. 0, no. June, 2010. [Online]. Available: https://www.bluetooth.org/en-us/specification/adopted-specifications

[96] K. Mikhaylov, N. Plevritakis, and J. Tervonen, “Performance Analysis and Comparison of Bluetooth Low Energy with IEEE 802.15.4 and SimpliciTI,” J. Sens. Actuator Networks, vol. 2, no. 3, pp. 589–613, Aug. 2013. [Online]. Available: http://www.mdpi.com/2224-2708/2/3/589/

[97] C. Gomez, J. Oller, and J. Paradells, “Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology,” Sensors, vol. 12, no. 12, pp. 11 734–11 753, Aug. 2012. [Online]. Available: http://www.mdpi.com/1424-8220/12/9/11734/

[98] Gogo6, “gogoCLIENT Download Page.” [Online]. Available: http://www.gogo6.com/profiles/profile/show?id=gogoCLIENT

[99] litech.org, “Linux ipv6 router advertisement daemon (radvd),” 2014. [Online]. Available: http://www.litech.org/radvd/

[100] Gogo6, “Freenet6 Tunnel Broker.” [Online]. Available: http://www.gogo6.com/freenet6/tunnelbroker)

[101] R. Gilligan and E. Nordmark, “Transition Mechanisms for IPv6 Hosts and Routers,” 2000. [Online]. Available: https://www.ietf.org/rfc/rfc2893.txt

[102] Raspberry Pi Foundation, “Raspbian.” [Online]. Available: http://www.raspbian.org

[103] T. Narten, E. Nordmark, and W. Simpson, “RFC 2461 - Neighbor Discovery for IP Version 6 (IPv6)),” 1998. [Online]. Available: https://datatracker.ietf.org/doc/rfc2461/

[104] C. Duiadns, “D.U.I.A. - Dynamic Updates for Internet Addressing.” [Online]. Available: www.duiadns.net

[105] CETIC, “6lbr - A deployment-ready 6LoWPAN Border Router solution based on Contiki,” 2014. [Online]. Available: http://cetic.github.io/6lbr/

[106] ——, “CETIC - Centre d’Excellence en Technologies de l’Information et de la Communication.” [Online]. Available: https://www.cetic.be

[107] ——, “RaspberryPi Software Configuration,” 2014. [Online]. Available: https://github.com/cetic/6lbr/wiki/RaspberryPi-Software-Configuration

[108] L. Deru, S. . Dawans, M. Oca, B. Quoitin, and O. Bonaventure, “Redundant Border Routers for Mission-Critical 6LoWPAN Networks,” Proc. Fifth Work. Real-World Wirel. Sens. Networks, 2013.

[109] D. Thaler, C. Patel, and M. Talwar, “Neighbor Discovery Proxies (ND Proxy),” pp. 1–18, 2006. [Online]. Available: http://tools.ietf.org/pdf/rfc4389.pdf

[110] L. M. Ara, “Neighbor Discovery Proxy-Gateway Wireless Sensor Networks Neighbor Discovery Proxy-Gateway for 6LoWPAN-based Wireless Sensor Networks,” 2011. [Online]. Available: http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/111221-Luis_Maqueda-with-cover.pdf

[111] P. Thubert, “6LoWPAN Backbone Router,” pp. 1–20, 2013. [Online]. Available: http://tools.ietf.org/html/draft-thubert-6lowpan-backbone-router-03

[112] CETIC, “6lbr - ND Proxy.” [Online]. Available: https://github.com/cetic/6lbr/wiki/Nd-Proxy

[113] ——, “6lbr - 6LBR Modes.” [Online]. Available: https://github.com/cetic/6lbr/wiki/6LBR-Modes

[114] L. Deru, “6LBR Web Page - 6LBR Modes,” 2013. [Online]. Available: https://github.com/cetic/6lbr/wiki/6LBR-Modes

[115] nylen. request node.js module web page. [Online]. Available: https://www.npmjs.org/package/request

[116] nfarina. xmldoc node.js module web page. [Online]. Available: https://www.npmjs.org/package/xmldoc

[117] pkrumins. base64 node.js module web page. [Online]. Available: https://www.npmjs.org/package/base64

[118] dougwilson. express node.js module web page. [Online]. Available: https://www.npmjs.org/package/express

[119] M. Collina, “mosca - MQTT broker as a module.” [Online]. Available: https://www.npmjs.org/package/mosca

[120] ——, “Use MQTT from the Browser, based on MQTT.js and websocket-stream.” [Online]. Available: https://www.npmjs.org/package/mows

[121] Eclipse, “Paho- Open Source messaging for M2M.” [Online]. Available: http://www.eclipse.org/paho/

[122] ——, “Paho - JavaScript Client.” [Online]. Available: http://www.eclipse.org/paho/clients/js/)

[123] Télecom Bretagne, “Arduino-IPv6Stack,” 2012. [Online]. Available: https://github.com/telecombretagne/Arduino-IPv6Stack/

[124] ——, “Arduino-pIPv6Stack,” 2013. [Online]. Available: https://github.com/telecombretagne/Arduino-pIPv6Stack

[125] R. Meier, “CoolTerm.” [Online]. Available: http://freeware.the-meiers.org/CoolTerm_ReadMe.txt.html

[126] Arduino, “Arduino IDE.” [Online]. Available: http://arduino.cc/en/main/software#toc1

[127] Creative Commons, “Creative Commons Attribution-ShareAlike 3.0.” [Online]. Available: http://creativecommons.org/licenses/by-sa/3.0/us/

[128] SparkFun Electronics, “Temperature Sensor - TMP36.” [Online]. Available: https://www.sparkfun.com/products/10988

[129] oomlout.com, “Tmp36 temperature sensor source code.” [Online]. Available: http://www.oomlout.com/ARDX/CIRC10/CIRC10-code.txt

[130] J. M. H. F. L. M. P. L. T. B.-L. R. Fielding, J. Gettys. (1999, June) Rfc 2616 - hypertext transfer protocol – http/1.1. [Online]. Available: http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.5

[131] Cisco. The iot opportunity. [Online]. Available: http://www.cisco.com/web/solutions/trends/iot/indepth.html

[132] O. Bergmann, “tinydtls Web Page,” 2013. [Online]. Available: http://tinydtls.sourceforge.net/

[133] O. Bergmann, S. Gerdes, and C. Bormann, “Simple Keys for Simple Smart Objects,” Work. Smart Object Secur., 2012. [Online]. Available: http://www.lix.polytechnique.fr/hipercom/SmartObjectSecurity/papers/OlafBergmann.pdf

[134] Z. Cao, R. Cragie, and B. Haberman, “Light-Weight Implementation Guidance (lwig) Working Group,” 2011. [Online]. Available: https://datatracker.ietf.org/doc/charter-ietf-lwig/

[135] D. F. Aranha and C. P. L. Gouvêa, “RELIC is an Efficient LIbrary for Cryptography,” 2013. [Online]. Available: https://code.google.com/p/relic-toolkit/

[136] M. Sethi, J. Arkko, A. Keranen, and H. Rissanen, “Practical Considerations and Implementation Experiences in Securing Smart Object Networks,” 2012. [Online]. Available: http://www.arkko.com/publications/draft-aks-crypto-sensors.txt

[137] Thingsquare, “Contiki: The open source os for the internet of things,” 2014. [Online]. Available: http://www.contiki-os.org/

[138] M. S. David Navarro, Julien Vermillard, “Wakaama (lwm2m library),” 2013. [Online]. Available: http://eclipse.org/proposals/technology.liblwm2m/