University of Pisa
and
Scuola Superiore Sant'Anna

Master Degree in Computer Science and Networking
Master Thesis

Performance of data serialization methods for wireless communication in resource-constrained devices

Candidate

Francesca Pacini

Supervisors

Prof. Piero Castoldi

Dr. Paolo Pagano

Dr. Femi Aderohunmu

Academic Year 2013/2014

Contents

1 Introduction **7**
- 1.1 Background: the Internet of Things . . . 7
- 1.2 The Internet of Things Service Domain . . . 8
  - 1.2.1 Intelligent Transport Systems . . . 9
- 1.3 Enabling Technologies . . . 11
  - 1.3.1 Internetworking of smart devices . . . 11
  - 1.3.2 Machine-To-Machine . . . 12
- 1.4 Objectives . . . 14
- 1.5 Contribution of this work . . . 14
- 1.6 Methodology . . . 15
- 1.7 Structure of the thesis . . . 16

2 Enabling Technologies **17**
- 2.1 Introduction . . . 17
- 2.2 IEEE 802.15.4 . . . 18
  - 2.2.1 IEEE 802.15.4 Physical Layer . . . 19
  - 2.2.2 IEEE 802.15.4 MAC Layer . . . 20
- 2.3 IPv6 over Low Power Wireless Personal Area Networks . . . 20
- 2.4 The Transport Protocol . . . 22
- 2.5 Constrained Application Protocol . . . 22
  - 2.5.1 CoAP Message Layer . . . 24
  - 2.5.2 CoAP Request/Response model . . . 24
  - 2.5.3 Message Structure . . . 26
  - 2.5.4 CoAP Methods . . . 27
  - 2.5.5 Transmission Reliability . . . 27
  - 2.5.6 Resource Discovery . . . 28
  - 2.5.7 CoAP Observer pattern . . . 28
  - 2.5.8 Blockwise Transfer . . . 32
- 2.6 The ETSI M2M Communication Paradigm . . . 34
  - 2.6.1 M2M Resources . . . 38
  - 2.6.2 M2M Operations . . . 38

2.6.3 M2M remote execution and data distribution 40
2.7 Data Serialization Formats 45
2.7.1 JavaScript Object Notation (JSON) 46
2.7.2 Efficient XML Interchange (EXI) 46

3 Related Work 52
3.1 EXIP based implementations 52
3.2 Non-EXIP based implementations 53
3.3 Original contribution 55

4 IoT in the ICSI project 56
4.1 Introduction 56
4.2 ICSI System Architecture 57
4.2.1 Central Sub-System 58
4.2.2 Roadside Sub-System 59
4.2.3 Vehicular Sub-System 59
4.2.4 Personal Sub-System 59
4.2.5 The ICSI System 59

5 Implementation and performance assessment 62
5.1 Introduction 62
5.2 The Development Platform 63
5.2.1 Contiki 63
5.3 Preliminary performance assessment 64
5.4 System Parameters 65
5.5 Compression performance evaluation 66
5.5.1 The Measurement process 67
5.5.2 Energy measurements 67
5.6 Compression Gain 68
5.6.1 Size in terms of CoAP blocks 72
5.7 Serialization Complexity 74
5.7.1 Serialization time 74
5.7.2 Deserialization time 74
5.7.3 Serialization memory usage 75
5.8 ETSI M2M CoAP Interoperability Tests 75
5.9 application Create 77
5.10 application Retrieve 79
5.11 application Update 83
5.12 subscription Create 88
5.13 contentInstance Retrieve 91
5.14 Comparison between GET and POST performances 95

5.15 Performance in lossy configurations 98
5.15.1 applicationCreate POST with 95% PDR 98
5.16 Summary of Results 101
5.17 Further Optimizations 103
5.18 Comparison with the state-of-the-art 103
5.19 Fulfillment of ICSI time constraints 104

6 Conclusions **105**
6.1 Future direction 106

Appendices **108**

A DATEX2 notification **109**

B <application> profiles **111**

C <subscription> profiles **113**

D <contentInstance> profiles **115**

Bibliography **118**

List of Figures

1.1 Closed Loop ITS 10
1.2 The Open Loop model in classical transportation systems 10
1.3 Protocol Suite for Constrained Networks 12
1.4 IoT Protocol Suite 13

2.1 CoAP abstract layers 23
2.2 Reliable and unreliable message transmission 24
2.3 The three paradigms for CoAP request/response 25
2.4 CoAP message format 26
2.5 Discovery message exchange example 28
2.6 Observing a resource in CoAP 29
2.7 High level architecture for M2M 36
2.8 Deployment scenarios and reference points 37
2.9 Execution of a mgmtCmd resource 41
2.10 Execution of a mgmtObj resource 42
2.11 Resource Announcement 44
2.12 Long polling and notifications 45

4.1 ICSI system components 58
4.2 ICSI system mapped on ETSI M2M 60
4.3 ICSI logical areas 60

5.1 *Cooja* simulation 66
5.2 <application> resource serialization - size of serialized data . . 70
5.3 <subscription> resource serialization - size of serialized data . . 71
5.4 <contentInstance> resource serialization - size of serialized data 72
5.5 applicationCreate execution time 80
5.6 applicationCreate energy consumption 81
5.7 applicationRetrieve execution time - Scenario 1 84
5.8 applicationRetrieve execution time 85
5.9 applicationRetrieve execution time - Scenario 2 86

5.10 applicationUpdate execution time 88
5.11 applicationUpdate energy consumption 89
5.12 subscriptionCreate execution time 91
5.13 subscriptionCreate execution time 92
5.14 contentInstanceRetrieve execution time - Scenario 1 95
5.15 contentInstanceRetrieve energy consumption - Scenario 1 96
5.16 contentInstanceRetrieve execution time - Scenario 2 97
5.17 applicationCreate execution time, 95% PDR 99
5.18 applicationCreate energy consumption, 95% PDR 100

Chapter 1

Introduction

1.1 Background: the Internet of Things

The Internet is an entity which is in constant evolution. From the initial *"Internet of Computers"* with services such as the World Wide Web built on top of it, to the *"Internet of People"* which saw the emergence of a Social Web, the Internet is now once again witnessing major change.

Cheap and ubiquitous broadband connectivity and the availability of low cost embedded systems with on-board sensors is leading to a reality where physical objects themselves are included in the Internet, and are able to provide services. In order to handle the large amount of data produced, physical objects are also expected to become "smarter" and provide applications with complex data that may be the result of information either acquired locally or retrieved from other objects.

The *"Internet of Things"* [1] is expected to provide an enhanced interaction with the objects that surround us. Sensing, memory and processing capabilities will enable the user to receive information that would not be available or easily accessible otherwise, regarding for instance their history or their non-obvious characteristics.

Ubiquitous connectivity would on the one hand make such information available to the user remotely. On the other hand it would enable objects themselves to exchange and make use of their information independently of their location or ownership to a given provider or network. It would allow them to take management decisions and trigger automated reaction to conditions, including notifications to the human users.

This chapter gives an overview of the service domain of the *"Internet of Things"* and introduces the enabling technology stack, which will be further explained in the following chapters. It also aims at positioning the present

work in relation to the objectives that lie before industry and research in order to exploit the *"Internet of Things"* paradigm fully.

1.2 The Internet of Things Service Domain

The IoT vision is driven by the concepts expressed in the previous section: objects are "smart" and most importantly, Internet-connected. They also are highly energy autonomous. These three characteristics make it possible to develop a platform where services can be easily deployed and maintained. The domains where IoT is expected to play a key role in services enablement are[1]:

- *Smart Environments:* management systems for homes, buildings and other infrastructures can take advantage of schedules and usage patterns of the inhabitants in order to balance supply and demand in a predictive manner; management functions and procedures related to utility consumption, heating, lighting, safety and security are made available remotely. It should be possible to identify and manage risks such as fire hazards, as well as to trigger physical intervention. Moreover, sensors placed in proximity of structural and mechanical components will reduce the need for routine inspections and preventive maintenance.

- *Smart Cities:* sensor collected information can be used to adjust the intensity of light to weather conditions or presence of citizens or cars, resulting in significant energy savings; traffic can be decreased by using sensor data to provide alerts related to traffic jams and serve as input for enhanced routing applications; visual or magnetic sensors can detect free slots and guide cars to the available parking spaces. In the domain of waste management, filling sensors can be used to elaborate more efficient routes for container pickup services, resulting in a cost effective optimization as well as in a reduction of CO_2 emissions.

- *Asset management, transportation and logistics:* supply chains can themselves become "smart". Tracking items involved in a supply chain through sensor devices instead of RFIDs allows keeping track of additional information such as temperature and current location, thus bringing the benefits of automating the supply chain, gaining efficiency, reducing human error, and enhancing the process by managing information regarding transport and stock conditions of goods.

[1] This list has been elaborated on the basis of a network survey and consists of some of the most common use cases envisioned.

- *Retail industry and smart shopping:* in automated distribution, collecting real time data and inventory of a single machine makes it possible to detect failures as soon as they arise, to adapt refilling logistics before the stock breaks, and to change prices remotely. Shopping analytics can be improved by monitoring the number of visitors as well as shopping patterns in order to detect hot and cold zones and time spent; and it can enhance shop design and functionality.

- *Industry and agriculture:* assets used in industrial installations such as vehicles, drillers, pumps, pipes etc. can be remotely monitored, enabling automated inventory, remote locations and geo-fence alerts, remote diagnostics and maintenance to reduce costs. The irrigation process can increase its efficiency, and the combined use of humidity sensors and remotely manageable water valves enables the development of applications aimed at reducing water consumption, augmenting crop productivity and detecting unauthorized water usage.

This work has been executed within the research activities related to the field of Intelligent Transport Systems (ITSs). As explained in the following paragraph, ITS domain is analogous to the one to which Smart City systems belong. Still, ITSs have been developed independently from IoT and are characterized by their own system architecture. For this reason they are addressed in a separate section.

1.2.1 Intelligent Transport Systems

Intelligent Transport Systems (ITSs) evolved in parallel to IoT. Very similar use cases have been developed in the domain of IoT with the name of *Smart Cities*.

From an architectural point of view ITS systems are characterized by a closed loop interaction between the user and the transportation infrastructure as depicted in Figure 1.1.

This pattern of interaction is an evolution of the classical open loop transportation architecture depicted in Figure 1.2. In an ITS system the user provides input to the transportation infrastructure system, and receives a feedback. An ITS system is often structured according to the "Model-View-Controller" pattern. The system formed by the *User*, the *Application*, the *Co-operative Intelligence* and the *ITS Logical Abstraction* resembles the "Model-View-Presenter with Passive View" pattern, since the presentation logic is decoupled from the "view" component.

The use cases that properly define ITS systems according to the European Telecommunications Standards Institute (ETSI) are described below [9].

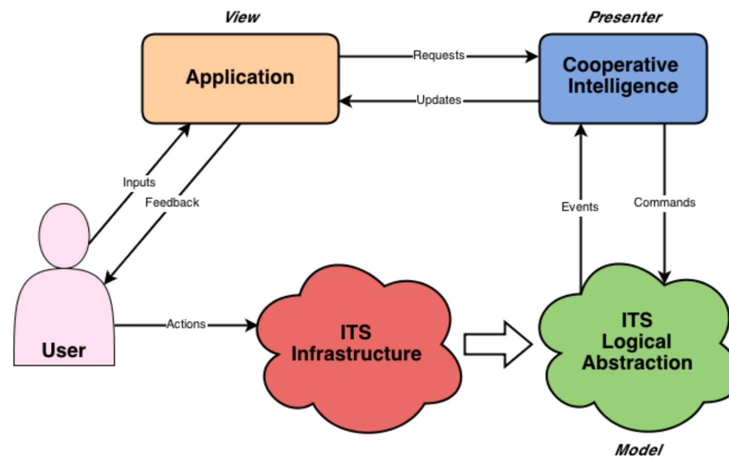


Figure 1.1: Closed Loop ITS

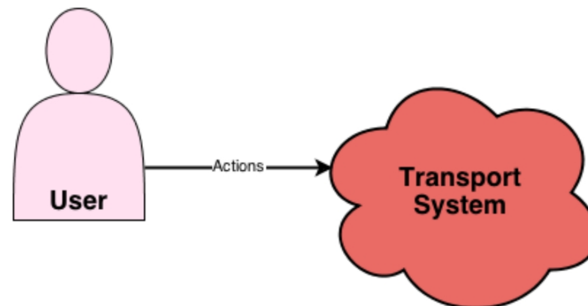


Figure 1.2: The Open Loop model in classical transportation systems

A first group of use cases is related to vehicle safety and security:

- Breakdown Call: the use case is triggered by the user pushing a switch which activates the service. Vehicle diagnostic information as well as vehicle position are sent to a roadside assistance unit, allowing the fault to be diagnosed and permitting a quicker and more effective response.
- Stolen Vehicle Tracking: this use case is initiated by a user providing a police report number, and is activated through retrieval of location data from the on-board Transmission Control Unit communication with the police.
- Remote Diagnostic: this set of use cases encompasses a *maintenance minder* use case, a *health check* and a *fault triggered* use case. In the

first use case, when a mileage threshold has been reached, the user is prompted to schedule a vehicle check-up. The second one enables the on-board *Transmission Control Unit* to compile the vehicle's status diagnostic using inbuilt reporting functions, prompting the user if intervention is needed. The *fault triggered* use case alerts the vehicle owner or manufacturer to the fact that a fault has been detected within one of the vehicle's systems.

In all the use cases mentioned the user can be prompted through interfaces such as phone and a Mobile UI.

A second group of use cases is related to connected navigation:

- Traffic Reports: according to this use cases, the user is warned of a traffic congestion on the current or intended route.
- Route Planning: a route to a given destination that has been elaborated for instance on a PC may be imported into an vehicle navigation system.
- Information Provisioning: this use case provides information to the vehicle driver and passenger, and includes for instance mobile TV, web browsing and email service.

While on the one hand the IoT stack is clearly shaped, the ETSI standardization of Intelligent Transport Systems generically recommends to follow the modularity of the ISO/OSI stack but does not in any way mandate a specific choice for the technologies used at each layer.

According to the ETSI use cases definition, ITS systems can be expected to share two of the three key requirements of IoT. In addition since ISO recommends the use of IPv6 throughout all types of systems the third requirement is enforced as well.

For these reasons, ITS systems can be configured as a specific case of IoT systems: the adoption of the IoT stack is the object of a recent proposal to ETSI.

1.3 Enabling Technologies

1.3.1 Internetworking of smart devices

The driving requirements for IoT enabling technologies can be summarized as[13]:

- *Energy efficiency:* device sensors with limited processing and memory capabilities need to have a conservative energy management.

- *Reliability On-Demand:* it should be possible to establish reliable communication, but also to switch seamlessly to non-reliable communication whenever this option is preferred – which could be due to loose requirements of data delivery and a higher time performance expectation.

- *Internet-connectivity:* objects should have IP addresses so that they can be accessible directly at IP level.

Several attempts at shaping the IoT stack have been made over the last two decades. A big part of the achievements and the design choices that were perceived as most successful have recently been consolidated in a set of protocols through an extensive standardization activity. The Institute of Electrical and Electronics Engineering (IEEE) has defined a Link Layer protocol for efficient radio transmission (IEEE 802.15.4); at the Application Layer the Internet Engineering Task Force (IETF) has defined the Constrained Application Protocol (CoAP) and the adaptation layer for IPv6 over constrained networks 6LoWPAN.

The technology stack is shaped as depicted in Figure 1.3: the principles of each component and the compliancy to the aforementioned core requirements will be discussed in Chapter 2.

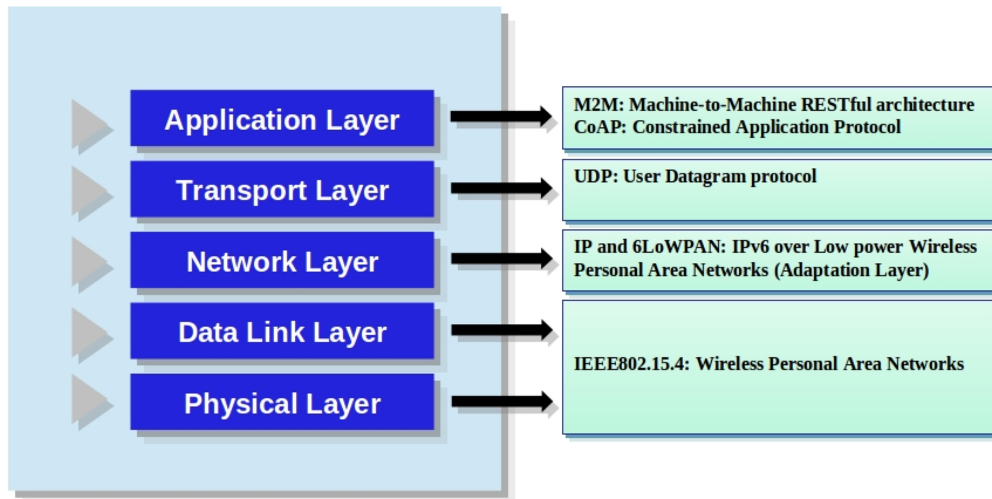


Figure 1.3: Protocol Suite for Constrained Networks

1.3.2 Machine-To-Machine

Machine-To-Machine (M2M) protocols emerge from an approach that is orthogonal but complementary to the one that drove the development of the IoT

stack, and it is centered on the concept of service enablement at the application level.

As shown in Figure 1.4, M2M systems aim at building a framework for data access that allows vertical services to be simply developed and “plugged-in” without needing to know where data is generated, pre-processed or stored.

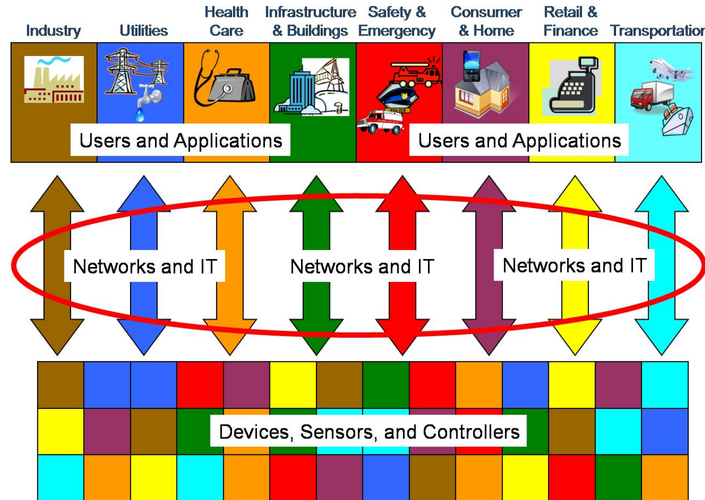


Figure 1.4: IoT Protocol Suite

From a system point of view this vision can be translated into the following two goals:

1. To enable applications to work with de-localized data, i.e. the data used for processing may be located anywhere in the network and be composed of parts displaced in multiple physical locations.

2. To enable applications to be built independently of communication protocols necessary to access those data.

In line with the idea of web service *mashups*, applications become independent of who or what generates a data item, as long as it is accessible through a URI.

M2M systems find their joining link with IoT in correspondence with:

- *RESTful architecture:* REST is the architectural style of choice for the design of CoAP, as well as the mandated data access pattern for M2M systems.

- *IPv6-based Internet connectivity:* according to ISO guidelines that recommend a horizontal IP addressability.

Among the M2M protocols available, the ETSI M2M Technical Specification has particular relevance, in that it is the product of an effort at standardization. For this reason it has been adopted in this work and will be discussed in the next chapter.

1.4 Objectives

The following key objectives in order to fully realize the ITS paradigm in a IoT perspective were identified:

- Align ITS and IoT evolution: the overall degree of interoperability would increase, since ITS would then be able to benefit from the traction offered by the IoT ecosystem development due to the creation of integrated services.
- Address the question as to whether the IoT technology stack outlined so far is able to support successfully the services envisioned, and which further modifications will be necessary to realize an effective M2M architecture. This research effort should be integrated with the standardization process itself providing active input.
- Design, build and test freely available cases of study which can be used as a reference and if possible converge into compact library functions. Within the industry proprietary IoT solutions exist[2], whereas an open-source implementation as well as performance measurements that can guide design and dimensioning of new deployments are not yet available.

1.5 Contribution of this work

The contributions of the work described in this thesis to the objectives mentioned are:

- An ITS solution based on the state-of-the-art of IoT OSI stack technologies has been considered as a case study for performance assessment, which adapts the most promising candidate technology for the IoT application layer to ITS systems, that is the ETSI Machine-To-Machine protocol.

[2]for example: `http://www.sensinode.com/EN/technology.html` Last Access: June 12, 2014.

- We conducted a comparative study on the impact of the data serialization format choice on the system performance from the points of view of energy consumption, channel usage and operation execution time. We focus on the data serialization formats recommended by ETSI Machine-To-Machine Specification, which are XML, JSON or EXI[3], and elaborate a performance assessment for the last two within a Wireless Sensor Network IoT system deployment.

- The work has been carried out using an open-source platform, since it has been developed using the *Contiki* operating system, and the performance assessment was conducted using a simulation tool. It constitutes the first open-source realization of the complete IoT stack for ITS systems.

1.6 Methodology

The approach we followed can be summarized as:

- Real Testbed: ETSI M2M CoAP Interoperability Tests [8] have been selected as a test benchmark. These are the standard tests recommended by ETSI and are executed in order to determine whether the implementation makes proper use of CoAP primitives, and their execution is a necessary condition for stating that a system correctly integrates ETSI M2M onto the IoT stack. They are expected to be executed very often and adopting them as messaging benchmark facilitates the comparison with our results.

- Performance Assessment Criteria: the criteria adopted consist of three key indicators for an M2M system following the IoT paradigm and realized over a Wireless Sensor Network. The measurement of channel usage for a given operation directly relates to the concern of limiting interference in a given wireless network and therefore being able to host a higher number of communications. Energy efficiency is a priority for IoT because objects will often be battery-powered and they are required to be highly energy autonomous. Operation execution time provides an indicator of which time constraints can be met by the M2M data distribution platform.

- Use of a Simulator: the *Cooja* simulator has been used because it supports the emulation of a variety of *Contiki* chipsets. The tests can therefore be executed on different platforms and in different conditions of packet loss at physical level.

[3]Annex D, section 1, [12]

1.7 Structure of the thesis

This thesis addresses the mentioned topics as follows:

- Chapter 2 discusses IoT enabling technologies. It describes IoT-specific features according to a set of core requirements and highlights the characteristics having greatest impact on the measured data, that is constraints on packet sizes at the various OSI layers. It also considers the Efficient XML Interchange (EXI) serialization format recommended by ETSI as an alternative to JSON for constrained networks.
- Chapter 3 describes the work which has been done so far in the scientific community towards the objectives previously expressed.
- Chapter 4 outlines the framework in which the present work has been carried out: the Intelligent Cooperative Sensing for Improved Traffic Efficiency (ICSI) project.
- Chapter 5 describes the concrete contribution of this work and the performance measurement results of EXI versus JSON.
- Chapter 6 presents the conclusions of this work and possible ideas for future work.

Chapter 2

Enabling Technologies

2.1 Introduction

This chapter gives an overview of IoT enabling technologies, highlighting the underlying requirements, explaining their design rationale and what in turn makes them right for IoT.

The evolution of Mobile Internet, based on smart phones, tablets and netbooks, towards the Internet of Things where connectivity is provided by everyday objects of any kind, affects the requirements for the technologies used: the goals of efficiency and interoperability now become pre-conditions for success.

The need for efficiency derives from the fact that while most people are acquainted with the idea of charging their mobile phone or laptop every day, charging the batteries of millions of objects is clearly unfeasible.

The requirement of interoperability derives instead from the fact that objects will not only be interconnected but, more precisely, Internet-connected. That is, IoT will be IP enabled.

This rationale can be expressed through three core requirements [13]:

- **Low Power Communication Stack:** when energy saving is not a key requirement, energy can be easily wasted in transmitting unessential data, overheads, and using non-optimized communication patterns. Existing and widely adopted protocols such as TCP and HTTP bring verbosity and often unnecessary overhead in terms of additional headers and of a high number of message exchanges. While it remains desirable to preserve interoperability with existing technologies to as great an extent as possible, it is also necessary to elaborate suitable alternatives at each layer of the protocol stack.
- **Efficient reliability "on-demand":** currently the best-effort Internet medium is made reliable at different levels through error detection,

retransmissions and flow control. The fact that these techniques are applied at multiple layers often results in inefficient overall functioning. It is required instead to achieve the same reliability at a much higher efficiency. While on the one hand it should be possible to achieve reliability whenever it is a requirement in a given deployment, on the other hand it should also be possible to achieve better performance in case such a requirement is dropped, while still using the same protocol set. For example, if a sensor node samples values from a continuous variable and stores them in a remote repository a certain degree of packet loss may be tolerable, and system designers may choose not to use reliable protocols that would imply extra transceiver on-time while waiting for acknowledgements. In the case of event detection, instead, being able to ensure reliability becomes crucial.

- **Internet-Enabled Communication Stack:** objects should be able to be singularly and seamlessly connected to the Internet, and this essentially means that it should be possible to identify them at IP level.

After mid-nineties projects demonstrated the feasibility of the core ideas of low-power WSNs, several companies started pioneering the market. Since it soon became clear that engineering the technology principles would rapidly lead to having a number of proprietary systems which would impede scalability as well as a wide ecosystem growth, the start of the standardization effort from the Institute of Electrical and Electronics Engineers (IEEE) and the Internet Engineering Task Force (IETF) in 2003 was followed with high expectations.

Currently the approved or proposed standards cover all of the technology layers: IEEE 802.15.4 at the Physical and Link Layer, 6LoWPAN as adaptation Layer between IPv6 and the underlying protocols, CoAP at the Application Layer.

2.2 IEEE 802.15.4

IEEE 802.15.4 addresses low-rate personal area networks (LR-WPAN), and comprises Physical and Link Layer specifications.

A radio consumes energy for transmission, reception and listening: during transmission, the signal needs to be modulated and amplified; during reception, the signal needs to be amplified through a Low-Noise-Amplifier and demodulated.

Transmitting can be slightly more energy demanding than receiving and listening; but since the current consumption does not depend significantly on whether a radio is listening or receiving bytes, the characteristic that impacts

the most is the duty cycle, meaning the percentage of time the radio is actually on, which is expected to be lower than 1%.

The second most relevant parameter is the operating current: while the duty cycle is determined by the PHY and MAC layer protocols adopted, the operating current depends on the design of the device, meaning how well it is designed in order to achieve power efficiency.

The amount of time spent transmitting and receiving when in absence of communications the device would be listening also impacts in terms of energy consumption, but is less relevant than the duty cycle: this depends on the protocols of the upper layers.

A typical low-power radio operates on a current of the order of 10mA.

Let us consider the case when the device is powered through a couple of AA batteries, which allow 3000 mAh of functioning. This means that a device would typically be able to work for 300 hours, that is 12 days. With a duty cycle of 1%, instead, it would be able to work for 3 years and a half, and 7 years if we have a more power efficient device operating on a 5 mA current.

2.2.1 IEEE 802.15.4 Physical Layer

IEEE 802.15.4 defines several possible frequency bands that can be used at the Physical layer, as listed in Table 2.1: the most frequently used one is the 2.4–2.485 GHz frequency band, where 16 channels spaced by 5MHz are defined.

A transceiver can send and transmit over any of these channels, and switching between channels takes at most 192μs.

Data rate	Symbol rate	Frequency Band	Number of Channels	Modulation
20 Kbps	20 Ksyms	868 - 868.6 MHz	1	BPSK
20 Kbps	20 Ksyms	905 - 928 MHz	10	BPSK
250 Kbps	62.5 Ksyms	2.405 - 2.480 GHz	16	O-QPSK

Table 2.1: PHY Bit rates

The Physical layer specifies how frame detection occurs: after the transmission of a 128 μs preamble allowing the receiver to lock onto the signal, a well-known Start Frame Delimiter (SFD) is sent to indicate the start of the frame. The first byte representing the length of the payload itself is sent: the payload length has a maximum value of 127 bytes which makes the size of the frame 128 bytes including the header.

After receiving the indicated number of bytes and buffering them, the radio switches off and notifies the micro-controller of the reception.

Apart from using the appropriate modulation schemes, a device that is 802.15.4 compliant at the Physical layer must simply be able to lock onto a signal, recognize the preamble (SFD), and buffer data for a length indicated by the first byte received.

