UNIVERSITÀ
DEGLI STUDI
FIRENZE

PhD Program in Smart Computing
Dipartimento di Ingegneria dell'Informazione (DINFO)

Device Management at the Network Edge to Support QoS-aware IoT Services

Martina Pappalardo

Dissertation presented in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Smart Computing

PhD Program in Smart Computing
University of Florence, University of Pisa, University of Siena

Device Management at the Network Edge to Support QoS-aware IoT Services

Martina Pappalardo

Advisor:

Prof. Enzo Mingozzi

Head of the PhD Program:

Prof. Stefano Berretti

Evaluation Committee:
Prof. Marco Di Felice, *Università di Bologna*
Prof. Antonio Puliafito, *Università di Messina*

XXXV ciclo — October 2022

Acknowledgments

Firstly, I would like to thank my advisor Prof. Enzo Mingozzi for his guidance during the PhD. I'm also particularly grateful to Ing. Antonio Virdis for his valuable advice.
I would like to thank the members of my Supervisory Committee, Prof. Marco Avvenuti and Prof. Stefano Chessa, for their suggestions and feedbacks.
I would like to express my gratitude to all my friends and in particular to Carlo, Chiara, Francesca, Francesco, Giada, Gionatan, Leonardo, Marco, Marco, Maurizio, and Michele. Last but not least, thanks to my parents and to my whole family.

Abstract

The Internet of Things (IoT) brings Internet connectivity to everyday devices. The M2M systems generated by these IoT devices produce a large amount of data which is then forwarded to the IoT applications for processing, decision-making, etc. Best-effort delivery is sufficient in some circumstances; yet, in others, like in the context of Industrial IoT (IIoT), IoT applications may have strict Quality of Service (QoS) requirements, particularly in terms of latency. The severe energy, memory, processing, and communication constraints of IoT devices and networks, however, make it difficult to enable the deployment of QoS-aware services.

To support QoS-aware services provided by IoT devices, we propose providing a set of improved functionalities between IoT devices and IoT applications, which are frequently set up in the cloud. We propose integrating these functions into an edge-based IoT proxy that can efficiently utilize network, computing, and storage resources at the edge and take advantage of the close proximity to IoT devices to enhance their communication with IoT applications.

The proposed functions aim (i) to reduce and control the flow of requests sent to the IoT devices, resulting in benefits such as reduced and more predictable end-to-end latency and reduced packet loss; (ii) to minimize the energy consumed by the IoT device for sensing the environment and for transmitting the status update messages, while taking into account the information freshness requirements of the IoT applications; and (iii) to lower the size of the packets to comply with the required maximum packet length of energy-constrained IoT networks.

We evaluate the proposed functions both by simulation and experimentally, showing that they can strongly improve the QoS of the IoT system.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Internet of Things (IoT) is a paradigm of growing importance in the scenario of wireless communications (Atzori et al., 2010). The basic idea of IoT is the pervasive presence around us of interconnected objects and devices, such as wearables, sensors, actuators, etc. IoT devices build large M2M systems that generate a huge amount of data that is then transmitted to the IoT applications, where they are analyzed to make output decisions. In some contexts, best-effort delivery is sufficient; in others, such as augmented reality, wearable cognitive assistants, ambient safety, cyber-physical systems (e.g., automated fault detection), remote automation or Industrial IoT (IIoT), instead, applications have strict Quality of Service (QoS) requirements, particularly in terms of latency, requiring continuous updates about real-time states of devices. For example, smart manufacturing systems include closed-loop control and real-time analytic services that require ultra-reliable and low-latency communications to deliver telemetry data and control commands from sensors and actuators deployed in the assembly line (NetWorld2020, 2020). The deployment of such systems raises several challenges as the timeliness of this large amount of status updates is limited by the constraints of the IoT devices and the low power and lossy networks they use to communicate. Indeed, an IoT system can be shared across different applications that may need to be aware of the state of the same IoT devices as timely as possible; but, multiple concurrent requests can easily overload device and network resources. Actually, IoT devices implement a small queue buffer to store incoming packets because they have a limited memory capacity, but also because a small buffer results in a lower channel contention. When a device starts receiving packets at a rate that is higher than its own transmit rate, the queue buffer can easily overflow causing packet drops. Subsequent possible re-transmissions due to packet drops may further degrade network performance. Moreover, packet drops cause energy wastage on the congested devices that may lead to power depletion. As a matter of fact, energy is a scarce and crucial resource, as devices may not have a fixed power supply, but they may rely on batteries, or they

may harvest energy. So, an IoT system needs to be managed effectively to also save energy on devices: prolonging the lifetime of a device is fundamental, especially in deployments where human intervention is limited. Within this context, the objective of IoT-system management is to minimize the energy consumption on devices considering that applications need timeliness of status updates of the process sensed by the devices. The energy consumption mainly depends on two factors: sensing and transmission (Huang et al., 2022); the amount of energy they consume depends on the type of device. For example, modern IoT devices can perform complex operations other than the typical simple monitoring tasks, e.g., they may use on-device artificial intelligence to pre-process the sensed information: generating a status update can be very expensive in terms of energy. A packet generated by one of these complex operations can convey more information than a packet generated by a simple monitoring task, so the energy cost for transmitting it can be higher (Zhou and Saad, 2018). Finally, even for simple monitoring tasks, the sensing energy cost may vary greatly: for passive sensors, such as temperature sensors, sensing power consumption is negligible in comparison to active sensors, such as gas sensors, where sensing power consumption can be significant (Razzaque and Dobson, 2014). The transmission energy cost also depends on the underlying transmission technology: some transmission protocols are more energy-efficient due to Low Power Wide Area Network (LPWAN) technologies (Kim-Hung and Le-Trung, 2020). However, these technologies impose severe limitations on traffic, e.g., a limited payload length, in order to provide low power operations (Farrell, 2018).

1.1 Motivation

We estimate how the limitations of the IoT devices and the constraints imposed by the low power and lossy wireless networks they use to communicate may affect system performance in two exemplary network deployments.
In the first example, we consider a sample scenario using the LightweightM2M (LWM2M) protocol (OMA, 2020a), (OMA, 2020b) for device management wherein a wireless sensor node receives multiple concurrent requests from different applications located outside of the sensor network. The sensor network is emulated using the COOJA network emulator (COOJA, 2022), it is deployed using the 6LoWPAN protocol (6LoWPAN, 2022) on top of the IEEE 802.15.4 MAC (914, 2020) operating in the 2.4 GHz band and runs RPL (Alexander et al., 2012) as routing protocol. The device that receives the requests is located three hops away from the 6LoWPAN Border Router. First, we run a baseline experiment in which an application sends 2000 requests back-to-back to the device, i.e., a new request is sent only when the response to the previous request is received, and we measure the service delay, defined as the time between the request is sent by an application and the time the response is re-

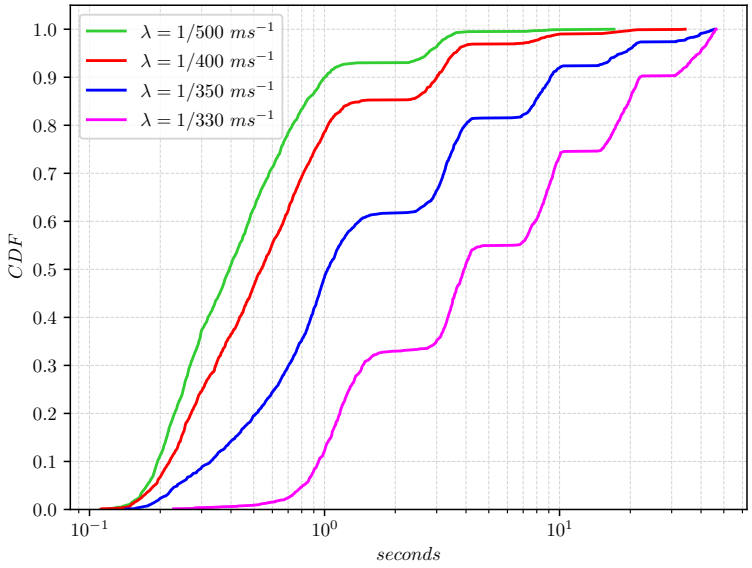


Figure 1.1: Service delay CDF of the sample scenario.

ceived. The resulting average service delay is 326.5 ms (95% CI [322.6, 330.4]). Then, we model the generation of the requests as a Poisson process with cumulative rate λ, and we run experiments varying λ. As the rate increases, the service delay distribution curve quickly shifts to the right and a growing, non-negligible, fraction of packets starts experiencing a large delay (see Fig.1.1). In addition to this, an increasing fraction of the requests starts not receiving a response (see Tab.4.1 in Sect.4.2). This happens because of the small queue buffer of the device that can quickly get overloaded as the device can take a non-negligible time to process and forward a packet due to its limited processing capabilities. Since requests are carried by CoAP CON messages, dropped requests are re-transmitted and the experienced delay increases as the number of re-transmissions increases.

As a second example, we consider a LoRaWAN network, one of the most adopted LPWAN technologies. LoRaWAN imposed strict limitations on packet payload length, having a minimum frame size of 59 bytes for the EU 868 MHz band and of 19 bytes for the US 915 MHz band (Farrell, 2018). Thus it can happen that a message does not fit in a LoRaWAN packet. We consider again an IoT system managed using the LWM2M protcol: it can often happen that a LWM2M message does not fit in a LoRaWAN packet, especially if it contains a LWM2M Object or a LWM2M Object Instance. As an example, consider the possible response to a request for the Device Object Instance of a device illustrated in Fig.1.2. We can notice that the LWM2M message may not fit in a LoRaWAN packet because the LWM2M payload alone is 399 bytes long using the JSON format. Even if the payload is encoded using a more

[{"bn": "/3/0/", "n": "0", "vs": "Open Mobile Alliance"},
"n": "1", "vs": "Lightweight M2M Client"},
"n": "2", "vs": "345000123"},
"n": "3", "vs": "1.0"},
"n": "6/0", "v": 1},
"n": "6/1", "v": 5},
"n": "7/0", "v": 3800},
"n": "7/1", "v": 5000},
"n": "8/0", "v": 125},
"n": "8/1", "v": 900},
"n": "9", "v": 100},
"n": "10", "v": 15},
"n": "11/0", "v": 0},
"n": "13", "v": 1367491215},
"n": "14", "vs": "+02:00"},
"n": "16", "v": "U"}]

Figure 1.2: JSON payload returned from an example request to Device Object (Read /3/0).

compact format, such as the TLV format, the message may not fit in a LoRaWAN packet because in this instance it is 121 bytes long.

We propose to improve system performance by adding one or more functions to support QoS-aware services; these functions should be implemented in between the IoT devices and the IoT applications, e.g., in an IoT proxy.
We propose

- a *core function* that implements an enhanced version of the application-layer management protocol to reduce the number of exchanged messages and the number of operations to be executed on the IoT device, without affecting IoT application operations;

and the following three additional functions:

- a *load-shaping function* that controls the flow of requests sent to the IoT devices over the IoT constrained network, so that the IoT applications experiment reduced delays and packet losses;

- a *compressing function* that compresses/decompresses the messages exchanged with the IoT device to be compliant with the limited packet payload length imposed by the energy-constrained networks;

- a *caching function* that reduces the number of exchanged messages in the IoT network, so that the IoT applications experiment reduced response times, and that reduces the power consumption of the IoT devices. To this aim, we propose a model-driven cache-management scheme that minimizes the device power consumption, while guaranteeing a given degree of data freshness. Data freshness is measured by the Age of Information (AoI) metrics (Yates et al., 2021) that assesses the timeliness of an IoT application's knowledge of the process running on an IoT device.

We evaluate the performance of the proposed functions both by simulation and experimentally, showing that they can strongly improve system performance and the QoS of the system.

Chapter 2

State of the Art

2.1 IPv6 over Constrained Node Networks

Introduction

Many IoT applications use resource-limited devices that are battery powered and that are connected to the Internet by low-power, low-bandwidth wireless networks. There are many advantages to using Internet protocols in these applications, including the possibility for IP-based devices to be connected easily to other IP networks, the ability to use the current network infrastructure, IP-based technologies, like the socket API, and IP-based tools for managing and troubleshooting networks. However, due to their characteristics, executing Internet protocols on wireless embedded devices and networks presents several difficulties. For example, devices have low duty cycles, multicast is often not supported, and the network bandwidth and frame size are constrained.
The IETF IPv6 over Low-Power WPAN (6LoWPAN) (6LoWPAN, 2022) working group enables IPv6 to be used with wireless embedded devices and networks. 6LoWPAN targets at IEEE 802.15.4-based networks providing an adaptation layer that includes the following functionalities: fragmentation and reassembly, address autoconfiguration, IPv6 and UDP header compression, and Neighbor Discovery Optimization. As IoT services are becoming more popular, the 6lo IETF working group (6lo, 2022) provides adaptation layer functions to support IPv6 over other link-layer technologies such Bluetooth Low Energy (Bluetooth LE) or Near Field Communication (NFC).
6LoWPAN are a crucial network in short-range low-power networks as 6LoWPAN devices will account for the majority of short-range, low-power things.
For these reasons, most of our experiments consider 6LoWPAN networks.

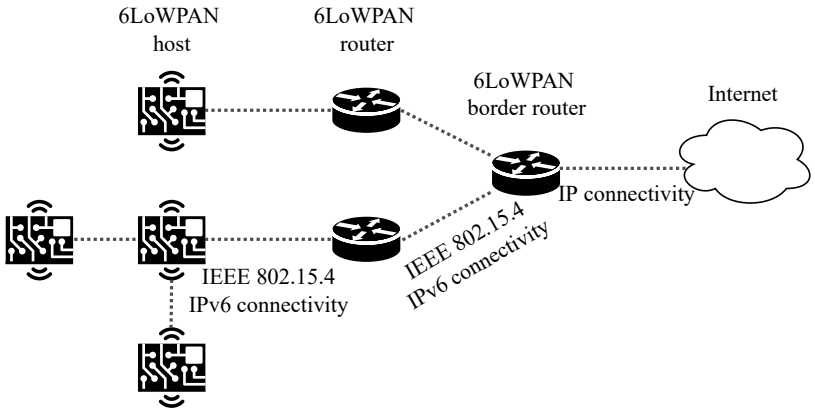


Figure 2.1: A 6LoWPAN architecture.

6LoWPAN

As shown in Fig.2.1, the 6LoWPAN architecture is made up of low-power wireless area networks (LoWPANs) that are IPv6 stub networks. A LoWPAN is composed of 6LoWPAN nodes which share a common IPv6 address prefix and it is connected to other IP networks, e.g., the Internet, through an edge router. The connection between 6LoWPAN nodes is implemented via IPv6 over IEEE 802.15.4. The edge router routes traffic in and out of the LoWPAN and provides traffic management, e.g., Neighbor Discovery. The other nodes of the LoWPAN can be host node or router. The hosts sense the environment and actuate devices. The routers are intermediate nodes that forward packets from the hosts to the edge router or to a destination inside the LoWPAN.

2.2 Low Power Wide Area Networks

Introduction

A growing number of IoT applications necessitates long-range, low-energy, and cost-effective solutions. Therefore, short-range radio technologies like Bluetooth or Zigbee are not appropriate for these applications; instead, cellular communication-based technologies like 3G or 4G offer greater coverage, but drain the device's battery too quickly. These requirements lead to the design of a new wireless communication technology: Low Power Wide Area Network (LPWAN) (Farrell, 2018). LPWANs are characterized by: (i) long-range connectivity: most LPWANs operate on low frequencies (sub-GHz band) and use modulation techniques to provide long communication range; (ii) low-power: LPWANs have a star topology, which eliminates the energy used by packet routing in multi-hop networks; they also use

duty cycling, a lightweight medium access control, and they offload complexity to the back-end system; (iii) low cost: LPWANs require no (or limited) infrastructure and may operate on unlicensed spectrum; (iv) reliability and robustness: the adopted modulation techniques and spread-spectrum techniques make the transmission hard to detect by an eavesdropper, more resilient to interference, and robust to jamming attacks; (v) potential to scale: avoidance of multi-hop gives potential to scale. Among the LPWAN technologies three leading solutions are (Haxhibeqiri et al., 2018), (Mekki et al., 2019): Sigfox (a proprietary solution standardized by the Sigfox company), LoRaWAN (standardized by the LoRa-Alliance) and Narrowband-IoT (NB-IoT, standardized by 3GPP). Sigfox and LoRaWAN use unlicensed ISM bands and bring advantages in terms of battery lifetime, capacity and cost; NB-IoT instead uses licensed LTE frequency and brings advantages in terms of latency and QoS. Unlike SigFox and NB-IoT, LoRaWAN allows private network deployments and integration with world-wide network platforms. LoRaWAN is one of the most widely used IoT technologies because of these factors as well as its open access specifications. Therefore, we carried out some experiments where devices use the LoRaWAN protocol to communicate.

LoRaWAN

LoRaWAN defines the communication protocol and the system architecture for the network (see Fig.2.2). It is a MAC protocol on top of LoRa (the physical layer).
LoRaWAN networks are organized in a star-of-stars topology in which gateways relay messages between devices and a Network Server in the back-end, so devices cannot communicate with each other but only with the gateway. The Network Server routes the packets to the Application Servers.
LoRaWAN radios make use of unlicensed ISM bands (for example, 868 MHz in Eu-

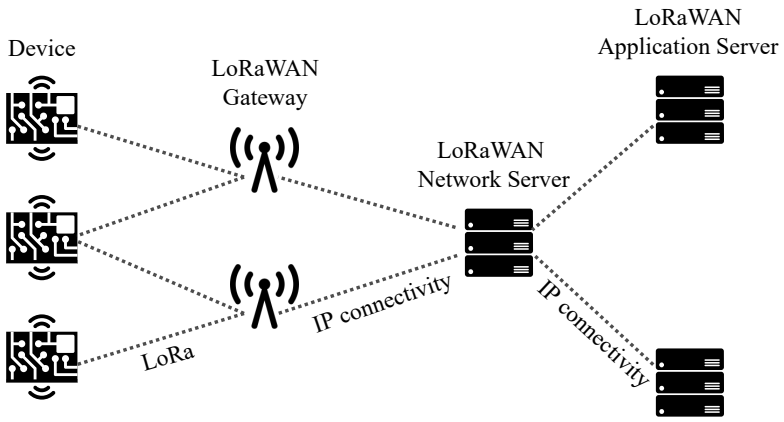


Figure 2.2: A LoRaWAN architecture.

rope, 915 MHz in North America) and due to the various spectrum allocations and needs in each region, the specification differs slightly between them.
As we can see from Tab.2.1, that shows some parameters of interest of LoRaWAN, strict traffic restrictions are imposed by this kind of technology to enable long range and low power operations. Indeed, it is possible to notice that, in case of the smallest frame size (19 bytes), 8 bytes are required for LoRa MAC layer headers, leaving only 11 bytes for the payload (including MAC layer options).
The FUOTA Working Group of the LoRa Alliance addresses this limitation on packet size by defining a fragmentation mechanism. They propose an application-layer messaging package (FUOTA-WG, 2022) running over LoRaWAN that sends a fragmented block of data to the device. This method splits a data block into multiple fragments to fit into LoRaWAN packets; then, these fragments must be reassembled on the device, handling potential losses. To do so, first a session establishment mechanism must be run to inform the device that a fragmentation session will start. However, fragmentation can affect the QoS of the system. Indeed, it introduces overhead because each fragment has to contain the L2 header, and it also introduces delay because a message can be reassembled only after all the fragments have been received. Moreover, LPWAN technologies are characterized by a high packet loss rate, so fragments can be lost causing re-transmissions and therefore performance degradation.

Modulation	chirp spread spectrum (CSS)
Minimum frame size (EU 868 MHz)	59 bytes
Maximum frame size (EU 868 MHz)	250 bytes
Minimum frame size (US 915 MHz)	19 bytes
Maximum frame size (US 915 MHz)	250 bytes
Range	5 km (urban), 20 km (rural)
Authentication and encryption	AES 128b
Duty cycle	min: 1%, max: no limit
Minimum data rate (EU 868 MHz)	250 bits/s
Maximum data rate (EU 868 MHz)	50000 bits/s
Minimum data rate (US 915 MHz)	980 bits/s
Maximum data rate (US 915 MHz)	21900 bits/s

Table 2.1: Overview of LoRaWAN.

2.3 Static Context Header Compression

Introduction

Since the characteristics of LPWAN technologies make the current IETF work (6lo) not easily applicable, the IETF LPWAN working group defined the procedures to enable IPv6 over LPWAN technologies. The Static Context Header Compression (SCHC) (Minaburo et al., 2020), (Gomez et al., 2020) technology is the core product of the IETF LPWAN WG, and it permits compression and fragmentation of packets to make them suitable for transmission over the constrained links of LPWANs. The working group's primary focus is on defining how to use SCHC to compress the headers of IPv6 and UDP as well as the headers of the Constrained Application Protocol (CoAP) (Shelby et al., 2014), a specialized web transfer protocol created by the IETF CoRE Working Group to realize the REST architecture in a way that is appropriate for constrained devices and networks. Indeed, now CoAP is implemented in the most popular operating systems for wireless sensor network nodes, such as Contiki (Contiki-NG, 2022).
SCHC is an adaptation layer between an upper layer, e.g., IPv6, and an underlying layer, e.g., an LPWAN technology. Before the upper layer packet is transmitted to the underlying layer, header compression is first performed. The inverse operation is done at the receiver. SCHC compression relies on a common static context, i.e., a set of rules, stored both in the back-end running the IoT application and in the IoT device. So, it is possible to define a specific context for each IoT device to take into account their heterogeneity. The context does not change during packet transmission: SCHC avoids the complexity of a synchronization mechanism. Moreover, SCHC supports fragmentation so, if the compressed-packet size still exceeds the maximum allowed size, it is possible to fragment the compressed packet with a reduced overhead and with fewer fragments needed than in the case of the uncompressed packet fragmentation.

SCHC

IoT devices embed built-in applications, hence the traffic flows are known in advance. So, the main idea of the SCHC scheme is to exploit the predictability of the traffic; in this way, it is possible to transmit the ID of a compression/decompression rule, i.e., the rule ID, instead of sending known packet-field values. SCHC classifies fields in (i) *static*: they are well-known, and hence they do not need to be sent on the link; (ii) *dynamic*: they need to be sent on the link; (iii) *computed*: they are rebuilt from other information.
At the top of Fig.2.3 we show an example of a compression rule: a rule contains a list of field descriptions composed of the following data:

- *Field Identifier* (*FID*), it designates a protocol and a field;
- *Field Length* (*FL*), it represents the length of the field;
- *Field Position* (*FP*), it indicates which occurrence of the field this field description applies to;
- *Direction Indicator* (*DI*), it indicates the packet direction;
- *Target Value* (*TV*), it is the value to match against the packet field;
- *Matching Operator* (*MO*), it is the operator used to match the field value and the target value;
- *Compression/Decompression Action* (*CDA*), it describes the compression and decompression actions applied on the field.

The matching operators are:

- *Equal*: true if the field value in the packet is equal to the target value
- *Ignore*: always true
- *MSB*(x): true if the most significant x bits of the packet field value are equal to the target value
- *Match-mapping*: true if the field value in the packet is equal to one of the values in the target value list

The main compression/decompression actions are detailed in Tab.2.2. The number of bits produced applying the CDA to the field is called the compression residue of the field. If the CDA is "not sent", then the compression residue is empty. If the field has a fixed length, then the compression residue has a fixed number of bits. If,

Action	**Compression**	**Decompression**
Not-sent	Elide	Use the value stored in the target value entry
Value-sent	Send	Build from received value
Mapping-sent	Send index of the value in the target value list	Use index to look up the target value list and restore the field value
LSB	Send least significant bits of the field value	Concatenate most significant bits stored in the target value entry and received bits

Table 2.2: Compression/decompression actions.