2.2.2 IEEE 802.15.4 MAC Layer

IEEE 802.15.4 MAC layer defines four possible frame types:

- **Data Frame:** is used for data transfer.
- **ACK Frame:** is used to confirm successful reception.
- **MAC Command Frame:** is used to control and remotely configure client nodes.
- **Beacon Frame:** if beacon-mode is enabled it allows synchronization of attached devices, Personal Area Network (PAN) identification, and superframe structure description.

Addresses can be of two types: either globally unique extended IEEE EUI-64 bit addresses, or short 16 bit addresses which refer to a given PAN network.

The IEEE 802.15.4 MAC layer is designed for a network with a star topology. If instead the target topology is multi-hop, the protocol does not perform as well: non end-devices will necessarily work at a very high duty cycle, and instabilities in a single channel may impede the network functioning in case there are no alternative channels available.

To address a wider set of topologies, the IEEE task group amended the MAC layer through IEEE 802.15.4e introducing Time Synchronized Channel Hopping (TSCH) based on a successfully engineered and proprietary adaptation of the 802.15.4 MAC layer.

Without going deeply into the amendment details, synchronization is used to achieve power efficiency and channel hopping to improve reliability over the physical medium.

2.3 IPv6 over Low Power Wireless Personal Area Networks

In order to interconnect IEEE 802.15.4 networks among themselves and with networks of different types, IPv6 support is needed. As often happens, mapping IPv6 on the Link Layer can be not trivial, and requires a layer in itself:

the IETF IPv6 over Low power WPAN (6LoWPAN) working group has been addressing this issue since 2007.

The key difficulties that motivate the existence of this adaptation layer are the following:

- **Size compatibility:** the IPv6 MTU of 1280 bytes is way too large to fit in a IEEE 802.15.4 frame without being fragmented. Moreover the 40 bytes IPv6 header is clearly a large overhead for short messages.
- **Subnet broadcast:** possibility of broadcasting Link Layer frames is required.
- **Management:** support for address auto-configuration is needed.

Let us consider how the first issue is addressed, since it is the one having the most evident impact in terms of communication efficiency.

A 6LoWPAN message consists in a stack of headers preceding the IPv6 Payload; the simplest situation is when the IPv6 packet is encapsulated and optionally compressed in a 6LoWPAN *Dispatch* message.

In case the IPv6 packet needs to be fragmented, a *Fragmentation Header* precedes the *Dispatch* header, and is constituted by fields indicating the original IPv6 packet size, an ID identifying the IPv6 packet, and the current fragmentation offset.

A *Mesh Addressing Header* may precede the *Fragmentation Header* in case the origin and destination of the 6LoWPAN message are not within a single-hop link. IEEE 802.15.4 does not define any routing protocol, but if an external routing capability is used the *Mesh Addressing Header* includes, in addition to the origin and destination link-layer addresses, the address of the forwarding node and the next-hop node.

A particular type of *Dispatch* message is the *Broadcast Header*: it occupies a different position in the header stack with respect to the other types of *Dispatch* headers, and includes a sequence number so that duplicate packets can be suppressed.

IPv6 header compression is based on address compression using shared states and variable-length encoding. In the best case, a single-hop IPv6 header can be brought to 2 bytes; a multi-hop one to 7 bytes.

A compression can also be applied to UDP headers: the length field is never present since it can obtained from the IEEE 802.15.4 frame header; port values can be compressed if they match a common set of ports, and the checksum can be omitted.

2.4 The Transport Protocol

Reliability in IP networks is generally provided at the transport layer, either through TCP or through an extension on top of UDP.

The reasons why TCP is not a sensible solution are:

- **Header Size:** it would be desirable to reduce the header size: TCP has a header that can range from 16 to 40 bytes;
- **Reliability "on-demand":** IoT applications do not always require reliability: reliability is not needed for example when periodic updates related to the value of a given non-critical continuous variable are received where only the last value matters, for example a temperature value in a temperature control system. On the other hand, if the heating control logic is delegated to the sensor node, a reliable event notification when the temperature crosses a given threshold value may be sent.
- **Performance degradation due to congestion control:** TCP does not differentiate between congestion and link errors, and interprets a packet loss as due to congestion and reacts by decreasing the sending rate: in wireless networks often packet loss is caused by either transmission errors or mobility, and may be temporary.

For these reasons, it is preferable to build reliability on top of UDP instead.

2.5 Constrained Application Protocol

As far as the application layer is concerned, the first option to be considered is HTTP, given its widespread use.

The main drawbacks of HTTP are:

- **User-oriented:** it has a strict request response messaging paradigm which works well with interactions with the user, since a user usually expects an application-level response even if just an acknowledgement, but may not necessarily fit interactions between machines, as explained above.
- **Heavy data:** it is designed for transmitting large quantities of data: the header alone goes from about 200 bytes to over 2KB. Often machines just need to exchange a few bytes.

- **“Pull” model:** it is based on a “pull” model of interaction. A push model may be suitable in a huge variety of situations.
- **No resource discovery:** it does not provide resource discovery.

Some of the drawbacks listed show that the problems are structural, and cannot be solved through HTTP compression.

The feature that cannot instead be discarded is the service-oriented nature of Internet applications expressed through the representational state transfer architecture of the web.

For these reasons CoAP has been developed within the Constrained RESTful Environment (CoRE) framework: CoRE aims at specifying a REST architecture suitable for constrained nodes with limited computing and storage capabilities and constrained networks, while allowing stateless HTTP mapping through proxies or directly at endpoints.

CoAP is a web protocol designed according to the principles of energy efficiency, performance over lossy networks, simplicity in message structure and parsing procedures, lightweight proxying and caching capabilities, security binding to Datagram Transport Layer Security (DTLS) [2] and straightforward HTTP mapping. CoAP can indeed be seen as a subset of REST procedures usually available over HTTP but optimized for classes of applications in constrained environments, with in addition built-in discovery, multicast support and asynchronous message exchange, and it uses UDP with optional reliability features.

CoAP can be thought of as being two-layered in that it comprises on the one hand a messaging layer whose goal is to deal with the unreliable nature of UDP, and on the other hand a Request-Response layer which deals with building a request-response pattern using asynchronous UDP interactions. CoAP abstract layers are shown in Figure 2.1.

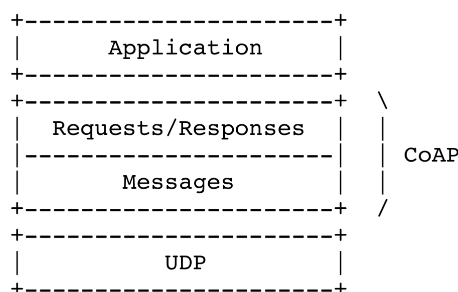


Figure 2.1: CoAP abstract layers

2.5.1 CoAP Message Layer

CoAP interactions are analogous to the HTTP client/server paradigm, but messages are sent asynchronously over the datagram-oriented transport layer protocol UDP.

CoAP [4] defines four types of messages: Confirmable, Non-Confirmable, Acknowledgement, Reset. Logical requests can be carried in Confirmable and Non-Confirmable messages, while logical responses can be carried in these as well as piggy-backed in Acknowledgement messages, as detailed in the next section.

Confirmable messages are acknowledged through an Acknowledgement message, while Confirmable messages are not. Either Non-Confirmable or Confirmable messages can be followed by a Reset message: a Reset message indicates that a specific message was received, but that it cannot be processed because some context is missing. For example this could happen when a node has rebooted and has forgotten some state that would be required in order to interpret the message received.

A identifier present in the CoAP header and called Message ID is used to match messages of Acknowledgement/Reset types, and messages of Non-Confirmable/Confirmable types as well as to detect duplication.

Reliability is essentially achieved by using Confirmable messages as shown in Figure 2.2. Confirmable messages are retransmitted using timeouts with an exponential back-off until an Acknowledgement or Reset is received carrying the same Message-ID.

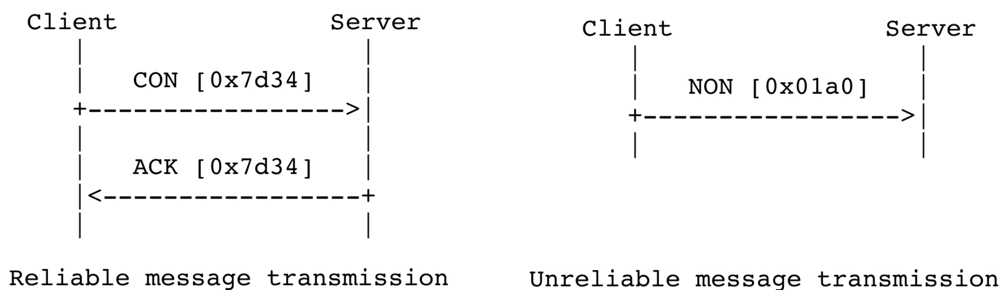
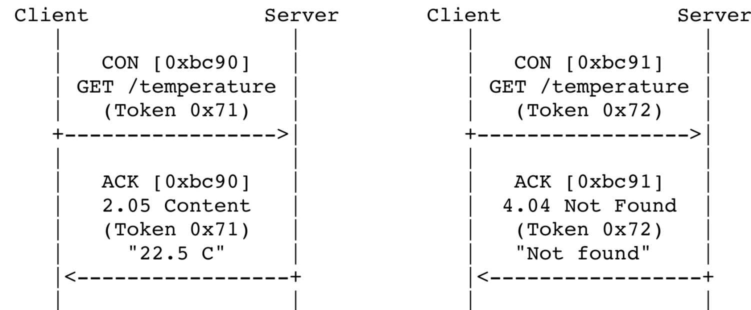


Figure 2.2: Reliable and unreliable message transmission

2.5.2 CoAP Request/Response model

CoAP requests and responses can follow three different paradigms, each of which fits deployment scenarios with diverse characteristics, and which are



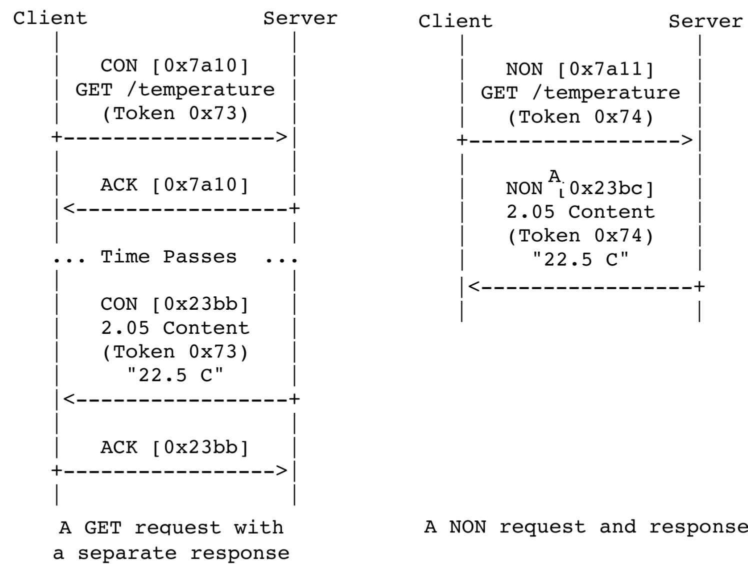


Figure 2.3: The three paradigms for CoAP request/response

depicted in Figure 2.3: a logical response is linked to the corresponding request through a matching *Token* value present in both.

In the first case, the one closest to HTTP, the response to a Confirmable message is piggy-backed in the Acknowledgement message. A server designer may choose this paradigm when for example the requested operation can be executed immediately in an amount of time considered appropriate from an application point of view.

The second case addresses the situation where the server is not able to respond immediately, but sends an Acknowledgement message so that the client knows the request has been received and stops retransmitting while the server elaborates the response: the server sends a Confirmable message with

the actual response to the client at a subsequent moment. This solution may be appropriate when for example a network application requests the execution of a command in a remote node, whose execution time is expected to exceed a given threshold.

In the third case, the client sends a Non-Confirmable request message, and the server returns a Non-Confirmable response: this paradigm can be used when a sensor keeps sending readings with a given frequency to a server, since the correctness of a single request operation does not depend on whether the previous request has been successfully received or not. Since at CoAP level there is no way for a sender to detect whether a Non-Confirmable message was received or not, it may transmit multiple copies, which will be identified as such through the Message ID.

2.5.3 Message Structure

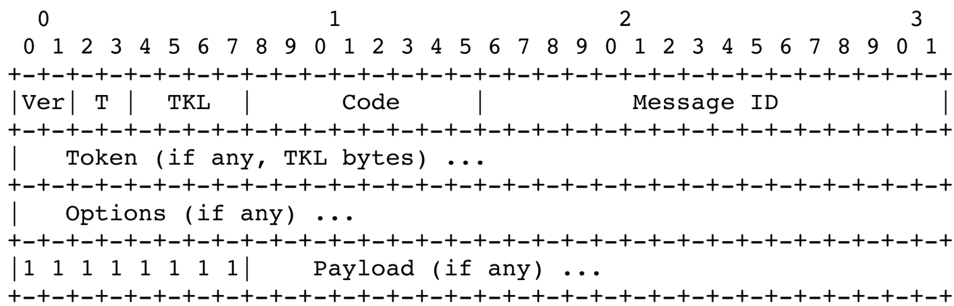


Figure 2.4: CoAP message format

The structure of a CoAP message is depicted in Figure 2.4. The 4 bytes-long header is constituted by the following structures:

- **Ver:** version number, over 2 bits.
- **T:** message type, among Confirmable, Non-Confirmable, Acknowledgement, Reset, over 2-bits
- **TKL:** Length of the Token field (0-8 bytes), over 4 bits
- **Code:** this is divided into class (3 bits) and detail (5 bits). The class sub-field allows distinguishing between a request, a success response, a client error response or a server error response; the detail sub-field instead contains sub-codes specific for each class sub-field value.

- **Message ID:** this identifier is used to detect message duplication and to match messages of Acknowledgement/Reset type to messages of Confirmable/Non-Confirmable type.

The Token value, which can be as long as 8 bytes, is used to correlate requests and responses, and is followed by zero or more Options.

Finally, there can be an empty payload or the Payload Marker followed by the actual payload.

This means that CoAP messages have a header that can be as small as 5 bytes (including the Payload Marker).

When the *Uri-Path* Option is specified, we can estimate adding 20 bytes.

The UDP header adds 8 bytes, which can be compressed at 6LowPAN level, as seen, making a total of around 32 bytes overhead at UDP level.

2.5.4 CoAP Methods

Request methods GET,POST, PUT, DELETE are distinguished through the reserved Code values 0.{01,02,03,04}, where the class sub-field 0 indicates a request.

The URI is instead specified through CoAP Options: a URI is split into host, path and port parts, which are respectively mapped to the Options *Uri-Host*, *Uri-Path*, *Uri-Port*.

A request is then expressed through a method, a URI and possibly a payload content.

A response is expressed through a response code and possibly payload content.

2.5.5 Transmission Reliability

The sender of a Confirmable message retransmits messages at exponentially increasing intervals until it receives an Acknowledgement or a Reset message, or runs out of attempts.

For each Confirmable message a new timeout is set, chosen randomly within a given interval. Each time the timeout is triggered, and the maximum number of retransmissions has not been reached, a new copy is sent and the timeout interval is doubled.

If a Reset is received or if the maximum number of retransmissions is reached, the transmission fails. Otherwise, if an Acknowledgement message is received the transmission is successful.

An indicative time for a retransmission interval using the default parameters, meaning the time from the first to last retransmission of a message, is 45

seconds: the default maximum number of attempts is 4.

$$timeout * rand_factor * \Sigma_{i=1}^{max_retransmit} 2^i = 1 * 1.5 * (2 + 4 + 8 + 16)s = 45s$$

2.5.6 Resource Discovery

Resource discovery is very important in machine-to-machine applications where no human intervention can be assumed and using static interfaces would be detrimental for the flexibility of the system.

CoRE Link Format [3] is used to carry information about resources hosted by a server: it essentially expresses the URI of the given resource plus a number of attributes which may for instance contain a human-readable name for the resource, an application-specific semantic type, or an interface definition to be used in order to interact with the target resource.

Discovery can be performed either in Unicast or Multicast mode. When the IP address of a server is known either *a priori* or though a DNS, Unicast Discovery takes place. Multicast Discovery is instead used when a client intends to locate resources within a scope over which IP multicast is supported.

In both cases a GET over the URI "/.well.known/core" is performed, and resources in a CoRE Link Format payload are returned.

For typical machine-to-machine scenarios where a server may be sleeping a remote resource directory may be used: resources are registered by a server through a POST over "/.well.known/core" and can be discovered by clients making a request to the resource directory lookup interface.

For example, when a sensor is plugged into a CoAP network it may send a GET request to a neighbor sensor which may have a sensing functionality for temperature and one for light, each listed as a <sensor> resource with therefore a "sensor" Interface Description (attribute "if") within a <sensors> collection: the message exchange is detailed in Figure 2.5.

```
REQ: GET /.well-known/core

RES: 2.05 Content
</sensors/temp>;if="sensor",</sensors/light>;if="sensor"
```

Figure 2.5: Discovery message exchange example

2.5.7 CoAP Observer pattern

The protocol described as CoAP Observer Pattern [5] extends the request/response pattern specified by the CoAP core protocol with a mechanism where

a CoAP client retrieves a representation of a resource and the server keeps this representation updated over a period of time. The protocol follows a best effort approach for providing the client with eventual consistency between the state client representation and the actual state of the resource.

According to the well-known observer design pattern, endpoints called "observers" register at a component hosting a resource called "subject", meaning that they are interested in being notified whenever the subject undergoes a change in state: a "subject" is essentially a resource in the namespace of the CoAP server.

Registration

As described in Figure 2.6 the client initially issues a GET message extended with the Observe header option indicating a registration request. The server responds with a regular GET response enclosing the resource representation and specifying the Observe option to confirm the registration request, and successively sends notification enclosing the actual resource state representation using the same token but different Observe option value.

In fact, the Observer pattern is equivalent to a GET request followed by multiple GET responses, or a number of GET request/response pairs where the ones following the first one assume an implicit request issued as soon as the resource undergoes a state change.

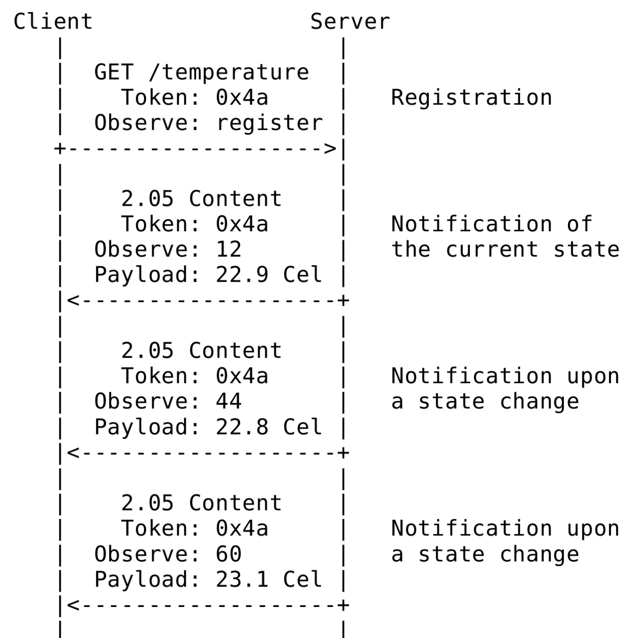


Figure 2.6: Observing a resource in CoAP

After being initialized a registration is assumed valid as long as the server can determine the client's continued interest in the resource, which is deduced by the client's acknowledgement of notifications carried in confirmable messages: the choice whether to use a Confirmable or a Non-Confirmable format for the notification is left to the server, but it is recommended that the server use at least one Confirmable message every 24 hours for the purpose of allowing de-registration in cases when for example the client experiences some malfunctioning that may prevent it de-registering in a different way.

An endpoint can list its resources in discovery registries as links that are useful for observation by using the "obs" attribute.

A server may of course decide to reject a registration request by processing the GET request as usual without including the Observe Option.

Deregistration

Deregistration can either happen explicitly through the Observe option or implicitly when a client rejects a confirmable or non-confirmable notification through a Reset message or when a confirmable notification is not answered with an Acknowledgement.

Consistency model

The goal of the CoAP Observer protocol is to keep the client local representation of the resource as close as possible to the resource's actual state. To do this, the protocol has to deal with the following aspects:

- Latency between the change of the resource state and the receipt of the notification at the client's side.
- Loss of a notification over the network leading the client to assume the old state is valid.
- The Server may erroneously deregister a client due to a repeated Acknowledgement message loss, leading the client to assume the old state representation is still valid.

The aforementioned issues are dealt with as follows:

- The client is issued a notification as soon as the resource changes, following a best-effort approach: according to the principle of eventual-consistency if a resource does not undergo a new change in state all registered observers will eventually have a consistent representation of the resource.

- Notifications are labeled with the maximum amount of time they should be considered valid unless a fresher notification is received, after which the client should issue a new GET request before using the resource representation value: this mechanism avoids the client being indefinitely out of sync with the server. This feature is supported by CoAP header's Max-Age option.

In order to allow the client to deduce the relative ordering of notifications the server should set the value of the Observe Option to the 24 least-significant bits of a strictly increasing sequence number which can either be the timestamp of a local clock or a variable per resource that is incremented every time the resource undergoes a state change.

CoAP recommends not to reset the value of the Observe Option more often than once every 256 seconds, which is well beyond the highest known design objective of around 1kHz in terms of notifications per second.

Reducing the number of notifications

There may be situations where clients may not be interested in receiving a notification upon every state change of a notification. Instead of defining procedures to provide *ad-hoc* notification patterns CoAP relies on the RESTful paradigm and suggests that the server exposes additional resources according to the desired semantic.

As an example a temperature sensor may expose two resources such as:

- `<coap://server/temperature>` which change its state every time a sensor reading is performed.
- `<coap://server/temperature/felt>` which has two possible states, WARM and COLD, and undergoes a state change when the temperature crosses two pre-configured threshold values.

Messaging optimizations

Since the CoAP Observe Protocol is an extension of the GET request/response pattern it supports the use of the ETag Option. A client may specify in the GET request a set of entity tags (hashed values of representations) any of which may be confirmed by the server through a 2.03 (Valid) response rather than a 2.05 (Content) notification carrying the entire resource representation.

This mechanism may be useful when the resource has a quite extended representation which can nevertheless assume a pre-defined set of states.

In order to perform request aggregation the use of intermediaries is also allowed as for a regular CoAP request-response: a client may address a GET request to an intermediary node which will act as a client towards the actual server.

2.5.8 Blockwise Transfer

In a wide number of IoT communications over constrained networks we can reasonably expect payloads to be relatively small: for example, when sensors handle temperature value, light switches or toll systems, data handled can be limited to a few bytes.

In other situations still within the IoT domain payloads may be significantly larger: for example in image processing systems even when data pre-processing and aggregation is performed, data transfered may be in the order of hundreds or thousands of kilo-bytes.

When data this size is transferred a number of difficulties may develop. With IPv4 even if datagram transport protocols of choice such as UDP or DTLS support message sizes larger than the nominal one of 64 KB through fragmentation the performance of the procedure over a constrained network may not be satisfactory. Since IPv6 instead never fragments packets, the tendency is that of using application level mechanisms that are aware of packet size constraints.

In constrained networks, datagram size is limited by the maximum datagram size *i.e.* 64 KB for UDP, by IPv6 MTU of 1280 KB and by the adaptation layer fragmentation mechanism which in the case of 6LoWPAN intervenes when the packet size is over 60 - 80 Bytes.

At the same time, while keeping the burden of implementing fragmentation mechanisms at the endpoints it should be remembered that nodes are constrained and that creation of communication state at the server's side should be avoided whenever possible: in case of GET requests this objective can be achieved, while for POST/PUT requests it is impossible to fully avoid creating a conversation state if the creation or replacement of resources is to be atomic.

CoAP Blockwise transfer [6] in turn specifies two Options and their use for block-wise transfers.

The main features of the Blockwise transfer mechanism are:

- Transfers of payloads that are larger than what can be accommodated at the link layer.
- No conversation state is created at the IP adaptation layer.
- Each block is explicitly requested and acknowledged: minimal conversation state is necessary at CoAP endpoints.
- Endpoints negotiate the dimension of the blocks to be transferred and the minimum among proposals is chosen.

- Random access to power-of-two-sized blocks can be performed by referring a GET request to a given block.

CoAP Blockwise Options

CoAP Blockwise transfer mechanism makse use of two Options: Block1 Option is used to handle payload fragmentation in requests, while Block2 Option is used to handle payload fragmentation in responses.

When Block1 and Block2 Options are present in request and response messages respectively they are said to be employed with a "descriptive use" since they describe the actual payload structure as part of the resource transferred. When Block2 is used in a request or Block1 is used in a response we can talk instead of a "control use" since they are part of a negotiation process regarding the block size that will be actually used.

Block Option structure

CoAP Block option is expressed as a zero to three byte long integer with the following parts:

- SZX: the three least significant bits express the block size through the relation 2^{4+SZX}.
- M: one bit that indicates whether more blocks are following (M set) or not (M unset).
- NUM: the remaining indicates the sequence number of the current block with respect to the overall sequence of blocks expressing the resource.

The use of Block1 and Block2 in "descriptive usage" is straightforward. When the Block2 Option is used in a request (e.g. GET) NUM refers to the block number that is being requested to be transmitted, M has no function and SZX suggests a block size to be used in the response. When Block1 is used in a response (e.g. PUT or POST) the NUM refers to the block that is being acknowledged; if the M bit was set in the request the server can choose whether to act upon each requested block or not: if the M bit is set in the response it means that the response does not carry the final response code to the request which will be elaborated atomically once all the request packets have been received by the server, while if the M bit is unset it means that the response carries the final response code to the request it refers to and the response code is set to 2.31 (Continue). The SZX field in this instance indicates the largest block size accepted by the server.

In general when a Block Option is used in "control usage" it is expected to express the maximum block size the endpoint is willing to accept, meaning that

block sizes inferior to that one are equally acceptable. The other endpoint will answer confirming the block size if its preferred block size is larger or proposing its own if not.

At any time a server can answer with the error code 4.13 if for example it does not have sufficient resources to support a Blockwise transfer of a large request it intends to execute atomically.

Blockwise transfer consistency

A Blockwise transfer is characterized by a single token, while a different messageID which is used for each GET request/response pair.

When resources have a constant representation, no consistency issues arise due to Blockwise transfer. In case of dynamically changing resources instead the decision on whether operations requested by clients should be executed atomically or not is left to the server. Nevertheless when resources are dynamic and may change during the execution of a Blockwise GET request the server can use the ETag option which can be used by the client to understand if all blocks refer to the same resource state or if some blocks are to be retransmitted.

2.6 The ETSI M2M Communication Paradigm

Within the IoT vision and thanks to the IoT stack described so far, applications are expected to rely on a RESTful architecture which has been designed, in terms of underlying protocols, in such a way that it can be successfully hosted by devices with limited resources.

In order to increase the level of interoperability of IoT systems, RESTful resources and access methods can be defined in a cross-service manner.

RESTful resources are used to identify and locate date items taking into account the following requirements:

- Existence of a data model where a single data item is expected to be associated with a specific entity, which may be either a sensor, a group of sensors, a gateway, a generic network node or even an abstract entity, which still should be identifiable in the network.
- It should be possible either to access data explicitly as is done through a web browser, or to enable automatic propagation through the application framework domain.

Operations to be executed on resources are designed taking into account the following requirements:

- Sensor devices may be powered off for long periods of time due to energy saving policies.
- Whenever possible, control procedures such as access methods, remote management procedures and access policies should be abstracted according to the RESTful paradigm.

A number of solutions have been proposed in order to address the requirements above. A particularly strong candidate is the ETSI M2M protocol since it is elaborated by a wide set of industry and research players within the IoT ecosystem. For this reason it can be expected not only to benefit from strong cross-domain support but also to offer a high degree of interoperability as well as flexibility since it constitutes the output of a working group formed by experts from a number of diverse ICT areas.

The ETSI M2M System addresses the requirements above and consists in an horizontal framework enabling applications to use data independently of their location and of the protocols needed to access them. It consists of a distributed system composed of M2M Applications, a M2M framework called Service Capability Layer (SCL), and a Communication Layer.