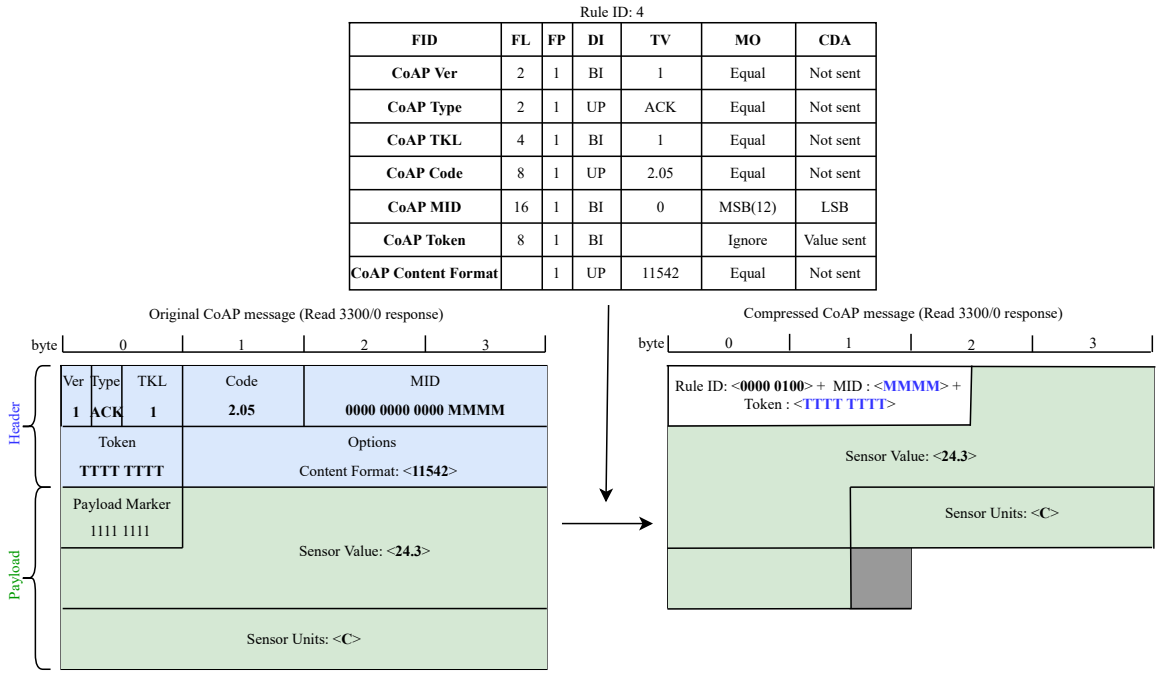


Figure 2.3: SCHC used in LoRaWAN.

instead, the field has a variable length, e.g., the uri-path, then, applying the CDA may produce either a fixed-size or a variable-size compression residue. The former case occurs when the field value is known and hence it is not sent, or when the field value belongs to a known list of values and hence only the index of the list is sent. In the latter case, instead, the compression residue is composed by the bits resulting from applying the CDA to the field, preceded by the size of the compression residue itself, encoded as specified in (Gomez et al., 2020). If a variable-length field is not present in the packet header being compressed, a size 0 must be specified to indicate its absence.

The compression algorithm consists of three steps as follows:

- *rule selection*: identify which rule will be used to compress the headers of the packets. The rule will be selected by matching the field descriptions to the packet header;
- *compression*: compress the fields according to the compression actions. The fields are compressed in the order specified by the rule and the compression residue is the concatenation of the residues for each field;
- *sending*: the rule ID is sent followed by the compression residue.

The decompression algorithm consists of a single step:

- *decompression*: the receiver selects the appropriate rule from the rule ID. Then it applies the decompression actions to reconstruct the original fields.

A SCHC rule has a fixed number of fields. If a variable number of fields is needed, e.g., there is a variable number of uri-path, there are two possible choices: create multiple rules to cover all cases, or, create one rule that defines multiple entries and send a compression residue with length zero to indicate that a field is empty. These two options introduce a trade-off between compression ratio and memory usage: indeed, if we define multiple rules that cover all possibilities, we increase the compression ratio, but we also need more memory on the device to store all the rules; if, instead, we define a single rule we obtain a lower compression ratio, but we also need less memory on the device.
To illustrate the SCHC mechanism, in Fig.2.3 we show an example of its possible use in a LoRaWAN network that uses the LWM2M protocol as application-layer protocol for device management: on the left, there is the original uncompressed packet, while, on the right, there is the packet compressed using the rule listed at the top of the figure. LWM2M is based on CoAP and, due to the limited bandwidth of LoRaWAN, it defines the possibility of using CoAP over LoRaWAN (LoRaWAN binding) by placing the CoAP packet containing the LWM2M message directly in the LoRaWAN packet payload. A CoAP message starts with a fixed size header of four bytes. This is followed by a variable length Token and by a sequence of zero or more options. Then, optionally, there is the payload, that in this case is the LWM2M message. The fields of the fixed header are defined in this way:

- Version (2 bits): indicates the CoAP version;
- Type (2 bits): indicates if the message is Confirmable (0), Non-confirmable (1), an Acknowledgment (2) or a Reset (3);
- Token Length (4 bits): indicates the length of the token (0-8 bytes);
- Code (8 bits): in case of a request, it indicates a Request Method, in case of a response, it indicates a Response Code;
- Message ID (16 bits): used to detect duplication and to match Acknowledgement or Reset to Confirmable or Non-confirmable messages.

Then, optionally, the token value is used to correlate requests and responses.

So, a set of compression rules for the CoAP header can be defined according to the following criteria:

- The CoAP version, type and token length fields have been elided, since their values are known.

- The CoAP code field has been reduced to the set of the used codes for each operation, defining a mapping list.

- The CoAP message ID has been reduced to a 4 bits value, using the MSB (Most Significant Bits) matching operator.

- The CoAP token field needs to be transmitted, but, since the token length is known, it is not necessary to send the size.

- The CoAP content format and accept fields have been elided when only a single value is possible; otherwise, they have been reduced to the set of used values using a mapping list.

- The CoAP uri path fields have been elided when only a single value is possible; otherwise, they have been reduced to the set of used values using a mapping list. Since the number of uri path elements may vary, it has been decided to define the length to variable and send a compression residue with a length of 0 when the uri path is empty.

- The CoAP uri query fields are used for setting the attributes, so they have been reduced to only their numeric values using the MSB matching operator. Since the number of uri query elements may vary, it has been decided to define the length to variable and send a compression residue with a length of 0 when the uri query is empty.

In the example of Fig.2.3, the receiver reconstructs the CoAP version, type, token length, code and content format using the values stored in the corresponding target value entries; it rebuilds the message-ID value concatenating the most significant bits stored in the rule and the received bits, and it builds the token value from the received value. We can notice that SCHC reduces the size of the header of the packet, from 8 bytes to 12 bits; however, the overall packet-size reduction is limited because the largest component of the packet, i.e., the LWM2M payload, is sent uncompressed. We can also notice that the payload marker used to indicate the end of header options and the start of the payload is not present in the compressed packet because the length of the compression residue of the header is known; moreover, even if the compression is bit-wise, the final size of the compressed packet is byte-aligned to be sent on the network.

2.4 LightweightM2M

Introduction

The IoT field has evolved with different types of applications, such as heath care, manufacturing, home automation, etc., that have domain specific technologies.
Thus, to achieve an interconnected network of things, the challenge is to manage and combine these heterogeneous technologies. Frameworks enhance IoT application development by rapid implementation, interoperability, maintainability and technology flexibility. So, for example, the IoTivity (IoTivity, 2022) framework enables device-to-device connectivity where wired and wireless IoT devices can connect to each other and to the Internet. It operates as a middleware and it is composed by four building blocks: discovery, data transmission, data management, device management. AllJoyn (AllJoyn, 2022) is an open source software framework that allows devices and applications to discover and communicate with each other. Among other examples, the Arrowhead framework (Arrowhead, 2022) aims to enable the Industrial IoT (IIoT) and, being based on Service Oriented Architecture (SOA), is characterized by the three following mandatory services: Service Register, Orchestration and Authorization. The Thread group framework (Thread, 2022) defines a protocol that is built on the standard IEEE 802.15.4 and performs peer-to-peer communications in a mesh network.
OMA SpecWorks' LightweightM2M (LWM2M) (OMA-SpecWorks, 2022) is a newly emerging framework. With the goal of defining technical specifications for the IoT, the IPSO Alliance combined with the Open Mobile Alliance (OMA) to create OMA SpecWorks. LWM2M from OMA SpecWorks was developed in response to the need for a uniform standard for controlling lightweight and low-power devices across various networks. LWM2M, which has an architectural design based on REST, defines an extensible resource and data model, and builds on CoAP, is intended for remote management of devices and related service enablement. LWM2M specification is freely available and today more than 25 companies are currently deploying LWM2M in their products and services, including ARM Pelion, AT&T IoT solutions, Huawei's OceanConnect IoT Platform, Microsoft's Azure and more: the support from many large organizations mitigates sudden end of support for the framework. Moreover, LWM2M provides several benefits that make it suitable for constrained devices: (i) it is a simple and stateless protocol; (ii) it is appropriate for low powered battery devices because of its low client footprint; (iii) it has a small RAM requirement; (iv) it has a transport-agnostic design and an increased bandwidth efficiency based on CoAP bandwidth optimization; and (v) it can be used both for data plane and device management.
For these reasons in our experiments we use the LWM2M protocol to manage IoT devices.

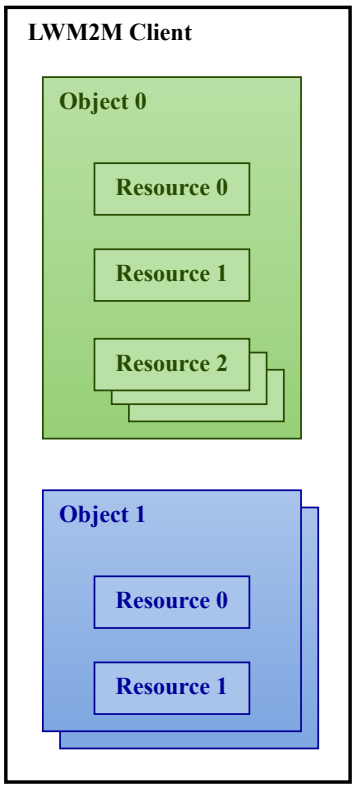


Figure 2.4: LWM2M Resource model.

LWM2M

LWM2M is an application-layer communication protocol between a LWM2M Server, i.e., the IoT application, and a LWM2M Client, i.e., the IoT device. Its functionalities include: (i) bootstrapping, e.g., key management, provisioning of access control lists, etc.; (ii) firmware update, e.g., update application and system software, apply bug fixes, etc.; (iii) remote management, e.g., changes to settings, trigger actuators, etc.; (iv) fault management, e.g., report errors from devices, query status of devices, etc.; (v) information reporting, e.g., notify changes in sensor values, retrieve configuration settings and device status, etc. LWM2M defines a set of REST-based interfaces and resources where each information made available by the LWM2M Client is a Resource. As depicted in Fig.2.4 Resources are organized into Objects. Objects and Resources can have multiple instances, and attributes, can be attached to them. A Resource which is instantiated within an Object Instance can either: (i) contain a value, or (ii) be used by the LWM2M Server to trigger an action in the LWM2M Client. The Resource can be identified by its path, expressed as follows: ObjectID/ObjectInstanceID/ResourceID. This object model is easily extensible, and the Object and Resource registry is open to the industry.
A LWM2M Client must support three mandatory Objects: (i) Security Object (ID:

Interface	Operation
Client Registration	*Register*: the LWM2M Client registers with the LWM2M Server
	Update: the LWM2M Client updates registration information
	De-register: the LWM2M Client de-registers with the LWM2M Server
Device Management and Service Enablement	*Read*: the LWM2M Server accesses the value of a Resource, a Resource Instance, an Object Instance or an Object
	Discover: the LWM2M Server learns the attributes of an Object, Object Instances, and Resources, and discovers which Resources are instantiated in a given Object Instance
	Write: the LWM2M Server changes the value of a Resource, a Resource Instance, and multiple Resources from an Object Instance
	Write-Attributes: the LWM2M Server modifies the notification attributes
	Execute: the LWM2M Server performs some action on individual Resources
	Create: the LWM2M Server creates Object Instances within the LWM2M Client
	Delete: the LWM2M Server deletes an Object Instance or a Resource Instance within the LWM2M Client
Information Reporting	*Observe*: the LWM2M Server initiates an observation request for changes of a specific Resource, Resources within an Object Instance or for all the Object Instances of an Object within the LWM2M Client
	Cancel-Observation: the LWM2M Server ends an observation relationship
	Notify: the LWM2M Client sends the new value of the Object Instance or Resource to the LWM2M Server

Table 2.3: LWM2M operations.

0), that contains keying material enabling access to a specified LWM2M Server; (ii) Server (ID: 1), that contains information relating to the connection of a LWM2M Server; (iii) Device (ID: 3), that contains device related information. Four interfaces are designed between the LWM2M Client and the LWM2M Server: (i) Bootstrap and (ii) Client Registration that are used to make the LWM2M Client accessible from the LWM2M Server, (iii) Device Management and Service Enablement that is used to read/write a Resource or an Object, or to execute an action on the LWM2M Client, (iv) Information Reporting that enables asynchronous information delivery. The main operations defined by these interfaces are illustrated in Tab.2.3.

2.5 Related Work

The future of IoT lies in the ability to orchestrate and program complex topologies of IoT devices taking into account their energy, memory, processing, and communication limitations and the constraints imposed by the low power and lossy networks they use to communicate.
IoT devices do not directly communicate with the IoT applications usually running in the cloud; indeed, the latency overhead alone makes the direct communication with the cloud impractical; and also, most devices use near-range technologies, so they cannot communicate directly with the cloud. Therefore, in a typical IoT architecture, an IoT gateway/proxy acts as an intermediary between devices and applications. It can also be used to effectively manage IoT devices by implementing some functionalities that aim to improve quality of experience and achieve better performance in terms of latency, throughput, energy consumption and response time (Beniwal and Singhrova, 2021).
Queue management and traffic shaping are typical functionalities implemented to support QoS services. Authors of (de Caldas Filho et al., 2019) propose a scheduling algorithm implemented by an IoT gateway that uses the Hierarchical Token Bucket algorithm to configure the output rate of a high priority queue for critical messages and of a low priority queue for non-critical messages. In (Mingozzi et al., 2014) authors implement a proxy virtualization framework to support scalability and the implementation of custom functionalities; in particular, they implement a prioritization policy to differentiate the service offered to two groups of applications using a priority-based scheduler that buffers the requests.
Another common functionality used to improve QoS is lowering the quantity of messages sent in order to save energy and avoid congestion. In (Ludovici and Calveras, 2015) authors consider a scenario in which applications need to continuously retrieve data from a WSN; they propose to implement the observe protocol in a CoAP proxy to receive status notifications from the device: since the proxy is the only observer, the WSN traffic is reduced. Additionally, they suggest adding a

cache that keeps received responses in memory until they expire, as stated by the Max-Age option; the cache is implemented by the proxy, reducing the number of interactions between the proxy and the WSN. Similarly, authors of (Lai et al., 2020) propose a CoAP-based framework implemented by a proxy that regulates the transmission of notifications in an IoT network. It groups clients, i.e., the applications, by their requirements and computes an optimal observation period for each group such that the device generates the minimum number of notifications. In addition to this, the framework computes a Max-Age value for each device to cache notifications, and it also combines the notifications of the device, so a client receives fewer notifications. Authors of (Robles et al., 2016) consider the LWM2M protocol for device management and they propose a group management function to minimize the traffic towards the LWM2M Server, i.e., the application. This function is implemented by a proxy that collects the messages coming from the LWM2M Clients, i.e., the devices, and aggregates them in on response to be sent to the LWM2M Server. However, in these solutions the implemented functions aim to reduce the traffic generated by the IoT devices. Instead, we propose to control and reduce the number of packets sent to the devices in order to cope with the constraints of IoT devices and networks.

System performance can be degraded also by the limited payload size imposed by constrained IoT network technologies. So, another functionality typically implemented in IoT systems is packet size reduction. Authors of (Hoebeke et al., 2018) propose a cloud-based virtual network operator for LPWANs that homogenizes deployments using the LWM2M protocol and that provides a way to communicate over different technologies using SCHC. However, they implement the compression mechanism for the CoAP/UDP/IPv6 stack, leaving the LWM2M payload uncompressed. In (Sanchez-Gomez et al., 2020) authors show the benefits of SCHC applied to constrained IoT networks: they consider a LPWAN network and, by applying the SCHC mechanism to the CoAP/UDP/IPv6 stack, they enable the transmission of packets that could not be transmitted previously. Also in this case, the compression is applied only to the headers of the packets. Instead, we propose an SCHC-based compression function that enables the payload of the message to be compressed as well, substantially reducing packet size.

Finally, also energy management is a critical issue in designing IoT networks; so, functionalities that provide energy-efficient solutions are key challenges in many IoT scenarios, e.g., in smart cities, smart grids, smart transport systems, etc. Indeed, many IoT devices can rely only on limited battery power, and it is often unfeasible to replace or recharge their batteries. Therefore, some efficient energy-management strategies have been implemented in IoT devices to prolong their lifetime. For example, in (Zhou and Saad, 2018) the authors design an optimal status sampling and updating policy for an IoT device to minimize data freshness, measured by the Age

of Information (AoI), at the destination, under an average energy cost constraint at the device; they show a trade-off between the average AoI and the sampling and updating costs. Authors of (Kaul et al., 2012) consider the challenge of providing all of their monitoring applications with device status updates as quickly as possible while taking into account the limited resources available. They consider the first-come-first-served queue policy and show the existence of an optimal rate at which a source should generate its updates. In (Abd-Elmagid et al., 2019) they consider an IoT network where devices sense different physical processes and send the status updates of these processes at a destination node; they investigate an optimal sampling policy that optimizes wireless energy transfer and scheduling of status update transmissions with the objective of minimizing a long-term weighted sum-AoI. The authors of (Chiariotti et al., 2022) define a new measure called Age of Information at Query (QAoI) to characterize the AoI available to the receiver when it needs it; they consider a sensor that needs to schedule transmissions over a constrained link and they maximize the freshness of the data at query time, considering that the sensor needs to limit the number of transmissions to prolong its lifetime. However, also caching can be used to provide efficient energy-management strategies. The ability to reduce the frequency of environmental sensing, and the frequency of data transmissions makes caching the data produced by IoT devices particularly beneficial in lowering power consumption. For example, authors of (Niyato et al., 2016) introduce a cache implemented in a wireless gateway that avoids activating the sensor too frequently, and hence reduces its energy consumption. Indeed, the cache implements a threshold-based refreshing mechanism: a cached item has a timer and, if the timer is larger than the threshold, the cache assumes that the item is expired, activates the sensor, and obtains a fresh sensing result. In (Xu et al., 2020) they propose a cache where data freshness is measured by the AoI metrics and they formulate a cache-update optimization problem that minimizes a cost that considers the users' AoI and the sensor's energy consumption. Other alternative approaches design cache refreshing schemes that balance AoI and latency because the experimented latency is also a significant problem. For example, authors of (Zhang et al., 2020) propose two cache-refreshing schemes: in the first one, the cached items are updated in a round-robin manner; in the second one, the cached items are updated upon requests with a certain probability. However, both the schemes may lead to unnecessary cache refreshing because they do not take into account the current state of the cached data. In (Zhang et al., 2018) authors propose a cache-assisted lazy update and delivery (CALUD) scheme to balance content freshness and service latency in vehicular networks. Authors of (Zhang et al., 2021) propose a cache-refreshing scheme where the cached items are refreshed upon requests if their AoI exceeds a given threshold called refreshing window. The value of the refreshing window is the solution of an optimization problem that minimizes the average delay under the

average AoI constraint of all devices. However, a requirement about the average AoI does not give any guarantee about how AoI values are distributed, so it is not possible to express a requirement in terms of the percentile of the distribution. This can be an issue in some IoT deployments such as ultra-reliable low-latency communication systems where it is needed to take into account the AoI tail behaviour in addition to optimizing the average metrics (Elgabli et al., 2019). Moreover, because they assume a device generates updates on demand, i.e., the AoI at the device is always zero, it is impossible to account for the energy consumption associated with the sampling process. So, for example, it is not possible to apply this solution for devices having a periodic sampling behaviour. Instead, we propose an AoI-based caching function that uses a model-driven management scheme that minimizes the energy consumption on the device under a freshness constraint expressed in terms of the percentile of the AoI distribution. The device energy consumption considers both the sampling energy consumption and the transmission energy consumption.

Chapter 3

Reference System

As depicted in Fig.3.1, a typical IoT architecture is composed of the following three main elements (Taivalsaari and Mikkonen, 2017):

1. *IoT Devices*
   IoT devices collect data or perform actuation, e.g., they are either sensors or actuators, and they have communication capabilities to submit the data to the broader IoT system through an access network. Sensors provide status updates of the physical process they monitor; actuators take energy and convert it in a state change.

2. *IoT Applications*
   IoT applications typically run in the cloud and have three main functionalities: (i) data acquisition, storage and access, to support the generation of a huge volume of data from devices; data are stored to be further processed and analyzed; (ii) data analytics on the collected data, which are examined to detect valuable information to support, for example, decision making; (iii) actuation support. In addition, they support several administrative functions, such as device management, user-account management, etc.

3. *IoT Gateways/Proxies*
   IoT gateways/proxies collect, process and transfer data from devices to the applications and deliver the actuation requests from the applications to the devices. They may also act as intermediaries between the devices and the applications, e.g., they may support data storage, service discovery, etc.

3.1 IoT Device

A simple data acquisition system is composed of, for example, a microelectromechanical system sensor, and the IoT application. In general, the data sample detect-

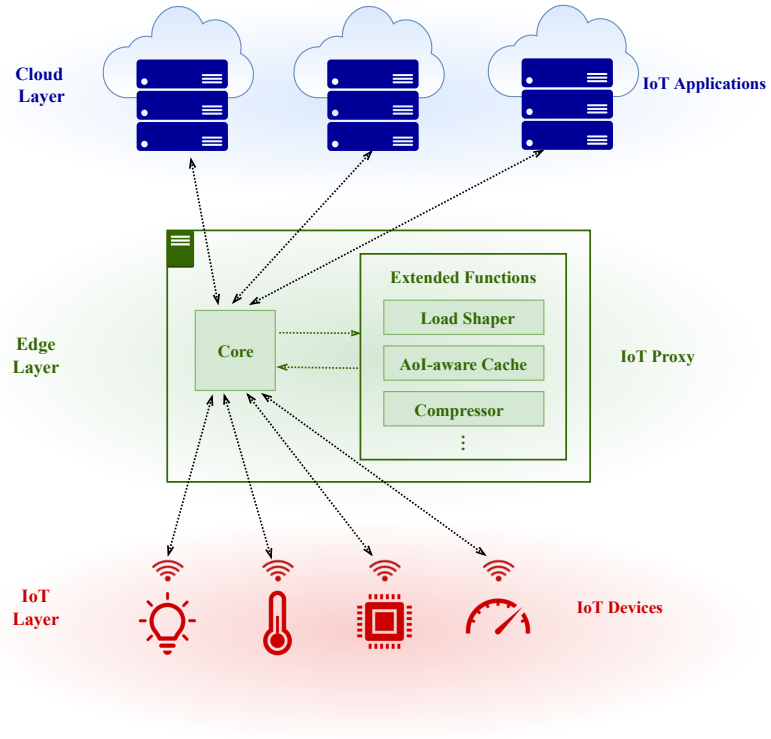


Figure 3.1: Reference system.

ed by the sensor is stored in a register of the sensor module and the IoT application reads out the data in the register. Typically, the register update interval of the sensor module is constant (Nishimura and Suzuki, 2020). In addition to this, the simplicity of periodic sampling is well-fitted with constrained devices, that have limited computational resources (Kim-Hung and Le-Trung, 2020). So, in many IoT systems, periodic sampling is the most prevailing behavior used by the applications (Li et al., 2019).
Hence, we assume that the IoT device collects information periodically with a sampling period s, that cannot be smaller than a given value called s_{min}, which is due to physical limitations on the device hardware. We also assume that the freshness of the samples is measured by the Age of Information (AoI) metrics. The fact that the device collects information at a given sampling rate implies that any external query on the device itself will produce data having an AoI in the range between 0 and s.

3.2 IoT Application

A server typically deployed in the cloud runs the IoT application that needs to retrieve the state of the IoT device as a part of, for example, a monitoring or control process. The server generates requests for state updates of the IoT devices and for-

wards them to the IoT proxy. We assume that the generation of requests is a process with mean rate λ.

3.3 IoT Proxy

As stated in the Introduction, we propose deploying a set of functions to provide QoS-aware services in order to address the limitations of IoT networks and devices and enhance system performance. These functions should be implemented in between the IoT devices and the IoT applications, e.g., in the IoT proxy. We propose a *core* function and the following three extended functions: (i) a *load-shaper*, (ii) a *compressor*, and (iii) a *cache*.
The core function improves network performance by implementing an enhanced version of the device management protocol, and orchestrates the other functions. The enhanced version of the device management protocol aims at reducing the number of exchanged messages and the amount of computations to be performed on the IoT device. The load-shaping function guarantees a better QoS by controlling the flow of requests sent to IoT devices and hence by controlling the delay experienced by the packets traveling in the constrained network and by reducing packet losses. The compression function improves QoS by reducing the packet size; in this way, the number of transmitted packets and the overhead of a fragmentation mechanism are reduced. Indeed, if the compressed packet can fit in a network frame, fragmentation is not needed; if, instead, the compressed packet still cannot fit in a network frame, the number of transmitted packets is reduced because its length is smaller than its uncompressed version, and the fragmentation overhead is reduced because fragmentation is defined as a mechanism internal to the compression function. Moreover, in case of a lack of fragmentation capabilities, the compression function can enable the transmission of packets that could not be transmitted previously. Finally, the caching function improves QoS because it reduces the power consumption of the IoT device and the number of packets to be transmitted in the constrained IoT network, saving network bandwidth and reducing response time, while guaranteeing information freshness.

Enhanced LWM2M

An enhanced version of the device management protocol is implemented by the core function. LWM2M is the device management protocol we are taking into account. The IoT application is a LWM2M Server, the IoT device runs a LWM2M Client and the IoT proxy is a LWM2M Proxy because we are considering a LWM2M architecture.

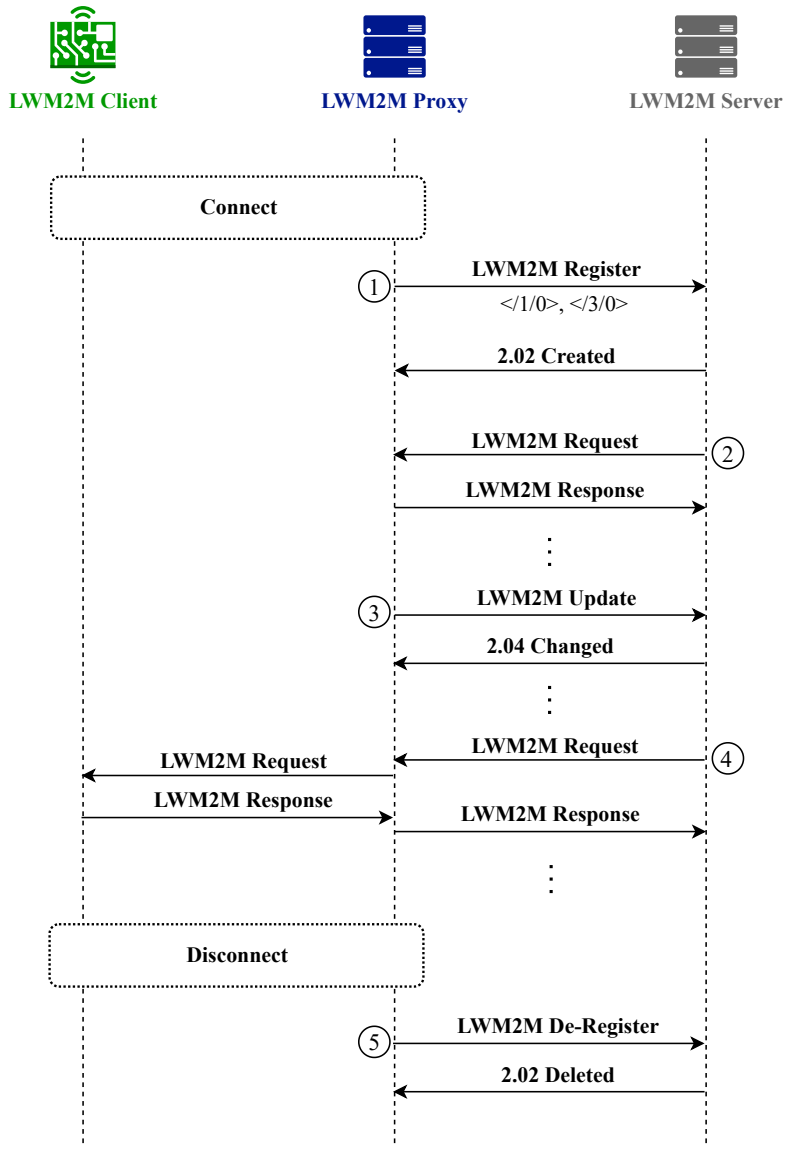


Figure 3.2: Enhanced LWM2M.

When a request of the LWM2M Server arrives at the LWM2M Proxy, the core function first attempts to use the enhanced version of LWM2M: in this case, it directly responds to the LWM2M Server, as showed in Fig.3.2, step 2. Then, if the enhanced version of LWM2M cannot be applied, i.e., the request has to be forwarded to the LWM2M Client, as showed in Fig.3.2 step 4, the core function determines the extended functions the request will go through, submits the request to them and finally responds to the LWM2M Server.
The enhanced version of LWM2M aims at reducing the number of exchanged messages and the amount of computation to be performed on the LWM2M Client. To do so, the enhanced LWM2M requires the LWM2M Proxy to be aware of the de-

vices present in the IoT network. The LWM2M Proxy discovers the devices of the network gathering information from the lower layers, exploiting technology specific functions. If, for example, the underlying communication technology is LoRaWAN, the LWM2M Proxy leverages the Join procedure to know which devices are connected; if instead a 6LoWPAN network is used, the LWM2M Proxy leverages the Neighbor Discovery procedure.
The rest of this Section describes when and how the enhanced LWM2M is applied.

Register, Update and De-Register

The registration procedure is performed by the LWM2M Proxy on behalf of the LWM2M Client. When a LWM2M Client connects to the network, the LWM2M Proxy sends the Register message to the LWM2M Server (see Fig.3.2, step 1). Then, it also sends an Update message when a refresh of the registration is needed (see Fig.3.2, step 3). When the LWM2M Client is no more connected, the LWM2M Proxy sends the De-Register message (see Fig.3.2, step 5).

Write-Attributes

The LWM2M Proxy rejects a Write-Attributes request if the conveyed Notification Condition Attributes are not consistent (see Fig.3.2, step 2). That is, when the "Change Value Conditions" attributes Less Than (lt), Greater Than (gt), Step (st), (OMA, 2020a) are set in a single Write-Attributes operation, the LWM2M Proxy does the required coherency checks: (i) *"lt" value* $<$ *"gt" value* and (ii) *"lt" value* + 2**"st" value* $<$ *"gt" value* on behalf of the LWM2M Client.

Discover

The LWM2M Proxy responds to a Discover request on behalf of the LWM2M Client (see Fig.3.2, step 2). To do so, the LWM2M Proxy maintains a list of the LWM2M Clients with their implemented Objects, Object Instances, Resources, Resource Instances and Attributes. The LWM2M Proxy is able to discover the LWM2M Clients in the network gathering information from the lower layers; and whenever a LWM2M Client connects to the network, the LWM2M Proxy retrieves and stores the list of its Objects, Object Instances, Resources and Resource Instances with their attached Attributes. The LWM2M Proxy updates this list every time a LWM2M Server creates or deletes an Object Instance or a Resource Instance and when it changes the notification Attributes of an Object, Object Instance, Resource or Resource Instance, i.e., when it issues a successful Write-Attributes request or a successful Create, Delete or Write request that modifies the list of Object Instances and Resource Instances.

Create

The LWM2M Proxy sends an error message in response to a Create request when the latter is not well-formed, i.e., if all the mandatory resources are not present in the conveyed message payload, or if the message payload conveys an Object Instance ID in conflict with one already present (see Fig.3.2, step 2). In this latter case, the LWM2M Proxy checks the correctness of the conveyed Object Instance ID using the list of Object Instances stored for each Client.

Write

The LWM2M Proxy can reject a Write request without forwarding it to the LWM2M Client if one of the following conditions is met: (i) the specified content format is not supported; (ii) the value of the incoming resource does not match the expected format; (iii) the value of the incoming resource is not in the range specified in the Object definition; (iv) the incoming resource is an Objlnk and its value is not valid; (v) the deletion or allocation of an instance of a multiple-instance resource is not allowed (see Fig.3.2, step 2).

Finally, the LWM2M Proxy checks all the requests to reject them if they are not allowed, or if they are incorrect, i.e., if the required parameters are not present in the uri-path/uri-query of the message; or if the specified Object ID, Object Instance ID or Resource does not exist or it is not supported by the LWM2M Client, or if the specified format is not supported (see Fig.3.2, step 2).

Moreover, the core function enables the support of QoS-aware services by orchestrating the extended functions. These functions are optional: the proxy can choose not to implement them, so, in this case its only component is the core; or it can choose to implement one, two or all of them. The proxy takes this decision according to the IoT application requirements. Indeed, before the application starts sending messages to the device, there is a configuration phase where the application and the proxy establish a service level agreement: during this phase the application specifies its QoS requirements in terms of service latency, data freshness or both.

Proxy Deployment

We propose to deploy the proposed functions on an IoT proxy at the network edge, in between the IoT devices and the IoT application. Hence, it can take advantage of the proximity with the devices and it can efficiently use network, computing and storage resources at the edge to overcome the limits imposed by IoT devices and networks.
However, the proxy should not be deployed even closer to the devices and within

the access network itself, e.g., within a node of the IoT network. Consider, for example, the case of dynamic IoT deployment, e.g., scenarios involving frequent topology changes due to channel variation or node mobility: a proxy deployed in an IoT node might fall in a sub-optimal placement with respect to the device-application path. If instead, an edge-placement is considered, topology changes can be easily managed by the proxy, which needs only to reconfigure few parameters of the extended functions that are influenced by the latency and the topology of the network, e.g., the load-shaper and the cache. As an example, let's consider the case of node mobility. When using an access network technology such as LoRaWAN or 6LoWPAN, node mobility is transparent to the proxy. Indeed, when a node moves within the 6LoWPAN domain, the mobility is supported by the routing protocol used within the 6LoWPAN and it is transparent to the proxy that is deployed at the edge, e.g., on the 6LoWPAN Border Router or on a node in the back-end near to the 6LoWPAN Border Router. Instead, LoRaWAN nodes are not associated with a specific gateway and data transmitted by a node can be received by multiple gateways. Each gateway will forward the packet to the network server, so complexity is transferred to the network server: if a node moves, there is no handover needed from gateway to gateway. Also in this case, node mobility is managed by the access network technology, and it is transparent to the proxy that runs the LoRaWAN Application Server. If the access-network technology is a cellular network, e.g., Narrowband IoT, node mobility among the base stations of the network is possible and is managed by the handover function of the technology itself. In this case, the proxy can still be deployed close to the serving base station, e.g., exploiting a Multi-Access Edge Computing (MEC) architecture.

3.4 Use Cases

As mentioned before, we assume that the IoT device collects information periodically with a sampling period s; hence, the IoT system is not event-based. We also assume that a REST-based communication protocol is used between the IoT device and the proxy; we employed the LWM2M protocol as a communication mechanism, as justified in 2.4. Moreover, we assume that the communication is pull-based. Pull-based applications must request what they want and pull it from the source, e.g., each application only receives the data it specifically asked for. Indeed, data quality is crucial in many IoT systems, such as those whose goal is to become learning systems, and the pull-based model makes it easier to keep dirty data from reaching the IoT applications and jeopardizing data quality.
Many applications could meet these requirements, such as: (i) smart agriculture applications (Sinha and Dhanalakshmi, 2022), that enable farmers to detect irrigation necessities based on weather recording and forecast, plant estimated needs, soil

moisture, etc.; (ii) smart city applications (Haque et al., 2022), that need to collect data from street lighting, parking sensors, air quality stations, waste and recycling containers, etc.; (iii) smart industry applications (Sisinni et al., 2018), that increase efficiency, reliability, safety and security within industrial processes and products. In fact, most of the decision-making process of these applications is dependent on efficient data analysis, therefore proper data collection is a crucial task.
All these applications might benefit of the proposed edge functionalities. Let us consider as an example the following scenario: an IoT network is composed of four LWM2M Clients, namely c_1, c_2, c_3, c_4, and three LWM2M Servers, namely s_1, s_2, s_3, need to communicate with some of them. In particular s_1 needs to communicate with c_1, c_2, c_3 and c_4; s_2 needs to communicate with c_2 and c_3; s_3 needs to communicate with c_2, c_3 and c_4. Using the standard architecture, the LWM2M Clients send nine messages just to register with the LWM2M Servers they need to communicate with. Then, suppose that the registration lifetime is 3600 s: this means that each LWM2M Client has to send an Update message every 3600 s. So, during a day, i.e., 24 h, a LWM2M Client sends 24 Update messages to each LWM2M Server it is registered with. Therefore, during a day the LWM2M Clients send 216 messages only to refresh their registrations. Instead, when we introduce the LWM2M Proxy, the Register and the Update messages are sent by the LWM2M Proxy on behalf of the LWM2M Clients, reducing the number of messages in the constrained network and saving energy on the device.
In the following, we describe each proposed extended function, highlighting its objectives, its benefits, and showing its possible implementation.

Chapter 4

Load-Shaping

Tiny devices with limited capabilities that use narrow-band communication protocols in energy-constrained networks (such as, LoRaWAN, 6LoWPAN, or Bluetooth) can have a significant impact on system performance. Since the energy-constrained networks are characterized by a limited capacity of the wireless medium, devices implement a small queue buffer (to store incoming packets) in order to have a lower channel contention but also in order to deal with a limited memory capacity, and they may take a non-negligible time to process and forward a packet because of limited processing capabilities. As a result, the queue buffer can fill up very quickly; when a node receives packets at a rate greater than its own transmit rate, the buffer overflows and packet drops happen. Hence, a load-shaping function can alleviate network congestion by controlling the flow of requests sent to the device. The load-shaping function implemented by the proxy is a window-based congestion-control mechanism. The congestion window is a standard mechanism that limits the number of bytes that can be sent out at any moment. We propose that the proxy implements a congestion window that limits the *number of outstanding requests* over the constrained network up to a fixed maximum. When the proxy receives a message request for a device, it enqueues it in a buffer, and forwards it only if the window is open, i.e., if the number of requests forwarded to any device and still waiting for a response is less than the maximum window size.

4.1 Load-Shaping Algorithm

The load-shaping algorithm behaves similarly to a token-bucket algorithm wherein a token represents a packet. The bucket is filled with a number of tokens equal to the size of the congestion window. Then, when a packet arrives, it is enqueued in the buffer and, if there is at least one token in the bucket, the proxy forwards the packet to the device and it removes one token from the bucket. When the proxy receives a response to a previous transmitted request, it adds one token to the bucket, checks

if a packet is queued for transmission, and possibly transmits it also removing one token from the bucket. In the extreme case the maximum size of the congestion window is set to one, i.e., there is only one outstanding request in the constrained network. However, in some cases, it can be more efficient to exploit the network resources by sending concurrent requests, i.e., having a maximum congestion window size greater than one. Indeed, the size of the congestion window depends on the number of requests that can be sent through the network without congesting it. This in turn, depends on several factors, including the following: the network/communication technology that is used; the network topology, i.e., the average number of hops to reach the destination; the memory and processing capabilities of the devices, i.e., the size of the buffer used to store incoming packets, and the time needed to process and forward a packet.

4.2 Performance Evaluation

To evaluate the advantages of introducing the proposed load-shaping function, we emulate an IoT system that uses the LWM2M protocol to manage the IoT devices and that is composed by a sensor network, a server, i.e., a LWM2M Server, and a proxy, i.e., a LWM2M Proxy, as depicted in Fig.4.1. A LWM2M Client runs on a wireless sensor node of the IoT network that uses the 6LoWPAN protocol (6LoWPAN, 2022) on top of the IEEE 802.15.4 MAC (914, 2020), operating in the 2.4 GHz band. The LWM2M Client is located three hops away from the 6LoWPAN border router. The system is emulated using the COOJA network emulator (COOJA, 2022) and uses RPL as routing protocol (Alexander et al., 2012). IoT devices run Contiki-NG as operating system. Contiki-NG implements the uIP stack, a small IP stack suitable for constrained devices. The uIP stack uses a single packet global buffer that can cause packet drops in case of concurrent requests. IoT devices are connected to the

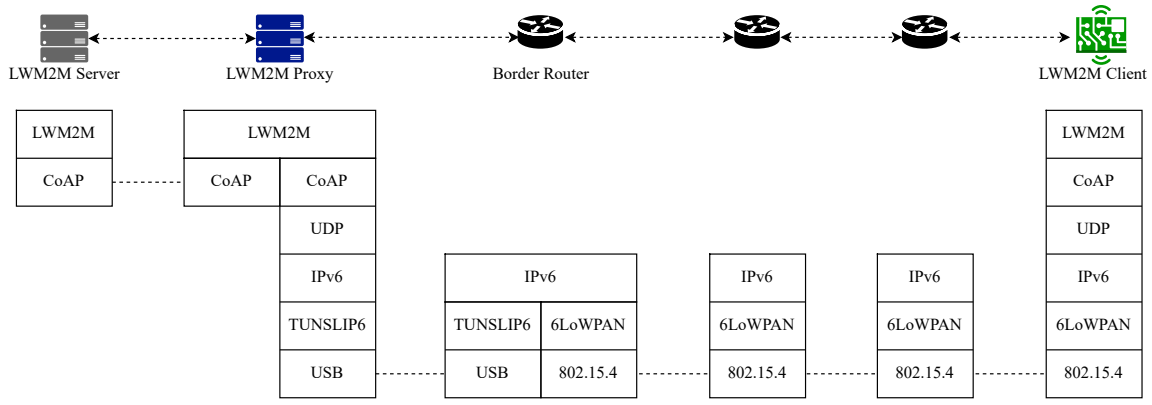


Figure 4.1: Proposed solution instantiated in a 6LoWPAN network.

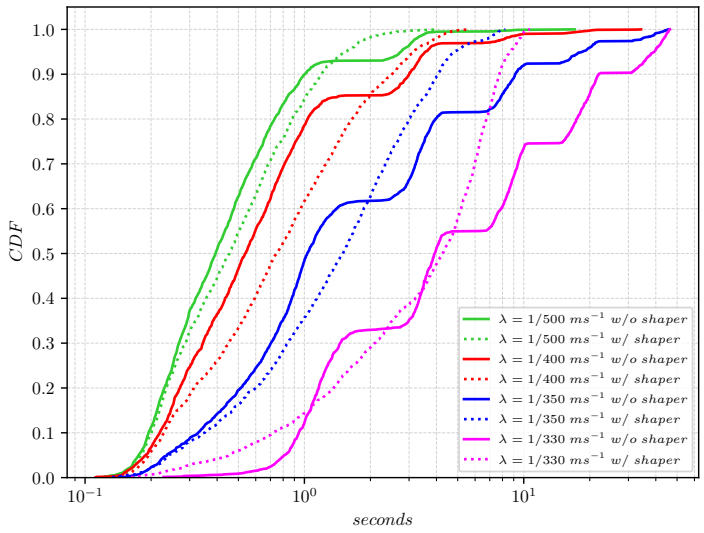


Figure 4.2: 3 hops - service delay CDF w/ and w/o the load-shaper.

$\lambda\ [ms^{-1}]$	w/o shaper	w/ shaper
1/500	0	0
1/400	0	0
1/350	4%	0
1/330	31%	0

Table 4.1: 3 hops - service loss.

Internet through the border router of the network, which uses a serial socket to connect to the outside network, which is managed by a tool called *tunslip6* on the fixed node, where the LWM2M Proxy is also running, while the LWM2M Server runs on a separate node connected to the proxy node by a LAN. The LWM2M Server and the LWM2M Proxy are developed using the Eclipse Leshan library (Leshan, 2022). We model the generation of the requests of the LWM2M Server as a Poisson process with cumulative rate λ.

We can take into account the following factors affecting the performance by emulating the sensor network: (i) the number of hops, (ii) the interfering traffic, (iii) the CoAP ACK timeout value, and (iv) the maximum size of the congestion window. In the first scenario, we compare the case where the proxy does not implement the load-shaper against the case where the proxy implements the load-shaper with the maximum size of the congestion window fixed to one. We consider the following metrics: (i) *service delay*, defined as the time between the request is sent by the application and the time the corresponding response is received, (ii) *service loss*, defined

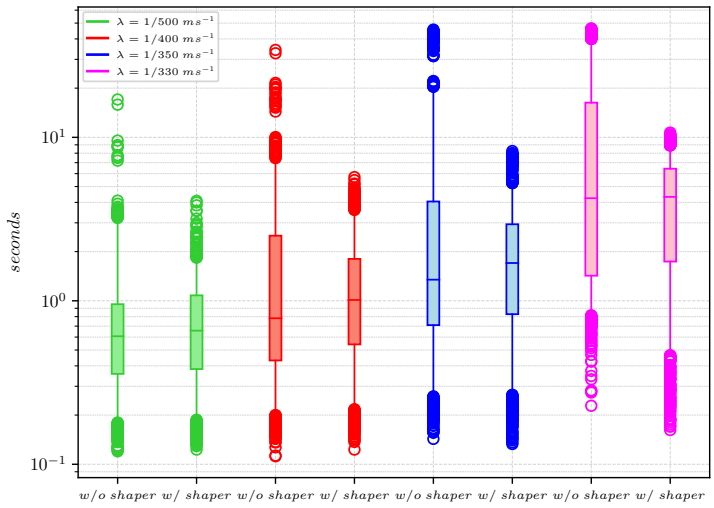


Figure 4.3: 3 hops - service delay boxplot w/ and w/o the load-shaper.

as the percentage of requests that did not receive a response according to requests sent. Fig.4.2 shows the cumulative distribution function of the service delay for different values of λ using a log scale on the x-axis. When the rate λ increases, if the proxy does not implement the load-shaper, the distribution shifts to the right and its tail is heavier; also the packet loss increases, as reported in Tab.4.1. Instead, when the load-shaper is used, the maximum experienced service delay decreases significantly, the tail of the distribution is shorter and there is no service loss. This happens because the proxy that uses the load-shaping function can control the traffic load in the network avoiding congestion. We can notice that the service delay experienced with the load-shaping function using a window value equal to one is not always better than the service delay experienced without the load-shaping function. So, in some cases, a load-shaping function using a larger window value could perform better.