M2M Applications can reside in a Device Application (DA), a Gateway Application (GA) or in a Network Application (NA). According to the ETSI definition a Device (D or D') is a piece of equipment that may collect a set of actuators and sensors with embedded electronic computing and communication capability, while a Gateway aims at translating and transferring information between two or more communicating entities, or at performing some routing and multiplexing function between the communicating entities [11]. It may also abstract functionalities of the related sensor devices which cannot be exposed directly by sensor devices themselves. A Network Node is instead a generic communication entity which does not fall into the two aforementioned categories.

The high level architecture of an M2M system distinguishes within the distributed system between a Network Domain and a Device and Gateway Domain, as depicted in Figure 2.7.

The M2M Service Capabilities Layer enables communication between M2M Applications: it is an application-level framework that abstracts the distributed system as a set of resources and operations to be executed on them. The Service Capabilities Layer is present in the Network Domain as Network SCL (NSCL), in M2M devices as Device SCL (DSCL) and in M2M gateways such as Gateway SCL (GSCL). While a Gateway and a Network Node always include a Service Capability Layer (SCL), a Device may (D) or may not (D') host a local SCL and instead access the M2M framework through a GSCL.

The SCL is a distributed system itself: it is composed of SCL instances, which can expose four types of interfaces: dIa, mId, mIa and mIm.

Figure 2.8 shows that while dIa constitutes the interface between a GA and a local GSCL or between a DA and a local DSCL, mId is the interface between a GSCL or DSCL and a NSCL, mIa is the interface between a NA and a local NSCL, and mIm is the interface between NSCLs. Figure 2.8 also presents the optional IP (Interworking Proxy) capabilities DIP, GIP and NIP, providing interworking with non ETSI compliant devices.

The reason for these four interfaces is to distinguish in the M2M system a number of actors with associated roles: such roles are defined by the interactions they have with the other system components which in turn can be summarized in a well defined functional interface. It can also be seen as a consequence of the strong centralization that characterized the first editions of the protocol specification (mId interface fully enabling a layered P2P structure within NSCL for example has been introduced only in Q4 2013).

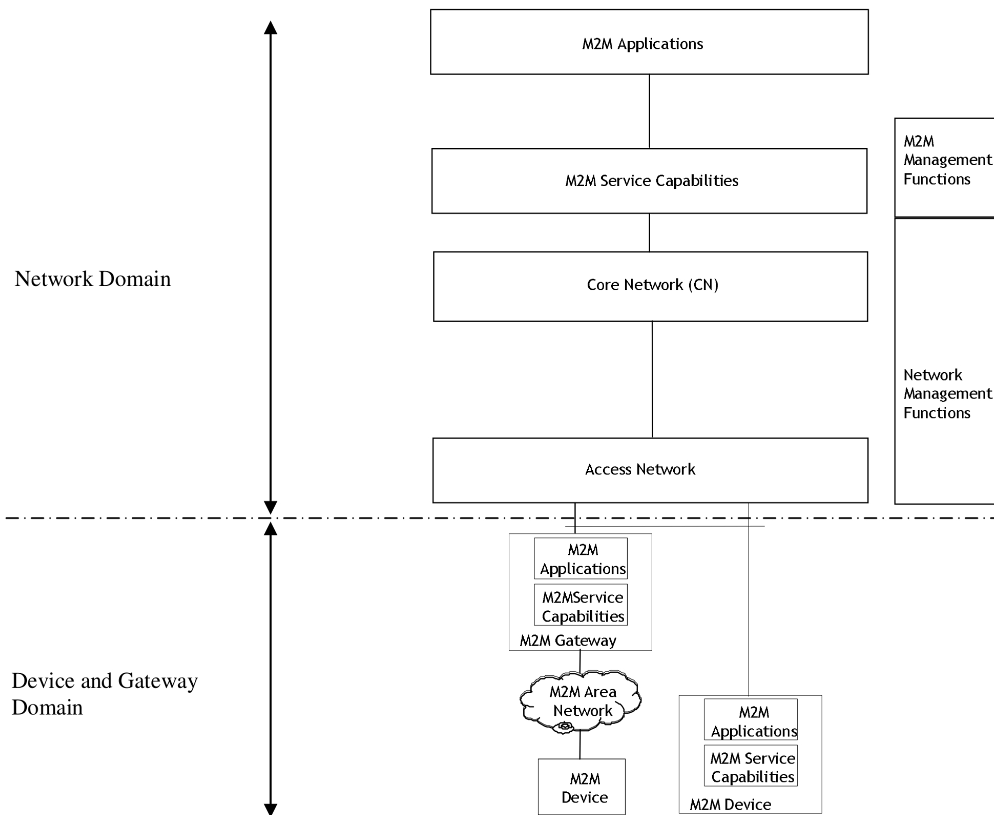


Figure 2.7: High level architecture for M2M

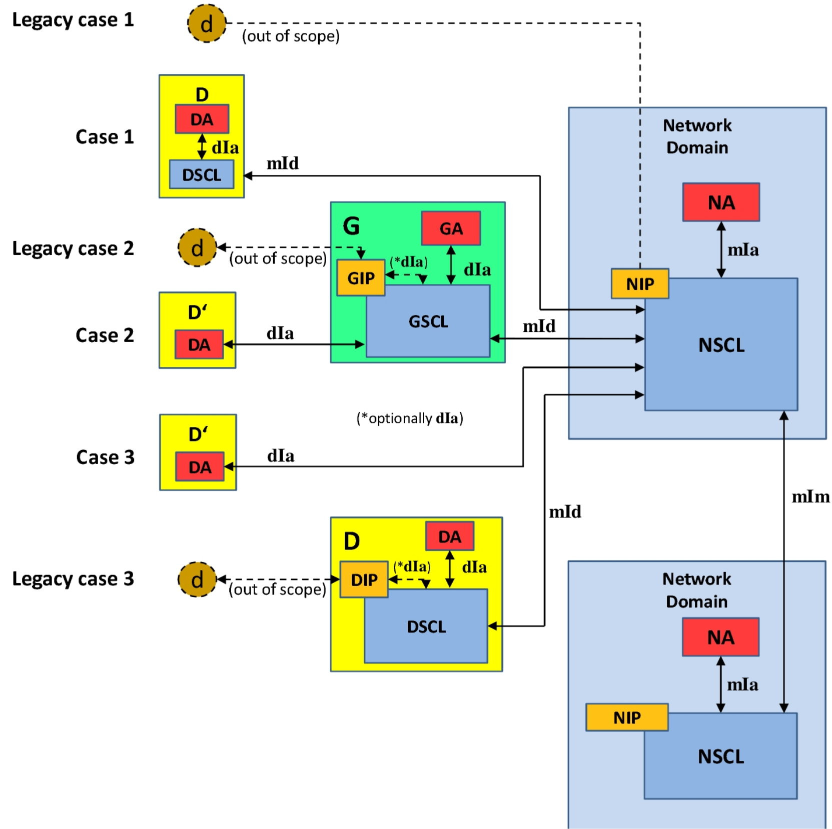


Figure 2.8: Deployment scenarios and reference points

For example in a Intelligent Transport System (ITS) a set of Devices may be constituted by a number of parking lot availability sensors; the Gateway may be a device with stable connectivity and continuous functioning due to a permanent power supply.

A NA could be an application hosted on a user's mobile phone which displays the available parking slots in the surrounding area; or a PC workstation where a software generating analytics related to city traffic is running.

These two NAs may have two different SCLs as Local SCL. A typical reason for this could be an end service constituted by two components each provided by a different service provider: e.g. a management company handling control room procedures and a company responsible for the interface with end-users.

The two NSCLs may communicate through mId for example in case of emergency situations which are detected by the first and rendered to the users by the second; moreover they both access the same Gateway from which they retrieve the data but with different policies corresponding to different access

rights.

2.6.1 M2M Resources

Each SCL instance exposes a set of REST resources hierarchically organized. In particular, it includes:

- <*scl*> resources representing remote SCLs.
- <*application*> resources representing remote M2M Applications
- <*container*> resources representing data exchange buffers
- <*accessRight*> resources representing access rights
- <*group*> resources representing a group of resources of the same or mixed type
- <*subscription*> resources representing a subscription to a resource
- <*mgmtObj*> resources representing management data corresponding to individual M2M remote management functions
- <*mgmtCmd*> resources representing non-RESTful management commands allowing M2M Applications to execute or cancel a non-RESTful Remote Procedure Call on a remote entity
- <*attachedDevices*> resources representing management information of D' or D devices attached to a M2M Gateway
- <*notificationChannel*> resources enabling a method for a non-server capable client to retrieve asynchronous notifications

2.6.2 M2M Operations

Following the RESTful architecture, the four basic methods or "verbs" that can be applied to resources are CREATE, DELETE, UPDATE, RETRIEVE. Through a combination of these basic operation, the following higher-level operations are possible:

- **SCL registration:** a SCL registers with another SCL in order to be able to interact with it, by creating an <scl> resource in the collection `<sclBase>/<scls>` resource of the registered-to SCLs representing the Issuer SCL. At the same time, an <scl> resource representing the registered-to SCL is created in the Issuer SCL.

- **Application registration:** every M2M Application will register itself to the Local SCL, meaning the SCL it uses to access the M2M System: as above, registration is carried out through the creation of an <application> resource in the collection `<sclBase>/local_scl/` `<applications>`.

- **Subscription to resources:** an M2M Application or an SCL may act as a subscriber requesting to be notified by the Hosting-SCL (the SCL hosting the subscribed-to resources) when resources are modified. The subscription is a resource itself, and it is created and customized by modifying the resource itself or its parts.

- **Execute Remote Procedures:** an Issuer may request to execute a specific management command represented by a <mgmtObj> on a remote entity through the UPDATE verb. Alternatively, a RPC call can be emulated by addressing a <mgmtCmd> and specifying the appropriate command parameters.

- **Resources Announcement:** an M2M Application or an SCL may decide to announce a resource to other SCLs through the appropriate child element <announceTo> of the resource. This element is interpreted as the list of SCLs that the SCL will try to announce to on behalf of the requestor. If this element is not provided, the Local SCL will decide where the resource will be announced.

As an example we can think of a smart city management system where a Device (D') is a taxi On Board Unit (OBU), a Gateways is a Road Side Unit (RSU) and Network Nodes are either users' mobile phones or a PC workstation where a Managing Application may run. Whenever a taxi OBU is switched on it will discover neighboring nodes including the nearest Gateway. It will then register its application in the GSCL by creating an <application> element and create a <container> resource within the `<sclBase>/scl/` `<scl>/containers` collection where it will store measured data (for example geolocation data). It will also register the management function it exposes by creating a <mgmtObj> resource in the path `<sclBase>/scls/<scl>` `/attachedDevices/<attachedDevice>/mgmtObjs`: each device has associated its own management commands.

A Network Application such as the Managing Application whose Local NSCL is subscribed to the Gateway SCL may be notified about the creation of the application resource and may subscribe to the newly created <container> resource in order to receive notification whenever the data representation undergoes a state change.

It is agreed to express an URL as composed by generic names (defined by the protocol specification) enclosed by brackets, which can be substituted by concrete instances to form the actual URI. In the example the corresponding actual URI may be "`coap://321.128.10.1:5683/gw01/scls/my_scl25/attachedDevices/parking_sensor1/mgmtObjs/obj1`".

2.6.3 M2M remote execution and data distribution

In terms of problem frames [7], a M2M System addresses the following:

- **Required Behavior:** the causal domain is constituted by a portion of the external world, which can be for example a gate, a dam water level, a car brake system, a vending machine.
- **Commanded Behavior:** the causal domain is constituted by a portion of the external world, and the operator biddable domain is the human user: for example, a user may want to issue a remote command to the heating system of his house: still, the procedure should be monitored and potentially corrected or inhibited to avoid, for example, the boiler heating system becoming a hazard.
- **Information Display:** the causal domain is constituted by a portion of the external world, while the causal display domain is a physical network terminal. Typical examples are measurements system for physical quantities such as temperature and pressure, number of vehicles per hour, parking lot availability.

An M2M System is capable of addressing these three problem frames and offers a number of built-in mitigation procedures to adapt to a set of specific domain configurations.

The architecture of an M2M System is constituted by a distributed system within the Network Domain, and a number of either distributed or remote systems within the Device and Gateway Domains.

Required Behavior and Commanded Behavior: both these patterns are enabled by the use of remotely executed commands, namely:

- **mgmtObj:** in order to execute a management command on a remote entity an Issuer executes the UPDATE method of a <mgmtObj> resource on the Hosting SCL. Subsequently the Issuer can retrieve the execution status of a command by executing the RETRIEVE method of the <mgmtObj> resource, as in Figure 2.10.

- **mgmtCmd:** non-RESTful management commands or a set of Remote Procedure Calls (e.g. Factory Reset, Reboot, Upload, Download, ScheduleDownload, ChangeDUState...) can be executed on a remote entity using <mgmtCmd> resources. Their use is analogous to that of <mgmtObj>: the Issuer requests the execution of a <mgmtCmd> in the Hosting SCL through the UPDATE method, and after the corresponding procedures have been carried out on the remote entity, a local <execInstance> representing a execution instance is created on the Hosting SCL, and its URI is returned to the Issuer. The Issuer can then use the <execInstance> to retrieve the execution results, as described in Figure 2.9.

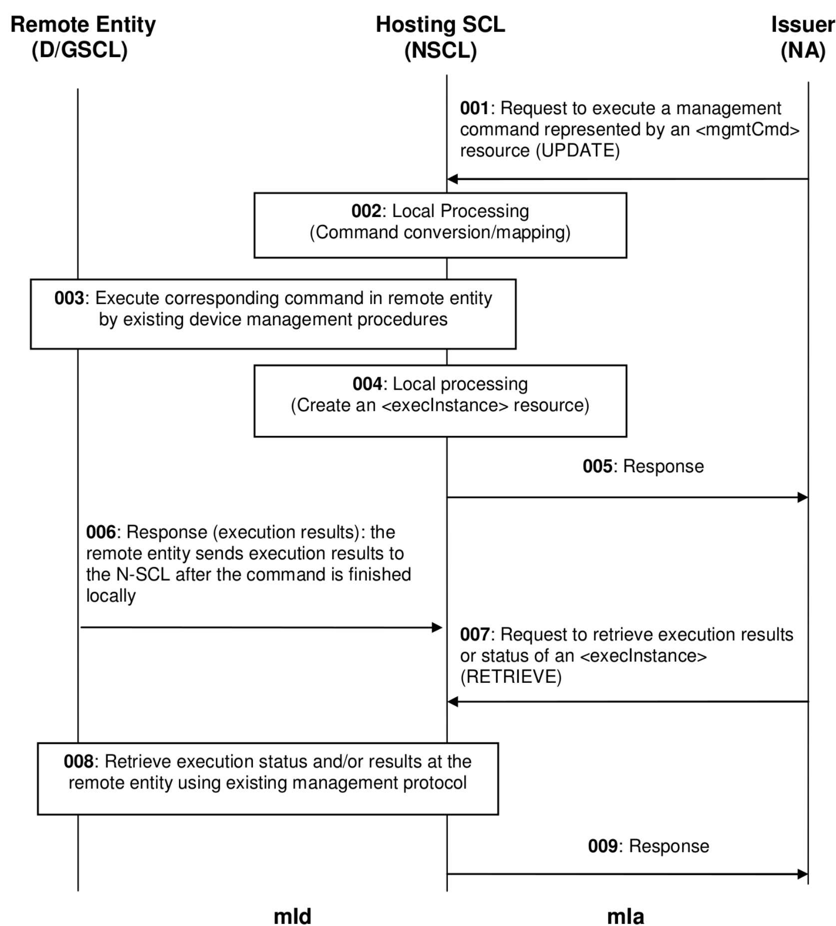


Figure 2.9: Execution of a mgmtCmd resource

Information Display:

For simplicity, we can put ourselves in the situation where a sensor device D' hosting an M2M Application DA is connected to a GSCL (Case 2 in Figure 2.8).

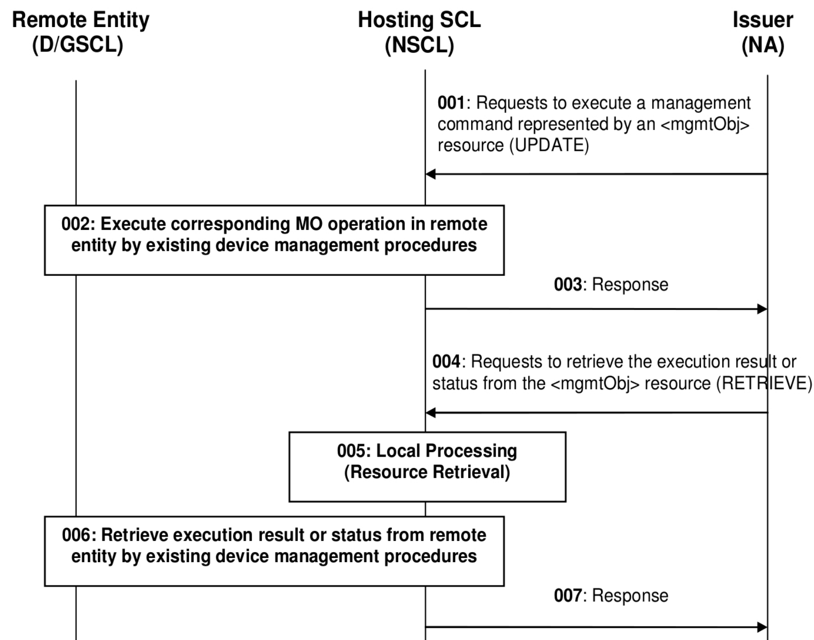


Figure 2.10: Execution of a mgmtObj resource

In this case, when the sensor is first connected to the network it will create in the GSCL an <application> resource, with an <application>/containers/ <container> resource representing the monitored value (e.g. the status of a monitored parking slot which can be free/occupied).

Each time there is a status change in the resource, the DA will create a new <contentInstance> under <application>/containers/<container>/contentInstances in the GSCL.

In the segment GSCL–NSCL, where NSCL is the SCL that is Local to the NA available to the user, three options are possible to address this problem frame:

- **Subscription:** in order to subscribe to a resource an Issuer can request the creation of a <subscription> element referred to the subscribed-to resource. Upon every resource modification, the Issuer will receive a notification to the contact-URI specified in the subscription request.

- **Subscription with long polling:** this offers a method for clients that are not server-capable to receive semi-asynchronous notifications regarding a subscribed-to resource. This mechanism is based on a <notification-Channel> resource: as depicted in Figure 2.12 an Issuer willing to subscribe to a resources of a remote SCL (Subscriber-to SCL) may use a

SCL (Hosting SCL) as a notification server.

The Issuer creates a <notificationChannel> in what becomes the Hosting SCL, and the newly created resource URI which is used as contact-URI in the subscription, and the long-polling URI which serves as the endpoint of a request-response operation are returned.

The Issuer can, in either order, send a long polling request and create a subscription in the Hosting SCL relative to a resource of the Subscribed-to SCL, using as callback URI the URI of the <notificationChannel> resource.

- **Announcement:** an announced resource is a resource whose creation and modification information is propagated to announced-to SCLs, meaning that for the announced-to SCL to be notified no explicit subscription action is required. The Issuer which creates an announcement enabled resource on a Hosting SCL delegated to the Hosting SCL the creation of a resource on the various announced-to SCLs, as described in Figure 2.11. The *003:Response* message can also be sent after the announced-to SCL *007:Response*, in which case a delayed response paradigm is adopted. As an example, the application hosted on a sensor device registering temperature values, as it is plugged in the M2M network, may create an <application> resource on the local SCL and a <container> resource which will contain the temperature measured values. If announcements are used, the <container> element may have as a child element an <announceTo> element specifying a list of SCLs. Upon <container> resource creation, the local SCL creates a <container> resource on each of the SCLs present in the list, and each time the sensor device will create a new temperature instance value after a measurement, the local SCL will update the corresponding value on each of those SCLs.

Continuing the smart city example, a NA hosted on the user's mobile phone discovers the local GSCL and subscribes to the <containers> collection.

The taxi OBU will periodically create a new <contentInstance> within the <containers> collection specifying its current location. Upon creation of a new <contentInstance> the user device's application will be notified and will display the taxi location on the screen map: the mobile application will then give the user the possibility to e.g. tap the icon and book the taxi.

When an application intends to access a resource which is not available in the local SCL, the request needs to be routed to the hosting SCL. It may happen that a single SCL receives a large amount of remote access requests: in

order to fulfill such demand in an efficient way, the SCL may perform requests aggregation in a store-and-forward manner.

Aggregation of access requests to remote resources can be controlled through two parameters: Tolerable Request Processing Delay Time (TRPDT) and Request Category (RCAT). Such parameters are applicable to requests issued by a D/GA or DSCL to access resources on a NSCL and to requests issued by NA on a NSCL to access resources hosted on a D/GSCL.

The local SCL that receives the request can delay the forwarding of the request to the Hosting SCL according to SCL-specific policies but within the upper bound defined by the TRPDT value specified in the request itself.

Request Categories instead control the forwarding policies of requests to access remotely hosted resources: if a connection cannot be established using the appropriate access network for a specific combination of RCAT value, issuer and destination, the local SCL can block the forwarding of the request.

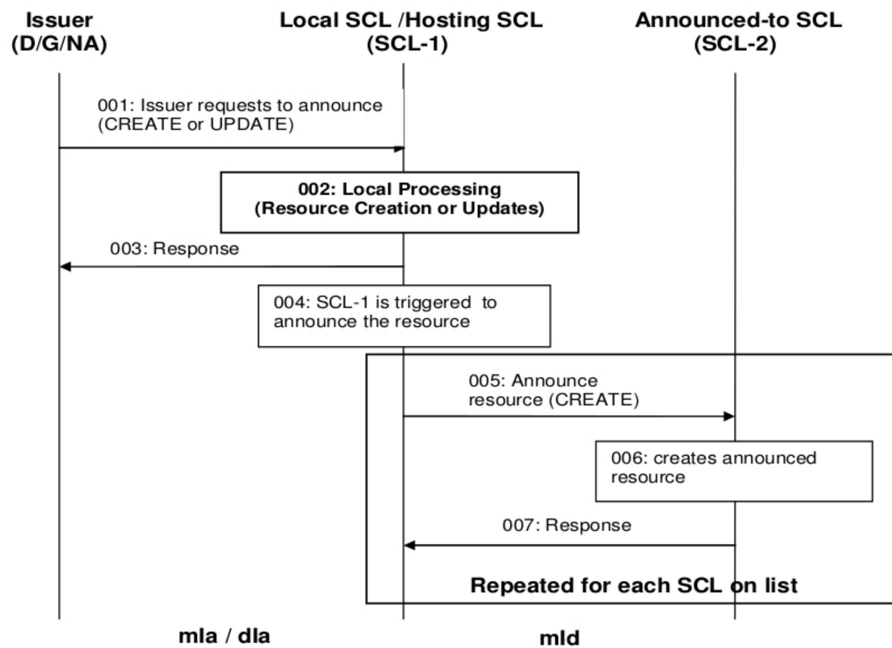


Figure 2.11: Resource Announcement

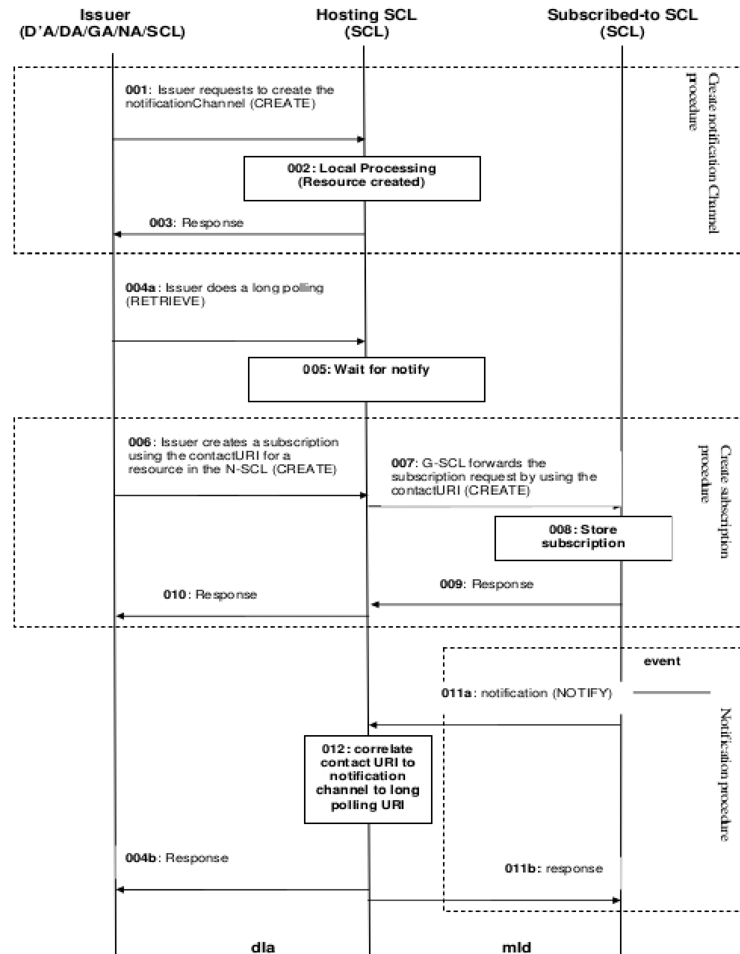


Figure 2.12: Long polling and notifications

2.7 Data Serialization Formats

Three data serialization formats are recommended[1] by ETSI M2M: Extensible Markup Language (XML), JavaScript Object Notation (JSON) [2] and Efficient XML Interchange (EXI). In our work on embedded systems only the JSON and EXI options have been considered. JSON and EXI are mentioned in this section, and a high level description of the EXI serialization mechanisms is given.

[1] Annex D, section 1, [12]
[2] http://tools.ietf.org/html/rfc7159 Last Access: June 12, 2014.

2.7.1 JavaScript Object Notation (JSON)

JavaScript Object Notation (JSON) is a data format widely used the interactions between servers and web-applications. Due to its simplicity and the fact that it is also human-readable, it is supported by a large variety of platforms.

2.7.2 Efficient XML Interchange (EXI)

EXI was developed by the Efficient XML Interchange working group within the World Wide Web Consortium (W3C), and was adopted as a Recommendation by the W3C in 2011.

The goal of EXI is to define a data format that satisfies the requirements of constrained networks, and that at the same time allows them to be interoperable with the legacy data formats XML and JSON. Its definition was guided by five design principles, namely generality, minimality, efficiency, flexibility and interoperability.

This section explains the main principles of the EXI format.

Even if EXI can use schema information to improve compactness and efficiency, schema documents do not have to be accurate or complete.

An EXI encoded data structure is called EXI Stream. It is composed of two parts: *EXI header* and *EXI body*.

EXI Header

The EXI header conveys version information, and can be used to communicate encoding properties that are needed to decode the EXI body: if options are omitted, the decoder should have access to the options used for encoding through out-of-band mechanisms, in order to decode the EXI Stream properly.

The minimal dimension of the EXI header is one byte, allowing to obtain a high efficiency for small data instances.

Options are formally described using an XML schema and are encoded using EXI as well; the most relevant options for the present work are:

- *alignment:* refers to the alignment of event codes and content. It can assume the values *bit-packed*, *byte-aligned* or *pre-compression*. The *pre-compression* option simply prepares the stream for a later compression, applying all the steps except the "deflate" algorithm [24].
- *strict:* increases compactness by ensuring a strict interpretation of the schema, meaning that elements and types are restricted to the ones declared in the schema.
- *preserve:* allows to retain or discard components such as comments, prefixes, XML processing instructions and lexical values.

- *schamId:* identifies schema information used to process the EXI body. *XML Schema* types are built-in and are always available.

EXI Body

The body of an EXI Stream is constituted by a sequence of events. For example an attribute named *foo* can be encoded through the event AT("foo"), and an element named "bar" as the couple of events SE("bar") EE.

The main event codes are reported in Table 2.2:

EXI Event Type	Grammar Notation
Start Document	SD
End Document	ED
Start Element	SE(*qname*) SE(*uri*:*) SE(*)
End Element	EE
Attribute	AT(*qname*) AT(*uri*:*) AT(*)
Characters	CH
Namespace Declaration	NS
Comment	CM

Table 2.2: EXI Event types

A single event type can convey a variable amount of information: SE("*qname*") for example represents the start element of an element with a precise name, while SE(*) indicates the start element of a generic element, whose name will be encoded separately and will constitute the *content* of the event.

Within an EXI Stream, an event is represented by an event code and an event content, if present. The event code is a sequence of one to three non-negative integers distinguishing the possible events that can occur at a given point of the EXI Stream.

Shorter events codes are used to represent events that are more likely to occur. When Schema-Enabled encoding is used, schema-derived events of the type SE ("*qname*") are encoded with fewer bits than generic events such as SE(*), which can even not be present if the *strict* option is set.

In order to represent content values, EXI uses built-in datatypes, as well as datatypes defined through external schema structures. If the *preserve* option is set for lexical values, all datatypes are represented as strings. Among the wide set of possible types, the following two are reported as example:

- *String:* string datatype representation consists of a length prefixed sequence of characters; length is represented as an unsigned integer.
- *Unsigned integer:* an unsigned integer is represented through a sequence terminated by a byte with its most significant bit set to 0, after zero or more octets with the most significant bit set to 1. The value of the unsigned integer is stored in the least significant 7 bits of each octet.

EXI Grammars

EXI encoding uses a set of grammar-derived structures, each representing the productions of a grammar. They are referred to as *grammars* in the EXI Specification; we will refer to them using the same terminology in this exposition. EXI uses the information provided through the XML schema structures available to build a *grammar* associated to each element and type defined. Such structures are used to determine which events are most likely to occur at any given point in an EXI Stream, mapping the most likely events to a lower entropy set of values, which are encoded using fewer bits.

The types of grammar structures that can be used are the ones provided explicitly through schema information, and the ones derived from built in schemas such as the XML Schema. Moreover EXI describes a mechanism by which built-in grammars can be dynamically extended, using information from the actual instance being encoded. For example, an $< accessRightID >$ element which occurs more than once during the encoding of a EXI Stream in Schema-Less mode (i.e. when no external schema documents are provided), will have its element name encoded only once. The first occurrence will be handled through the SE(*) event, its element name will be stored in a local table, and subsequently its element name will be referred to through an identifier.

Schema-informed grammars should be preferred to built-in grammars whenever sharing a grammar set (i.e. an XML schema) among the communicating endpoints is feasible: since value items such as element names and attribute names do not need to be encoded but simply referred to, processing speed increases and compactness is improved.

Encoding Example

As an example, we consider a binary data structure that corresponds to an XML $< application >$ resource such as:

```
<?xml version="1.0" encoding="UTF-8"?>
<p0:application xmlns:p0="http://uri.etsi.org/m2m">
   <p0:accessRightID>accessRight</p0:accessRightID>
   <p0:groupsReference>
    /gsclBase/applications/app/groups
   </p0:groupsReference>
```

```
</p0:application>
```

We assume performing *strict*, Schema-Enabled encoding, setting all *preserve* options to false and adopting *bit-packed* alignment. We also assume not compressing the EXI Stream, but simply serializing the data structure associated to the $<$ *application* $>$ resource in terms of EXI events. The body of the EXI Stream is composed of the following events:

```
SD
SE("application")
SE("accessRightID")
CH("accessRight")
EE
SE("groupsReference")
CH("/gsclBase/applications/app/groups")
EE
EE
ED
```

The EXI Stream is built as described below. The encoding procedure makes use of a stack of grammar-derived structures, where each grammar structure can be expressed as a state machine. During the encoding process, the current state can be represented through a set of states, one for each of the grammar structures currently present in what the EXI Specification refers to as the "grammar stack".

The notation A.B indicates the position within the stream after the serialization of the current element, where A is the byte number and B is the bit number.

1. EXI Header: 4.0 (4 bytes)
2. Header Distinguishing bits: 4.2 (2 bits)
3. Presence Bit for EXI Options: 4.3 (1 bit)
4. EXI Version: 4.4 (5 bits)
5. EXI Options: 5.3 (7 bits)
6. SD: 5.3 (0 bits). The *DocumentContent* grammar is pushed on top of the current stack of grammars, which is constituted by the *Document* grammar only, comprising a single state. The state relatively to the state machine associated with the topmost grammar is *DocumentContentStart*.

7. SE("application") code (9): 6.1 (6 bits). The number of productions available in the current state equals the number of elements defined by the schema, and the production for the *application* element is the 9th. The state becomes *DocumentContentEnd*, and the grammar *ApplicationGrammar* is pushed onto the grammar stack: the current state within this grammar is *StartTagApplication*.

8. SE("accessRightID") code (2): 6.5 (4 bits). The current grammar *ApplicationGrammar* has 15 productions. The second corresponds to the *accessRightID* element; the current state within the *ApplicationGrammar* becomes *AccessRightIDElement*, and the corresponding grammar *AccessRightIDGrammar* is pushed onto the grammar stack, where the current state becomes *StartTagAccessRightID*. The complete list of events available for *ApplicationGrammar* and corresponding to the various grammar productions is reported in Table 2.3, and has been obtained from the < *application* > element schema.

9. CH("accessRight"): 18.5 (0 bits CH event, 11 bytes string, 1 byte length). *StartTagAccessRightID* has only one transition, corresponding to the event CH. The event is encoded with 0 bits, and the state undergoes the transition toward the state *ElementAccessRightID*.

10. EE code (0): 18.5 (0 bytes) The state *ElementAccessRightID* has only one transition EE, encoded with 0 bits. The event is encoded and the current grammar *AccessRightIDGrammar* is popped from the grammar stack: the current state within the *ApplicationGrammar* is *AccessRightIDElement*.

11. SE("groupsReference") code (8): 19.1 (4 bits) is the eighth transition possible from the current state (it can be derived from Table 2.3 considering the fact that < *application* > is structured as a *sequence*). Again, the current state becomes *GroupsReferenceElement*; the corresponding grammar *GroupsReferenceGrammar* is pushed onto the stack where the state within the state machine of the current grammar becomes *StartTagGroupsReference*.

12. CH("/gsclBase/applications/app/groups"): 53.1 (0 bits CH event, 33 bytes string, 1 byte length). From *StartTagGroupsReference* the state undergoes the (only) transition towards *ElementGroupsReference*.

13. EE code (0): 53.1 (0 bytes). The grammar *GroupsReferenceGrammar* is popped from the stack and the current state becomes *GroupsReferenceElement*.

14. EE code (0) 53.3 (2 bits). The grammar *ApplicationGrammar* is popped from the stack of grammars and the current state becomes *Document-ContentEnd*.

15. ED code (0) 53.3 (0 bits). The grammar *DocumentContent* is popped from the stack of grammars; the *Document* grammar is also popped and the EXI Stream is closed.

Event	Event Code
AT("appId")	0
SE("expirationTime")	1
SE("accessRightID")	2
SE("searchStrings")	3
SE("creationTime")	4
SE("lastModifiedTime")	5
SE("announceTo")	6
SE("aPoC")	7
SE("aPoCPaths")	8
SE("locRequestor")	9
SE("containersReference")	10
SE("groupsReference")	11
SE("accessRightsReference")	12
SE("subscriptionsReference")	13
SE("notificationChannelsReference")	14
EE	15

Table 2.3: Event codes example: *ApplicationGrammar* grammar. Events are encoded using 4 bits.

Chapter 3

Related Work

The present chapter aims at showing how this work relates to research initiatives that approach themes analogous to the one we have been focusing on.

The results of this work are presented in Chapter 5, and they are compared to the research activities illustrated in Chapter 6.

3.1 EXIP based implementations

The Embeddable EXI Processor (EXIP) has been developed by Rumen Kyusakov [23] and is the work with the largest impact on our work, since we have adapted EXIP libraries to the embedded environment: this specific implementation was chosen because it is open-source, and even if performance measurements were not available in literature it is explicitly designed to be energy and memory efficient.

The most apparent characteristic of EXIP is what it is not[1]: it is not a tool for converting XML to W3C EXI and vice-versa. The reason for this is the need to avoid XML parsing, when the real goal is the translation between a binary structure and an EXI Stream.

The issue of XML-like communications within constrained devices has been tackled from a different point of view than the one from which our work originates: the motivation of the work as presented in Kyusakov's paper is that of enabling SOAP services on embedded devices providing support to manufacturing enterprises. The effort of facing the constant demands to change processes and products inevitably brings a huge complexity overhead.

[1] `http://exip.sourceforge.net/exip-user-guide.pdf` Last Access: June 12, 2014.

The first contribution we can relate to for a direct comparison is the work done by Domenico Caputo et al. [14], which depicts an implementation similar to the one we have realized. EXIP libraries[2] developed within the EISLAB research activities in the Lulea University of Technology have been adapted to an environment with limited RAM (no more than 8 KB available) by reducing memory occupation through code optimizations.

The platform used is constituted by an STM32W108 System-on-Chip (SoC), integrating a 32-bit $Cortex^{TM}$-M3 microprocessor @ 2.4 GHz and a IEEE 802.15.4-compliant transceiver. The RAM constraint is similar to the one we have been facing, since we have 16 KB of RAM available but also use the full communication stack.

The main contribution of Domenico Caputo's work is proving the fact that porting EXIP on a sensor node and meeting the 8 KB RAM constraint is actually possible.

Our work differs in that we provide performance measurements and in that we have been following an IoT-driven approach where ETSI M2M messages have been used as a payload benchmark.

3.2 Non-EXIP based implementations

In the work by Angelo P. Castellani et al. [17] an implementation on a *TelosB* sensor node inclusive of CoAP and EXI modules has been realized: the EXI module has in this case been developed from scratch resulting in a library called *libEXI*.

The main contributions of the article are:

- The scalability of a system composed of a variable number of CoAP servers on a single *TelosB* has been estimated. The article does not specify whether the server simply sends pre-computed EXI payloads or actually generates them from binary structures upon each request. While in the first case the source of failure would be channel congestion, in the second it would be impacted also by the capability of the server face requests from the point of view of the computational resources.

- The performance of *libEXI* is compared with that of *EXIficient*, a freely available Java implementation of the EXI specification. The time performance assessment consists of a measurement of compression time on a desktop PC in order to estimate the latency introduced by a proxy translating XML into EXI, which for *libEXI* is in the order of 10^{-2} ms.

[2]Embeddable EXI implementation in C. Available: http://exip.sourceforge.net/ Last Access: June 12, 2014.

The superposition of results between this work and our work is therefore limited to the compression gain in terms of data size, which in any case depends on the W3C EXI specification [16] and not on the specific implementation.

In the Master Degree Thesis by one of the authors [22] the RAM and ROM occupation of *libEXI* implementation on a *Tmote Sky* with *TinyOS* is reported: libEXI has a memory occupancy of 10 KB over a total 48 KB of ROM, and of 1.7 KB over a total 8 KB of RAM, leaving 1 KB free for stack and heap at runtime. The communication stack is not present.

In any case, the approach we have been using is different in that EXI serialization of data structures does not use XML as an intermediate data format (XML-less EXI), and performance measurements have been taken on an embedded system rather than on a PC workstation.

A very interesting article by Yusuke Doi et al. [18] describes a TOSHIBA proprietary implementation of EXI, called *EIGEN*.

First of all the article highlights the need to focus on XML-less EXI in order to limit implementation complexity, which is a feature that is not often considered crucial in EXI implementations.

Second of all, the article gives recognition to the concept that a node has often a limited set of purposes which do not necessarily need a full-spec EXI processor. The solution proposed is indeed that of developing a code generation module which takes as input an XML schema and a mapping of XML elements to binary data structures, and returns C code as output.

The generated code corresponds, in the EXIP case, to a module developed ad-hoc for the specific messaging set considered, plus a multipurpose EXI processor, plus external grammars if Schema-Enabled compression is performed.

The decoding instead is more similar to the EXIP approach, and is realized through a decoding module which uses external grammars.

A measurement of memory occupancy for a resource-constrained device with embedded OS[3] is also given: *EIGEN* uses 13 KB ROM for EXI and I/O related code, while the grammar brings a 50KB ROM occupancy at the decoder side. Moreover, the EXI decoder and encoder use approximately 9500 bytes of RAM, including 6KB of communication buffer.

Lastly, this work gives recommendations on sensible choices that are to be made when there is the chance of building an XML schema document for messages serialized using EXI Schema-Enabled, in order to achieve a particularly efficient EXI representation:

- Use as few optional elements and attributes as possible, in order to reduce the number of transitions.

[3]TOPPERS/ASP 1.3.2 on STM32F103ZE board with a Cortex-M3 processor

- Make the modular structure of the schema easily mappable to the set of application purposes envisioned. For example even if air conditioners do not need to understand the message parts for ovens, a broader schema may be used for both. The types used for each domain should then be clearly factorized: this can be done for example by using the `xsi:type` attribute defined in the XML schema specification [19].

This study mainly impacts our present work in the future works envisioned: since performance measurements are not executed, and the XML Schema used is not specified, it is not even possible to make a precise comparison for the RAM/ROM occupation with respect to our EXIP optimized implementation.

3.3 Original contribution

From this overview it emerges that several EXI implementations exist. Few of them are open source and target constrained nodes, and fewer are freely available as open source code.

In several among the mentioned works, the successful porting of an EXI implementation on a resource-constrained node is claimed.

Nevertheless, performance measurements in terms of time or energy have never been executed on sensor nodes, but only on PC workstations. Values that can be used to elaborate preliminary design and dimensioning estimates for resource-constrained networks are therefore not yet available in the literature.

The original contribution of the present work consists in considering a specific goal in terms of portability over IoT, which is ETSI M2M, and producing performance measurements for an explicitly declared set of messages, which have been deduced from a ETSI CoAP M2M Interoperability Test Suite.

Chapter 4

IoT in the ICSI project

4.1 Introduction

The present work has been carried out within the framework of the Intelligent Cooperative Sensing for Improved Traffic Efficiency (ICSI) project[1].

The goal of the ICSI project is to enable four use case scenarios related to the ITS domain using a closed loop REST-based architectural model jointly with state-of-the-art IoT and ITS technologies.

The use cases are related to both an urban environment and a highway environment. They are:

- **Area Access Control:** access rules to restricted urban areas can be either static, linked to transit permits owned for example by city center residents, or dynamic through real time monitoring of road traffic and pollution (level of CO_2 and other parameters) conditions detected by sensors. Intelligent cooperative sensing services provide collection, processing and distribution of real-time data regarding road traffic conditions and environmental parameters. These mechanisms enable the management of access permissions to restricted areas, according to the municipality's rules and road traffic congestion, and additional services such as alternative routing to vehicles with denied access.
- **Intermodal transportation:** in order to support new forms of transportation such as car-sharing and bike-sharing, the presence of a number of connected inter-modal transit nodes is needed. Such nodes consist in parking slots and e-vehicle recharging areas in case of electric cars and add up to a "system of sub-systems" that is "smart" in the sense that

[1] ICSI Project. Available: `http://www.ict-icsi.eu/index.html` Last Access: June 12, 2014.

it is constituted by cooperating components able to provide overarching coordination through distribution of real time data regarding parking availability, user service demand and management of charging points.

- **Route guidance:** aims at providing journey planning services with the added value of gathering real-time information regarding traffic conditions and detection of road accidents. The journey planning service is also comprehensive of information regarding points of interest, service areas and public transportation timetables.
- **Road tolling systems:** several road charging schemes such as area licensing, per access, distance based and time based are in use. In this scenario, intelligent cooperative sensing services collect, process and distribute real-time data on road use to provide a flexible and dynamic management of road pricing and charging systems.

4.2 ICSI System Architecture

ICSI is a pervasive system based on WSN and vehicular networks. This implies that it should be able to collect and process a large amount of sensed data in a scalable and reliable way and interconnect heterogeneous components as depicted in Figure 4.1:

- *Control Centres:* are able to collect, store and process large amounts of data from other components of the system and provide transportation management functions.
- *Road-Side Units (RSUs):* are positioned along the road and collect measured data by sensor devices (flow sensors, parking slots sensors), provide feedback (variable message signs, Electronic Vehicle charging spots) and act as a gateway towards the Internet.
- *On-Board Units (OBUs):* are placed within the vehicle and can be equipped with sensors and wireless networking equipment within the vehicular network as well as optional Internet connectivity.
- *User devices:* tablets, smart phones etc. are held by pedestrians or vehicle passengers and can provide transportation-related information to the users.

In order to be compliant with ITS state of the art the ETSI ITS and ETSI M2M technologies are considered. As discussed, ETSI ITS simply introduces a

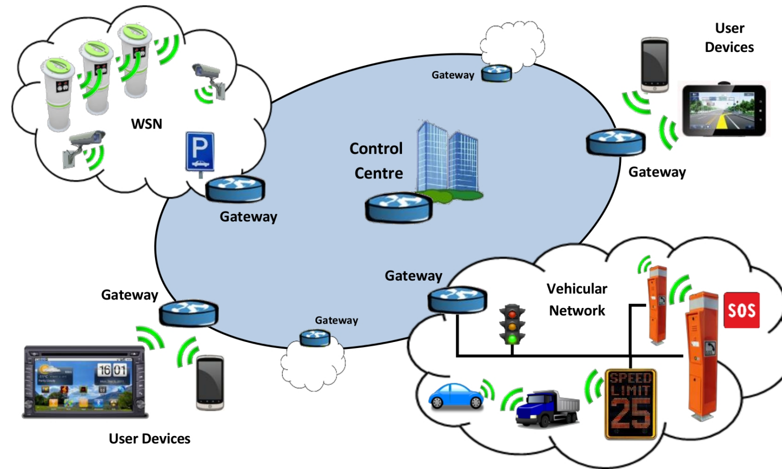


Figure 4.1: ICSI system components

functional scheme of an ITS system, a mapping of system components onto the OSI stack and a terminology based on four sub-systems which we will adopt in this instance:

- Central ITS sub-system (CS)
- Roadside ITS sub-system (RS)
- Vehicle ITS sub-system (CS)
- Personal ITS sub-system (PS)

4.2.1 Central Sub-System

A centralized architecture is the most common solution for a traffic management system: all information is gathered and analyzed in the CS, decisions are taken and infrastructure signals are sent back to the on-road infrastructure. Due to scalability issues a system like this is hard to maintain when the number of components and the amount of data increase.

The ICSI system follows a distributed approach where each Gateway component has CS functionalities and performs decentralized aggregation of traffic information analyzing the information gathered from the road infrastructure: Gateways may host for instance distributed algorithms for determining the best traffic strategies for dealing with roadway accidents. Despite this decentralization ICSI follows a layered approach in that a Gateway with a coordination role is envisioned.

4.2.2 Roadside Sub-System

A Roadside sub-system is composed of Roadside units i.e. sensor/actuator devices. Sensors/actuators consist of autonomous embedded devices in charge of extracting or influencing ITS parameters (e.g. number of cars in a road segment or number of free parking lots); they are mapped onto ETSI M2M D' devices hosting a Device Application whose local SCL is the SCL of the Gateway local to the VS itself.

4.2.3 Vehicular Sub-System

Vehicular equipment can use bidirectional communication channels (Vehicle-to-Infrastructure and Infrastructure-to-Vehicle) to exchange information with RSUs within the same or a different RS as well as On-Board Units. Components of a Vehicular Sub-system in scope are a processing unit, an interface to vehicular units such as Engine Control Unit, a Human-Machine interface, a mobile broadband with 3G/4G communication capabilities and an optional GPS system or similar; they can be seen as D' devices hosting an ETSI M2M Device Application.

4.2.4 Personal Sub-System

A Personal Sub-System consists of mobile devices (smartphones, etc...) used by pedestrians or vehicle passengers with 3G/4G Internet access following the D' paradigm and connected to an ICSI gateway.

4.2.5 The ICSI System

As can be seen from Figure 4.2 the ICSI system follows a distributed architecture which is mapped onto ETSI M2M in terms of Gateways and D' devices.

The ICSI system relies on distributed storage and intelligence which corresponds to the functionalities traditionally assigned to a Central Sub-System and now assigned to ETSI M2M Gateways. Each Gateway has a local scope, meaning that it receives data from and elaborates data for a limited number of components in a given area.

A single Gateway may communicate with Gateways of the same area handling data relevant to the processing of complex data for sensors and actuators of its own sub-system, or with gateways outside its area which may be issuing commands or be interested in the data collected locally. For example an

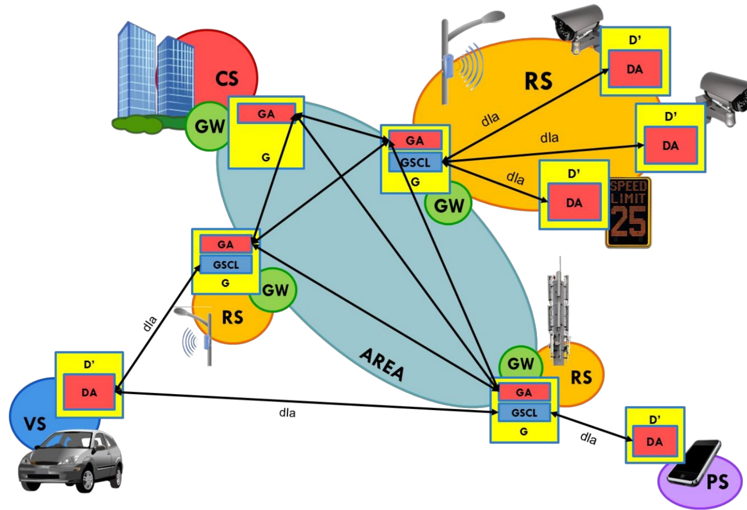


Figure 4.2: ICSI system mapped on ETSI M2M

accident detected locally may change the routing parameters of journey planning algorithms running on nodes far away which could be propagated through subscriptions.

A "local area" is a logical rather than a physical concept and its perimeter definition may depend on a number of elements such as population density, traffic and expected usage. A "global area" will support inter-area communication by interconnecting several "local areas" as shown in Figure 4.3.

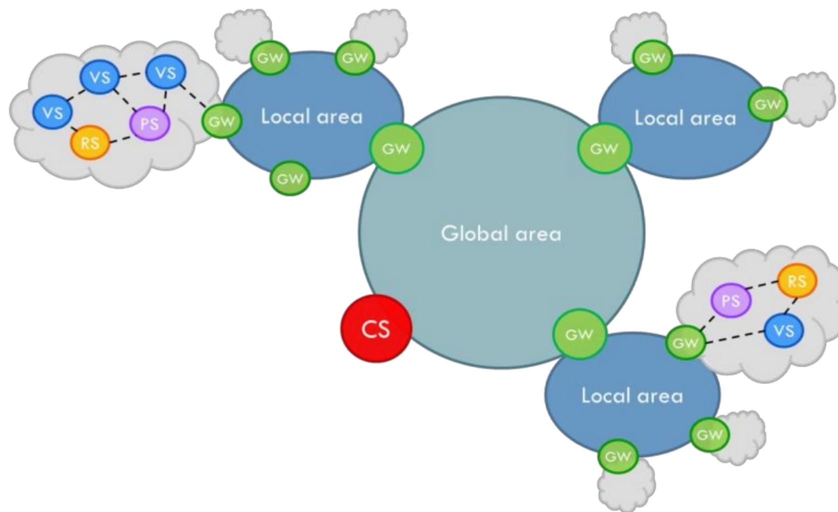


Figure 4.3: ICSI logical areas

In order to ensure the real-timeness within the application domains defined, which are not safety-critical but which do require effective data gathering and processing, the following time thresholds must be enforced:

Scope	**Actors**	**Gateway**	**Time Scale**	**Aggregation**
Local	RS,VS,PS	No	$\approx 10^{-2}$ sec	None
Intra-Area	RS,VS,PS	Yes	$\approx 10^{-1}$ sec	Low
Inter-Area	RS,VS,PS	Yes	10 sec	Medium
System	CS,RS,VS,PS	Yes	$\approx 10^{2}$ sec	High

Table 4.1: Cooperative sensing levels

Chapter 5

Implementation and performance assessment

5.1 Introduction

The present chapter explains the concrete contribution of this work to the ICSI project as well as the design and development work on which the system evaluation has been performed.

In order to complete the stack of IoT enabling technologies the feasibility of porting ETSI M2M onto sensor nodes needs to be proven: this is the main goal of our work.

According to the IoT vision where objects themselves should be "smart", the goal is to have ETSI M2M compliant sensor nodes hosting a Device Application and (possibly) a DSCL implementing a dIa interface as explained in Chapter 2. The first step is nevertheless that of having a Device following a D' model where the sensor device hosts a Device Application while referring to the Gateway SCL of the local area as Local SCL: the reference configuration for our work is Case 2 of Figure 2.8. The hardware platform used in the ICSI project is a *SEED-EYE* board[1], which has been developed within the research group. However, the performance measurements presented here have been executed on a *Wismote* sensor node[2] using *Cooja*, a simulator for WSN. This choice has been made so that other researchers can effortlessly replicate our measurements on the platforms made available by the simulator before being ported on actual hardware.

We based our software implementation on the *Contiki* Operating System

[1] `http://rtn.sssup.it/index.php/hardware/seed-eye` Last Access: June 12, 2014.

[2] `http://wismote.org/doku.php` Last Access: June 12, 2014.

which provides networking libraries for the IoT.

In order to achieve generality a set of ETSI M2M CoAP Interoperability Tests have been executed: instead of a test outcome consisting of a yes/no verdict, additional time and energy measurements have been executed. Or equivalently, ETSI M2M CoAP Interoperability Tests have been used as a benchmark for performance evaluation.

The results can for this reason be considered suitably representative of the performance of the ETSI M2M system.

5.2 The Development Platform

The open source operating systems for embedded devices targeted for IoT are essentially three: *TinyOS*, *Contiki* and *Riot*. All these operating systems are designed in order to work on severely resource-constrained classes of hardware, with memory on the order of kilobytes, power budgets on the order of milliwatts, processing speed measured in megahertz, and communication bandwidth on the order of hundreds of kilobits per second. Often 8 or 16-bit systems are targeted.

Riot has a real scheduler allowing concurrent flows of execution, while Contiki and TinyOS are based on the event-driven programming paradigm, and *Contiki* uses a mechanism called *Protothreads* [21]. Nevertheless Riot does not fully implement the IoT stack yet: for instance it has a partial implementation of CoAP.

The main difference between TinyOS and Contiki is historical: Contiki supported the IoT stack earlier and for this reason its implementation is more mature: the Contiki IPv6 stack for instance is fully certified under the IPv6 Ready Logo program, while the TinyOS IPv6 stack is not. For these reasons in the present work Contiki has been chosen as the development platform.

5.2.1 Contiki

Contiki[20] is an open source operating system for the Internet of Things: it provides low-power Internet communication by fully supporting IPv6 and IPv4 and the low-power wireless standards 6LoWPAN, RPL and CoAP so that even wireless routers can be battery operated.

Contiki is designed to operate in extremely low-power systems: systems that may need to run for years on a pair of AA batteries. It provides a set of mechanisms for memory allocation such as memory block allocation *memb*, a managed memory allocator *mmem*, as well as the standard C memory allocator *malloc*.

To save memory and at the same time provide a nice control flow in the code, Contiki uses Protothreads, which are a mixture of event-driven and multi-threaded programming mechanisms.

A customized Contiki environment is constituted by a number of components including the "application" being developed and the operating system itself which are compiled into a single executable file.

5.3 Preliminary performance assessment

Since DATEX2 [3] is the data model indicated by the European Committee for Standardization (CEN) as the format of choice for interoperable ITS systems[4], we adapted the DATEX2 model to the notifications which will be sent by the sensors. Such notification messages will constitute the body of a <contentInstance> POST operation performed by the sensor device itself every time a data item is sensed and sent to the Gateway for aggregation.

As a back-of-the envelope calculation we considered the M2M notifications since they will likely be the type of message most frequently exchanged and we tried to estimate which data format it would be reasonable to explore.

A DATEX2-compliant notification included within an ETSI M2M <notify> resource and referred to parking slot occupation is reported in Appendix A.

The values reported in Table 5.1 are the size of the sample notification evaluated using the data formats considered. The corresponding number of CoAP blocks for a block size of 64 bytes is also shown[5]. The reason why we report the number of blocks is that each block corresponds to an additional network overhead, since it requires a header set corresponding to CoAP, UDP, 6LoWPAN and IEEE 802.15.4.

DATEX2 notification serialization				
Profile	XML	JSON	Schema-Less EXI	Schema-Enabled EXI
Payload (bytes)	**2308**	**1739**	**1453**	**307**
factor vs. XML	-	0.75	0.62	0.13
factor vs JSON	-	-	0.83	0.17
Number of 64 byte blocks	37	28	23	5

Table 5.1: DATEX2 notification serialization - size of serialized data

[3] http://www.datex2.eu/ Last Access: June 12, 2014.