Fig.4.3 compares the service delays of the scenarios that use the load-shaper against the service delays of the scenarios that do not use the load-shaper; the comparison is done using a box plot representation: in the box, the 25th, the 50th and the 75th percentiles are represented, respectively, and the ends of the whiskers represent the 5th and 95th percentiles. The circles are the outliers. We can notice that especially for lower values of λ, i.e., when $\lambda = 1/500$ ms^{-1} or when $\lambda = 1/400$ ms^{-1}, the 25th and 50th percentile values obtained with the load-shaper and without the load-shaper are almost the same; but we can also notice from the 75th and 95th percentile values that with the load-shaper the maximum experienced service delay decreases significantly (please note the log scale on the y-axis).

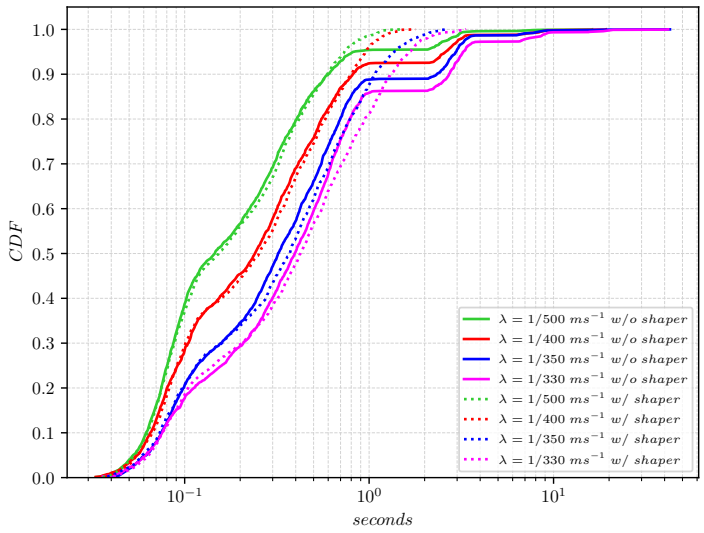


Figure 4.4: 1 hop - service delay CDF w/ and w/o the load-shaper.

In the next experiment, we test the performance of the load-shaping function by changing the number of hops between the LWM2M Client and the 6LoWPAN border router. In particular, we consider the limit case of the LWM2M Client located one hop away from the border router of the sensor network. Fig.4.4 shows the cumulative distribution function of the service delay for different values of λ for the one-hop case. Quite obviously, this scenario is less challenging. In fact, there is not service loss, both with and without the load-shaper. But, it can be noted that, even if the device is located only one hop away from the border router, the load-shaper is still effective in shortening the tail of the distribution, while there is practically no impact for smaller values of the service delay.

For the third scenario under consideration, we introduce an interfering traffic in the 6LoWPAN. Indeed, it often happens that an IoT network is shared between different applications. The LWM2M Client is located again three hops away from the 6LoWPAN border router, while the other two devices of the 6LoWPAN exchange UDP packets that interfere with the LWM2M traffic. More specifically, UDP packets are originated at one device following a Poisson process at an average rate equal to

$\boldsymbol{\lambda}$ **[ms^{-1}]**	**w/o shaper**	**w/ shaper**
1/500	0	0
1/400	0	0
1/350	15%	0

Table 4.2: 3 hops w/ UDP traffic - service loss.

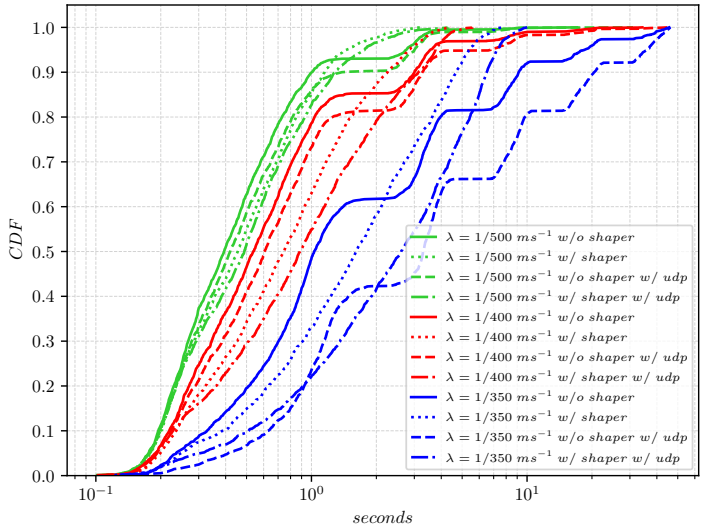


Figure 4.5: 3 hops - service delay CDF w/ and w/o UDP traffic.

1/350 ms^{-1} and sent to the other device. Fig.4.5 shows the cumulative probability distribution function of the service delay for different values of λ in this scenario. As expected, the introduction of extra traffic in the network further degrades the performance of the network in case of concurrent requests: the experienced service delay increases with respect to the case in which the only traffic present in the network is the LWM2M traffic in both cases, with and without the load-shaper. In the latter case, the service loss increases (see Tab.4.2). However, the benefits of introducing a load-shaper are the same as in the first scenario with no interfering traffic. In addition, the performance with the load-shaper is less affected by the presence of interfering traffic, as the shift of the curves to the right is smaller than in the case without the load-shaper.

The service delay and the service loss also depend on the value of the ACK timeout used by CoAP for the re-transmission of confirmable messages. The sender re-transmits a confirmable message at exponentially increasing intervals, until it receives an ACK (or a reset), or it exceeds the maximum number of re-transmissions. This means that when the timeout is triggered, the message is re-transmitted, and

$\lambda\ [ms^{-1}]$	w/o shaper	w/ shaper
1/500	0	0
1/350	0.7%	0
1/330	29%	0

Table 4.3: 3 hops and CoAP ACK Timeout = 4 s - service loss.

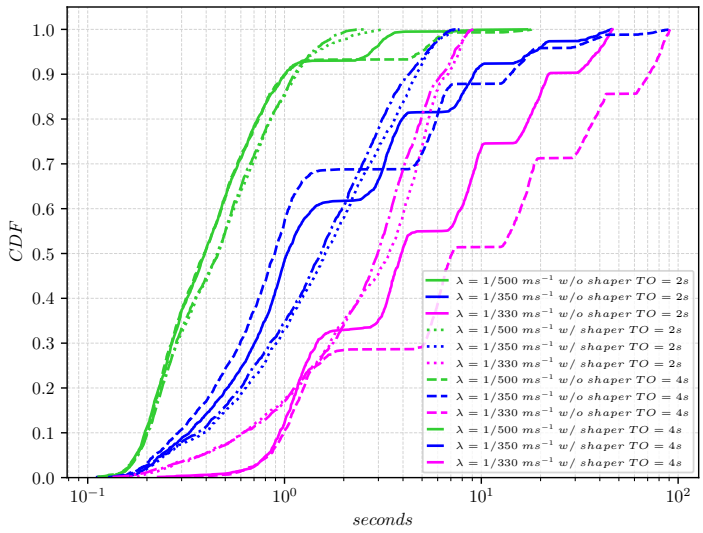


Figure 4.6: 3 hops - service delay CDF for different CoAP ACK timeouts.

the timeout is doubled. In the previous experiments the servers used a CoAP ACK timeout of 2 s (the default value). Consider again a LWM2M Client node located three hops away from the 6LoWPAN border router: Fig.4.6 shows the cumulative distribution function of the service delay for different values of λ and with a CoAP ACK timeout of 4 s. Without the load-shaper, results show that the maximum experienced service delay increases dramatically, especially for higher values of λ. For $\lambda = 1/350$ ms^{-1} or $\lambda = 1/330$ ms^{-1}, service delays can be more than 90 s. As expected, service loss instead decreases (see Tab.4.3). With the load-shaper, both the service loss and the experienced service delay are negligibly affected using a larger CoAP ACK timeout value.

Finally, we tested different values for the maximum congestion window size. Fig.4.7 shows the cumulative distribution function of the service delay for different values of the congestion window size in the case of a rate λ equal to $1/330$ ms^{-1}. We can observe that when the congestion window size is fixed to three the service delay experienced with the load-shaper is always better than the service delay experienced without the load-shaper. But, of course, when there are concurrent requests in the network it is possible that some of them experience a larger service delay with respect to the case in which there is always only one pending request in the network. Indeed, we can observe that the maximum service delay experienced by the packets in the case of a window value equal to three is larger than the case of a window value equal to one.

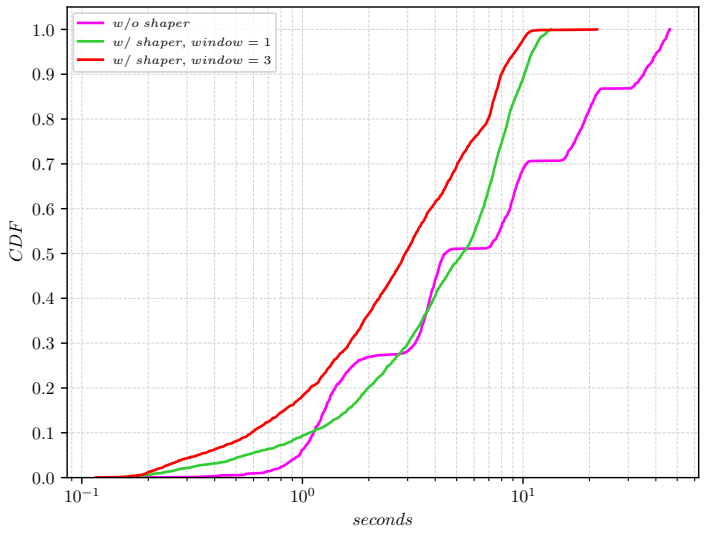


Figure 4.7: 3 hops - service delay CDF for different values of the congestion window size, $\lambda = 1/330$ ms^{-1}.

Chapter 5

Protocol Data Compression

IoT networks can impose strict limitations on traffic. For example, LPWAN technologies such as LoRaWAN, are characterized by a reduced data rate and a limited payload length in order to support long-range and low-power operations. To overcome the strict limits on the payload size imposed by this kind of technologies, we propose to reduce the packet size by applying a compression function based on the Static Context Header Compression framework that is executed both on the IoT device and on the IoT proxy.

5.1 Extended SCHC for LWM2M

SCHC provides only header compression, so the payload must be sent uncompressed after the SCHC compression residue. In constrained technologies such as LoRaWAN, the header compression alone might not be enough to fit the maximum allowed packet size. We propose an extension of the SCHC mechanism to also provide payload compression and we consider the case of an IoT network that uses the LWM2M protocol for device management.
The LWM2M message structure can be efficiently compressed using the SCHC mechanism. In fact, a LWM2M Object Instance can be represented by an array of entries where each entry is a Resource, i.e., a ResourceID and its corresponding value, as, for example, a sensor-measurement value. The ResourceID can be considered equivalent to the SCHC FID and, hence, we can straightforwardly apply the SCHC compression algorithm also to the Resource field. When the value of the Resource is dynamic, it is also possible to compress it using a standard data compression algorithm; we do not consider this latter case and we just apply the SCHC compression mechanism. If a LWM2M message contains multiple Instances of a given Object, the compression/decompression is applied for each Object Instance; the number of Object Instances present in the message is sent before the compression residue using the size encoding. More in details, the Object Instances and their Resources are

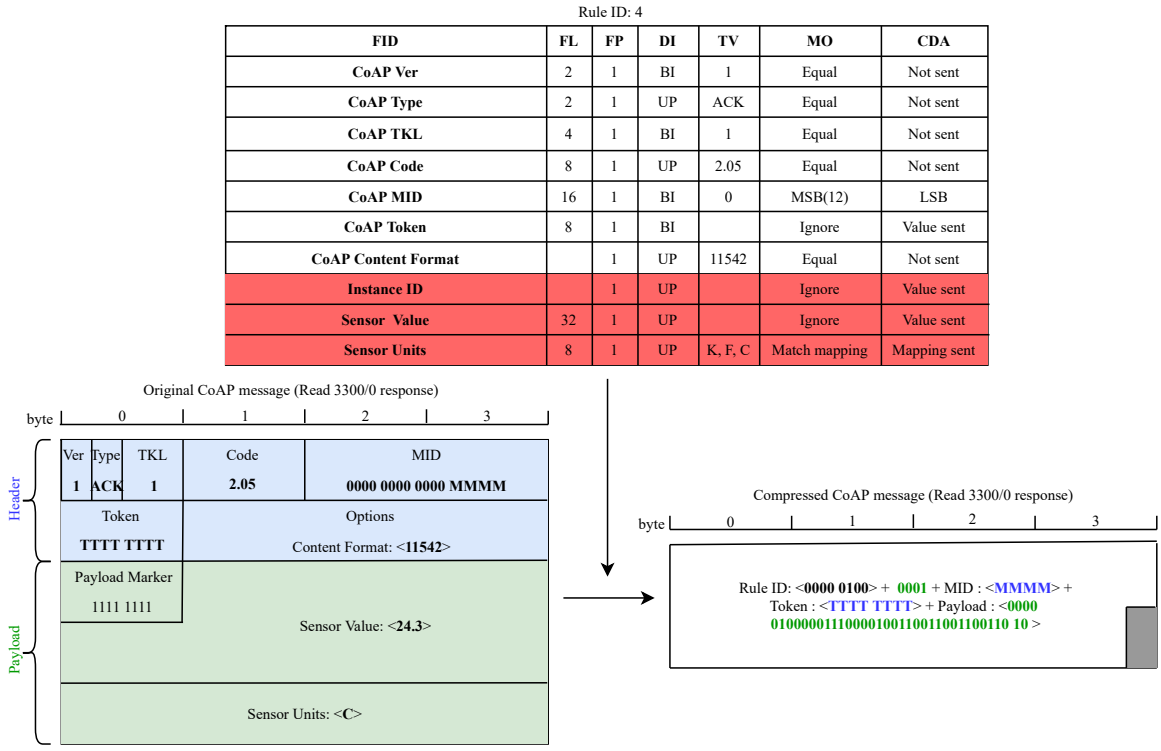


Figure 5.1: Extended SCHC used in LoRaWAN.

ordered according to their IDs, so that it is possible to either: (i) elide the value of a Resource when it is known a priori, (ii) define a mapping list for a Resource when its value belongs to a defined set of elements, (iii) send the value of the Resource in all the other cases. Instead, when Read/Write operations involve one single Resource, the value of the Resource is sent uncompressed after the CoAP compressed header.

Whenever a device connects to the network, the proxy retrieves and stores its Objects and, for each of them, the list of supported Resources. We assume that this list does not change during the transaction between the device and the proxy. After collecting the supported Resources, the proxy can create the rules needed for the compression of the messages. In the device, these rules are represented as LWM2M Objects, so the proxy can write the context in the device using LWM2M Write requests. As soon as this initialization phase has been completed, the proxy and the device can start exchanging compressed messages. In Fig.5.1 we apply the proposed extended SCHC framework to the same example proposed in Sect.2.3. In the figure, the additional field descriptions of the rule that allow the compression/decompression of the LWM2M message are highlighted in red. The compression residue of the CoAP header is now preceded by the number of Object Instances present in the message, i.e., 1, and followed by the compression residue of the LWM2M payload.

The LWM2M field descriptions of the rule specify that the Instance ID of the Object has a variable length and its value is sent; in the example, the Instance ID is not present in the LWM2M payload because the request is for a specific Instance of the Object, i.e., for the Instance 0, and not for the Object, so a compression residue with a length of 0 is sent; the value of the Sensor Value Resource is sent; the Sensor Units Resource can take only the three values listed in the TV, so the index of the value in the target value list is sent. We can notice that the packet size is further reduced with respect to the case in which the standard SCHC is applied (see Sect.2.3); indeed, when applying the proposed extended SCHC also the LWM2M message is compressed.

5.2 Performance Evaluation

To illustrate the benefits of introducing a compression function implemented in the proxy, motivated by the payload size issues illustrated in Sect.1.1, we deploy the proposed compression function in a LoRaWAN network (LoRaWAN-Specification, 2022) where devices are managed using the LWM2M protocol. We implemented the LoRaWAN binding, where the CoAP packet carrying the LWM2M message is embedded directly in the LoRaWAN packet payload, in addition to the compression mechanism, to deal with such constrained underlying technology. We assume that the device running the LWM2M Client implements only the LWM2M application, so the communication is end-to-end with the proxy and the IP and UDP layers become overhead and can be easily removed. As a consequence, the packet size is reduced and the number of computations and exchanged messages are reduced as well, because the device does not have to manage the IP protocol stack. The packet size is further reduced by applying the proposed compression mechanism based on the SCHC framework both on the device and on the proxy.
We implement a simple LoRaWAN system architecture composed of (see Fig.5.2):

1. a LWM2M Client. The device that runs the LWM2M Client is an Heltec WIFI

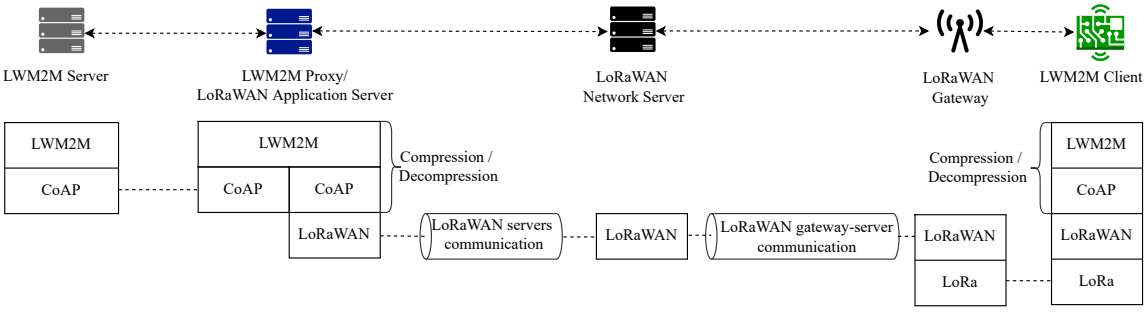


Figure 5.2: Proposed solution instantiated in a LoRaWAN network.

MicroController Unit	ESP32 Main processor: Tensilica LX6 Cores: 2 Clock frequency: 240 MHz
SRAM	520 KB
LoRa chip	SX1276 (868 and 915 version) SX1278 (433 and 470 version)
LoRa bands	EU_433, CN_470_510, EU_863_870, US_902_928 four bands optional
LoRa maximum output power	18 dB $\pm$ 2 dB
FLASH	4 MB

Table 5.1: Technical specifications of the device.

LoRa 32 (V2). The technical specifications of the device are detailed in Tab. 5.1. It implements a Server Object Instance (ObjectID: 1), a Device Object Instance (ObjectID: 3) and a Temperature Object Instance (ObjectID: 3303). Each of these Objects implements the mandatory resources. The Read, Write and Execute operations are implemented for the Server and Device Objects. The Observe, Notify and Cancel Observation are implemented for the Temperature Object.

2. a LoRaWAN Gateway. The gateway is a Laird Sentrius RG186. The gateway relays messages between the device and the network server in the back-end using the Gateway Message Protocol (GWMP) (Semtech, 2022a). So, communication between gateway and network server is via JSON/GWMP/UDP/IP.

3. a LoRaWAN Network Server. The network server routes the packets to the application server.

4. a LWM2M Proxy/LoRaWAN Application Server. The communication between the application server and the network server is via JSON over TCP over IP (Semtech, 2022b).

5. a LWM2M Server.

All the servers are Java servers and the LWM2M Proxy and the LWM2M Server use the Eclipse Leshan library (Leshan, 2022) to implement the LWM2M protocol. Tab.5.2 shows compression results. Compression is applied both to Object Instances, i.e., when reading the Device Object Instance, the Server Object Instance and the Temperature Object Instance, and when writing the Server Object Instance, and to Resources or to a set of Resources, i.e., when reading a single Resource of the Device or Server Object Instance and when writing one or more Resources of the

Type of message	Read Object		Notify Object	Read Resource						Write Object	Write Resources			Write Resource		
	3/0	1/0	3303/0	3/0/11	3/0/16	1/0/0	1/0/1	1/0/6	1/0/7	1/0	1/0/1 and 1/0/6	1/0/1 and 1/0/7	1/0/6 and 1/0/7	1/0/1	1/0/6	1/0/7
Percentage reduction	58.8%	75%	44.4%	35.7%	37.5%	37.5%	21.4%	37.5%	37.5%	56%	54.5%	54.5%	63.2%	45%	64.3%	64.3%
Compression ratio	2.43	6	1.8	1.56	1.6	1.6	1.27	1.6	1.6	2.27	2.2	2.2	2.71	1.81	2.8	2.8
Fit in a LoRaWAN packet	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 5.2: Compression results.

Server Object Instance. The compression rules that have been used are detailed in Appendix C. When considering operations involving Resources, we obtain different compression ratios depending on the number of bytes used to represent the resource itself, e.g., the Lifetime Resource (ID: 1) of the Server Object is an integer, while the Binding Resource (ID: 7) of the Server Object is a string. The compression function is effective because the size of the compressed packet is always smaller than that of the uncompressed one. This compression function can greatly reduce the packet size, especially when it conveys an Object Instance or an Object; indeed, from Tab.5.2 we can observe that the maximum percentage reduction of the packet is 75% and is obtained when considering the response to a Read request for the Server Object Instance of the LWM2M Client; and the mean percentage reduction of packets containing an Object Instance is 58.6%. However, the compression function can bring remarkable benefits also when the packets convey just one or more Resources: results show that for these packets the mean percentage reduction is 46.1%. Moreover, the compressed message always fits in a LoRaWAN packet.

Chapter 6

IoT Data Caching

IoT devices generate a huge amount of data that has to be transmitted along constrained networks possibly causing congestion and long delays. One possible solution is using an information caching system: indeed, caching the data generated by the devices can be very effective in reducing the number of packet transmissions. However, caching can lead to staleness of information, so the cache needs a refreshing scheme, as IoT applications need to receive fresh information. Several measures have been analyzed in order to measure the freshness of the cached data (Zhong et al., 2018) and consequently to design a refreshing scheme for the cache; one of the most used is the Age of Information (AoI) metrics (Yates et al., 2021), (Kosta et al., 2017), since it is a suitable metrics for describing the freshness at the receiver with respect to the sender. Introduced for the first time in (Kaul et al., 2011), the AoI is defined as the elapsed time for an item between current time and the time the item was generated at the source, i.e., the IoT device.
We propose to implement a caching function in the proxy whose refreshing scheme is a typical scheme that associates a validity lifetime, expressed in terms of AoI, to each cached data item; when the data item expires, it is not useful anymore and it must be discarded.