[4] http://www.datex2.eu/sites/www.datex2.eu/files/Datex_Brochure_2011.pdf Last Access: June 12, 2014.

[5] The Blockwise transfer is described in Chapter 2

The preliminary performance assessment of Schema-Enabled and Schema-Less EXI has been performed using OpenEXI[6], a Java tool which converts an XML structure into an EXI stream.

The outcome of the evaluation is that the use of XML is unfeasible given the high number of CoAP blocks it is necessary to transmit: in a scenario with non-negligible packet loss it would impede communication; using JSON and Schema-Less EXI is also unreasonable.

Schema-Enabled EXI is an interesting candidate since the high compression achieved could allow reducing the REST method execution time provided that the overhead in terms of message encoding does not keep from having an overall benefit due to the smaller number of blocks.

In any case even in the case of Schema-Enabled EXI, DATEX2 introduces a very high structural overhead and therefore should be adopted only if strictly necessary in order not to impede communication performance.

As a conclusion, the adoption of DATEX2 as the data model on top of ETSI M2M introduces an excessive overhead. Nevertheless EXI has been proven the only viable choice in comparison with the other options considered and for this reason should be evaluated as a candidate for ETSI M2M communications not necessarily relying on DATEX2.

The implementation of EXI on embedded systems requires meeting the constraints on RAM availability. Moreover the time performance of the encoding procedure needs to be evaluated directly on the sensor itself since it could differ significantly from the values measured on a PC workstation.

5.4 System Parameters

The simulation set up comprises of three *Wismote* nodes: a client, a server, and a border router, as depicted in Figure 5.1.

The border router creates the Routing Protocol for Low Power and Lossy Networks (RPL) Directed Acyclic Graph. We consider the situation when client and server communicate directly in a single-hop manner. EXI compression has been carried out with the *bit-packed* alignment option: in order to achieve maximum compression in size, event codes and content items are aligned at bit-level.

For Schema-Enabled EXI encoding the *Strict* encoding option has been enabled: a deviation from the schema results in an error. This brings the benefit of reducing the length of event codes.

In the simulations the parameter *Packet Delivery Rate* (PDR) has been used. This parameter is expressed as a percentage and it represents the fraction

[6]http://openexi.sourceforge.net/ Last Access: June 12, 2014.

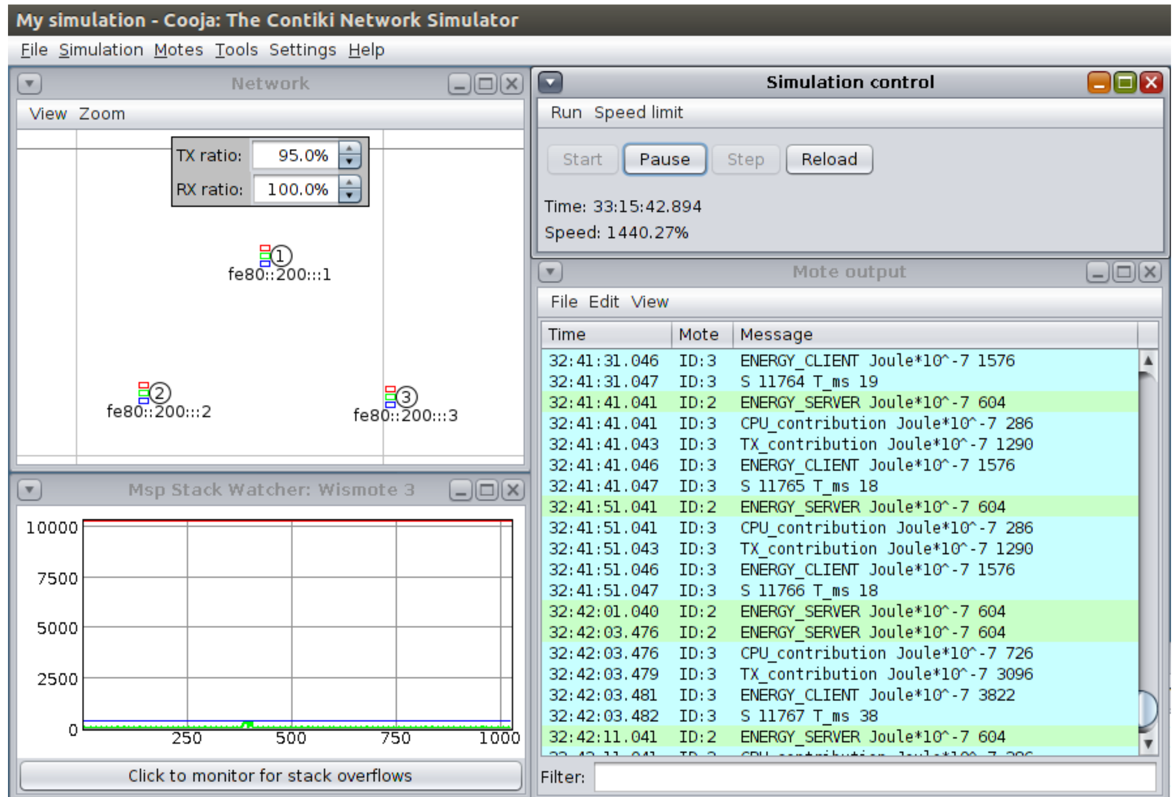


Figure 5.1: *Cooja* simulation

of the frames at physical level sent by the source mote which are successfully received by the destination mote.

ETSI M2M CoAP Interoperability tests have been executed in conditions of 100% success rate in delivery at physical level (PDR, Packet Delivery Ratio). Additional tests have then been executed at 95% PDR. *Cooja* implements loss using a uniform distribution over the messages exchanged: it does not distinguish between scenarios where there are an equal number of packets which differ in length.

5.5 Compression performance evaluation

The performance of the two M2M data serialization modes alternative to XML (JSON and EXI) has been compared according to the following criteria:

1. *Compression gain.*
2. *Serialization complexity*, in terms of both memory usage and time.

3. *Performance over ETSI M2M CoAP Interoperability Tests*: execution time, channel usage and energy consumption of a CoAP operation (GET, POST, PUT methods over RESTful resources) between two nodes.

4. *Performance in a lossy configuration.*

While evaluating the EXI format two configurations have been considered, corresponding to Schema-Enabled serialization and Schema-Less serialization.

5.5.1 The Measurement process

The time values reported are the result of an average of a set of independent measurements where the uncertainty has been taken as $3 * \hat{\sigma}$ where $\hat{\sigma} = \sqrt{\frac{1}{N(N-1)} * \sum_{i=1}^{N}(x_i - \langle x \rangle)^2}$, x_i are the measured value and $\langle x \rangle$ is the mean value[7].

The number of samples has been estimated for each average value[8] and is on the order of 10^3 for measurements with a 100% PDR and on the order of 10^4–10^5 for measurements with a 95% PDR. It has been then verified *a posteriori* that the relative uncertainty is always less than 5%: this condition does not apply to JSON serialization/deserialization times which are too small in comparison with the resolution of the measurement itself.

5.5.2 Energy measurements

Energy measurements are referred to the client node. Server-side energy consumption is out of scope in the present work.

Energy consumption has been estimated using the *energest* library. With *energest* we can perform time measurements relatively to the CPU and the radio module separately. Moreover, we can distinguish between CPU active time, and low power mode time. For the radio, we have measured transmission time and listening time.

The values reported as energy measurements are the sum of the CPU consumption and the energy used for transmitting and receiving messages. We have chosen not to consider idle time and listen time because they can depend highly on the duty cycle pattern used, and their inclusion would have made our results less general. Moreover, the sensor node radio is not necessarily inactive while waiting for an answer: in an analogous way the low power mode

[7] $\hat{\sigma}$ is sometimes called standard error in the scientific literature.

[8] Using the empirical formula $\Delta x \approx \frac{x_{max} - x_{min}}{\sqrt{N}}$, aiming at a relative uncertainty in the order of 1%

energy consumption (when the Protothread is suspended, waiting for an event such as the reception of a message or a timeout) is not considered.

The values measured correspond to the energy spent for executing an operation, but it should be noted that they do not represent the additional energy spent for operation execution with respect to a situation where the CPU is in low power mode and the radio is listening according to its duty cycle.

The operating currents have been obtained from *Wismote* and CC2520 transceiver data sheets, and are reported in Table 5.2:

Operating Currents	
Mode	**Value**
Transmission	25.8 mA
Reception	18.5 mA
CPU active	2.2 mA

Table 5.2: Operating currents for *Wismote* and CC2520

The formula describing energy dissipated by the system:

$$E = \int_{t_1}^{t2} W(t)dt = \int_{t_1}^{t2} V(t)I(t)dt$$

in our case can be simplified as:

$$E = \int_{t_1}^{t2} VI(t)dt$$

In a *Wismote* sensor device, the operating voltage for both CPU and the radio modules is 3.3V.

In order to obtain the corresponding results for a sensor device operating at a different voltage, it is sufficient to rescale the results reported in this Chapter.

5.6 Compression Gain

In this section we discuss the size of the payloads used throughout the performance assessment, for the three data serialization methods considered.

With respect to JSON, EXI performance in compression is expected to depend in a non-trivial way on the structure of the grammar used, on the structure of the message itself, on the content of the message[9] and on the data types used.

[9] "structure" and "content" terms are used referring to the element name and the element content in the XML version of a message, i.e. < *structure* > *content* < /*structure* >

The size of the serialized data is reported for <application>, <subscription> and <contentInstance> M2M resources[10]; the corresponding data used is reported in Appendices B, C and D respectively.

The message profiles chosen increase both their content and structure components as the profile number goes from 1 to 4, except the <contentInstance> resource, where only the content part of the XML increases in size.

A trend should not be deduced from the data reported: the high gain for short messages is due to the absence of an EXI counterpart for the XML declaration, as seen in Subsection 2.7.2, which has a smaller impact as messages get larger.

We would expect instead the situation to become stable after a given threshold, and to have peaks of size gain for EXI with respect to JSON whenever structures are repeated (EXI Schema-Less) or content is repeated (EXI Schema-Less and EXI Schema-Enabled). Moreover, for large data items with highly repetitive structures we would expect EXI Schema-Less and EXI Schema-Enabled to become closer in performance, due to EXI Schema-Less dynamic learning mechanisms. EXI Schema-Less builds up grammar structures when a given structure is processed the first time, in such a way that they can then be referred to indirectly when such structure is subsequently encountered, without re-encoding the structure item.

These situations have not been explored experimentally due to the RAM limitations of the *Wismote* mote used for testing.

<application> resource serialization						
Profile	XML	JSON	Schema-Less EXI		Schema-Enabled EXI	
	payload (bytes)	payload (bytes)	payload (bytes)	size factor vs.JSON	payload (bytes)	size factor vs.JSON
1.	**103**	**56**	**42**	0.75	**7**	0.12
2.	**235**	**142**	**111**	0.78	**33**	0.23
3.	**424**	**279**	**236**	0.84	**113**	0.41
4.	**628**	**424**	**377**	0.89	**195**	0.46

Table 5.3: <application> resource serialization - size of serialized data

[10]The grammars have been generated from the ETSI XSD documents available at the URL: `http://www.etsi.org/deliver/etsi_ts/102900_102999/102921/01.01.01_60/ts_102921v010101p0.zip` Last Access: June 12, 2014.

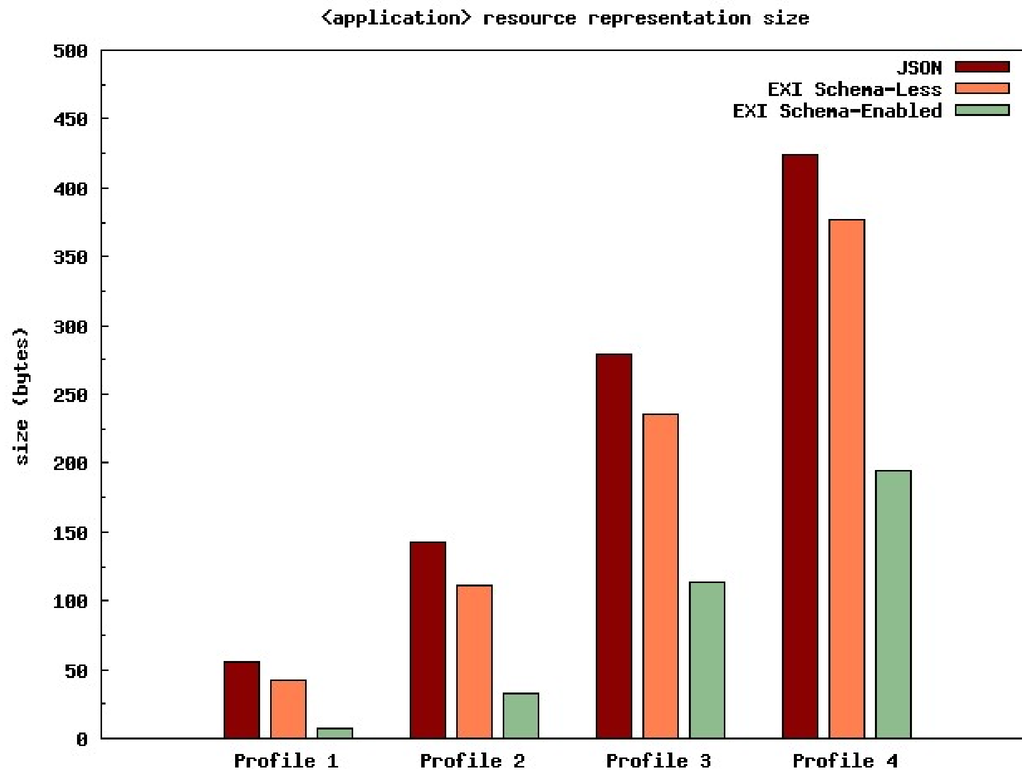


Figure 5.2: <application> resource serialization - size of serialized data

<subscription> resource serialization						
Profile	XML	JSON	Schema-Less EXI		Schema-Enabled EXI	
	payload (bytes)	payload (bytes)	payload (bytes)	size factor vs.JSON	payload (bytes)	size factor vs.JSON
1.	**105**	**57**	**43**	0.75	**7**	0.12
2.	**170**	**104**	**86**	0.83	**16**	0.15
3.	**235**	**151**	**129**	0.68	**25**	0.16
4.	**291**	**196**	**170**	0.73	**57**	0.29

Table 5.4: <subscription> resource serialization - size of serialized data

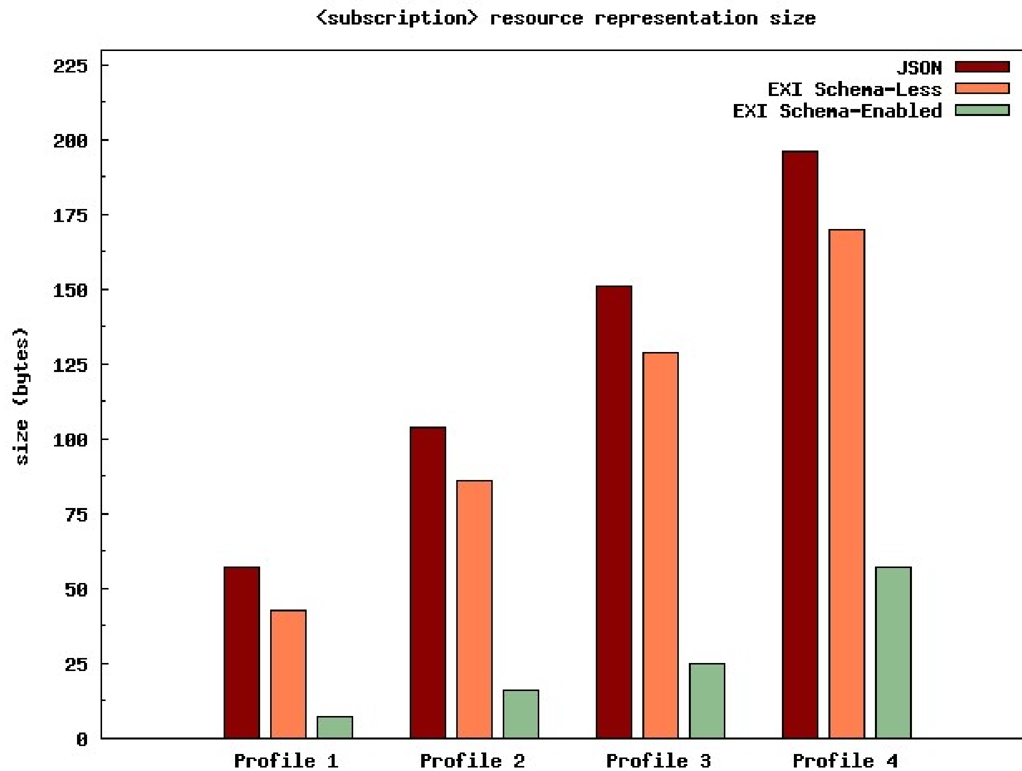


Figure 5.3: <subscription> resource serialization - size of serialized data

<contentInstance> resource serialization						
Profile	XML	JSON	Schema-Less EXI		Schema-Enabled EXI	
	payload (bytes)	payload (bytes)	payload (bytes)	size factor vs.JSON	payload (bytes)	size factor vs.JSON
1.	**233**	**147**	**127**	0.96	**44**	0.30
2.	**300**	**199**	**179**	0.90	**96**	0.48
3.	**363**	**277**	**258**	0.93	**175**	0.63
4.	**467**	**381**	**362**	0.95	**279**	0.73

Table 5.5: <contentInstance> resource serialization - size of serialized data

Comparing the compression gains, we can see that the size gain obtained for an EXI Schema-Enabled <subscription> resource is higher than the one achieved for an EXI Schema-Enabled <application> resource. This is due to the high size gain achieved by EXI for *DateTime* data types, in comparison to the one achieved in the case of plain strings, which constitute the content of the <application> resource. EXI Schema-Less does not benefit from this fact because it encodes *DateTime* data types as plain strings.

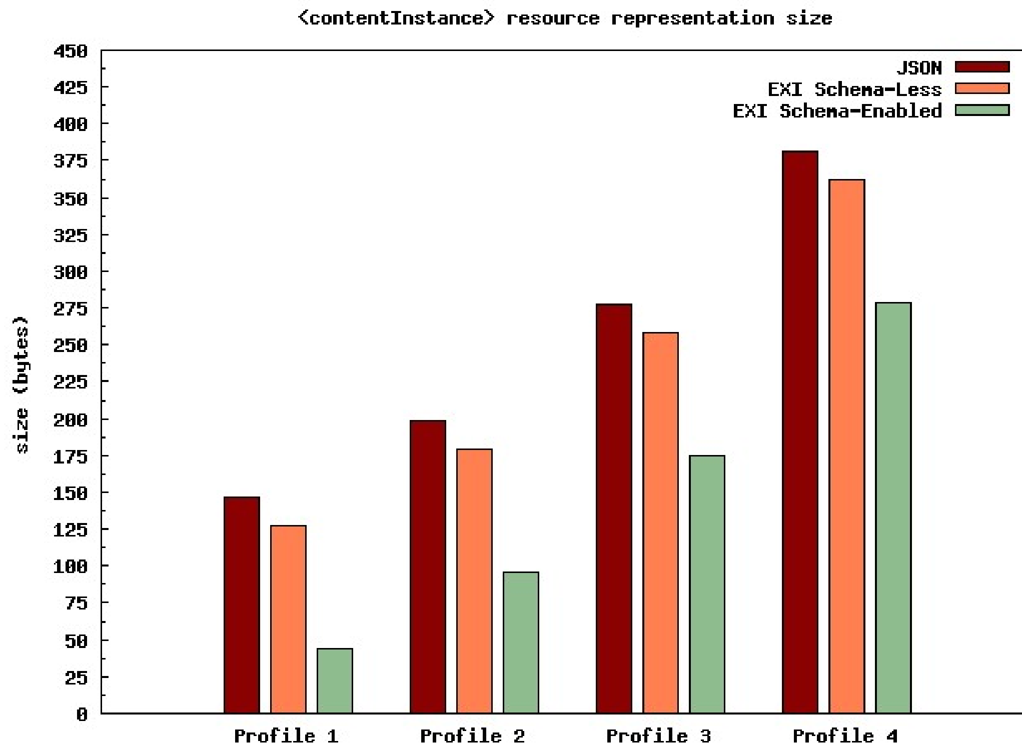


Figure 5.4: <contentInstance> resource serialization - size of serialized data

The lower size gain obtained for a <contentInstance> resource is due to the fact that this resource has a high content size over structure size ratio, compared to the other two cases, and the content is not repetitive. This is the situation where EXI performs worse.

5.6.1 Size in terms of CoAP blocks

In a constrained network, payloads resulting in frames larger than 127 bytes are subject to fragmentation at IP layer, as seen in Chapter 2. Contiki for example fragments UDP packets larger than 90 bytes, in order to keep some margin and avoid fragmentation in the pessimistic case of a 6LoWPAN header expansion while crossing a multi-hop 6LoWPAN network.

The first level of fragmentation takes place at CoAP level. The default block size is 64 bytes, but it can be increased or decreased as long as it is expressed as a power of two, is greater or equal to 16 bytes and smaller or equal to 1024 bytes (SZX∈[0,6], SZX=7 is reserved).

We focus CoAP fragmentation in terms of blocks, because not all 6LoW-

PAN networks support fragmentation at IP level, due to the limited resources available.

Below we show the dimensions of the packets in terms of number of blocks: they will be recalled in ETSI M2M CoAP Interoperability Test tables to allow a quicker interpretation of the performance graphs.

Profile	Number of Blocks			
	XML	JSON	Schema-Less EXI	Schema-Enabled EXI
	#	#	#	#
1.	2	1	1	1
2.	4	3	2	1
3.	6	5	4	2
4.	10	7	6	3

Table 5.6: <application> resource - Number of Blocks for blocks of 64 bytes

Profile	Number of Blocks			
	XML	JSON	Schema-Less EXI	Schema-Enabled EXI
	#	#	#	#
1.	2	1	1	1
2.	3	2	2	1
3.	4	3	3	1
4.	5	4	3	1

Table 5.7: <subscription> resource - Number of Blocks for blocks of 64 bytes

Profile	Number of Blocks			
	XML	JSON	Schema-Less EXI	Schema-Enabled EXI
	#	#	#	#
1.	4	3	2	1
2.	5	4	3	2
3.	6	5	5	3
4.	8	6	6	5

Table 5.8: <contentInstance> resource - Number of Blocks for blocks of 64 bytes

We can observe from Table 5.6.1 that the EXI Schema-Enabled serialization of a <subscription> resource is particularly efficient, indeed all message profiles map onto a single CoAP block. The EXI Schema-Enabled serialization of a <contentInstance> resource, on the contrary, shows a lower gain with respect to JSON.

5.7 Serialization Complexity

The Serialization complexity has been measured for an $< application >$ resource.

5.7.1 Serialization time

The serialization time has a relevant weight in the overall operation performance because EXIP is not currently engineered to exploit the parallelism between block serialization and block delivery.

We can see that JSON serialization performs an order of magnitude better compared to EXI, with respect to time, whereas EXI Schema-Less and EXI Schema-Enabled differ by a factor of two, and EXI Schema-Enabled is faster.

<application> resource serialization time			
	JSON	Schema-Less	Schema-Enabled
Profile	(ms)	(ms)	(ms)
1.	0.17 ± 0.02	5.64 ± 0.03	2.50 ± 0.01
2.	0.42 ± 0.01	14.80 ± 0.07	6.07 ± 0.03
3.	0.62 ± 0.02	31.5 ± 0.2	15.01 ± 0.07
4.	1.10 ± 0.01	48.0 ± 0.2	23.1 ± 0.1

Table 5.9: <application> resource serialization - Serialization time complexity

5.7.2 Deserialization time

Deserialization times for EXI Schema-Less, EXI Schema-Enabled and JSON are reported below for completeness. EXI deserialization is faster compared to EXI serialization, whereas for JSON the opposite is true.

<application> resource deserialization time			
	JSON	Schema-Less	Schema-Enabled
Profile	(ms)	(ms)	(ms)
1.	0.27 ± 0.01	3.60 ± 0.01	2.05 ± 0.01
2.	0.73 ± 0.01	9.70 ± 0.04	4.45 ± 0.02
3.	1.09 ± 0.01	19.17 ± 0.09	8.46 ± 0.04
4.	1.86 ± 0.01	27.5 ± 0.1	12.05 ± 0.06

Table 5.10: <application> resource deserialization - Deserialization time complexity

5.7.3 Serialization memory usage

The measurements of memory usage are specific for an *applicationCreate* operation. The estimate of the stack size at run time is inclusive of:

- Contiki OS and its communication stack: IEEE 802.15.4, 6LoWPAN, *Erbium* (CoAP)
- EXIP

The ETSI M2M implementation is not considered since its dimension is highly dependent on the specific deployment.

Memory usage - applicationCreate			
	JSON	Schema-Less	Schema-Enabled
Text + Data (ROM)	5 KB	43 KB	50 KB
Data + BSS (RAM)	210 B	8 KB	9 KB
Stack at run-time (RAM)	(500 ± 70) B	(680 ± 40) B	(650 ± 40) B

Table 5.11: Memory usage - applicationCreate

With EXI, the stack is used at 30% for Schema-Less encoding and at 40% for Schema-Enabled encoding: the total RAM available for stack and heap at run-time is respectively 2250 and 1710 bytes. Out of the initial *Wismote* RAM size of 16 Kb, 8 Kb are used by EXI and 5.7 Kb are used by Contiki OS and the communication stack.

5.8 ETSI M2M CoAP Interoperability Tests

ETSI M2M CoAP Interoperability Tests consist in the execution of a set of operations (create, retrieve, update) over ETSI M2M resources (<application>, <subscription>, <contentInstance>) and in verifying the output of the server: the client is seen as the sensor node M2M Device Application and the server as the Gateway SCL.

In this instance the goal is not to test the Gateway implementation which has been built using the *Cooja* simulation tool for the sole purpose of executing the tests themselves. The goal is to understand the performance of the data serialization methods using ETSI M2M CoAP Interoperability Tests as a benchmark.

To summarize, the tests executed are:

- *applicationCreate:* ETSI M2M CoAP Test TD_M2M_COAP_01.
- *applicationRetrieve:* ETSI M2M CoAP TD_M2M_COAP_02.
- *applicationUpdate:* ETSI M2M CoAP Test TD_M2M_COAP_03.
- *subscriptionCreate:* ETSI M2M CoAP Test TD_M2M_COAP_04.
- *contentInstanceRetrieve:* ETSI M2M CoAP Test TD_M2M_COAP_08.

Hereby we list the tests we did not execute, and comment on the reasons for exclusion:

- *subscriptionNotify:* ETSI M2M CoAP Test TD_M2M_COAP_05, refers to a use case that will not be considered in ICSI.
- *subscriptionDelete:* ETSI M2M CoAP Test TD_M2M_COAP_06, does not involve any payload.
- *applicationDelete:* ETSI M2M CoAP Test TD_M2M_COAP_07, does not involve any payload.
- *MultipleQueryOptions:* ETSI M2M CoAP Test TD_M2M_COAP_09, considered as message benchmark item, it is equivalent to TD_M2M_COAP_08.
- *PartialAddressing:* ETSI M2M CoAP Test TD_M2M_COAP_10, considered as message benchmark item, it is equivalent to TD_M2M_COAP_08.
- *Announcement:* ETSI M2M CoAP Test TD_M2M_COAP_11, considered as message benchmark item, it is equivalent to TD_M2M_COAP_01.

The performance of the various serialization methods has been evaluated through the *Cooja* simulation tool using a 100% PDR. Motes have been displaced in a triangular network where the server and client communicate directly in single-hop.

The message profiles used for the tests are reported in Appendix B, C and D: XML data are shown just for reference. Indeed in the analytical assessment XML has not been considered because of its redundancy, making it not suitable for constrained networks.

In the following we have considered the contributions in terms of energy and time of CPU usage, Transmission and Reception. CPU usage is referred to the serialization time, and to the OS and communication stack. It should be noted that Network and MAC layers in Wireless Sensor devices are implemented in software.

Deserialization at the client side has not been taken into account because the pattern used by an endpoint to extract information from a message depends on the specific implementation and use case.

5.9 application Create

For a POST request, the time interval measured includes the serialization and delivery time of the request, and the delivery time of the response which does not carry a payload.

The response carries an optional payload depending on whether the server accepts the application resource as proposed by the client, or modifies some or all of its parts: we put ourselves in the scenario where the application proposal by the client is accepted by the server without modifications: for this reason the server response does not contain a payload.

Profile	JSON	EXI		Number of blocks
		Schema-Less	Schema-Enabled	
	(ms)	(ms)	(ms)	
1	19.06 ± 0.09	22.7 ± 0.1	12.00 ± 0.06	/1/1/1/
2	42.3 ± 0.2	45.7 ± 0.2	19.05 ± 0.09	/3/2/1/
3	75.1 ± 0.4	94.2 ± 0.5	46.0 ± 0.2	/5/4/2/
4	111.3 ± 0.5	143.5 ± 0.7	80.1 ± 0.4	/7/6/3/

Table 5.12: applicationCreate execution time

Profile	JSON	EXI	
		Schema-Less	Schema-Enabled
	bytes	bytes	bytes
1.	245	232 (-5%)	155 (-36%)
2.	640	469 (-27%)	181 (-72%)
3.	1135	953 (-16%)	471 (-58%)
4.	1679	1448 (-14%)	870 (-48%)

Table 5.13: applicationCreate channel usage

Energy Type	JSON	EXI	
		Schema-Less	Schema-Enabled
	μJ	μJ	μJ
	Profile 1		
TOTAL	719 ± 4	638 ± 3	340 ± 1
TX	426 ± 2	341 ± 2	170.3 ± 0.8
RX	199 ± 1	171.3 ± 0.8	103.9 ± 0.5
CPU	94.4 ± 0.5	125.7 ± 0.6	65.3 ± 0.3
	Profile 2		
TOTAL	1642 ± 8	1337 ± 7	512 ± 3
TX	1107 ± 5	851 ± 4	255 ± 1
RX	282 ± 1	189.9 ± 0.9	154.1 ± 0.8
CPU	252 ± 1	297 ± 1	101.6 ± 0.5
	Profile 3		
TOTAL	2970 ± 10	2775 ± 14	1337 ± 7
TX	2043 ± 10	1788 ± 9	851 ± 4
RX	470 ± 2	375 ± 2	187.8 ± 0.9
CPU	456 ± 2	611 ± 3	298 ± 1
	Profile 4		
TOTAL	4490 ± 20	4230 ± 20	2410 ± 10
TX	3150 ± 15	2724 ± 14	1532 ± 8
RX	657 ± 3	566 ± 3	373 ± 2
CPU	686 ± 3	936 ± 5	508 ± 2

Table 5.14: applicationCreate energy consumption

Table 5.12 shows *applicationCreate* execution times. The measured data is also reported in Figure 5.5.

From Figure 5.5 we can see that when Profile 1 is adopted, we use just one CoAP block of 64 bytes with all data formats considered. Schema-Enabled EXI decreases the disadvantage in encoding time with respect to JSON with a lower transmission time, while Schema-Less EXI is not able to decrease it enough to result in a overall faster POST.

Throughout Profiles 2 to 4 Schema-Less EXI keeps suffering from the high encoding time without having a decisive advantage in transmission with respect to JSON; we expect this situation to change if:

1. The block serialization and block delivery activities are executed in parallel.

2. The communication takes place over a lossy network.

Schema-Enabled EXI has instead a clear advantage with respect to the other formats which increases with payload complexity: it benefits from efficiency in terms of transmission time of a smaller payload which largely compensates its encoding time.

As a consistency check it has been verified that the results referring to an equal number CoAP packets once the serialization time is subtracted are compatible within the interval defined by the transmission time of a CoAP single CoAP block (since CoAP does not use padding) which is 2 ms.

This relation holds except for the comparison EXI Schema-Less vs. EXI Schema-Enabled in Profile 1: this is due to 6LoWPAN fragmentation which is performed when the UDP packet as a whole exceeds 90 bytes, and happens in the EXI Schema-Less case, but not in the EXI Schema-Enabled one.

The 90 bytes threshold is Contiki-specific: Contiki estimates the overhead introduced by IP, 6LoWPAN, IEEE 802.15.4 as 57 bytes, in order to avoid fragmentation even in a pessimistic scenario where for example the 6LoWPAN header may be extended in length. This could happen if the packet crosses a network that requires the insertion of a *Fragmentation Header* or a *Mesh Addressing Header* as described in Chapter 2 due to the presence of a multi-hop path.

Figure 5.6 shows the energy consumption of a *applicationCreate* operation, and the contributions in terms of transmission, reception and CPU consumptions.

From Figure 5.6 we can see that even if EXI Schema-Less performs worse in time, it brings a benefit in comparison to JSON from an energy standpoint. This is due to the fact that a smaller payload implies a lower radio usage, which has an higher power consumption than the one deriving from the CPU usage. EXI Schema-Enabled brings a large advantage compared to both JSON and EXI Schema-Less.

From Table 5.13 we can see that while the use of EXI Schema-Less reduces channel usage by a factor of approximately 15% with respect to JSON, EXI Schema-Enabled reaches an average of 50%.

5.10 application Retrieve

For an *applicationRetrieve* operation, in addition to the situation where the server has high computational capabilities (Scenario 1), we also explore the case where the GSCL is hosted by a sensor node (Scenario 2). In this last case the serialization time cannot be considered negligible with respect to the operation execution time.

While the *applicationRetrieve* operation execution time depends on whether

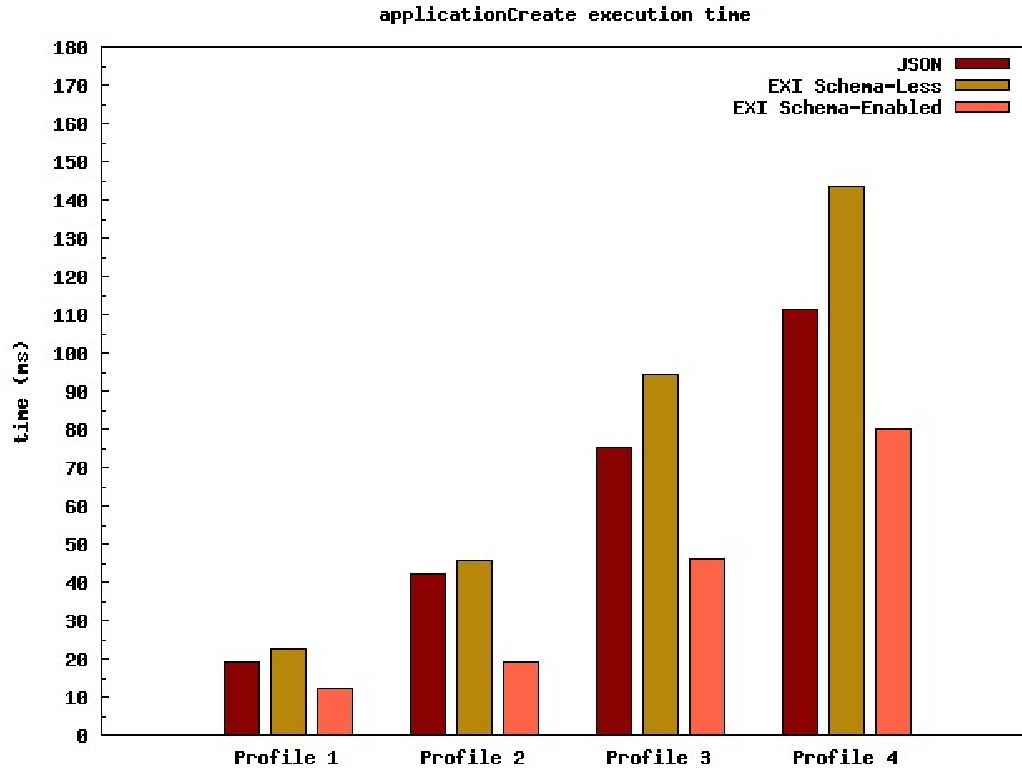


Figure 5.5: applicationCreate execution time

the data serialization is negligible with respect to data delivery, the channel usage and the energy consumption do not.

For a GET request, the time interval measured includes the delivery time of the request and the serialization and delivery time time of the response: the request does not carry a payload.

Profile	JSON	EXI		Number of blocks
		Schema-Less	Schema-Enabled	
	(ms)	(ms)	(ms)	
1	12.70 ± 0.06	11.73 ± 0.05	9.35 ± 0.04	/1/1/1/
2	35.0 ± 0.2	24.7 ± 0.1	11.12 ± 0.05	/3/2/1/
3	60.8 ± 0.3	49.7 ± 0.2	24.8 ± 0.1	/5/4/2/
4	87.2 ± 0.4	75.5 ± 0.4	48.3 ± 0.2	/7/6/3/

Table 5.15: applicationRetrieve execution time - Scenario 1

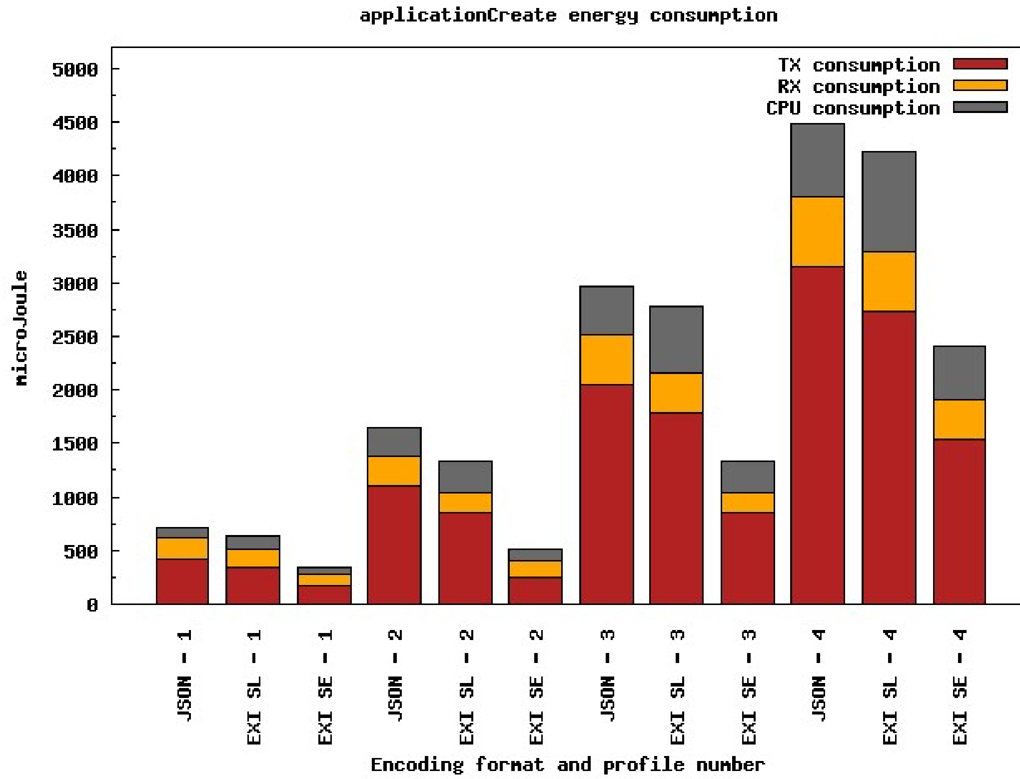


Figure 5.6: applicationCreate energy consumption

Profile	JSON	EXI		Number of blocks
		Schema-Less	Schema-Enabled	
	(ms)	(ms)	(ms)	
1	12.86 ± 0.06	17.44 ± 0.08	11.95 ± 0.05	/1/1/1/
2	35.41 ± 0.17	39.6 ± 0.2	17.29 ± 0.08	/3/2/1/
3	61.5 ± 0.3	81.1 ± 0.4	39.9 ± 0.2	/5/4/2/
4	88.2 ± 0.4	123.5 ± 0.6	71.5 ± 0.3	/7/6/3/

Table 5.16: applicationRetrieve execution - Scenario 2

Profile	JSON	EXI	
		Schema-Less	Schema-Enabled
	bytes	bytes	bytes
1.	185	171 (-7%)	136 (-26%)
2.	533	371 (-30%)	162 (-70%)
3.	932	758 (-18%)	373 (-60%)
4.	1339	1158 (-14%)	757 (-43%)

Table 5.17: applicationRetrieve channel usage

Energy Type	JSON	EXI	
		Schema-Less	Schema-Enabled
	μJ	μJ	μJ
	Profile 1		
TOTAL	403 ± 2	375 ± 2	300 ± 1
TX	170.3 ± 0.8	170.3 ± 0.8	170 ± 0.8
RX	189.4 ± 0.9	161.4 ± 0.8	93.6 ± 0.4
CPU	43.6 ± 0.2	43.6 ± 0.2	36.3 ± 0.2
	Profile 2		
TOTAL	1242 ± 6	803 ± 4	359 ± 2
TX	595 ± 3	340 ± 2	170.3 ± 0.8
RX	517 ± 3	375 ± 2	143.2 ± 0.7
CPU	130.7 ± 0.6	87.1 ± 0.4	43.6 ± 0.2
	Profile 3		
TOTAL	2105 ± 10	1726 ± 9	807 ± 4
TX	937 ± 5	766 ± 4	341 ± 2
RX	943 ± 5	780 ± 4	379 ± 2
CPU	225 ± 1	179 ± 0.9	87.1 ± 0.4
	Profile 4		
TOTAL	3059 ± 15	2668 ± 13	1718 ± 9
TX	1362 ± 7	1192 ± 6	766 ± 4
RX	1385 ± 7	1207 ± 6	776 ± 4
CPU	312 ± 2	269 ± 1	176 ± 1

Table 5.18: applicationRetrieve energy consumption

From the point of view of channel usage, reported in Table 5.17 and of the energy consumption in Figure 5.8, it is clear that EXI Schema-Less is better than JSON, and that EXI Schema-Enable is better than EXI Schema-Less.

Scenario 1

Figure 5.7 shows the execution times for a *applicationRetrieve* operation in relation with the serialization method adopted.

Again, the situation is clear: EXI Schema-Enabled performs better than EXI Schema-Less and JSON, and EXI Schema-Less performs better than JSON.

Scenario 2

From Figure 5.9 we can see that in Scenario 2 JSON tends to perform better than Schema-Less EXI: Schema-Less EXI never overcomes the initial disadvantage in encoding time.

On the one hand, the trend for increasing payload within the observed range lengths does not suggest that the compression ratio will become relevant enough to bring a significant benefit in transmission time and counterbalance the long encoding time.

On the other hand, in situations such as Profile 2 when Schema-Less EXI maps to a number of blocks inferior to the one used instead by JSON, the two serialization formats do become closer. It should be verified whether for larger payloads the size savings add up and Schema-Less EXI becomes competitive with JSON or not, and under which conditions of packet loss.

Schema-Enabled EXI performs significantly better than the other encoding formats due to the smaller payload: a Schema-Enabled EXI performs as well as a JSON Profile 2 (they both map the payload onto 3 CoAP blocks) and a Schema-Enabled EXI performs as well as a Schema-Less EXI Profile 2 (they both map the payload onto 2 CoAP blocks) while having comparable serialization time.

It is interesting to observe that Profile 1 EXI Schema-Enabled and EXI Schema-Less measures are compatible once the serialization time is subtracted, unlike the *applicationCreate* case: now 6LoWPAN fragmentation never occurs due to the fact that the CoAP header never contains the *Uri-Path* option when it contains a payload and therefore a 64 byte payload never generates a UDP packet larger than 90 bytes.

5.11 application Update

In this case we intend to consider a situation where both client and server adopt a Blockwise communication: this happens when a client requests an *applicationUpdate* operation through a PUT method and the server (hosting SCL) does not accept the values proposed. The server returns the resource representation actually modified in the response.

We considered a situation where the server does not accept any of the

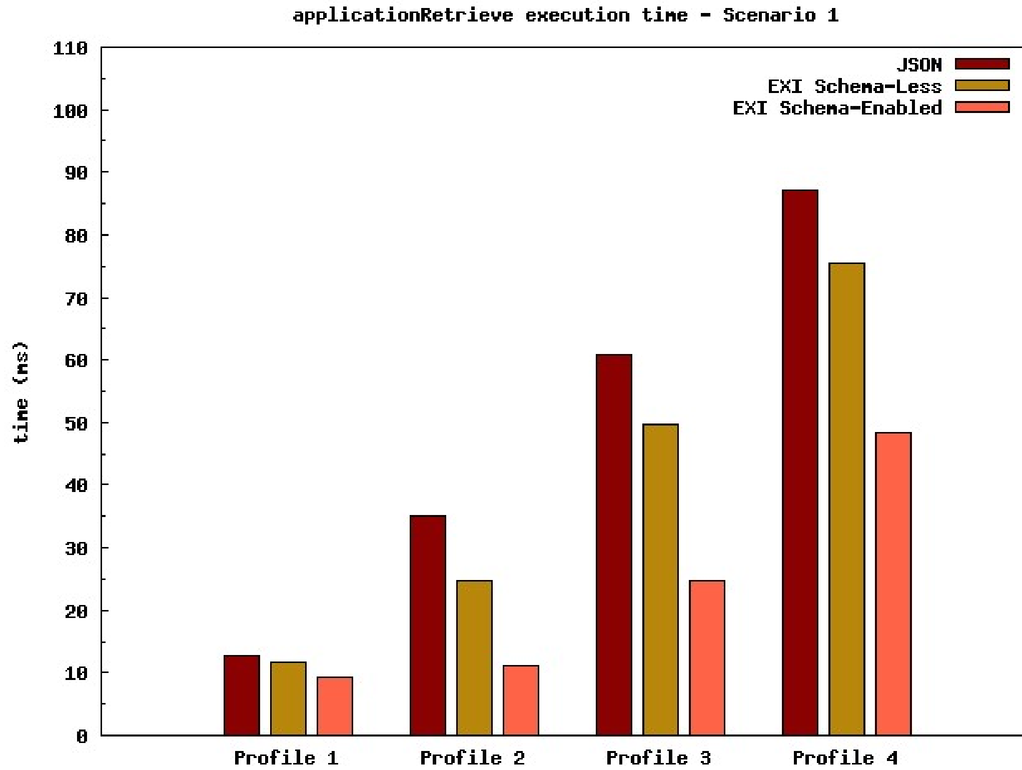


Figure 5.7: applicationRetrieve execution time - Scenario 1

parameters provided, and responds with a message having a payload of the same length of the one present in the request.

For this test suite item, we have considered the case where the server computation time is negligible compared to the overall operation execution time.

Profile	JSON	EXI		Number of blocks
		Schema-Less	Schema-Enabled	
	(ms)	(ms)	(ms)	
1	19.48 ± 0.09	23.2 ± 0.1	12.39 ± 0.06	/1/1/1/
2	72.5 ± 0.4	65.5 ± 0.3	22.4 ± 0.1	/3/2/1/
3	134.8 ± 0.7	139.7 ± 0.7	65.9 ± 0.3	/5/4/2/
4	195 ± 1	216 ± 1	124.2 ± 0.6	/7/6/3/

Table 5.19: applicationUpdate execution time

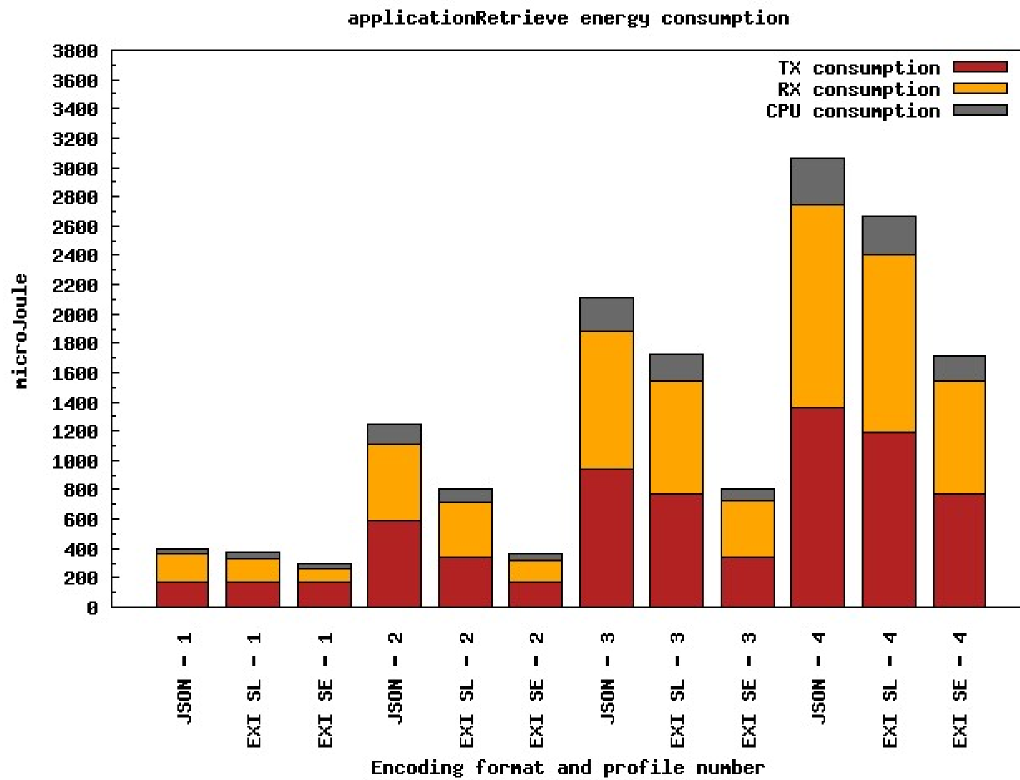


Figure 5.8: applicationRetrieve execution time

Profile	JSON	EXI	
		Schema-Less	Schema-Enabled
	bytes	bytes	bytes
1.	253	238 (-6%)	163 (-36%)
2.	1108	929 (-16%)	230 (-79%)
3.	2055	1823 (-11%)	1326 (-35%)
4.	2977	2738 (-8%)	2144 (-28%)

Table 5.20: applicationUpdate channel usage

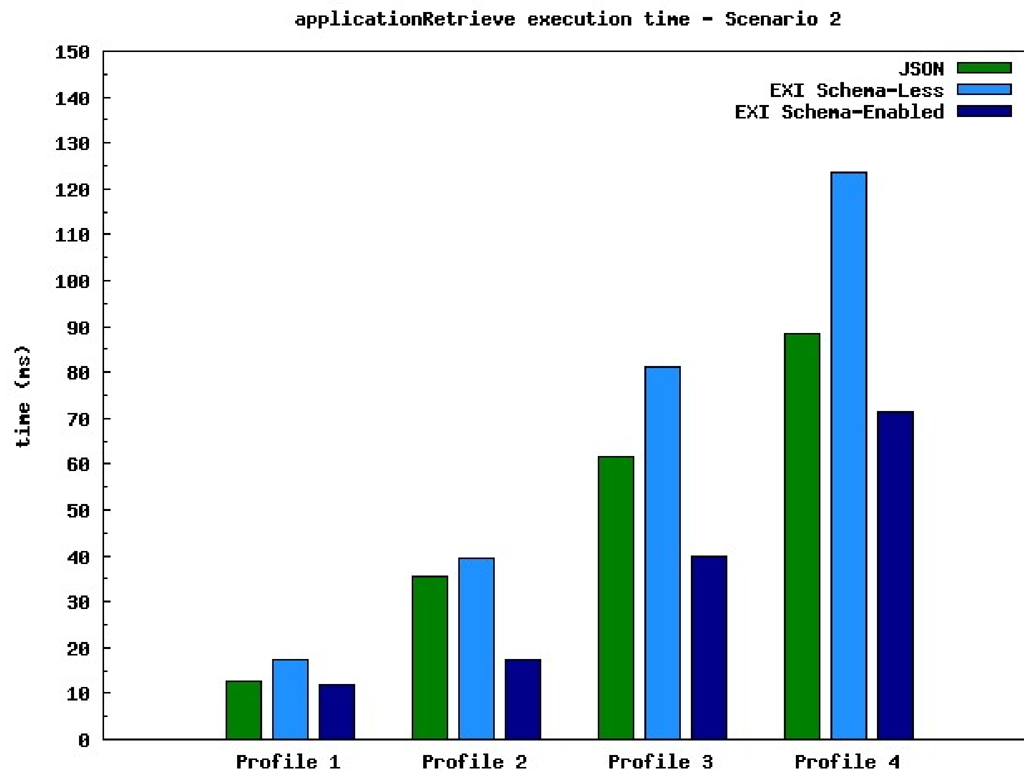


Figure 5.9: applicationRetrieve execution time - Scenario 2

Energy Type	JSON	EXI	
		Schema-Less	Schema-Enabled
	μJ	μJ	μJ
	Profile 1		
TOTAL	720 ± 4	643 ± 3	340 ± 2
TX	426 ± 2	341 ± 2	170.3 ± 0.9
RX	199 ± 1	171.3 ± 0.8	103.9 ± 0.5
CPU	94.4 ± 0.5	130.7 ± 0.6	65.3 ± 0.3
	Profile 2		
TOTAL	2760 ± 14	2008 ± 10	624 ± 3
TX	1618 ± 9	1107 ± 6	340 ± 2
RX	787 ± 4	553 ± 3	152 ± 0.8
CPU	352 ± 2	348 ± 2	131 ± 0.6
	Profile 3		
TOTAL	5150 ± 30	4380 ± 20	2019 ± 11
TX	3065 ± 15	2470 ± 10	1107 ± 6
RX	1404 ± 7	1146 ± 6	557 ± 3
CPU	677 ± 3	762 ± 4	356 ± 2
	Profile 4		
TOTAL	7520 ± 40	6780 ± 30	4020 ± 20
TX	4510 ± 20	3830 ± 20	2210 ± 10
RX	2029 ± 10	1765 ± 9	1142 ± 6
CPU	984 ± 5	1183 ± 6	660 ± 3

Table 5.21: applicationUpdate energy consumption

Figure 5.10 shows the execution times for the *applicationUpdate* operation.

Comparing Figure 5.10 with Figure 5.5 we can see that, as could be easily anticipated, the presence of a response payload decreases the difference between JSON and Schema-Less EXI, while the difference between JSON and Schema-Enable EXI increases. Of course this happens only because the Gateway is assumed to be a high-end system.

The channel usage gain obtained is inferior to the one reported in Table 5.17 because the two-way Blockwise transfer piggybacks the first block of the response in the Acknowledgement of the last block of the request.

The client energy consumption, which does not depend on whether the server has or has not limited computing resources, clearly shows that EXI Schema-Less performs better than JSON, and that EXI Schema-Enabled performs better than EXI Schema-Less.

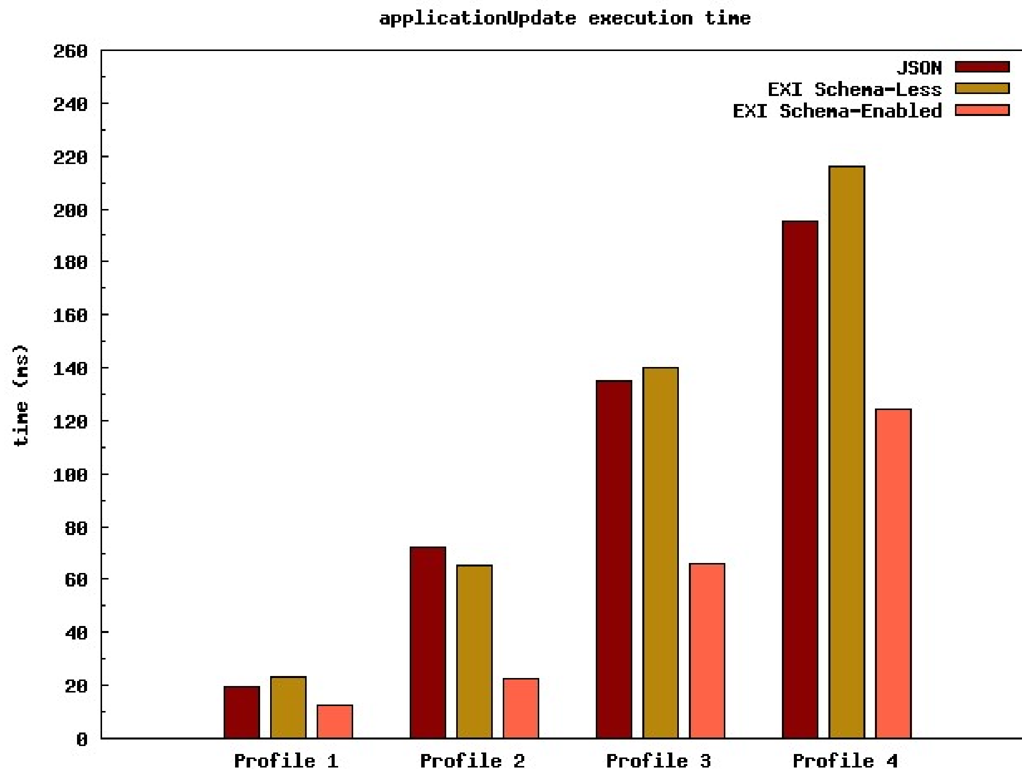


Figure 5.10: applicationUpdate execution time

5.12 subscription Create

The *subscriptionCreate* measurement is analogous to *applicationCreate*: the only difference lies in the different payload transmitted.