6.1 Cache-management Scheme

The IoT proxy that implements the caching function attempts to serve IoT application requests taking the responses from its cache, thus reducing network load. The freshness of a cached item is quantified by the AoI metrics: we assume that the AoI is the QoS parameter established by the service level agreement about data freshness between the IoT application and the IoT proxy. So, a cached item is valid if its AoI is smaller than a value called refresh window (W). We also assume that the refresh window is a multiple of the device sampling period, i.e., $W = ws$, with $w \geq 1$.
More in details, the proxy responds to the application request using its cached data

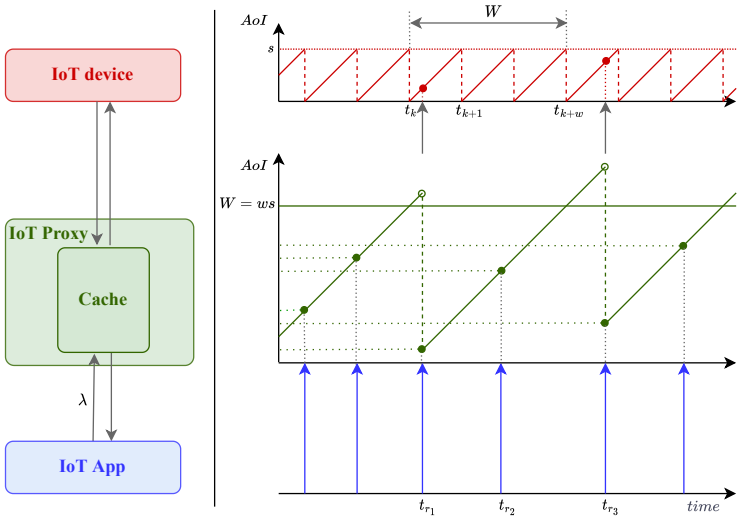


Figure 6.1: Cache-management scheme.

item if it is valid; otherwise, it relays the request message from the application to the device, then, it relays the response message back from the device to the application and updates the cache.
We call t_k the time when the $k-th$ sample is collected. Assume that a request arrives at time t_{r_1} as depicted in Fig.6.1 and that the cached item has expired. Since the data item is not valid, the proxy fetches the latest version of the item, whose AoI is therefore $t_{r_1} - t_k$, and delivers it to the application. Then assume that a second request arrives at time t_{r_2}. In this case, the cache contains the item previously fetched, whose AoI is now $s + (t_{r_2} - t_{k+1})$, and, since the AoI is less than the refresh window W, the proxy can directly deliver the item to the application. When a third request arrives at time t_{r_3}, the cached item is expired because its AoI is greater than the window W, so the proxy must refresh the cache before delivering the item to the application.

6.2 Performance Evaluation

To assess the performance of the caching function, we emulate an IoT system where we use the LWM2M protocol to manage the IoT devices and we consider a scenario consisting of an IoT network, a LWM2M Server that runs the IoT application and a cache-enabled LWM2M Proxy that implements our proposed cache-management scheme. We consider the same scenario of Sec.4.2, i.e., a LWM2M Client runs on a node of the 6LoWPAN network and is located three hops away from the border router (see Fig.6.2).
We model the arrival of requests as a Poisson process with cumulative rate $\lambda = 1/10$

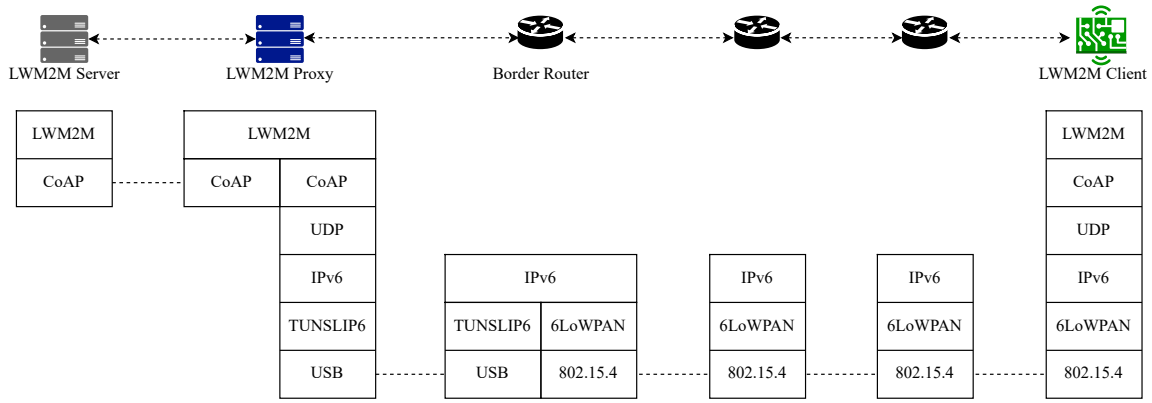


Figure 6.2: Proposed solution instantiated in a 6LoWPAN network.

s^{-1}, we assume the sensor sampling period equal to 60 s and we compare the case in which the proxy does not implement the cache against the case in which the proxy implements the cache. For this latter case we consider two different refresh window values: (i) $W = 60$ s, i.e., the window value is equal to the sampling period; and (ii) $W = 240$ s. Each scenario is run for 250000 s. To evaluate the performance of the caching function we consider the following metrics: (i) the number of exchanged messages in the sensor network and (ii) the AoI of the data at the application. Fig.6.3 shows the number of exchanged messages, while Fig.6.4 shows the cumulative distribution function of the AoI. It is possible to notice that the case of no cache and the case of a cache with a window value equal to the sampling period guarantee the same AoI at the application, but the cache can reduce the number of exchanged

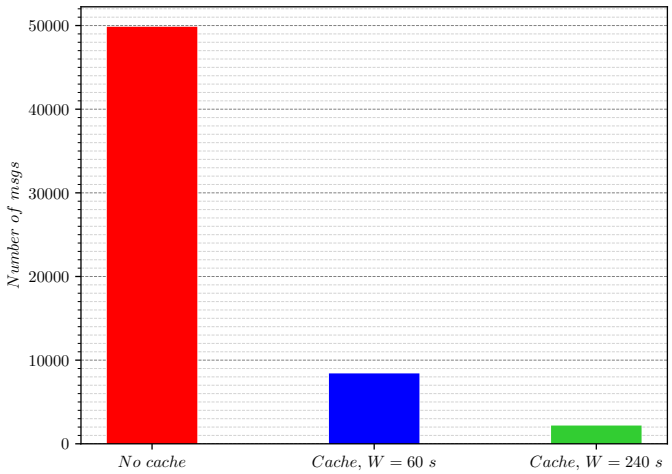


Figure 6.3: Number of exchanged messages w/ and w/o the cache, $\lambda = 1/10$ s^{-1}.

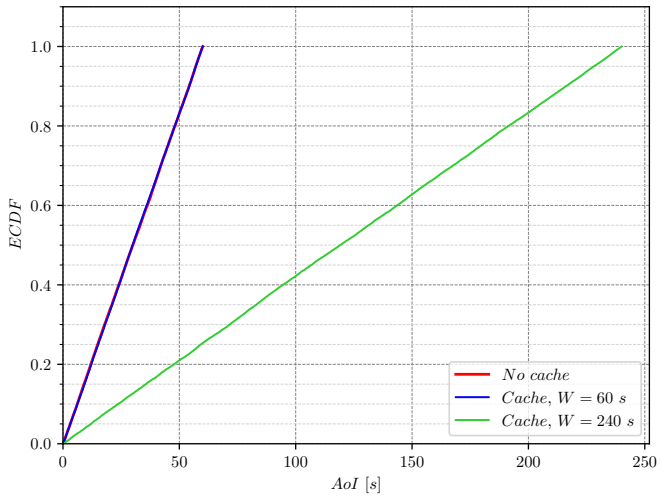


Figure 6.4: AoI empirical CDF w/ and w/o the cache, $\lambda = 1/10\ \mathrm{s}^{-1}$.

messages with the device. Moreover, results show the trade-off between the number of exchanged messages and the AoI of the data: when W increases, the number of exchanged messages decreases because the probability that a request gets a hit in the cache is higher, but the AoI increases. Minimizing the number of exchanged messages and minimizing the AoI of the data are opposite objectives.

6.3 Energy-Optimized Cache Refreshing

IoT devices generate a large volume of data that has to be transmitted to the IoT applications that analyze and process the received information to make some output decisions. These decisions are directly related to the freshness of the received information. Indeed, delivering fresh status information of the underlying physical process of interest is critical for many IoT applications for effective monitoring and control, and the number of scenarios in which devices send time-stamped status updates to applications is increasing. For example, sensor data are analyzed to detect anomalies; environmental sensor data can help to predict and control calamities; or, as another example, vehicles share their positions, velocities, accelerations, etc. to assist drivers in an intelligent transportation system. This phenomenon is even more evident in the Industrial IoT (IIoT) context; indeed, IIoT applications require continuous updates about real-time states of a huge volume of devices. Ideally, we would want a device generate status updates as fast as possible and transmit them to the application; but the deployment of such systems raises several challenges as the timeliness of this huge amount of status updates is limited by the severe en-

ergy, memory, processing, and communication constraints of IoT devices and networks. In particular, energy is a scarce and crucial resource; so, the generation and the transmission of the device status updates need to be managed effectively to save energy on the device and prolong its lifetime. The energy consumption of the device depends mainly on two components: the sensing energy consumption, i.e., the energy used to obtain a new status update, and the transmission energy consumption, i.e., the energy used to transmit the status update. Therefore, the objective of IoT-system management is to minimize the energy consumed by the device for sampling the physical process of interest and for transmitting the data, while ensuring the requested level of information freshness, i.e., AoI, is provided.
Caching the data generated by the devices can be very effective in reducing the sensing frequency and the transmissions frequency. However, applications can establish a threshold value on the freshness, i.e., on the AoI, of the received data; hence, when designing a management scheme for the cache, it may be necessary to optimize the system so that the AoI remains below this threshold with a certain probability (Costa et al., 2016). For example, in ultra-reliable low-latency communication systems it is necessary to take into account also the AoI tail behavior in addition to the AoI average value (Elgabli et al., 2019). Therefore, the application-freshness requirements are formulated as follows: *the IoT application requires that at least a fraction α of the requests receives a data item whose AoI is not larger than a target value denoted by AoI_α.* For example, an IoT application may require that at least the 90% of the requests, i.e., $\alpha = 0.9$, receives a data item whose AoI is not larger than a given target value, i.e., AoI_α.
We consider a cache managed using the scheme presented in Sect.6.1 and we define a model-driven optimization problem to find the values of the sampling period of the device and of the refresh window of the cache that minimize device power consumption, while satisfying the AoI requirement expressed by the IoT application.

Model of the Cache-Management Scheme

As mentioned before, we assume that W is a multiple of the device sampling period, i.e., $W = ws$, with $w \geq 1$. The system can be modelled as a $2w$-states Discrete Time Markov Chain (DTMC) $\{X_k\}_{k \in \mathbb{N}}$, where transitions occur at time instants t_k, i.e., the time when the $k-th$ sample is collected. We assume that the generation of requests follows a Poisson distribution with an aggregate rate λ. The DTMC is a simple yet effective model that allows us to compute in closed forms the following KPIs: the average AoI of the items and the cumulative distribution function of the AoI at the steady-state, and the number of transmissions per time unit needed to refresh the cache. This model only assumes that the proxy experiences small delays as compared to AoI requirements, and that the link between the proxy and the server running the application is almost deterministic.

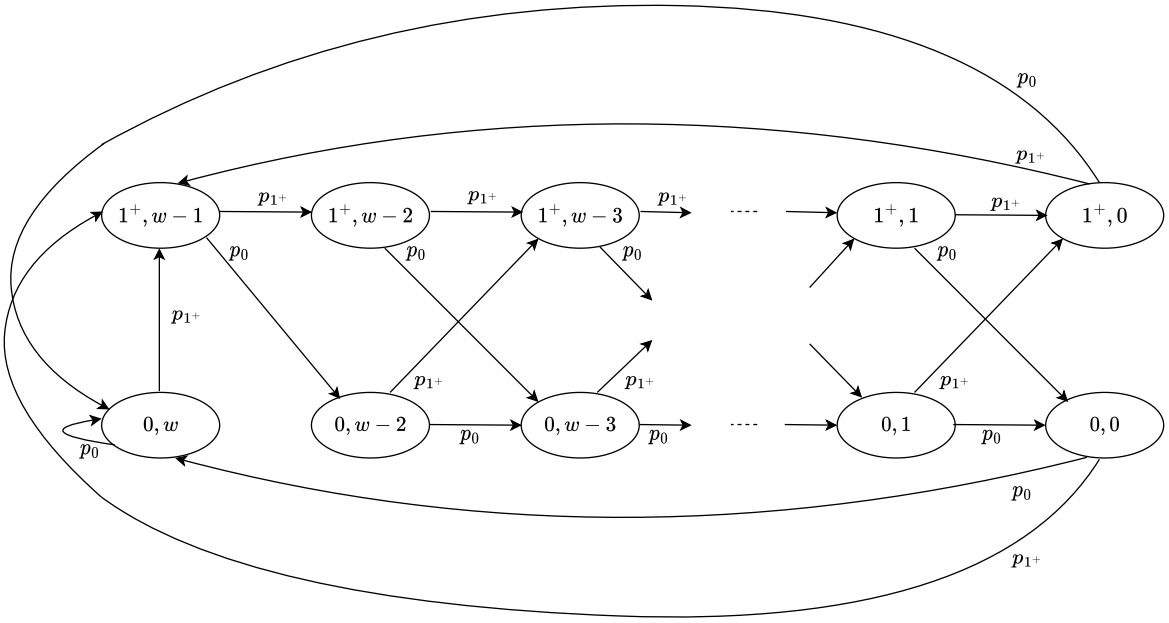


Figure 6.5: Discrete Time Markov Chain.

States

Denote X_k as the state of the DTMC at time t_k. X_k is defined by two components: (i) the first component specifies if in the previous interval (t_{k-1}, t_k) at least one request has arrived; (ii) the second specifies the number of remaining intervals during which the cached item is still considered fresh. Therefore, the states of the DTMC are the following:

1. $X_k = (0, w)$: no request arrived during the interval (t_{k-1}, t_k) and there is not a fresh item in the cache. If a request arrives in the interval (t_k, t_{k+1}), the proxy fetches the latest update from the device, which will be cached and will expire in w intervals.

2. $X_k = (0, w-j)$, $1 < j \leq w$: no request arrived in the interval (t_{k-1}, t_k), and the cached item will expire in $w-j$ intervals.

3. $X_k = (1^+, w-j)$, $1 \leq j \leq w$: at least one request arrived during the interval (t_{k-1}, t_k), and the cached item will expire in $w-j$ intervals.

Transition Probabilities

The model needs only to keep track if any request has arrived or not during the current interval, hence the transition probabilities are the probability of having no requests in a period of length s, denoted as p_0, and the probability of having at

least one request in a period of length s, denoted as p_{1^+}, as shown in Fig.6.5. Since requests follow a Poisson distribution, it is $p_0 = e^{-\lambda s}$ and $p_{1^+} = 1 - e^{-\lambda s}$.

Steady-State Probabilities

The DTMC is irreducible and positive recurrent, so it is possible to compute the steady-state probabilities, denoted as π. More in details, the steady-state probabilities are: $\pi_{0,w}$, $\pi_{0,w-j}$ with $1 < j \leq w$, and $\pi_{1^+,w-j}$ with $1 \leq j \leq w$, and, according to Markov chain theory, are derived as follows:

$$\pi_{0,w} = \frac{p_0}{p_0 + w p_{1^+}} \tag{6.1}$$

$$\pi_{0,w-j} = p_0 \frac{p_{1^+}}{p_0 + w p_{1^+}}, \quad 1 < j \leq w \tag{6.2}$$

$$\pi_{1^+,w-1} = \frac{p_{1^+}}{p_0 + w p_{1^+}} \tag{6.3}$$

$$\pi_{1^+,w-j} = p_{1^+} \frac{p_{1^+}}{p_0 + w p_{1^+}}, \quad 1 < j \leq w \tag{6.4}$$

Network Cost

Denote $E\{T\}$ as the average time between two requests that trigger a cache refresh. $E\{T\}$ is equal to the average time between two subsequent visits to the state $(1^+, w-1)$, which is given by the inverse of its steady-state probability:

$$E\{T\} = s \frac{1}{\pi_{1^+,w-1}} \tag{6.5}$$

$$= W + \frac{s}{e^{\lambda s} - 1} \tag{6.6}$$

Then, denote the number of exchanged messages to refresh the cache as n. Thus, the network cost, i.e., the average number of messages per time unit, is:

$$\overline{msgs} = \frac{n}{E\{T\}} \tag{6.7}$$

Average AoI

The average AoI is denoted as $\overline{AoI}$. The AoI is measured only in intervals where at least one request has arrived, and it depends on the state of the DTMC and on the instant of the arrival within the interval. The state of the DTMC is given by the steady-state probabilities; the average time instant of arrival is $s/2$, because Poisson

arrivals are uniformly distributed in a time interval. For any j, $1 \leq j \leq w$, the average AoI seen by requests arriving in the interval (t_{k-1}, t_k) is:

$$\overline{AoI_k} = E\{AoI|X_k = (1^+, w-j)\} \tag{6.8}$$
$$= \frac{s}{2} + (j-1)s \tag{6.9}$$

Unconditioning over all states for which at least one request arrived in the previous interval, it is, at the steady state:

$$\overline{AoI} = \frac{\sum_{j=1}^{w} E\{AoI|X_k = (1^+, w-j)\}\pi_{1^+,w-j}}{\sum_{j=1}^{w} \pi_{1^+,w-j}} \tag{6.10}$$
$$= \frac{s}{2} + \frac{1}{2}\frac{(W-s)W(e^{\lambda s}-1)}{W(e^{\lambda s}-1)+s} \tag{6.11}$$

Probability Distribution of AoI

The probability distribution function of AoI at the steady-state is denoted as $P_{AoI}(\delta W)$. It is:

$$P_{AoI}(\delta W) = P\{AoI \leq \delta W|X_k = (1^+, w-j), 1 \leq j \leq w\} \tag{6.12}$$

for any δ, $0 \leq \delta \leq 1$.
Therefore:

$$P_{AoI}(\delta W) = \frac{\sum_{j=1}^{w} P\{AoI \leq \delta W|X_k = (1^+, w-j)\}\pi_{1^+,w-j}}{\sum_{j=1}^{w} \pi_{1^+,w-j}} \tag{6.13}$$

If in state $X_k = (1^+, w-j)$ at time instant t_k, $1 \leq j \leq w$, the cached sample has been collected at a time instant $t_{k-j} = t_k - js$, so, it is for any j, $1 \leq j \leq w$:

$$P\{AoI \leq \delta W|X_k = (1^+, w-j)\} = \begin{cases} 1 & j \leq \lfloor \delta w \rfloor, \\ \delta w - \lfloor \delta w \rfloor & \lfloor \delta w \rfloor < j \leq \lfloor \delta w \rfloor + 1, \\ 0 & j > \lfloor \delta w \rfloor + 1. \end{cases} \tag{6.14}$$

Finally:

$$P_{AoI}(\delta ws) = \begin{cases} \dfrac{\delta w}{w(1-e^{-\lambda s})+e^{-\lambda s}} & 0 \leq \delta < \frac{1}{w}, \\ \dfrac{\delta w - e^{-\lambda s}(\delta w - 1)}{w(1-e^{-\lambda s})+e^{-\lambda s}} & \frac{1}{w} \leq \delta \leq 1. \end{cases} \tag{6.15}$$

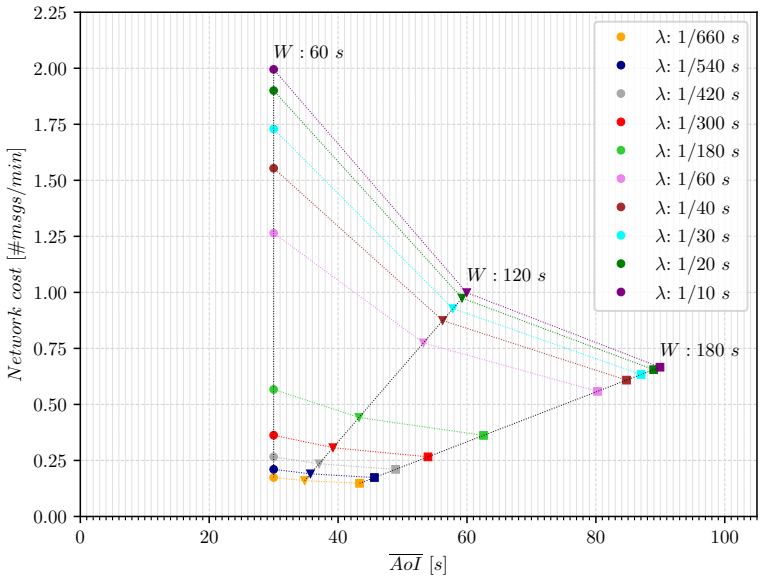


Figure 6.6: Network cost vs average AoI.

Analysis

To evaluate the considered cache-management model, we firstly analyze it analytically. More in details, assuming that each cache refresh costs two messages - one request message sent by the proxy to ask the device for the data item and one response message sent by the device to deliver the data item to the proxy - we analyze the metrics previously presented: (i) the network cost, or the average number of messages per time unit in the constrained sensor network, i.e., $2/E\{T\}$; ii) the average AoI of the data items sent to the application, i.e., $\overline{AoI}$; and (iii) the probability distribution function of AoI, i.e., $P_{AoI}(\delta ws)$. Fig.6.6 shows the average AoI against the network cost expressed in number of exchanged messages per minute. Both metrics are estimated using our model, for different values of λ and W. The value of s is fixed at 60 s, whereas w assumes the following values: $w = 1$, $w = 2$ and $w = 3$, which are represented by points, triangles, and squares, respectively. It can be noted that when varying λ while keeping the same value of W, there is a linear relation between the cost and the average AoI. To explain this, we can rewrite 6.11 and express $\overline{AoI}$ as a function of the inverse of $E\{T\}$, which is a measure of the network cost, as follows:

$$\overline{AoI} = \frac{s}{2} + \frac{(W-s)W}{2} E\{T\}^{-1} \tag{6.16}$$

When $W = s$, it is $\overline{AoI} = s/2$; this is confirmed by the figure, where we can see that for $w = 1$ the average AoI is always 30 s, regardless of the value of λ; while

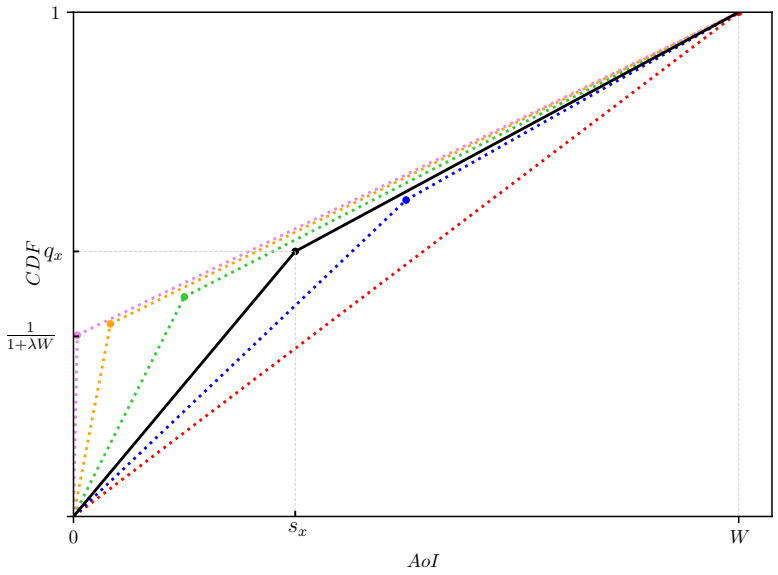


Figure 6.7: AoI CDF for different values of s, fixed W.

the cost decreases as λ decreases. Moreover, for $\lambda \to 0$ it is $\overline{AoI} \to s/2$ and for $\lambda \to \infty$ it is $\overline{AoI} \to W/2$; while, for $\lambda \to 0$ it is $E\{T\} \to \infty$ and for $\lambda \to \infty$ it is $E\{T\} \to W$. Indeed, when the request rate is very low ($\lambda \to 0$), each request triggers cache refreshing with high probability, so, the network cost mainly depends on the interval between two successive requests, and the average AoI seen by the requests is the average AoI at the device; while, when the request rate is extremely high ($\lambda \to \infty$), the cache is refreshed almost periodically with period W, so the network cost and the average AoI depend on W. From Fig.6.6 we can also have a graphical representation of the trade-off between network cost and average AoI; we can observe that for the same value of λ, if the value of the window W increases, the network cost decreases. This happens as the larger the value of W, the higher is the probability a request gets a hit in the cache, thus reducing the transmission load of the network. However, doing so results in an increased average AoI. So, minimizing the network cost and minimizing the average AoI are two contrasting objectives, and we can balance them leveraging the value of the window W: if we have a high communication cost, we can choose a higher value of W, because it is better not to frequently retrieve data from the device; while, if the application needs to be aware of the state of the sensor as timely as possible, we can choose a lower value of W.
Fig.6.7 shows the cumulative distribution function of the AoI for different values of w and s, while keeping their product constant. Denote q as $P_{AoI}(s)$, i.e.,

$$q = \frac{s}{W(1 - e^{-\lambda s}) + se^{-\lambda s}} \tag{6.17}$$

Function $P_{AoI}(s)$ is continuous; indeed, for $\delta \to (1/w)^-$, it is $P_{AoI}(\delta ws) \to q$. It is also piece-wise linear: the first line passes by $(0,0)$ and (s,q), and the second line passes by (s,q) and $(ws,1)$, as shown in the figure. We can also observe that when $s \to 0$ it is $q \to 1/(1+\lambda W)$. Moreover, the slope of the line passing by $(0,0)$ and (s,q) is $m = wq$; and, for $s \to \infty$ it is $m \to 1$ and for $s \to 0$ it is $m \to w$. Consequently, the probability distribution function is described by the three points $(0,0)$, (s,q) and $(ws,1)$, and can be used to compute any AoI percentile.

Emulation

Then, we compare the results obtained with our model with the experimental results obtained using the same scenario of Sec. 6.2, i.e., we emulate an IoT system consisting of an IoT network, an IoT application and a cache-enabled proxy that implements our cache-management scheme, and we consider a device located three hops away from the border router of the IoT network.
The device sampling period is equal to 60 s and experiments are performed for different values of λ; for each of these values, we have considered a refresh time window W of 60 s ($w = 1$), 120 s ($w = 2$), 180 s ($w = 3$) and 240 s ($w = 4$). Each simulation lasted 250000 s to have a sufficiently large number of requests also for smaller values of λ. Fig.6.8 compares the experimental results with the results estimated from the model: "x" markers represent the values obtained via emulation,

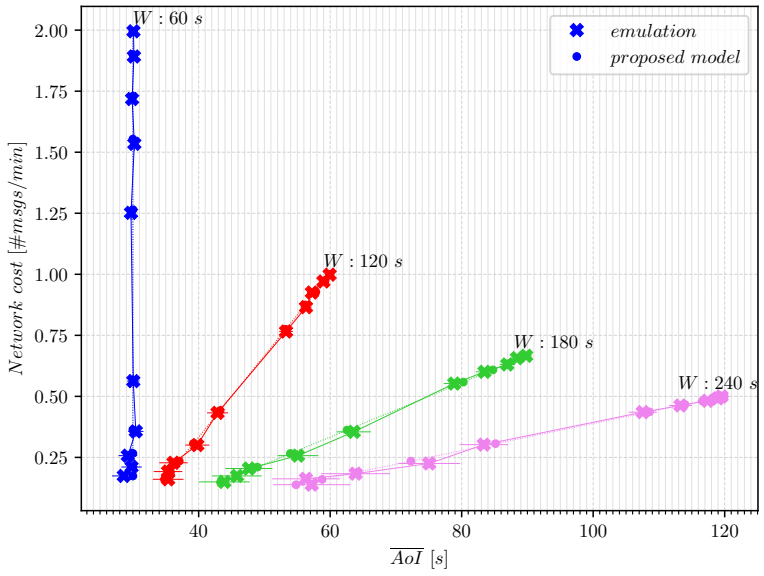


Figure 6.8: Network costs: "x" markers are obtained via emulation (95% CI is shown), whereas point markers are obtained through the proposed model.

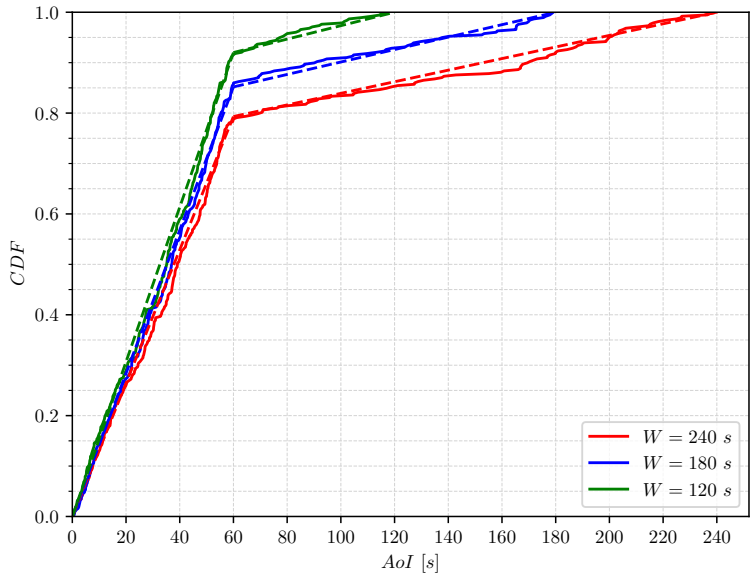


Figure 6.9: AoI CDFs for different values of w: dotted lines are obtained through the proposed model, whereas solid lines are obtained via emulation.

whereas point markers represent the values obtained through the proposed model. Considering a line representing the values obtained with a given W and starting from the bottom up, λ assumes the following values: $1/660$ s^{-1}, $1/540$ s^{-1}, $1/420$ s^{-1}, $1/300$ s^{-1}, $1/180$ s^{-1}, $1/60$ s^{-1}, $1/40$ s^{-1}, $1/30$ s^{-1}, $1/20$ s^{-1}, $1/10$ s^{-1}. The simulation and theoretical results are shown to be very close to each other.
Then we consider a request arrival rate of $1/660$ s^{-1}, the following values of w: (i) w=4; (ii) w=3; (iii) w=2, and we run experiments using these three configurations. Fig.6.9 shows the empirical CDFs for the three cases and the CDFs obtained using the proposed model. Also in this case, the simulation and theoretical results are shown to be very close to each other, indicating that the proposed model closely approximates a real IoT system.

Model of Power Consumption

Devices consume energy when performing three main tasks: (i) data sampling, e.g., sensing from the environment, for example, the temperature, the humidity, the pressure, the fluid flow, etc.; (ii) data processing, performed after sampling and involving operations like storage, denoising, etc.; (iii) data communication, which includes all necessary networking tasks like packet transmissions and receptions, protocol overheads due to control traffic, etc. We model the energy consumption as depending on two components: (i) sampling energy consumption, due to the sens-

ing operation and the processing of the sampled data; (ii) communication energy consumption, due to the transmission of the updates. The computational energy cost can be considered negligible, as it becomes significant only in some specific cases involving complex mathematical operations or very long sleep times. Denote c_T as the energy consumption for transmitting an update message and denote c_S as the energy consumption for generating a new sample. The energy consumption on the device depends on the frequency of these two operations, i.e., on the transmission frequency and on the sampling frequency. In our system model, the transmission frequency, that we denote as f_T, is the poll frequency, i.e., it is the inverse of the average time between two requests that trigger a cache refresh: $f_T(w,s) = 1/E\{T\}$. The sampling frequency, that we denote as f_S, is instead the inverse of the sampling period: $f_S(w,s) = 1/s$. So, the energy consumption per time unit c on the device is

$$c = c_T f_T + c_S f_S \tag{6.18}$$

Given the wide diversity of the IoT devices, c_T and c_S can range from very small values to very large values, relative to each other. Without losing generality, we normalize c with respect to the sum of c_T and c_S, i.e., with respect to the sum of the energy cost of one transmission operation and the energy cost of one sampling operation:

$$\frac{c}{c_T + c_S} = \frac{c_T}{c_T + c_S} f_T + \frac{c_S}{c_T + c_S} f_S \tag{6.19}$$

Denote $c/(c_T + c_S)$ as $c_{\overline{\beta}}$, and $c_T/(c_T + c_S)$ as $\overline{\beta}$, it is:

$$c_{\overline{\beta}} = \overline{\beta} f_T + (1 - \overline{\beta}) f_S \tag{6.20}$$

with $\overline{\beta} \in [0,1]$. This means that $c_{\overline{\beta}}$ takes into account the relationship between the energy consumption of a transmission operation and of a sampling operation, but it does not depend on their absolute values. Indeed, the parameter $\overline{\beta}$ indicates the energy cost of a transmission operation with respect to the sum of the energy costs of a transmission operation and of a sampling operation. The value of $\overline{\beta}$ depends on the type of device, e.g., for a device where the energy cost of a sampling operation is negligible with respect to the energy cost of a transmission operation $\overline{\beta}$ tends to one; on the contrary, for a device where the energy cost of a transmission operation is negligible with respect to the energy cost of a sampling operation $\overline{\beta}$ tends to zero. In (Razzaque and Dobson, 2014) they compute the operational energy costs in wireless sensor networks focusing on energy consumption during a single sampling period. They consider several commercial sensors, and they present a comparison of their sensing and communication energy costs. Comparisons are normalized with respect to the communication energy. In Tab.6.1 we report the results for six exemplary sensors for which we compute the corresponding value of $\overline{\beta}$ starting from

Sensor	c_S/c_T	$\overline{\beta}$
MMA7269Q (Accelerometer)	0.0000268	0.97
GE/Telaire 6004 (CO_2 sensor)	1249.25	0.0008
SHT1X (H) (Humidity sensor)	0.4	0.71
SHT1X (T) (Temperature sensor)	1.5	0.4
CP 18 (Proximity sensor)	0.267	0.8
LUC-M10 (Level sensor)	9.22	0.098

Table 6.1: Exemplary values of $\overline{\beta}$ computed from commercial-sensors parameters (Razzaque and Dobson, 2014).

the given values of c_S/c_T. Clearly, for sensors where the cost of a sensing operation is much higher than the cost of a transmission, the value of $\overline{\beta}$ is close to zero; instead, for sensors where the cost of a transmission is much higher than the cost of a sampling operation, the value of $\overline{\beta}$ is close to one.

Model-Driven Cache-Management Optimization

We propose a model-driven method to choose the values of the two parameters w and s that minimize $c_{\overline{\beta}}$, under the constraint given by the AoI requirement. More in details, the values of w and s are the solutions of the following optimization problem:

$$\min_{w,s} \quad c_{\overline{\beta}} \tag{6.21a}$$

$$\text{subject to} \quad s \geq s_{min}, \tag{6.21b}$$

$$P_{AoI}(AoI_\alpha) \geq \alpha \tag{6.21c}$$

The constraint 6.21b is the hardware constraint of the device, while the constraint 6.21c is the AoI requirement of the application. The latter can be expressed in a solvable form using the proposed model. Indeed, since the model allows us to compute the closed form of the probability distribution function, we can derive a condition on w such that the probability distribution function goes through the point (AoI_α, y), with $y \geq \alpha$. So, we need to find δ, s and w such that $AoI_\alpha = \delta W$ and $P_{AoI}(\delta W) \geq \alpha$. We obtain:

- If $1 \leq \delta w \leq w$ (i.e., $s \leq AoI_\alpha \leq W$):

$$w \leq \frac{AoI_\alpha}{\alpha s} + \frac{e^{-\lambda s}(1-\alpha)}{(1-e^{-\lambda s})\alpha} \tag{6.22}$$

- If $0 \leq \delta w < 1$ (i.e., $0 \leq AoI_\alpha < s$):

$$w \leq \frac{\frac{AoI_\alpha}{\alpha s} - e^{-\lambda s}}{1-e^{-\lambda s}} \tag{6.23}$$

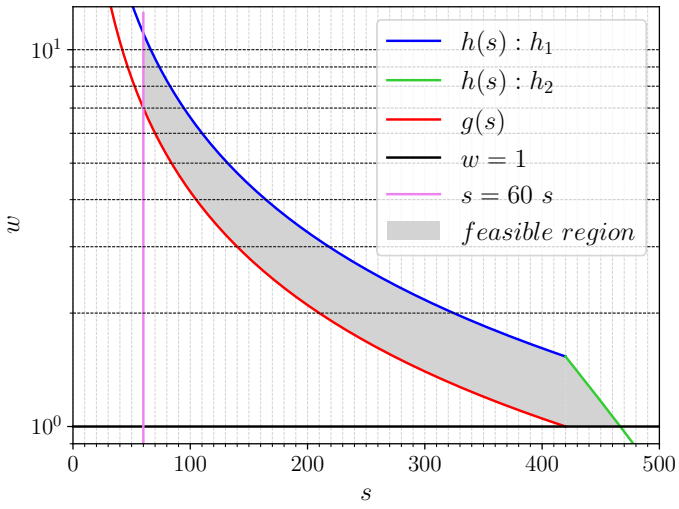


Figure 6.10: Constraints for $AoI_\alpha = 420$ s, $\alpha = 0.9$, $\lambda = 1/1800$ s^{-1} and $s_{min} = 60$ s (log-scale on the y-axis).

Therefore, $g(s) \leq w \leq h(s)$, with:

$$h(s) = \begin{cases} h_1(s) = \frac{AoI_\alpha}{\alpha s} + \frac{e^{-\lambda s}(1-\alpha)}{(1-e^{-\lambda s})\alpha} & s \leq AoI_\alpha \\ h_2(s) = \frac{\frac{AoI_\alpha}{\alpha s} - e^{-\lambda s}}{1-e^{-\lambda s}} & s > AoI_\alpha \end{cases} \tag{6.24}$$

and

$$g(s) = \frac{AoI_\alpha}{s} \tag{6.25}$$

An example of the feasible region where the values of w and s satisfying the AoI requirement must fall is shown in Fig.6.10. Note that, although the region is highlighted as a two-dimensional area, the admissible solutions are only those on the segments for which w takes an integer value.
Finally, the optimization problem can be reformulated as follows:

$$\min_{w,s} \quad c_{\overline{\beta}} \tag{6.26a}$$

$$\text{subject to} \quad s \geq s_{min}, \tag{6.26b}$$

$$g(s) \leq w \leq h(s), \tag{6.26c}$$

$$w \in \mathbb{Z}^+, s \in \mathbb{R}^+ \tag{6.26d}$$

In Fig.6.11 we show the values of $c_{\overline{\beta}}$ inside the feasible region for different types of sensors, expressed by different values of $\overline{\beta}$. For clarity, in the figure we reported the values of $c_{\overline{\beta}}$ for all the pairs of values of w and s inside the feasible region; however,

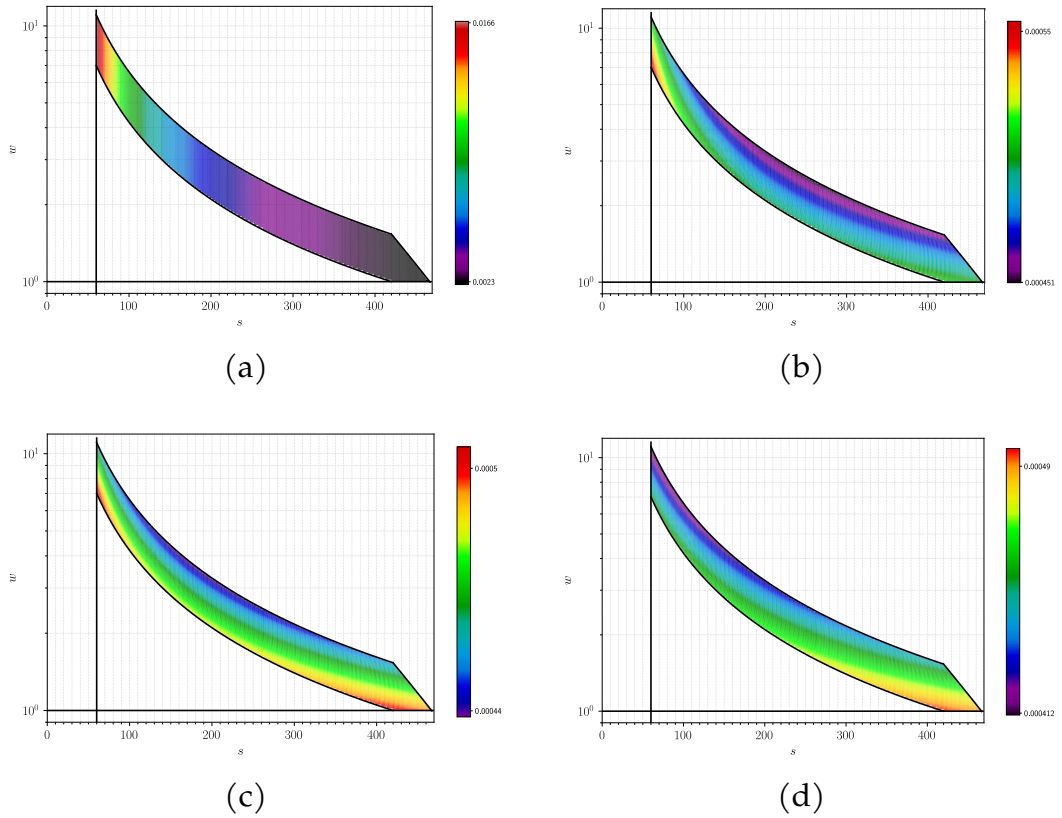


Figure 6.11: Values of $c_{\overline{\beta}}$ for $AoI_\alpha = 420$ s, $\alpha = 0.9$, $\lambda = 1/1800$ s^{-1} and $s_{min} = 60$ s, varying $\overline{\beta}$: (a) $\overline{\beta} = 0$, (b) $\overline{\beta} = 0.994$, (c) $\overline{\beta} = 0.997$, (d) $\overline{\beta} = 1$.

the only admissible pairs are those having $w \in \mathbb{Z}^+$. We can notice that for $\overline{\beta} = 0$ (the energy consumption for transmitting is zero, i.e., $c_T = 0$) the minimum value of $c_{\overline{\beta}}$ is on the lower right corner of the feasible region; as $\overline{\beta}$ increases, the minimum value starts shifting on the left, up to the top left corner of the feasible region when $\overline{\beta} = 1$ (the energy consumption for sampling is zero, i.e., $c_S = 0$). In the following, we study the objective function for the extreme cases of a device for which the energy consumption for transmitting is zero and a device for which the energy consumption for sampling is zero, having respectively $\overline{\beta} = 0$, $\overline{\beta} = 1$; and then for the general case of $0 < \overline{\beta} < 1$, representing hybrid sensors.

Devices with transmission energy consumption equal to zero: $\overline{\beta} = 0$

When $\overline{\beta} = 0$, i.e., $c_{\overline{\beta}} = f_S$, it is possible to compute the optimum values of w and s in closed form (see Appendix A), obtaining $w^* = 1$ and $s^* = AoI_\alpha/\alpha$, as can be also seen in Fig.6.11. Since $c_{\overline{\beta}} = f_S$, it follows that $c_{\overline{\beta}}$ does not depend on the rate of requests λ, but depends only on the AoI requirement, i.e., AoI_α and α. When the

value of AoI_α increases, the optimum value of $c_{\overline{\beta}}$ decreases, and when $AoI_\alpha \to \infty$, the objective function tends to 0:

$$\lim_{AoI_\alpha \to +\infty} f_S(s^*, w^*) = 0$$

Indeed, if $AoI_\alpha \to \infty$, there is not the need for the refreshing the data.
When the value of α decreases, the optimum value of $c_{\overline{\beta}}$ decreases as well, whereas when $\alpha \to 0$, the objective function tends to 0:

$$\lim_{\alpha \to 0} f_S(s^*, w^*) = 0$$

Indeed, $\alpha \to 0$ means that the fraction of requests that needs to receive a data item whose AoI is not larger than the target value tends to zero. However, typical real use cases will require higher values of α, e.g., 0.8, 0.9 or 0.95.

Devices with sampling energy consumption equal to zero: $\overline{\beta} = 1$

When $\overline{\beta} = 1$, i.e., $c_\beta = f_T$, if we remove the integer constraint on w, it is possible to compute the optimum values of w and s in closed form (see Appendix B), obtaining $s^* = s_{min}$ and $w^* = f(s_{min}) = \frac{AoI_\alpha}{\alpha s_{min}} + \frac{e^{-\lambda s_{min}}(1-\alpha)}{(1-e^{-\lambda s_{min}})\alpha}$ if $s \leq AoI_\alpha$, or $w^* = f(s_{min}) = \frac{\frac{AoI_\alpha}{\alpha s_{min}} - e^{-\lambda s_{min}}}{1-e^{-\lambda s_{min}}}$ if $s > AoI_\alpha$. The same result can be also seen graphically in Fig.6.11. In this case, $c_{\overline{\beta}}$ depends both on the requests rate λ and on the AoI requirement, i.e., AoI_α and α. When the value of λ decreases also the optimum value of $c_{\overline{\beta}}$ decreases (see Fig.6.12), and when $\lambda \to 0$ it is:

$$\lim_{\lambda \to 0} f_T(s^*, w^*) = 0$$

Indeed, in this case each request triggers a refresh with high probability, but, since the request rate is extremely low, only few messages are exchanged in the network. Instead, when $\lambda \to \infty$, it is (see Fig.6.12):

$$\lim_{\lambda \to +\infty} f_T(s^*, w^*) = \frac{\alpha}{AoI_\alpha}$$

In this case, the cache is refreshed almost periodically with period W.
Clearly, also in this case, when the value of AoI_α increases, the optimum value of $c_{\overline{\beta}}$ decreases, and when $AoI_\alpha \to \infty$, the objective function tends to 0:

$$\lim_{AoI_\alpha \to +\infty} f_T(s^*, w^*) = 0$$

And when the value of α decreases also the optimum value of $c_{\overline{\beta}}$ decreases, and when $\alpha \to 0$, the objective function tends to 0:

$$\lim_{\alpha \to 0} f_T(s^*, w^*) = 0$$

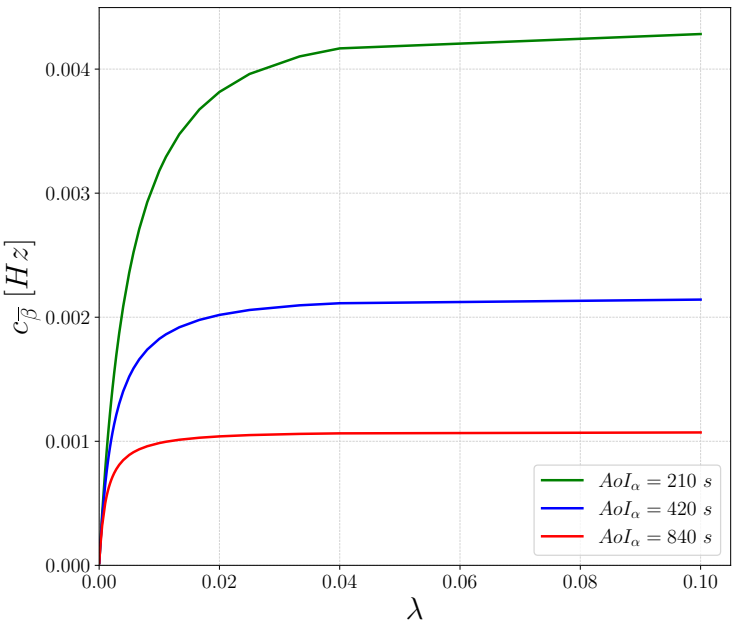


Figure 6.12: Values of $c_{\overline{\beta}}$ for $\overline{\beta} = 1$ varying λ and AoI_α.

These conclusions remain essentially the same also when considering the integer constraint on w: in this case we cannot compute the optimum values of w ans s in closed form, but we can only find a numerical solution using some optimization techniques and, as we can observe from Fig.6.13, the optimal value of s can be slightly greater than s_{min} to satisfy the integer constraint on w.