Profile	JSON	EXI		Number of blocks
		Schema-Less	Schema-Enabled	
	(ms)	(ms)	(ms)	
1	20.1 ± 0.1	23.8 ± 0.1	14.02 ± 0.07	/1/1/1/
2	32.5 ± 0.2	42.4 ± 0.2	18.95 ± 0.09	/2/2/1/
3	48.4 ± 0.2	60.8 ± 0.3	21.0 ± 0.1	/3/3/1/
4	61.2 ± 0.3	71.6 ± 0.3	28.7 ± 0.1	/4/3/1/

Table 5.22: subscriptionCreate execution time

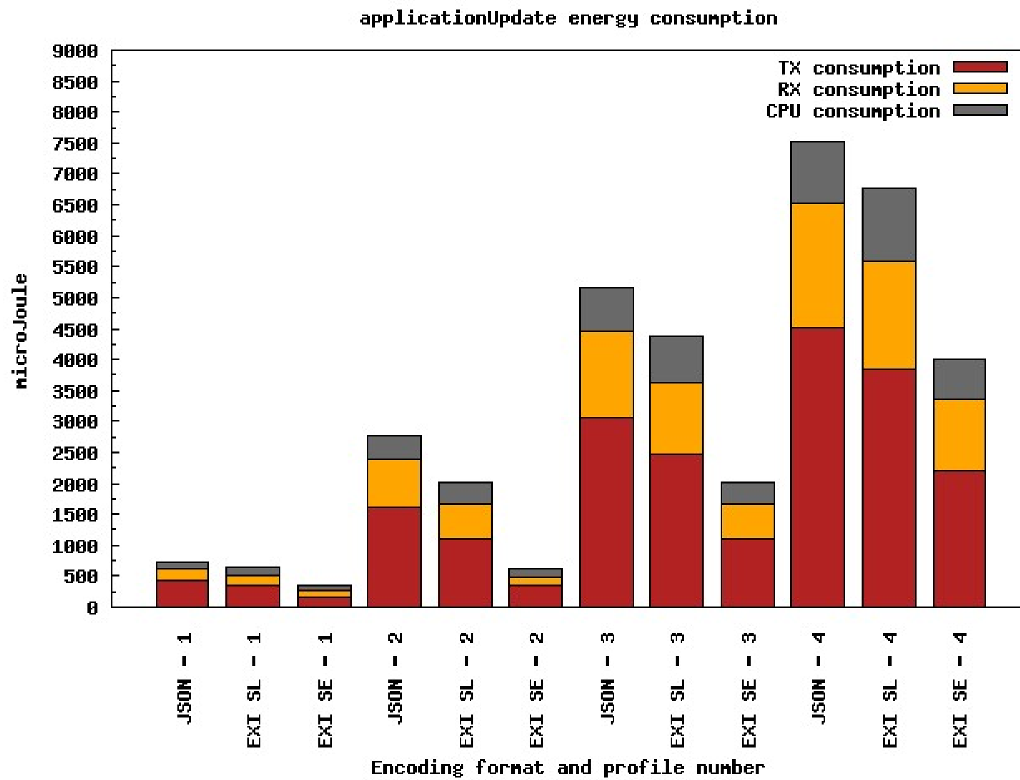


Figure 5.11: applicationUpdate energy consumption

Profile	JSON	EXI	
		Schema-Less	Schema-Enabled
	bytes	bytes	bytes
1.	265	251 (-5%)	174 (-34%)
2.	500	482 (-2%)	224 (-55%)
3.	745	682 (-8%)	233 (-69%)
4.	947	764 (-19%)	265 (-72%)

Table 5.23: subscriptionCreate channel usage

Energy Type	JSON	EXI	
		Schema-Less	Schema-Enabled
	μJ	μJ	μJ
	Profile 1		
TOTAL	728 ± 4	730 ± 4	438 ± 2
TX	426 ± 2	426 ± 2	255 ± 1
RX	201 ± 1	173.2 ± 0.9	102.3 ± 0.5
CPU	101.6 ± 0.5	130.7 ± 0.6	79.5 ± 0.4
	Profile 2		
TOTAL	1319 ± 7	1309 ± 6	569 ± 3
TX	936 ± 5	851 ± 4	341 ± 2
RX	186.1 ± 0.9	189.9 ± 0.9	119 ± 0.6
CPU	196 ± 1	269 ± 1	109.9 ± 0.5
	Profile 3		
TOTAL	1940 ± 10	1940 ± 10	603 ± 3
TX	1362 ± 7	1277 ± 6	340.6 ± 2
RX	279 ± 1	281 ± 1	138.6 ± 0.7
CPU	297 ± 1	384 ± 2	123.4 ± 0.6
	Profile 4		
TOTAL	2529 ± 12	2192 ± 11	786 ± 4
TX	1788 ± 9	1447 ± 7	426 ± 2
RX	372 ± 2	280 ± 1	201 ± 1
CPU	369 ± 2	464 ± 2	160 ± 0.8

Table 5.24: subscriptionCreate energy consumption

Figure 5.12 shows *subscriptionCreate* operation execution time.

From Figure 5.12 we can observe that the use of EXI Schema-Enabled consistently decreases the execution time with respect to JSON, due to the reduction in terms of the number of CoAP blocks. As for EXI Schema-Less, the overall behavior already registered for *applicationCreate* and shown in Figure 5.5 is confirmed.

The energy consumption is shown in Figure 5.13. Whereas the benefit of using EXI Schema-Enabled instead of JSON is clear, EXI Schema-Less suffers from the fact that JSON and EXI Schema-Enabled map the payload into an equal number of CoAP blocks for Profile 1, Profile 2 and Profile 3.

In this case EXI Schema-Less Profile 3 has a higher compression gain with respect to JSON, compared to the one measured for EXI Schema-Less Profile 3 for an < *application* > resource. Due to the mapping in terms of CoAP blocks, the same relationship does not hold when we consider the energy consumption gain with respect to JSON.

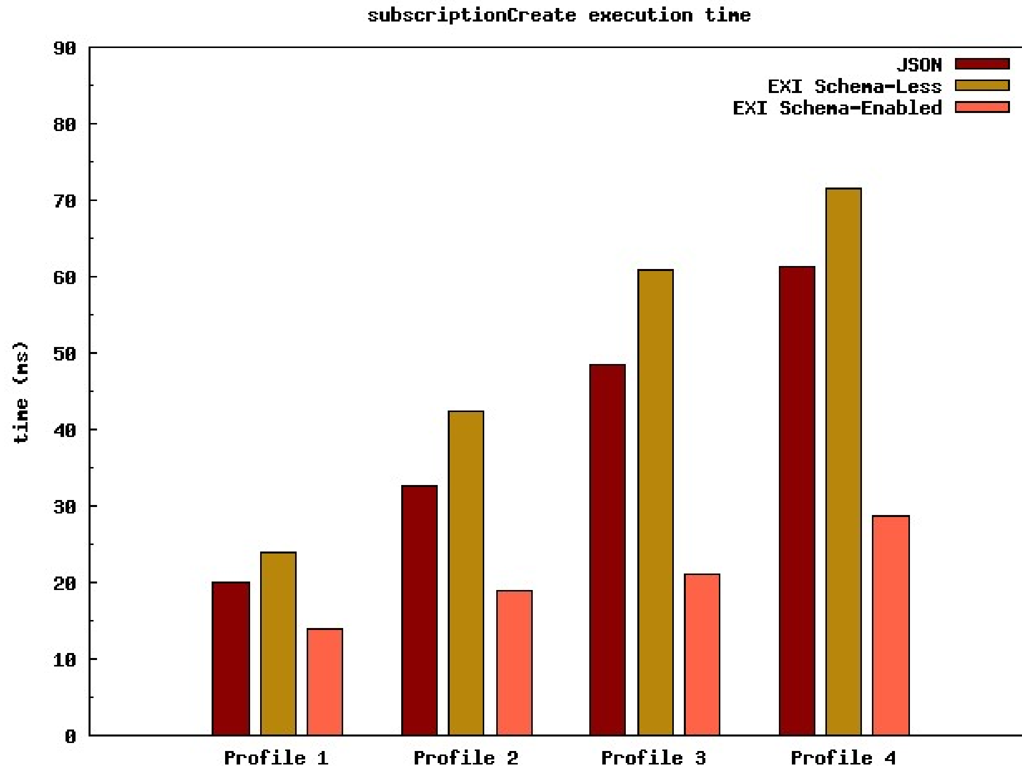


Figure 5.12: subscriptionCreate execution time

5.13 contentInstance Retrieve

As we did for the *applicationRetrieve* operation, also for the *contentInstanceRetrieve* operation we explore the case where the GSCL is hosted by a sensor node (Scenario 2), in addition to the first scenario where the server has high computational capabilities (Scenario 1).

Again, the channel usage and the energy consumption do not depend on whether the serialization time is negligible or not compared to the overall operation execution time.

From Table 5.27 we can see that as the profile number increases, the channel usage for EXI Schema-Less and JSON becomes comparable. This is due to the fact that for Profile 3 and Profile 4, JSON and EXI Schema-Less result in the same number of blocks.

This also affects the energy consumption: JSON and EXI Schema-Less have a similar energy consumption in both Profile 3 and Profile 4.

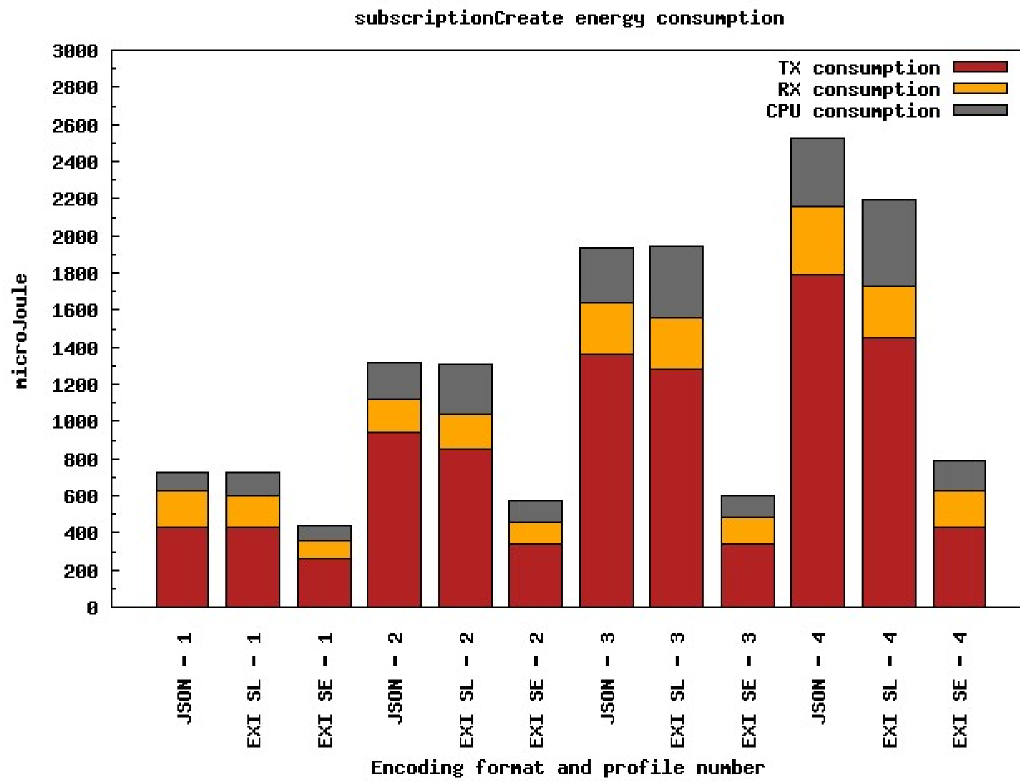


Figure 5.13: subscriptionCreate execution time

Profile	JSON	EXI		Number of blocks
		Schema-Less	Schema-Enabled	
	(ms)	(ms)	(ms)	
1	38.2 ± 0.2	27.6 ± 0.1	12.79 ± 0.06	/3/2/1/
2	50.9 ± 0.3	40.3 ± 0.2	25.5 ± 0.1	/4/3/2/
3	65.4 ± 0.3	65.5 ± 0.3	40.1 ± 0.2	/5/5/3/
4	81.6 ± 0.4	80.4 ± 0.4	65.6 ± 0.3	/6/6/5/

Table 5.25: contentInstanceRetrieve execution time - Scenario 1

Profile	JSON	EXI		Number of blocks
		Schema-Less	Schema-Enabled	
	(ms)	(ms)	(ms)	
1	38.6 ± 0.2	43.7 ± 0.2	18.7 ± 0.1	/3/2/1/
2	51.4 ± 0.2	60.9 ± 0.3	36.1 ± 0.2	/4/3/2/
3	66.1 ± 0.3	92.9 ± 0.5	57.5 ± 0.3	/5/5/3/
4	82.6 ± 0.4	116.7 ± 0.6	91.9 ± 0.4	/6/6/5/

Table 5.26: contentInstanceRetrieve execution time - Scenario 2

Profile	JSON	EXI	
		Schema-Less	Schema-Enabled
	bytes	bytes	bytes
1.	604	431 (-29%)	195 (-68%)
2.	809	636 (-21%)	400 (-51%)
3.	1040	1060 (+2%)	632 (-39%)
4.	1297	1278 (-1%)	1042 (-20%)

Table 5.27: contentInstanceRetrieve channel usage

Energy Type	JSON	EXI	
		Schema-Less	Schema-Enabled
	μJ	μJ	μJ
	Profile 1		
TOTAL	1445 ± 7	1019 ± 6	468 ± 2
TX	766 ± 4	511 ± 3	255 ± 1
RX	527 ± 3	407 ± 2	166.3 ± 0.8
CPU	152.5 ± 0.8	101.6 ± 0.5	45.8 ± 0.3
	Profile 2		
TOTAL	1932 ± 10	1507 ± 8	957 ± 5
TX	1022 ± 5	766 ± 4	511 ± 3
RX	708 ± 4	588 ± 3	347 ± 2
CPU	203 ± 1	152 ± 0.8	99.0 ± 0.5
	Profile 3		
TOTAL	2470 ± 12	2512 ± 13	1499 ± 7
TX	1277 ± 6	1277 ± 6	766 ± 4
RX	938 ± 5	981 ± 5	580 ± 3
CPU	254 ± 1	254 ± 1	152.5 ± 0.8
	Profile 4		
TOTAL	3059 ± 15	3022 ± 15	2474 ± 12
TX	1533 ± 8	1532 ± 8	1277 ± 6
RX	1221 ± 6	1184 ± 6	943 ± 5
CPU	305 ± 2	305 ± 2	254 ± 1

Table 5.28: contentInstanceRetrieve energy consumption

Scenario 1

We can observe from Figure 5.14 that for Profile 3 and Profile 4 the operation execution time for EXI and JSON is comparable, whereas in analogous situations such as Figure 5.7 the advantage of using EXI Schema-Less in a Scenario 1 type of situation is clear.

Scenario 2

In scenario 2, as we can see from Figure 5.16, using EXI Schema-Enabled does not bring a benefit from the point of view of the operation execution time any more. This is the only situation where this happens in the message benchmark we have considered, and is due to the fact that EXI performs worse in a situation where the message content is larger.

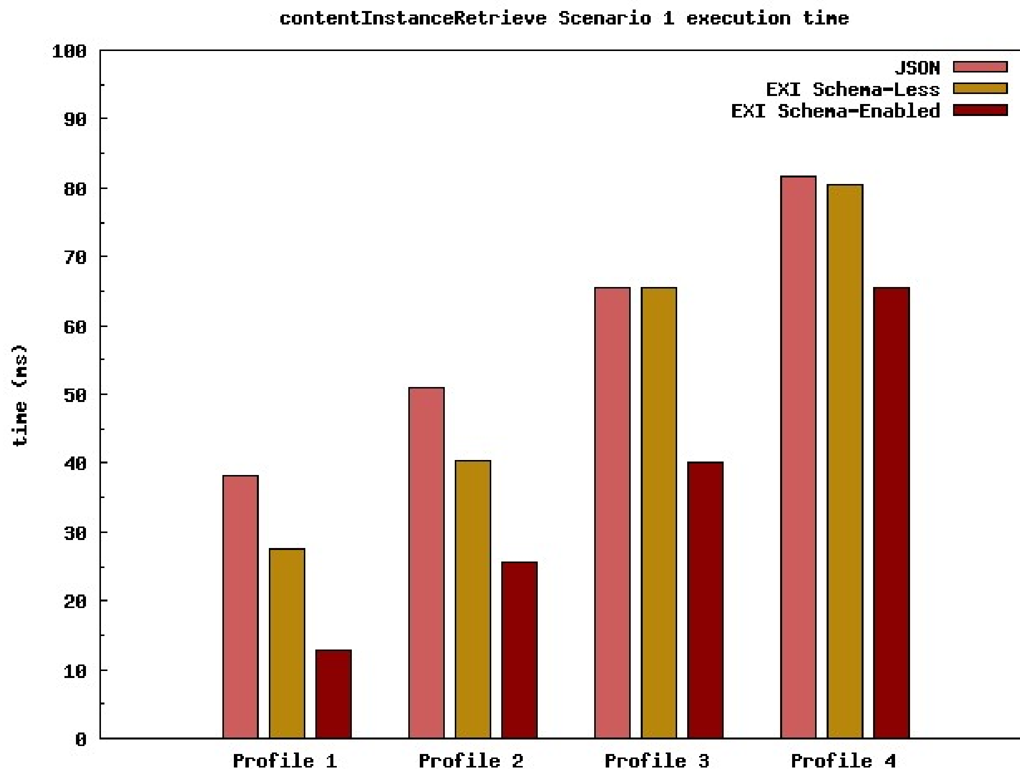


Figure 5.14: contentInstanceRetrieve execution time - Scenario 1

5.14 Comparison between GET and POST performances

By comparing Figure 5.5 and Figure 5.7 we can observe that POST execution suffers from an overall delay with respect to GET execution. Looking at the packets transmitted with a packet analyzer tool we can see that using a block size of 64 bytes results in a frame that is larger than 127 bytes and therefore is subject to 6LowPAN fragmentation into two frames. The reason why this happens in the POST and not in the GET is that the POST request carries a "Uri-Path" Option for the resource for a total overhead of 33 bytes (25 bytes Uri-Path string plus additional overhead). Using blocks of 32 bytes does not avoid fragmentation, as can be verified in Table 5.29: the first CoAP payload size that avoids fragmentation is 16 bytes. There may be situations when fragmentation at IP level is not available, since not all 6LoWPAN devices have enough resources to perform it: if this were true in our POST example, CoAP blocks of 16 bytes would be the only option.

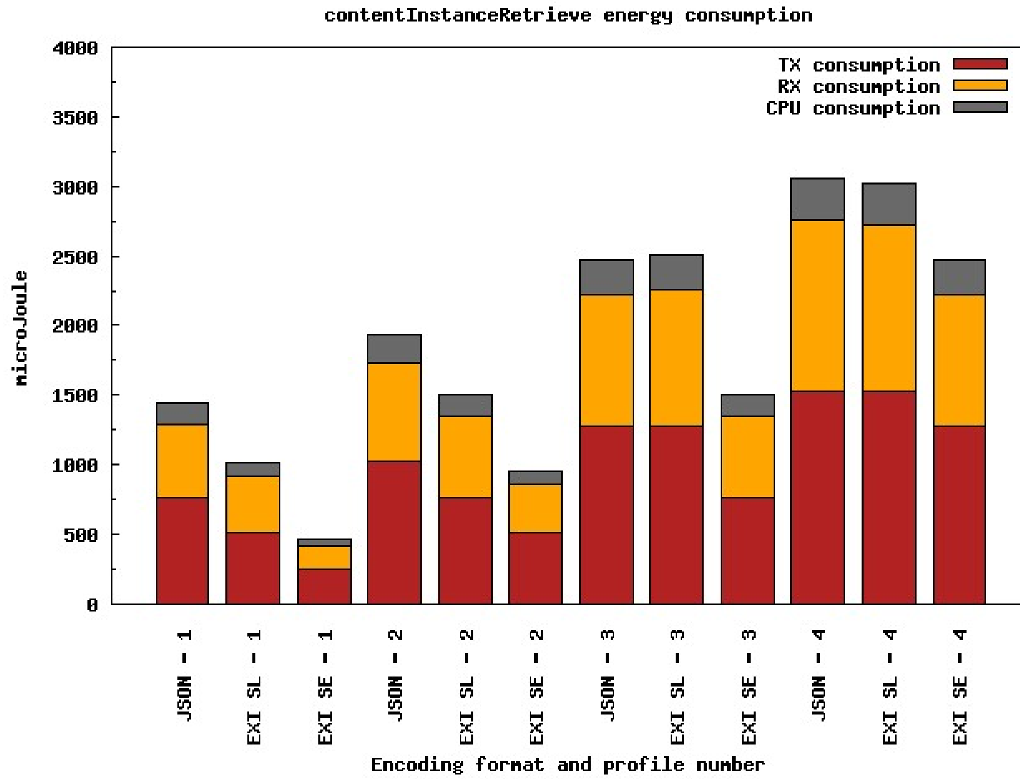


Figure 5.15: contentInstanceRetrieve energy consumption - Scenario 1

Profile	JSON	EXI		Number of blocks
		Schema-Less	Schema-Enabled	
	(ms)	(ms)	(ms)	
1.	24.5 ± 0.1	28.9 ± 0.1	12.0 ± 0.05	/2/2/1/
2.	65.8 ± 0.3	65.8 ± 0.3	28.7 ± 0.1	/5/4/2/
3.	106.6 ± 0.5	137.4 ± 0.7	66.2 ± 0.3	/9/8/4/
4.	190.8 ± 0.9	210 ± 1	114.7 ± 0.6	/14/12/7/

Table 5.29: applicationCreate execution time, 100% PDR - 32 bytes blocks

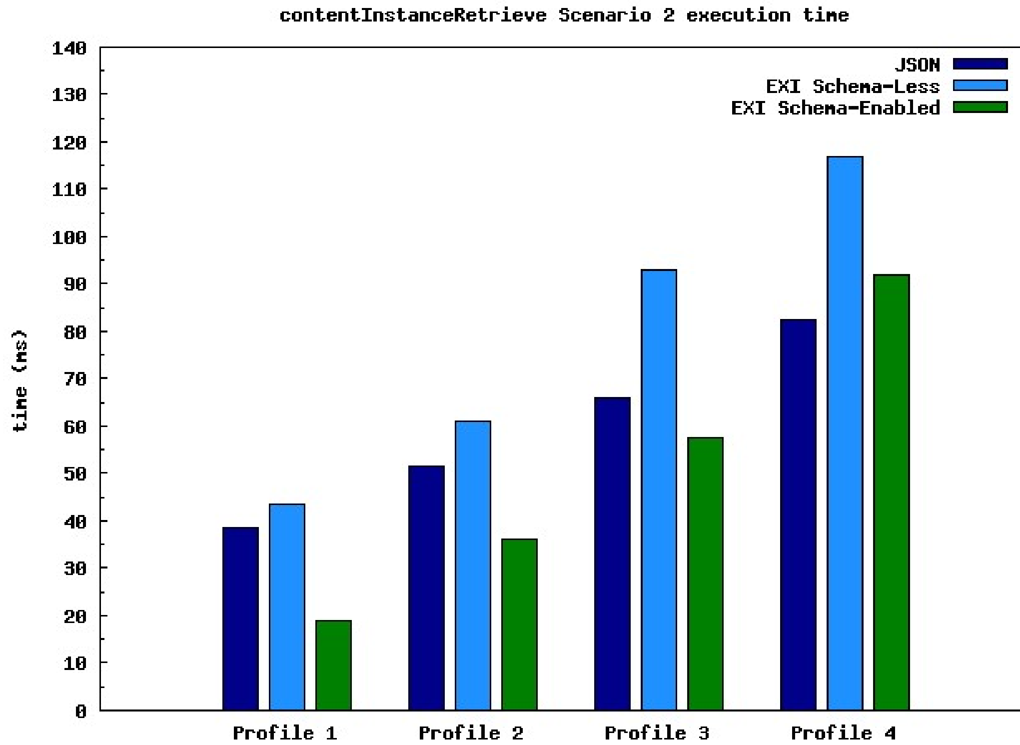


Figure 5.16: contentInstanceRetrieve execution time - Scenario 2

Profile	JSON	EXI		Number of blocks
		Schema-Less	Schema-Enabled	
	(ms)	(ms)	(ms)	
1.	40.5 ± 0.2	35.9 ± 0.2	12 ± 0.05	/4/3/1/
2.	90.7 ± 0.4	84.5 ± 0.4	35.8 ± 0.2	/15/7/3/
3.	149.2 ± 0.7	179.1 ± 0.9	93.7 ± 0.5	/18/15/8/
4.	269.75 ± 1.4	283.8 ± 1.4	150.7 ± 0.7	/27/23/13/

Table 5.30: applicationCreate execution time, 100% PDR - 16 bytes blocks

From this example we can argue that the presence of long Uri-Path Options in every instance of a POST request which also carries a payload has a negative effect on the performance of the system.

POST operations are executed very often, for example when sensors register their reading at the Local SCL creating a new <contentInstance> resource.

In our opinion this is an issue for integrating M2M over constrained networks: ETSI M2M builds its resources using particularly long URIs, and CoAP

does not offer a mechanism for not repeating the URI-path multiple times during Block1 Blockwise transfers.

From an ETSI M2M standpoint, the need for URIs with a complex structure is understandable and is related to the requirements of flexibility and scalability. At the same time, reducing the length of collection names, which are included unchanged in URIs would be beneficial to the operation performance.

Even if the ETSI M2M Technical Committee were willing to change the architecture of REST resources identifiers introducing such optimizations, a higher degree of flexibility from CoAP is desirable.

A solution from the CoAP side could be that of making the option of not repeating the URI-path for Block1 Blockwise transfers an object of negotiation between client and server nodes. Whenever the server implements the POST operation in an atomic way, communication at CoAP level is not stateless and implementing this option would only result in an additional state. If the operation is not atomic, the server may be willing to create a state in order to save energy globally at a network level depending on the internal resource availability. The power and time saving would be not only limited to transmission and reception, but also to fragmentation both at CoAP level and 6LoWPAN levels, and retransmissions.

From the point of view of CoAP, this issue was raised in 2012. A solution that was proposed consisted in returning a temporary resource in the first POST response, and in returning the final URI only at the end of the Blockwise transfer, with the general recommendation of keeping URIs short, which in turn pushed the issue back to the higher application layers. Still, it could be a temporary workaround that could be considered, if not by ETSI M2M, from industry-specific bodies such as the Open Mobile Alliance (OMA), which has been working on a lightweight version of M2M since the end of Q4 2013 (OMA LWM2M).

5.15 Performance in lossy configurations

5.15.1 applicationCreate POST with 95% PDR

Figure 5.17 shows the execution time of the *applicationCreate* operation in a lossy configuration. We can observe that EXI Schema-Less always performs better than JSON and EXI Schema-Less. EXI Schema-Less performs better than JSON whenever it maps the message in an inferior number of CoAP messages. The impact packet losses on the execution time is relevant: this is due to the use of default CoAP transmission parameters.