Hybrid sensors: $0 < \overline{\beta} < 1$

In this case, both the energy consumption of transmissions and the energy consumption of sensing are different from zero, and therefore there is a trade-off between minimizing the average poll frequency and minimizing the sampling frequency on the device, as the first leads to minimize s, indeed the optimization problem chooses $s \to s_{min}$, whereas the second leads to maximize s. It is possible to compute the optimum values of w and s solving 6.26a using optimization techniques for non-linear integer programming, e.g., branch and bound. Tab. 6.2 and Fig. 6.13 show the optimum values of w and s computed using the APMonitor solver (Hedengren et al., 2014), (Beal et al., 2018) considering $\alpha = 0.9$, $AoI_\alpha = 420$ s, and three exemplary cases where $\lambda = 1/90$ s^{-1}, $\lambda = 1/180$ s^{-1} and $\lambda = 1/360$ s^{-1}. We can notice that the optimum values are on the top edge of the feasible region, as in the cases showed in Fig.6.11. Moreover, we can also notice that several values of $\overline{\beta}$ can result in the same

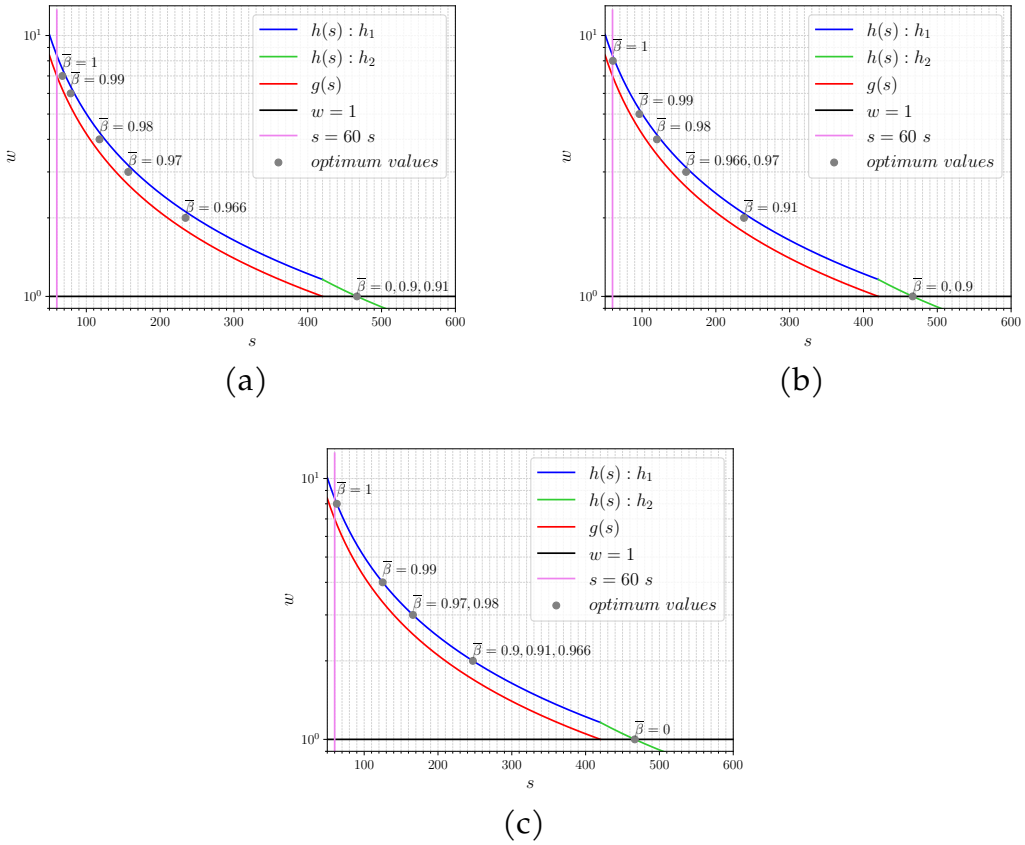


Figure 6.13: Optimum values of w and s for (a) $\lambda = 1/90$ s^{-1}, (b) $\lambda = 1/180$ s^{-1}, (c) $\lambda = 1/360$ s^{-1}, $\alpha = 0.9$, $AoI_\alpha = 420$ s, varying $\overline{\beta}$.

configuration of the parameters w and s: Tab. 6.2 shows that in all the considered scenarios, there are different values of $\overline{\beta}$, i.e., different types of devices, that have the same optimum values of w and s. This follows from the model constraint that w can only take integer values. However, the resulting values of $c_{\overline{\beta}}$ are different, as they

λ	$\overline{\beta}$	0	0.9	0.91	0.966	0.97	0.98	0.99	1
1/90 s^{-1}	w	1	1	1	2	3	4	6	7
	s	466.67	466.67	466.67	234.37	156.79	117.88	78.82	67.63
1/180 s^{-1}	w	1	1	2	3	3	4	5	8
	s	466.67	466.67	238.14	159.7	159.7	120.18	96.36	60.44
1/360 s^{-1}	w	1	2	2	2	3	3	4	8
	s	466.67	247.25	247.25	247.25	166.05	166.05	125.03	62.91

Table 6.2: Optimum values of w and s for $\lambda = 1/90$ s^{-1}, $\lambda = 1/180$ s^{-1}, $\lambda = 1/360$ s^{-1}, $\alpha = 0.9$, $AoI_\alpha = 420$ s, varying $\overline{\beta}$.

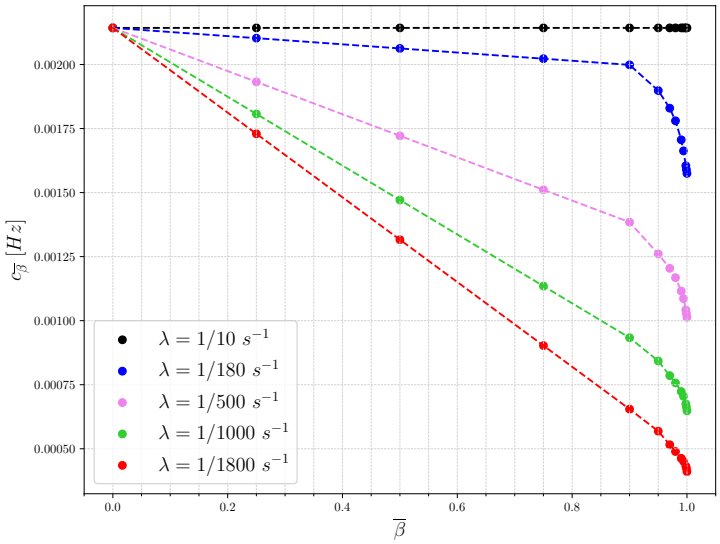


Figure 6.14: Values of $c_{\overline{\beta}}$ for $\alpha = 0.9$, $AoI_\alpha = 420$ s, $\lambda = 1/10$ s^{-1}, $\lambda = 1/180$ s^{-1}, $\lambda = 1/500$ s^{-1}, $\lambda = 1/1000$ s^{-1}, $\lambda = 1/1800$ s^{-1}.

depend on the value of $\overline{\beta}$, i.e., the energy consumption still depends on the type of device. Fig. 6.14 shows $c_{\overline{\beta}}$ for different types of devices and for the same AoI requirement and five possible network loads: $\lambda = 1/10$ s^{-1}, $\lambda = 1/180$ s^{-1}, $\lambda = 1/500$ s^{-1}, $\lambda = 1/1000$ s^{-1}, $\lambda = 1/1800$ s^{-1}. Clearly, as the request rate decreases also the transmission energy cost decreases. As mentioned above, in all the considered scenarios for values of $\overline{\beta}$ going from 0 up to 0.9-0.91-0.97, the optimum values of w and s are the same, hence their values of $c_{\overline{\beta}}$ differ only for $\overline{\beta}$, showing a linear behavior. Instead, when $\overline{\beta} \to 1$ the optimum values of w and s change and, at lower rates we also have that $f_T \to 0$, so $c_{\overline{\beta}} \to 0$, causing the steep change in the slope of the curve.

Sensitivity Analysis

The cache optimizer needs to receive as inputs the AoI requirements of the application, i.e., α and AoI_α, and the request rate λ. The proxy receives the values of α and AoI_α from the server during the initial configuration phase; instead, it needs to estimate the value of the request rate λ. So, the estimated value of λ may be affected by an estimation error, or the actual value of λ may not be constant but have some small fluctuations that are not seen by the proxy. To assess the sensitivity of the proposed model to variations of the parameter λ, we evaluate our model in a sample scenario in which we assume that the AoI requirements are $\alpha = 0.9$ and $AoI_\alpha = 420$ s, for different values of $\overline{\beta}$ and λ. Typically, estimating the rate of a Poisson process

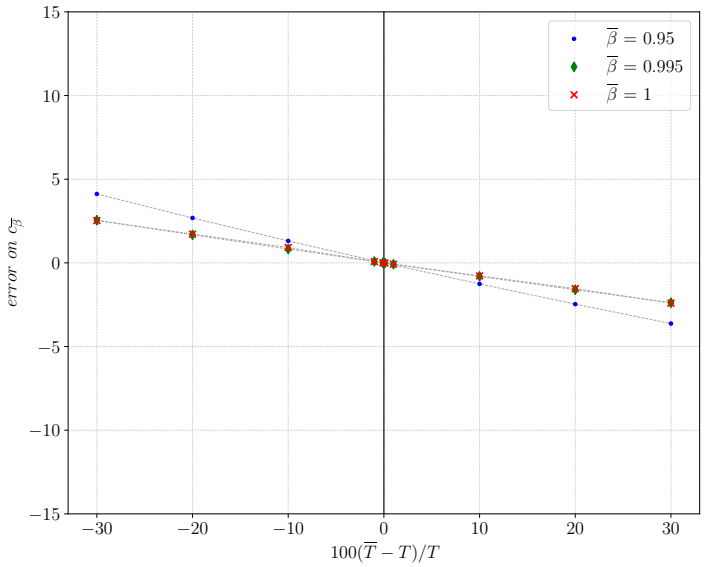


Figure 6.15: Percentage variation of the normalized energy cost for $\lambda = 1/1800$ s^{-1}, $\overline{\beta} = 0.95$ (circles), $\overline{\beta} = 0.995$ (diamonds), $\overline{\beta} = 1$ (crosses).

requires estimating the mean inter-arrival time; so, in the following we show the sensitivity of our model when the estimated mean inter-arrival time, denoted as $\overline{T}$, is different from the actual mean inter-arrival time, denoted as T. Since we want to evaluate the impact of an estimation error of T, we are considering devices where the predominant energy cost is the transmission energy cost, i.e., $\overline{\beta} \to 1$: indeed, in these cases $c_{\overline{\beta}}$ depends also on the request rate and hence is more affected by estimation errors on T. Call w^* and s^* the optimum values obtained considering the mean inter-arrival time T and call $\overline{w}$ and $\overline{s}$ the optimum values obtained considering the estimated inter-arrival time $\overline{T}$. Fig. 6.15 shows the percentage variation of $c_{\overline{\beta}}$, calculated as follows:

$$\frac{c_{\overline{\beta}}(T, \overline{w}, \overline{s}) - c_{\overline{\beta}}(T, w^*, s^*)}{c_{\overline{\beta}}(T, w^*, s^*)} \cdot 100 \tag{6.27}$$

as a function of the percentage variation of T, i.e., $100 \cdot (\overline{T} - T)/T$.

Moreover, call π_α the $\alpha - th$ percentile of P_{AoI}. Fig.6.16 shows the percentage variation of π_α, calculated as follows:

$$\frac{\pi_\alpha(T, \overline{w}, \overline{s}) - \pi_\alpha(T, w^*, s^*)}{\pi_\alpha(T, w^*, s^*)} \cdot 100 \tag{6.28}$$

as a function of the percentage variation of T. From Fig. 6.15 we can notice that when T is underestimated, the resulting energy consumption cost is larger than the

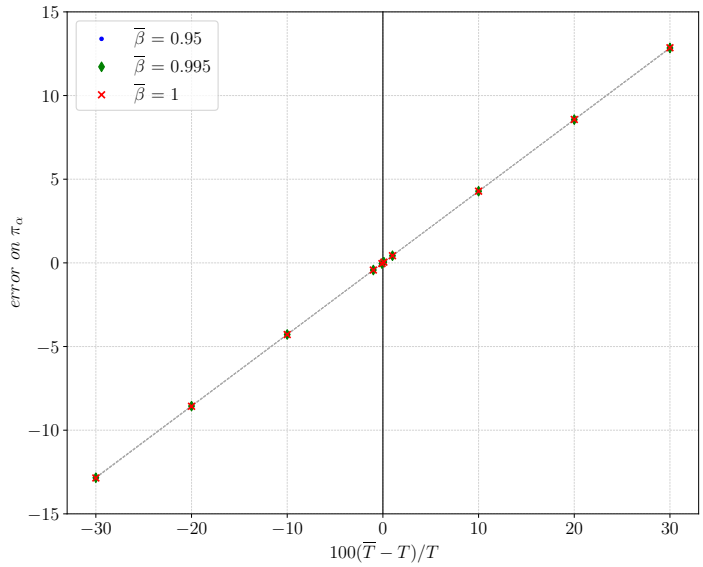


Figure 6.16: Percentage variation of AoI_α for $\lambda = 1/1800$ s^{-1}, $\overline{\beta} = 0.95$ (circles), $\overline{\beta} = 0.995$ (diamonds), $\overline{\beta} = 1$ (crosses).

minimum value, though the variation is small. On the other hand, when T is overestimated, the energy consumption is smaller, but the AoI requirement is not satisfied (see Fig. 6.16). However, for small variations of T, also the percentage variation of π_α is small.
Finally, Fig. 6.17 shows the percentage variation of $c_{\overline{\beta}}$ for different values of $\lambda = 1/T$. We can notice that lower rates are more affected by estimation errors, because as λ tends to zero, also the transmission energy cost tends to zero, and hence larger errors on T have a larger impact on $c_{\overline{\beta}}$. However, the graphs show that the model is robust, indeed if we assume that T varies up to 30%, in the worst case the error is slightly more than 10%.

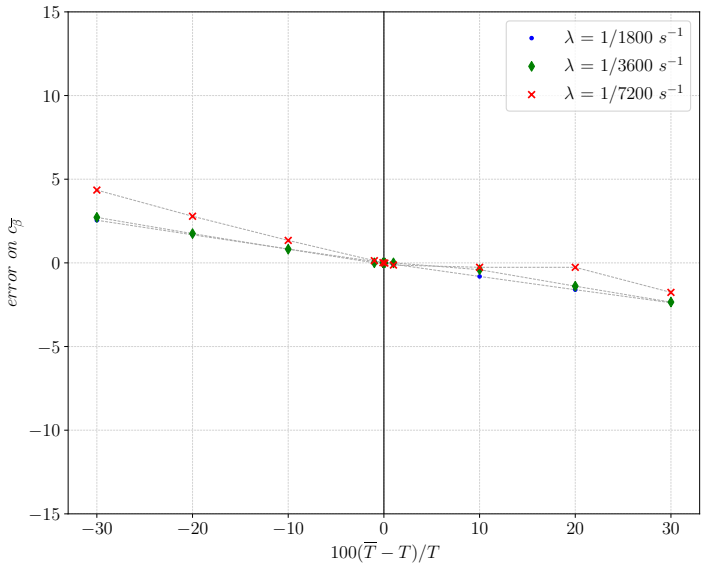


Figure 6.17: Percentage variation of the normalized energy cost: $\lambda = 1/1800$ s^{-1} (circles), $\lambda = 1/3600$ s^{-1} (diamonds), $\lambda = 1/7200$ s^{-1} (crosses), $\overline{\beta} = 0.995$.

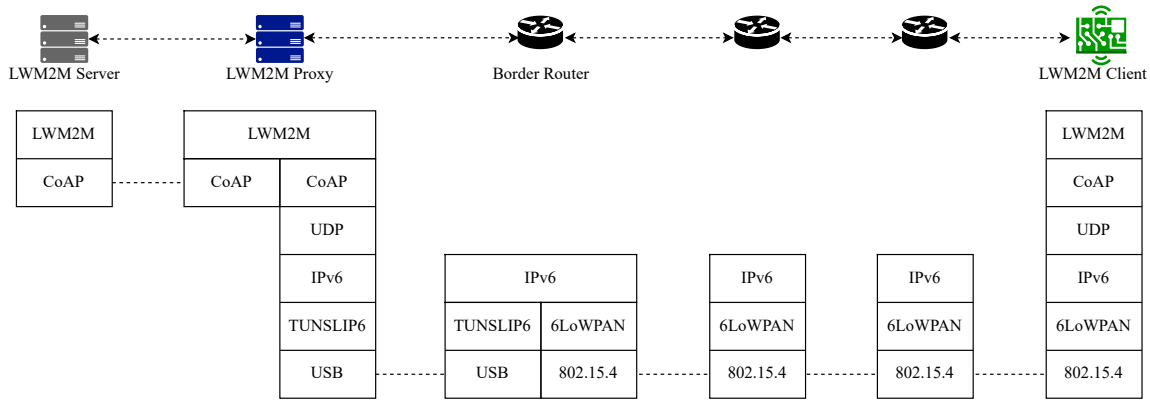


Figure 6.18: Proposed solution instantiated in a 6LoWPAN network.

Performance Evaluation

To assess the performance of our proposed optimized cache, we emulate an IoT system where we use the LWM2M protocol to manage the IoT devices, and we consider a scenario consisting of an IoT network, a LWM2M Server, i.e., the IoT application, and a LWM2M Proxy that implements our proposed optimized cache. We consider the same scenario used in Sect. 6.2 (see Fig. 6.18): an IoT network is emulated using the COOJA network emulator and uses the 6LoWPAN protocol on top of the IEEE 802.15.4 MAC, and the RPL routing protocol. The wireless devices of the IoT network run the Contiki-NG operating system and are connected to the Internet through the 6LoWPAN Border Router. A device located three hops away from the border router runs the LWM2M Client, exposing a LWM2M Object representing the sensor. The LWM2M Proxy is located outside the IoT network and manages the requests for the LWM2M Client sent by the LWM2M Server using the proposed cache-management scheme. The LWM2M Server and the LWM2M Proxy are implemented using the Eclipse Leshan library. Each experiment lasted 400000 s, and the resulting average service delay, i.e., the interval between the time an application issues a request and the time it receives response, is 326.5 ms (95% CI [322.6, 330.4]), which is negligible compared to the chosen value of AoI_α, i.e., 420 s.
In our first experiment α is 0.9 and the generation of application-requests follows a Poisson distribution with a cumulative rate $\lambda = 1/180$ s^{-1}. Fig.6.19 shows the empirical cumulative distribution functions and the cumulative distribution functions obtained through the model for the following values of $\overline{\beta}$: 0.5, 0.97, 1. Tab. 6.3 shows the values of w and s chosen by the optimizer. We can see that the empirical and the theoretical results are very close to each other, so the empirical cumulative distribution functions obtained with the values of w and s chosen by the model always satisfy the AoI requirements. Moreover, Tab. 6.3 also shows the values of $\overline{AoI}$ in all the considered cases: we can notice the trade-off between minimizing the average

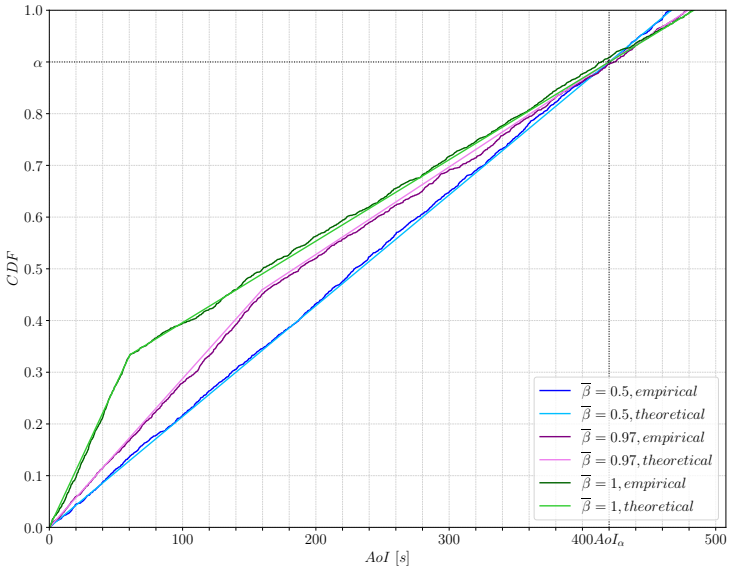


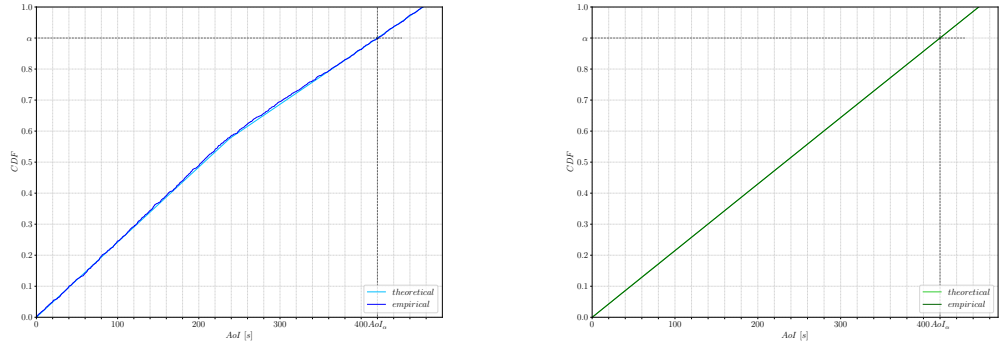
Figure 6.19: Theoretical AoI CDF and empirical AoI CDF for $\lambda = 1/180$ s^{-1}, $\alpha = 0.9$, $AoI_\alpha = 420$ s.

$\overline{\beta}$	**0.5**	**0.97**	**1**
w	1	3	8
s	466.67	159.7	60.44
$\overline{AoI}$	231.53 95% CI[221.93, 241.16]	212.31 95% CI[203.51, 221.11]	189.98 95% CI[182.08, 197.88]

Table 6.3: w, s, and $\overline{AoI}$ for $\lambda = 1/180$ s^{-1}, $\alpha = 0.9$, $AoI_\alpha = 420$ s, varying $\overline{\beta}$.

poll frequency and minimizing the sampling frequency, indeed when $\overline{\beta} = 1$, it is $s \to s_{min}$, that results in a lower value of $\overline{AoI}$, but this comes at the cost of a higher sampling power consumption.

In our second experiment we consider the same configuration as the previous experiment, but now multiple applications send periodic requests for the state of the device. We consider two cases: (i) in the first scenario ten applications have request periods of the same order of magnitude, e.g., the applications have similar characteristics, and so all the request periods are randomly extracted from a uniform distribution between 1000 and 3000 seconds; (ii) in the second scenario, instead, five applications have periods uniformly distributed between 50 and 100 seconds, while the remaining five have periods uniformly distributed between 5000 and 10000 seconds, e.g., we consider two different classes of applications. In the first scenario we consider $\overline{\beta} = 0.95$, in the second scenario we assume $\overline{\beta} = 1$. The optimizer computes



(a) (b)

Figure 6.20: AoI CDF for the case of periodic requests: (a) first scenario, (b) second scenario.

the optimum values using λ as the sum of the inverses of the periods and chooses (i) $w = 2$ and $s = 238.1$ s for the first scenario and (ii) $w = 7$ and $s = 66.68$ s for the second scenario. Fig. 6.20 shows the empirical cumulative distribution functions and the cumulative distribution functions obtained through the model for the two scenarios: we can notice that the empirical cumulative distribution functions and the theoretical cumulative distribution functions are very close to each other, so the optimum values of w and s obtained through the model can be applied also in this case.