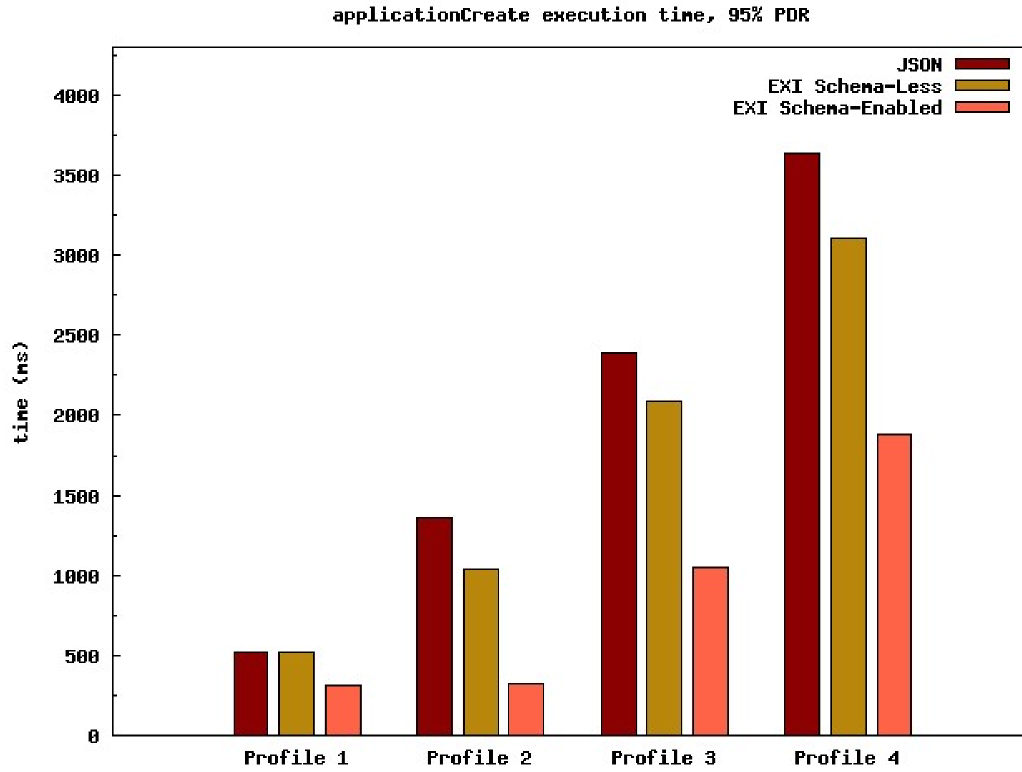


Figure 5.17: applicationCreate execution time, 95% PDR

Figure 5.17 shows the client energy consumption for a *applicationCreate* operation. EXI Schema-Enabled always performs better than JSON and EXI Schema-Less, and EXI Schema-Less always performs better than JSON.

Profile	JSON	EXI		Number of blocks
		Schema-Less	Schema-Enabled	
	(ms)	(ms)	(ms)	
1	516 ± 20	515 ± 20	312 ± 14	/1/1/1/
2	1357 ± 40	1039 ± 30	317 ± 20	/3/2/1/
3	2392 ± 60	2086 ± 60	1047 ± 30	/5/4/2/
4	3633 ± 100	3108 ± 80	1878 ± 50	/7/6/3/

Table 5.31: applicationCreate GET time with 95% PDR

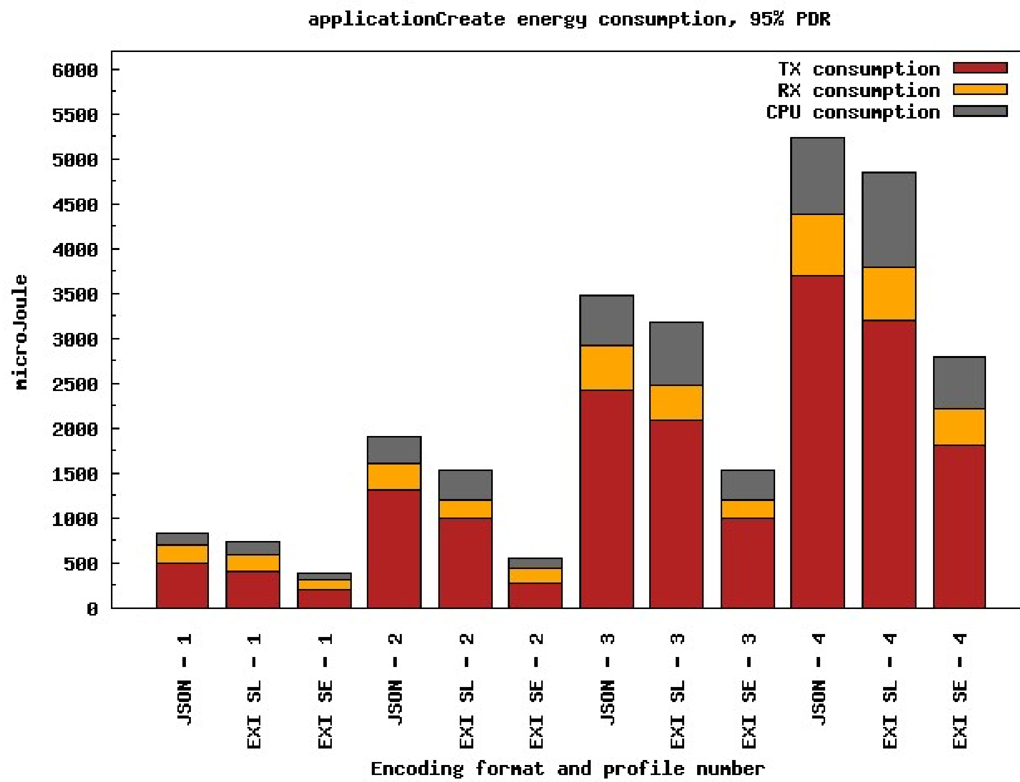


Figure 5.18: applicationCreate energy consumption, 95% PDR

Energy Type	JSON	EXI	
		Schema-Less	Schema-Enabled
	μJ	μJ	μJ
	Profile 1		
TOTAL	824 ± 4	742 ± 4	381 ± 2
TX	500 ± 3	412 ± 3	198 ± 1
RX	210 ± 1	180.2 ± 0.9	109 ± 0.5
CPU	114 ± 0.7	148.5 ± 0.7	73.6 ± 0.4
	Profile 2		
TOTAL	1912 ± 9	1535 ± 8	558 ± 3
TX	1311 ± 7	1001 ± 5	285 ± 2
RX	297 ± 1	197 ± 1	162.3 ± 0.8
CPU	304 ± 2	336 ± 2	110.9 ± 0.6
	Profile 3		
TOTAL	3474 ± 20	3180 ± 16	1541 ± 8
TX	2426 ± 10	2090 ± 10	1004 ± 5
RX	497 ± 2	392 ± 2	198 ± 1
CPU	550 ± 3	696 ± 3	338 ± 2
	Profile 4		
TOTAL	5231 ± 30	4850 ± 20	2796 ± 14
TX	3700 ± 20	3193 ± 16	1816 ± 10
RX	695 ± 3	592 ± 3	396 ± 2
CPU	835 ± 4	1065 ± 5	584 ± 3

Table 5.32: applicationCreate energy consumption, 95% PDR

5.16 Summary of Results

This section summarizes the results obtained so far. The three data serialization methods considered are placed in relative order whenever possible. Situations which, based on the benchmark we chose, need to be evaluated singularly are marked with a line (“-”).

Criteria	JSON	EXI Schema Less	EXI Schema Enabled
memory occupation			
ROM	1	2	3
RAM	1	2	3
applicationCreate			
Execution Time	2	3	1
Energy Consumption	3	2	1
Channel Usage	3	2	1
applicationCreate - lossy network			
Execution Time	-	-	1
Energy Consumption	3	2	1
applicationRetrieve			
Execution Time Sc. 1	3	2	1
Execution Time Sc. 2	2	3	1
Energy Consumption	3	2	1
Channel Usage	3	2	1
applicationUpdate			
Execution Time	-	-	1
Energy Consumption	3	2	1
Channel Usage	3	2	1
subscriptionCreate			
Execution Time	2	3	1
Energy Consumption	-	-	1
Channel Usage	3	2	1
contentInstanceRetrieve			
Execution Time Sc. 1	3	2	1
Execution Time Sc. 2	-	3	-
Energy Consumption	-	-	1
Channel Usage	-	-	1

Table 5.33: Summary of Results

From the measurements made, it emerges that:

1. EXI Schema-Enabled allows to achieve better performance in terms of energy consumption, compared to JSON and EXI Schema-Less. In terms of execution time, EXI Schema-Enabled performs better in the vast majority of cases. This is possible at the cost of a higher memory occupancy, moreover endpoints need to share grammar structures.

2. The use of EXI Schema-Less achieves better or equal performance to JSON from the point of view of the energy consumption. In our implementation, EXI Schema-Less execution time is better than JSON when serialization time is negligible, while otherwise the opposite is true. EXI Schema-Less performs better than JSON in terms of execution time also in the configurations considered at 95% PDR.

5.17 Further Optimizations

As a next step, we intend to optimize the current implementation, in order to exploit the parallelism between message serialization and transmission for the EXI format. The constraints imposed by such optimization on the system (e.g. on the class of ETSI M2M security algorithms and procedures that can be supported), should be assessed.

Considering the *Low Power Mode* component in our measurements, in the example of *applicationCreate* operation, we expect a 10% improvement for the time performances of both EXI Schema-Less and EXI Schema-Enabled.

For 95% PDR, we expect to have improvements in time performances in the order of at most 1%.

Our results fully justify the development of an optimized communication module, the most suitable platform to perform a fair comparison between the three serialization methods considered.

5.18 Comparison with the state-of-the-art

Regarding the works presented in Chapter 3, we can conclude that:

- *libEXI*[17] and EXIP have a similar ROM occupancy in the Schema-Less case (respectively 48 KB and 43 KB for Schema-Enabled EXI). Since ROM occupancy is highly dependent on the dimension of the grammar structures, a comparison with the Schema-Enabled ROM occupancy cannot be performed. *libEXI* RAM consumption is lower than the one we registered with EXIP (1.7 Kb vs. 8 KB for Schema-Less EXI). This would make our implementation not portable on a *TelosB* mote, which only has 8 KB of RAM, unless further optimizations are made. It should be noted that *libEXI* is not XML-less, so the memory occupancy of a parser needs to be taken into account for an effective comparison.

- *EIGEN*[18] has a considerably lower ROM occupancy than *libEXI* (13 KB vs. 43 KB or 45 KB), which is due to that fact that grammars are

implicitly present in the generated code. In our case ROM is not the limiting factor, and for this reason this approach has not been explored. EIGEN has not been considered a viable option as EXI library because it is not freely available.

5.19 Fulfillment of ICSI time constraints

From the measured data, we can see that the ICSI time constraint of a 10^{-2} s time scale for a *local scope* as expressed in Table 4.1 can not always be granted for the message payloads considered, but only for small messages. While the use of EXI Schema-Less does not dramatically alter the time performance registered with JSON in terms of execution time, the use of EXI Schema-Enabled consistently increases the complexity of the payload that can be sent while still enforcing the 10^{-2} s time constraint.

Chapter 6

Conclusions

This is the first extensive experimental work done on the ETSI M2M messaging benchmark having adopted a reference set of serialization methodologies.

The effectiveness of porting the ETSI Machine-To-Machine protocol on constrained access networks following the Internet of Things paradigm is still debated in the scientific community. In the present work we have assessed the impact of the serialization method in a Wireless Sensor Network. The experimental work has been conducted through simulation in order to ease the execution of similar performance assessments on different platforms. We have made use of the *Wismote* emulated chipset integrated in the *Cooja* simulation tool.

From the present work we can conclude the following:

- *ETSI M2M operations execution time*: our results show that EXI Schema-Enabled performs better than JSON in most configurations considered, and always better than EXI Schema-Less. JSON performs better than EXI Schema-Less whenever the encoding time is not negligible, whereas JSON performs worse than EXI Schema-Less in the opposite case. In the configuration of lossy network, EXI Schema-Less performs better than JSON even when the serialization time is not negligible.

- *Channel usage*: in all scenarios considered the channel usage adopting EXI is less than in the case of JSON, especially when EXI Schema-Enabled is used.

- *Energy consumption*: EXI Schema-Enabled outperforms JSON and EXI Schema-Less in all configurations considered. EXI Schema-Less outperforms JSON in most situations considered and achieves comparable results in the rest.

From the results of our experiments, the performance of EXI Schema-Enabled reveals that it is the most effective choice of data serialization over constrained wireless systems following the IoT paradigm. Similarly, EXI Schema-Less registers interesting results especially from the points of view of energy consumption and of channel usage.

We envision that these results could be beneficial to researchers within the scientific community.

6.1 Future direction

As a follow up to this work we propose:

- The development of an automated engine capable of producing EXI Streams from a given custom XML Schema or equivalent. A pre-processor should replace the custom library we implemented for each message considered in our experimental work; indeed this would drastically reduce the adaptation effort in order to customize the libraries in a novel implementation.
- Further optimizations, according to the considerations discussed in Section 5.17.
- From a software engineering perspective it is recommended to test the validation environment prior to executing actual tests. It would therefore be worthwhile to test the CoAP environment before executing the test suite to avoid any bias in the testing results.

The result of this work will be adopted for the implementation of the ITS architecture designed within the ICSI Project approved by the European Commission under the grant #317671.

Ringraziamenti

E' con grande piacere che ringrazio i miei relatori, Piero Castoldi, Paolo Pagano e Femi Aderohunmu, per aver reso possibile il mio lavoro di tesi, e per essere stati così disponibili durante questi mesi. Tutto il gruppo di ricerca è stato preziosissimo, ed in particolare Daniele Alessandrelli, Andrea Azzarà, Matteo Pertracca e Stefano Bocchino. Un grazie a tutti voi.

Grazie a tutte le persone che mi sono state vicine in questi anni, a partire dai miei compagni di Fisica Else, Alessia, Leonardo, Davide, Valerio e Daniele. Gli anni dell'università sono stati fantastici, e condividerli con tutti voi è stato davvero speciale! Grazie a Gian, Francesca, Simone, Daniele, Davide, Sina, Filippo, Angela, Francesco, Daniele, Bob e Tudor.

Grazie a tutti i compagni del lab e ad Alessandra per i nostri appuntamenti fissi del mercoledì!

Grazie a Valentina, Celeste e Silvia per la vostra amicizia! Siete davvero insostituibili.

Grazie a Federico, sei il migliore vicino di sempre!

Thank you to my canadian friends Anand, Levent, Kevin, Meron, Tseganesh, and Michael, Taras, Dennis and Reena for the awesome experience, I hope to see all of you again very soon.

Else, non ci sono parole! Grazie di tutto.

Thank you John for being with me.

Infine un grazie immenso ai miei genitori, Kathy e Arturo, ad Alessandro, Casimiro e Anna per il vostro affetto e sostegno incondizionato.

Appendices

Appendix A

DATEX2 notification

Car Park Slot occupation

```
<?xml version="1.0" encoding="UTF-8"?>
<d2LogicalModel xmlns="http://datex2.eu/schema/2/2_0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 modelBaseVersion="2">
   <exchange>
      <supplierIdentification>
         <country>it</country>
         <nationalIdentifier>CNIT_WSN</nationalIdentifier>
      </supplierIdentification>
   </exchange>
   <payloadPublication xsi:type="SituationPublication" lang="it">
      <publicationTime>2006-09-28T16:00:00+01:00</publicationTime>
      <publicationCreator>
         <country>it</country>
         <nationalIdentifier></nationalIdentifier>
      </publicationCreator>
      <situation id="GUID2A22530C-D452-4ae8-B942-993BC2923D13"
          version="1">
         <headerInformation>
            <confidentiality>noRestriction</confidentiality>
            <informationStatus>real</informationStatus>
            <urgency>normalUrgency</urgency>
         </headerInformation>
         <situationRecord xsi:type="CarParks"
         id="GUID2A22530C-D452-4ae8-B942-993BC2923D14"
             version="1">
            <situationRecordCreationTime>
              2006-09-28T16:00:00+01:00
            </situationRecordCreationTime>
            <situationRecordVersionTime>2006-09-28T16:05:00+00:00
            </situationRecordVersionTime>
            <situationRecordFirstSupplierVersionTime>
```

```
             2006-09-28T16:05:00+00:00
            </situationRecordFirstSupplierVersionTime>
            <probabilityOfOccurrence>certain</probabilityOfOccurrence>
            <validity>
               <validityStatus>
                definedByValidityTimeSpec
               </validityStatus>
               <validityTimeSpecification>
                  <overallStartTime>2006-10-17T14:00:00+02:00
                  </overallStartTime>
                  <overallEndTime>2006-10-17T16:00:00+02:0
                  0</overallEndTime>
               </validityTimeSpecification>
            </validity>
            <groupOfLocations xsi:type="Point"></groupOfLocations>
            <carParkConfiguration>singleLevel</carParkConfiguration>
            <carParkIdentity></carParkIdentity>
            <carParksExtension>
               <SlotData occupied="true" id="324"/>
            </carParksExtension>
         </situationRecord>
      </situation>
   </payloadPublication>
</d2LogicalModel>
```

Appendix B

<application> profiles

Profile 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<p0:application xmlns:p0="http://uri.etsi.org/m2m">
</p0:application>
```

Profile 2.

```
<?xml version="1.0" encoding="UTF-8"?>
<p0:application xmlns:p0="http://uri.etsi.org/m2m">
   <p0:accessRightID>accessRight</p0:accessRightID>
   <p0:searchStrings>
      <p0:searchString>searchString</p0:searchString>
   </p0:searchStrings>
</p0:application>
```

Profile 3.

```
<?xml version="1.0" encoding="UTF-8"?>
<p0:application xmlns:p0="http://uri.etsi.org/m2m">
   <p0:accessRightID>accessRight</p0:accessRightID>
   <p0:searchStrings>
      <p0:searchString>searchString</p0:searchString>
   </p0:searchStrings>
   <p0:aPoC>apoc_</p0:aPoC>
   <p0:groupsReference>
    /gsclBase/applications/app/groups
   </p0:groupsReference>
   <p0:accessRightsReference>
    /gsclBase/applications/appaccessRights
   </p0:accessRightsReference>
</p0:application>
```

Profile 4.

```
<?xml version="1.0" encoding="UTF-8"?>
<p0:application xmlns:p0="http://uri.etsi.org/m2m">
   <p0:accessRightID>accessRight</p0:accessRightID>
   <p0:searchStrings>
      <p0:searchString>searchString</p0:searchString>
   </p0:searchStrings>
   <p0:aPoC>apoc_</p0:aPoC>
   <p0:groupsReference>
    /gsclBase/applications/app/groups
   </p0:groupsReference>
   <p0:accessRightsReference>
    /gsclBase/applications/appaccessRights
   </p0:accessRightsReference>
   <p0:subscriptionsReference>
    /gsclBase/applications/app/subscriptions
   </p0:subscriptionsReference>
   <p0:notificationChannelsReference>
    /gsclBase/applications/app/notifications
   </p0:notificationChannelsReference>
</p0:application>
```

Appendix C

<subscription> profiles

Profile 1.


```
<?xml version="1.0" encoding="UTF-8"?>
<p0:subscription xmlns:p0="http://uri.etsi.org/m2m">
</p0:subscription>
```


Profile 2.

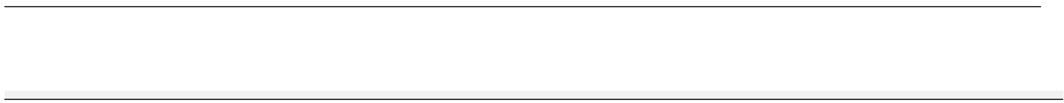

```
<?xml version="1.0" encoding="UTF-8"?>
<p0:subscription xmlns:p0="http://uri.etsi.org/m2m">
   <p0:expirationTime>
    2012-07-31T13:33:55.000839</p0:expirationTime>
</p0:subscription>
```


Profile 3.


```
<?xml version="1.0" encoding="UTF-8"?>
<p0:subscription xmlns:p0="http://uri.etsi.org/m2m">
   <p0:expirationTime>
    2012-07-31T13:33:55.000839</p0:expirationTime>
   <p0:delayTolerance>
    2012-08-31T13:33:55.000839</p0:delayTolerance>
</p0:subscription>
```


Profile 4.


```
<?xml version="1.0" encoding="UTF-8"?>
<p0:subscription xmlns:p0="http://uri.etsi.org/m2m">
   <p0:expirationTime>
    2012-07-31T13:33:55.000839</p0:expirationTime>
   <p0:delayTolerance>
    2012-07-31T13:33:55.000839</p0:delayTolerance>
   <p0:contact>coap://DA\IP_Addr:Port/da_notif</p0:contact>
</p0:subscription>
```



Appendix D

<contentInstance> profiles

Profile 1.


```
<?xml version="1.0" encoding="UTF-8"?>
<p0:contentInstance xmlns:p0="http://uri.etsi.org/m2m">
   <p0:lastModifiedTime>
    2012-07-31T00:01:56.000839</p0:lastModifiedTime>
   <p0:content>{
   1200234523454
   1240365423427}
   </p0:content>
</p0:contentInstance>
```


Profile 2.


```
<?xml version="1.0" encoding="UTF-8"?>
<p0:contentInstance xmlns:p0="http://uri.etsi.org/m2m">
   <p0:lastModifiedTime>
    2012-07-31T00:01:56.000839</p0:lastModifiedTime>
   <p0:content>{
      "application":{
   1200234523454
   1240365423427
   1330187635432
   1345109843751
   1355287165632
   1405173646721}
   </p0:content>
</p0:contentInstance>
```


Profile 3.


```
<?xml version="1.0" encoding="UTF-8"?>
<p0:contentInstance xmlns:p0="http://uri.etsi.org/m2m">
   <p0:lastModifiedTime>
```

```
    2012-07-31T00:01:56.000839</p0:lastModifiedTime>
   <p0:content>{
   1200234523454
   1240365423427
   1330187635432
   1345109843751
   1355287165632
   1405173646721
   1510438484381
   1830383838381
   1950325216232
   2010384374371
   2135843784375
   2140484389434}
   </p0:content>
</p0:contentInstance>
```

Profile 4.

```
<?xml version="1.0" encoding="UTF-8"?>
<p0:contentInstance xmlns:p0="http://uri.etsi.org/m2m">
   <p0:lastModifiedTime>
    2012-07-31T00:01:56.000839</p0:lastModifiedTime>
   <p0:content>{
   1200234523454
   1240365423427
   1330187635432
   1345109843751
   1355287165632
   1405173646721
   1510438484381
   1830383838381
   1950325216232
   2010384374371
   2135843784375
   2140484389434
   0010474743743
   0020437843784
   0130438843846
   0140484848434
   0230948444748
   0315294848432
   0512747474747
   0530474374364}
   </p0:content>
</p0:contentInstance>
```

Acronyms

CoAP Constrained Application Protocol.

CoRE Constrained RESTful Environment.

DA Device Application.

DSCL Device SCL.

DTLS Datagram Transport Layer Security.

ETSI European Telecommunications Standards Institute.

EXI Efficient XML Interchange.

EXIP Embeddable EXI Processor.

GA Gateway Application.

GSCL Gateway SCL.

ICSI Intelligent Cooperative Sensing for Improved Traffic Efficiency.

IEEE Institute of Electrical and Electronics Engineers.

IETF Internet Engineering Task Force.

ITS Intelligent Transport System.

M2M Machine-To-Machine.

NA Network Application.

NSCL Network SCL.

OBU On Board Unit.

PDR *Packet Delivery Rate*.

RPL Routing Protocol for Low Power and Lossy Networks.

RSU Road Side Unit.

SCL Service Capability Layer.

W3C World Wide Web Consortium.

Bibliography

[1] Luis Coetzee, Johan Eksteen, *"The Internet of Things - Promise for the Future? An Introduction"* in IST-Africa Conference Proceedings, May 2011.

[2] *Datagram Transport Layer Security*, RFC 4347 [Online]. Available: http://tools.ietf.org/html/rfc4347 Last Access: June 12, 2014.

[3] *Constrained RESTful Environments (CoRE) Link Format* RFC 6690" [Online]. Available: http://tools.ietf.org/html/rfc6690 Last Access: June 12, 2014.

[4] *Constrained Application Protocol (CoAP) draft-ietf-core-coap-18* [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-core-coap/ Last Access: June 12, 2014.

[5] *Observeing Resources in CoAP draft-ietf-core-observe-13* [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-core-observe/ Last Access: June 12, 2014.

[6] *Blockwise transfers in CoAP draft-ietf-core-block-14* [Online]. Available: http://tools.ietf.org/html/draft-ietf-core-block-14 Last Access: June 12, 2014.

[7] Michael Jackson *Problem Frames: Analysing & Structuring Software Development Problems* Addison-Wesley Professional (November 12, 2000).

[8] ETSI TS 103 104 V1.1.1 *Machine-to-Mchine communications (M2M); Interoperability Test Specification for CoAP Binding of ETSI M2M Primitives*. 2013-04 [Online]. Available: http://www.etsi.org/deliver/etsi_ts/103100_103199/

103104/01.01.01_60/ts_103104v010101p.pdf Last Access: June 12, 2014.

[9] ETSI TS 102 898 V1.1.1 *Machine-to-Mchine communications (M2M); Use Cases of Automotive Applications in M2M capable networks.* 2013-04 [Online]. Available: http://www.etsi.org/deliver/etsi_tr/102800_102899/102898/01.01.01_60/tr_102898v010101p.pdf Last Access: June 12, 2014.

[10] ETSI TS 102 732 V1.1.1 *Machine-to-Mchine communications (M2M); Use Cases of M2M applications for eHealth.* 2013-09 [Online]. Available: http://www.etsi.org/deliver/etsi_tr/102700_102799/102732/01.01.01_60/tr_102732v010101p.pdf Last Access: June 12, 2014.

[11] ETSI TS 102 690 V2.1.1 *Machine-to-Mchine communications (M2M); Functional architecture.* 2013-06 [Online]. Available: http://www.etsi.org/deliver/etsi_ts/102600_102699/102690/02.01.01_60/ts_102690v020101p.pdf Last Access: June 12, 2014.

[12] ETSI TS 102 921 V2.1.1 *Machine-to-Mchine communications (M2M); mIa, dIa and mId interfaces.* 2013-12 [Online]. Available: http://www.etsi.org/deliver/etsi_ts/102900_102999/102921/02.01.01_60/ts_102921v020101p.pdf Last Access: June 12, 2014.

[13] Maria Rita Palattella, Nicola Accettura, Xavier Vilajosana, Thomas Watteyne, Luigi Alfredo Grieco, Gennaro Boggia, Mischa Dohler, *Standardized Protocol Stack for the Internet of (Important) Things.* Communications Surveys & Tutorials, IEEE, 2012-12.

[14] Domenico Caputo, Luca Mainetti, Luigi Patrono, Antonio Vilei, *Implementation of the EXI schema on wireless sensor nodes using Contiki*, Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, 2012.

[15] SENSEI FP7 Project “Integrating the Physical with the Digital World of the Network of the Future” [Online]. Available: http://www.ict-sensei.org/ Last Access: June 12, 2014.

[16] W3C Recommendation "Efficient XML Interchange (EXI) Format 1.0" [Online]. Available: `http://www.w3.org/XML/EXI/` Last Access: June 12, 2014.

[17] Angelo P. Castellani, Mattia Gheda, Nicola Bui, Michele Rossi, Michele Zorzi, *Web Services for the Internet of Things through CoAP and EXI*, IEEE International Conference on Communications Workshops (ICC), 2011.

[18] Yusuke Doi, Yumiko Sato, Masahiro Ishiyama, Yoshihiro Ohba, Keiichi Teramoto, *XML-Less EXI with Code Generation for Integration of Embedded Devices in Web Based Systems*, Third International Conference on the Internet of Things (IOT), 2012.

[19] H.S. Thompson, D. Beech, M. Maloney, N.Mendelsohn, "XML schema part 1: Structures second edition" [Online]. Available: `http://www.w3.org/TR/xmlschema-1/` Last Access: June 12, 2014.

[20] Adam Dunkels, Björn Grönvall, Thiemo Voigt, *Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors*, 29th Annual IEEE International Conference on Local Computer Networks, 2004.

[21] Adam Dunkels, Oliver Schmidt, Thiemo Voigt, Muneeb Ali, *Protothreads: Simplifying event-driven programming of memory-constrained embedded systems*, ACM SenSys, 2006.

[22] Mattia Gheda, *LibEXI Analisi e implementazione del formato EXI su reti di sensori wireless*, 2010.

[23] Rumen Kyusakov, Jens Eliasson, Jerker Delsing, *Efficient Structured Data Processing for Web Service Enabled Shop Floor Devices*, 2010.

[24] Peter Deutsch, "DEFLATE Compressed Data Format Specification version 1.3" IETF [Online]. Available: `http://www.ietf.org/rfc/rfc1951.txt` Last Access: June 12, 2014.