In the next experiments, we assess the performance of the optimized cache for different types of devices. We compare the case in which the proxy implements the optimized cache against the case in which the proxy does not implement the cache. In the latter case we consider $s = AoI_\alpha/\alpha$, i.e., the maximum value of s that satisfies the AoI constraint. In Fig. 6.21 we report the value of $c_{\overline{\beta}}$ obtained for different types of devices both without the cache and with the optimized cache. We can notice that the optimized cache can significantly reduce the energy cost of devices for which the transmissions cost is the prevalent cost, because it can significantly reduce the number of transmissions, especially for high values of λ. We can also notice that using the optimized cache makes the system less sensitive to higher rates, as the number of exchanged messages depends on the refresh window.

Finally, we evaluate the service delay, i.e., the time between a request is sent by the application and the time its response is received. Fig. 6.22 shows the cumulative distribution function of the service delay for $\lambda = 1/180$ s^{-1} both for the case in which the proxy does not implement the cache and for the case wherein the proxy implements the optimized cache. For the latter we consider three values of $\overline{\beta}$, namely 0.5, 0.97 and 1. All the scenarios satisfy the same AoI requirements: $AoI_\alpha = 420$ s and $\alpha = 0.9$. Clearly, we can notice that the cache-enabled proxy always provides

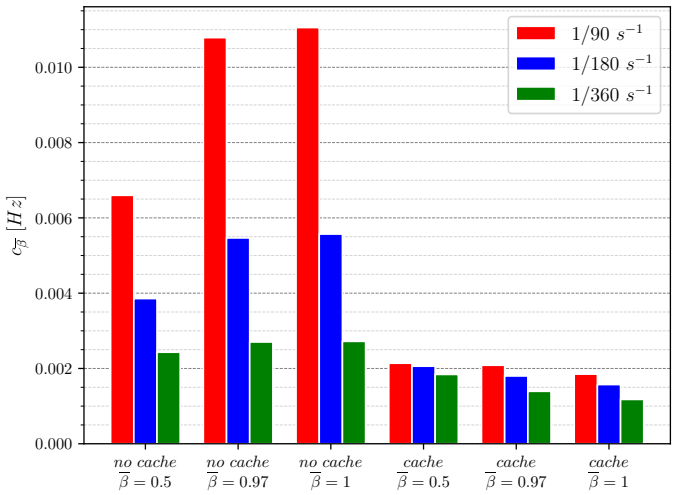


Figure 6.21: $c_{\overline{\beta}}$ w/ and w/o the cache for $\alpha = 0.9$, $AoI_{\alpha} = 420$ s, $\lambda = 1/360$ s^{-1}, $\lambda = 1/180$ s^{-1} and $\lambda = 1/90$ s^{-1}, varying $\overline{\beta}$.

quicker responses with respect to the case in which the cache is not implemented, as some responses are taken from the cache, as showed by the CDFs. Indeed, the CDFs obtained with the cache show a bi-modal behavior: some responses are taken from the cache and hence have smaller service delays, while some responses are forwarded to the device and hence have larger service delays. Moreover, we can also notice that the case $\overline{\beta} = 1$ is the configuration that minimizes the service delay. As a matter

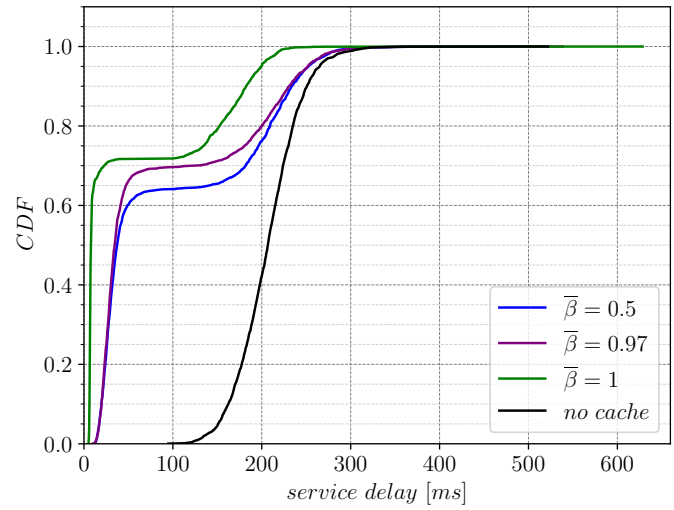


Figure 6.22: Service delay for $\lambda = 1/180$ s^{-1}.

of fact, $\overline{\beta} = 1$ is the case in which the predominant energy cost is the transmission cost, so it minimizes the number of exchanged messages with the device and, hence, it is also the configuration that minimizes the service delay.

Finally, it is worth to notice that this optimized per-device cache uses a simple yet effective management scheme that does not pose any limitation on the number of applications issuing requests on the device and that has constant complexity, i.e., it only involves a comparison between the AoI of the cached item and its refresh window. So, our solution can be easily applied in deployments involving multiple IoT devices just scaling vertically, i.e., adding more resources to the proxy, or scaling horizontally, i.e., replicating the proxy.

Chapter 7

Conclusions

The ubiquitous presence of connected things all around us is the fundamental tenet of IoT. The systems created by these IoT devices produce enormous amounts of data that are subsequently communicated to IoT applications for analysis and decision-making. Best-effort delivery is sufficient in some circumstances; however, in others, such as Industrial IoT, applications may be subject to severe QoS standards, notably in terms of latency, as they must be informed of the state of the devices as soon as feasible. The implementation of such systems presents a number of difficulties since the IoT devices and the lossy, low-power networks they communicate over impose severe energy, memory, processing, and communication constraints that limit how quickly this high volume of status updates can be transmitted.
We suggest implementing a set of functions to provide QoS-aware services in order to increase system performance; these functions should be implemented between IoT devices and IoT applications, for example, in an IoT proxy. We propose:

- a core function that implements an enhanced version of the application-layer device-management protocol to reduce the number of exchanged messages and the number of operations to be executed on the device. It also orchestrates the other three functions:

- a load-shaping function that regulates the flow of requests sent over the IoT network, so that the IoT application experiments reduced delays and packet losses. To do so, this function implements a congestion window that limits the number of pending requests in the IoT network;

- a compressing function that compresses and decompresses the messages exchanged with the devices by means of a SCHC-based compression mechanism, to overcome the limits on packet size imposed by the energy-constrained networks. We propose an extension of SCHC that offers both headers and payload compression and we define a set of SCHC compression rules to compress the

messages exchanged in a network managed by means of the LWM2M device-management protocol;

- a caching function that minimizes the energy consumed by the device, while taking into account the information freshness requirements. The freshness of a cached item is quantified using the Age of Information (AoI) metrics, and a cached item is no longer considered fresh when its AoI exceeds the value of the cache parameter denoted as refresh window, W; in this case the cache is refreshed by fetching the last update from the device. In order to configure the value of W, we developed and solved an optimization problem that minimizes energy consumption while satisfying an AoI requirement specified by the IoT application. This problem took into account both the sampling frequency and the average frequency of requests sent to the device for refreshing the cache.

These functions can be applied singularly or together depending on the IoT application demands.
Experiments examining an IoT proxy that implements each of the proposed functions show that these functions can considerably reduce the size of exchanged packets, can alleviate the traffic load of the IoT network, improving the QoS of the system in terms of a reduced latency and a reduced packet loss, and can minimize the device energy consumption while satisfying the AoI requirements.

Appendix A

Devices with $\overline{\beta} = 0$

When $\overline{\beta} = 0$, i.e., $c_{\overline{\beta}} = f_S$, the optimization problem maximizes s under the constraint given by the AoI requirement:

$$\max_{w,s} \quad s \tag{A.1a}$$

$$\text{subject to} \quad s \geq s_{min}, \tag{A.1b}$$

$$g(s) \leq w \leq h(s), \tag{A.1c}$$

$$w \in \mathbb{Z}^+, s \in \mathbb{R}^+ \tag{A.1d}$$

The model tends to maximize s, but, when $s \to +\infty$, it is $w \to 0$:

$$\lim_{s \to +\infty} \frac{\frac{AoI_\alpha}{\alpha s} - e^{-\lambda s}}{1 - e^{-\lambda s}} = 0$$

It must be $w \geq 1$, so:

$$\frac{\frac{AoI_\alpha}{\alpha s} - e^{-\lambda s}}{1 - e^{-\lambda s}} \geq 1 \tag{A.2}$$

That results in:

$$s \leq \frac{AoI_\alpha}{\alpha} \tag{A.3}$$

The maximum value is obtained for $w = 1$ and $s = \frac{AoI_\alpha}{\alpha}$.

Appendix B

Devices with $\overline{\beta} = 1$

When $\overline{\beta} = 1$, i.e., $c_{\overline{\beta}} = f_T$, the optimization problem maximizes $E\{T\}$ under the constraint given by the AoI requirement:

$$\max_{w,s} \quad ws + \frac{s}{e^{\lambda s} - 1} \tag{B.1a}$$

$$\text{subject to} \quad s \geq s_{min}, \tag{B.1b}$$

$$g(s) \leq w \leq h(s), \tag{B.1c}$$

$$w \in \mathbb{Z}^+, s \in \mathbb{R}^+ \tag{B.1d}$$

For a given s, the model chooses the maximum possible value of w to maximize the objective function, so it is necessary to study the objective function when $w = h(s)$. We denote $\varphi(w, s)$ as the objective function, i.e., $\varphi(w, s) = ws + \frac{s}{e^{\lambda s}-1}$, and we define $F(s) : F(s) \triangleq \varphi(f(s), s)$.
Therefore:

- If $s \leq AoI_\alpha \leq sw$
  It is:

$$w = \frac{AoI_\alpha}{\alpha s} + \frac{e^{-\lambda s}(1-\alpha)}{(1-e^{-\lambda s})\alpha} \tag{B.2}$$

  and

$$F(s) = \left[\frac{AoI_\alpha}{\alpha s} + \frac{e^{-\lambda s}(1-\alpha)}{(1-e^{-\lambda s})\alpha}\right] s + \frac{s}{e^{\lambda s} - 1} \tag{B.3}$$

$$= \frac{AoI_\alpha}{\alpha} + \frac{1}{\alpha}\frac{se^{-\lambda s}}{(1-e^{-\lambda s})} \tag{B.4}$$

  So,

$$F^{'}(s) = \frac{1}{\alpha}\frac{(1-e^{-\lambda s} - s\lambda)e^{-\lambda s}}{(1-e^{-\lambda s})^2} \tag{B.5}$$

And $F^{'}(s) \leq 0$ results in:

$$e^{-\lambda s}(1 - e^{-\lambda s} - s\lambda) \leq 0 \tag{B.6}$$

That is:

$$(1 - e^{-\lambda s} - s\lambda) \leq 0 \tag{B.7}$$

We define $x = \lambda s$ and $l(x) = 1 - e^{-x} - x$.
It is:

$$l(0) = 1 - 1 = 0 \tag{B.8}$$

and

$$l^{'}(x) = e^{-x} - 1 \tag{B.9}$$

So $l^{'}(x) \leq 0$ results in:

$$e^{-x} \leq 1 \tag{B.10}$$

for $x \geq 0$.
Therefore, $l(x)$ is always ≤ 0 for $x \geq 0$, because $l(x)$ is decreasing and it is $l(0) = 0$. This means that $F^{'}(s)$ is always ≤ 0 for $x \geq 0\,(\lambda s \geq 0)$, so $F(s)$ is decreasing and the maximum value is obtained for $s = s_{min}$. Then, we have to consider the integer constraint on w.

- If $0 \leq AoI_{\alpha} < s$
  It is:

$$w = \frac{\frac{AoI_{\alpha}}{\alpha s} - e^{-\lambda s}}{1 - e^{-\lambda s}} \tag{B.11}$$

and

$$F(s) = \left[\frac{\frac{AoI_{\alpha}}{\alpha s} - e^{-\lambda s}}{1 - e^{-\lambda s}}\right] s + \frac{s}{e^{\lambda s} - 1} \tag{B.12}$$

$$= \frac{\frac{AoI_{\alpha}}{\alpha} e^{\lambda s} - \frac{AoI_{\alpha}}{\alpha}}{e^{\lambda s} + e^{-\lambda s} - 2} \tag{B.13}$$

So,

$$F^{'}(s) = \frac{-\frac{AoI_{\alpha}}{\alpha}\lambda e^{\lambda s} - \frac{AoI_{\alpha}}{\alpha}\lambda e^{-\lambda s} + 2\frac{AoI_{\alpha}}{\alpha}\lambda}{(e^{\lambda s} + e^{-\lambda s} - 2)^2} \tag{B.14}$$

And $F^{'}(s) \leq 0$ results in:

$$-\frac{AoI_{\alpha}}{\alpha}\lambda e^{\lambda s} - \frac{AoI_{\alpha}}{\alpha}\lambda e^{-\lambda s} + 2\frac{AoI_{\alpha}}{\alpha}\lambda \leq 0 \tag{B.15}$$

That is:

$$-e^{\lambda s} - e^{-\lambda s} + 2 \leq 0 \tag{B.16}$$

We define $x = \lambda s$ and $l(x) = -e^{x} - e^{-x} + 2$
It is:

$$l(0) = -1 - 1 + 2 = 0 \tag{B.17}$$

and

$$l(x) = -e^{x} + e^{-x} \tag{B.18}$$

So $l^{'}(x) \leq 0$ results in:

$$e^{-x} \leq e^{x} \tag{B.19}$$

for $x \geq 0$.
So, $l(x)$ is always ≤ 0 for $x \geq 0$, because $l(x)$ is decreasing and it is $l(0) = 0$. This means that $F^{'}(s)$ is always ≤ 0 for $x \geq 0\,(\lambda s \geq 0)$, so $F(s)$ is decreasing and the maximum value is obtained for $s = s_{min}$. Then we have to consider the integer constraint on w.

Appendix C

Compression Rules

C.1 Read Request

FID	FL	FP	DI	TV	MO	CDA
CoAP Version	2	1	BI	1	equal	not sent
CoAP Type	2	1	DW	CON	equal	not sent
CoAP Token Length	4	1	BI	1	equal	not sent
CoAP Code	8	1	DW	GET	equal	not sent
CoAP Message ID	16	1	BI	0	MSB(12)	LSB
CoAP Token	8	1	BI		ignore	value sent
CoAP Uri Path		1	DW	1, 3	match mapping	mapping sent
CoAP Uri Path		2	DW		ignore	not sent
CoAP Uri Path		3	DW		ignore	value sent
CoAP Uri Path		4	DW		ignore	value sent
CoAP Accept		1	DW	0, 11542	match mapping	mapping sent

Table C.1: Read request compression rule.

C.2 Read Device

FID	FL	FP	DI	TV	MO	CDA
CoAP Version	2	1	BI	1	equal	not sent
CoAP Type	2	1	UP	ACK	equal	not sent
CoAP Token Length	4	1	BI	1	equal	not sent
CoAP Code	8	1	UP	2.05	equal	not sent
CoAP Message ID	16	1	BI	0	MSB(12)	LSB
CoAP Token	8	1	BI		ignore	value sent
CoAP Content format		1	UP	11542	equal	not sent
Instance ID		1	UP		ignore	value sent
Error Code		1	UP		ignore	value sent
Supported Bindings and Modes		1	UP	N	equal	not sent

Table C.2: Read Device Object response compression rule.

C.3 Read Server

FID	FL	FP	DI	TV	MO	CDA
CoAP Version	2	1	BI	1	equal	not sent
CoAP Type	2	1	UP	ACK	equal	not sent
CoAP Token Length	4	1	BI	1	equal	not sent
CoAP Code	8	1	UP	2.05	equal	not sent
CoAP Message ID	16	1	BI	0	MSB(12)	LSB
CoAP Token	8	1	BI		ignore	value sent
CoAP Content Format		1	UP	11542	equal	not sent
Instance ID		1	UP		ignore	value sent
Short Server ID		1	UP	1	equal	not sent
Lifetime		1	UP	2592000	equal	not sent
Notification Storing When Disabled or Offline		1	UP	0,1	match mapping	mapping sent
Binding		1	UP	N	equal	not sent

Table C.3: Read Server Object response compression rule.

C.4 Observe and Cancel Observation

FID	FL	FP	DI	TV	MO	CDA
CoAP Version	2	1	BI	1	equal	not sent
CoAP Type	2	1	DW	CON	equal	not sent
CoAP Token Length	4	1	BI	1	equal	not sent
CoAP Code	8	1	DW	GET	equal	not sent
CoAP Message ID	16	1	BI	0	MSB(12)	LSB
CoAP Token	8	1	BI		ignore	value sent
CoAP Observe		1	DW	0, 1	match mapping	mapping sent
CoAP Uri Path		1	DW	1, 3, 3303	match mapping	mapping sent
CoAP Uri Path		2	DW		ignore	value sent
CoAP Uri Path		3	DW		ignore	value sent
CoAP Uri Path		4	DW		ignore	value sent

Table C.4: Observe and Cancel Observation request compression rule.

C.5 Notify Temperature

FID	FL	FP	DI	TV	MO	CDA
CoAP Version	2	1	BI	1	equal	not sent
CoAP Type	2	1	UP	ACK, NON	match mapping	mapping sent
CoAP Token Length	4	1	BI	1	equal	not sent
CoAP Code	8	1	UP	2.05	equal	not sent
CoAP Message ID	16	1	BI	0	MSB(12)	LSB
CoAP Token	8	1	BI		ignore	value sent
CoAP Observe		1	UP		ignore	value sent
CoAP Content Format		1	UP	11542	equal	not sent
Instance ID		1	UP		ignore	value sent
Sensor Value		1	UP		ignore	value sent

Table C.5: Notify Temperature Object response compression rule.

C.6 Write Server

FID	FL	FP	DI	TV	MO	CDA
CoAP Version	2	1	BI	1	equal	not sent
CoAP Type	2	1	DW	CON	equal	not sent
CoAP Token Length	4	1	BI	1	equal	not sent
CoAP Code	8	1	DW	POST, PUT	match mapping	mapping sent
CoAP Message ID	16	1	BI	0	MSB(12)	LSB
CoAP Token	8	1	BI		ignore	value sent
CoAP Uri Path		1	DW	1	equal	not sent
CoAP Uri Path		2	DW	0	equal	not sent
CoAP Content Format		1	DW	11542	equal	not sent
Lifetime		1	DW		ignore	not sent
Notification Storing When Disabled or Offline		1	DW		ignore	value sent
Binding		1	DW		ignore	value sent

Table C.6: Write Server Object request compression rule.

C.7 Write a Server Resource

FID	FL	FP	DI	TV	MO	CDA
CoAP Version	2	1	BI	1	equal	not sent
CoAP Type	2	1	DW	CON	equal	not sent
CoAP Token Length	4	1	BI	1	equal	not sent
CoAP Code	8	1	DW	PUT	equal	not sent
CoAP Message ID	16	1	BI	0	MSB(12)	LSB
CoAP Token	8	1	BI		ignore	value sent
CoAP Uri Path		1	DW	1	equal	not sent
CoAP Uri Path		2	DW	0	equal	not sent
CoAP Uri Path		3	DW	1, 6, 7	match mapping	mapping sent
CoAP Content Format		1	DW	0	equal	not sent

Table C.7: Write a Server Resource request compression rule.

Appendix D

Publications

Journal papers

1. **M. Pappalardo**, A. Virdis, E. Mingozzi, "An Edge-Based LWM2M Proxy for Device Management to Efficiently Support QoS-Aware IoT Services", *IoT 3.1*, pages: 169–190, 2022. **Candidate's contributions**: conceptualization, software, performing the experiments, writing - original draft, review and editing

2. **M. Pappalardo**, A. Virdis, E. Mingozzi, "Energy-Optimized Content Refreshing of Age-of-Information-Aware Edge Caches in IoT Systems", *Future Internet 14.7*, pages: 197, 2022. **Candidate's contributions**: conceptualization, software, performing the experiments, writing - original draft, review and editing

Workshop papers

1. **M. Pappalardo**, G. Tanganelli, E. Mingozzi, "Enhanced Support of LWM2M in Low Power and Lossy Networks", *2020 IEEE International Conference on Smart Computing* (*SMARTCOMP*), pages: 344–349, 2020. **Candidate's contributions**: conceptualization, software, performing the experiments, writing - original draft, review and editing

2. **M. Pappalardo**, E. Mingozzi, A. Virdis, "A Model-Driven Approach to AoI-Based Cache Management in IoT", *2021 IEEE 26th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks* (*CAMAD*), pages: 1–6, 2021. **Candidate's contributions**: conceptualization, software, performing the experiments, writing - original draft, review and editing

Bibliography

(2020). Ieee standard for low-rate wireless networks. *IEEE Std 802.15.4-2020 (Revision of IEEE Std 802.15.4-2015)*, pages 1–800.

6lo (2022). https://datatracker.ietf.org/wg/6lo/about/.

6LoWPAN (2022). https://datatracker.ietf.org/wg/6lowpan/documents/.

Abd-Elmagid, M. A., Pappas, N., and Dhillon, H. S. (2019). On the role of age of information in the internet of things. *IEEE Communications Magazine*, 57(12):72–77.

Alexander, R., Brandt, A., Vasseur, J., Hui, J., Pister, K., Thubert, P., Levis, P., Struik, R., Kelsey, R., and Winter, T. (2012). RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550.

AllJoyn (2022). https://openconnectivity.org/technology/reference-implementation/alljoyn/.

Arrowhead (2022). https://arrowhead.eu/why-how/.

Atzori, L., Iera, A., and Morabito, G. (2010). The internet of things: A survey. *Computer networks*, 54(15):2787–2805.

Beal, L. D., Hill, D. C., Martin, R. A., and Hedengren, J. D. (2018). Gekko optimization suite. *Processes*, 6(8):106.

Beniwal, G. and Singhrova, A. (2021). A systematic literature review on iot gateways. *Journal of King Saud University-Computer and Information Sciences*.

Chiariotti, F., Holm, J., Kalør, A. E., Soret, B., Jensen, S. K., Pedersen, T. B., and Popovski, P. (2022). Query age of information: Freshness in pull-based communication. *IEEE Transactions on Communications*, 70(3):1606–1622.

Contiki-NG (2022). https://github.com/contiki-ng/contiki-ng.

COOJA (2022). https://github.com/contiki-ng/cooja.

Costa, M., Codreanu, M., and Ephremides, A. (2016). On the age of information in status update systems with packet management. *IEEE Transactions on Information Theory*, 62(4):1897–1910.

de Caldas Filho, F. L., Rocha, R. L., Abbas, C. J., Martins, L. M. E., Canedo, E. D., and de Sousa, R. T. (2019). Qos scheduling algorithm for a fog iot gateway. In *2019 Workshop on Communication Networks and Power Systems (WCNPS)*, pages 1–6. IEEE.

Elgabli, A., Khan, H., Krouka, M., and Bennis, M. (2019). Reinforcement learning based scheduling algorithm for optimizing age of information in ultra reliable low latency networks. In *2019 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6. IEEE.

Farrell, S. (2018). Low-Power Wide Area Network (LPWAN) Overview. RFC 8376.

FUOTA-WG (2022). `https://lora-alliance.org/wp-content/uploads/2020/11/fragmented_data_block_transport_v1.0.0.pdf`.

Gomez, C., Minaburo, A., Toutain, L., Barthel, D., and Zuniga, J. C. (2020). Ipv6 over lpwans: Connecting low power wide area networks to the internet (of things). *IEEE Wireless Communications*, 27(1):206–213.

Haque, A. B., Bhushan, B., and Dhiman, G. (2022). Conceptualizing smart city applications: Requirements, architecture, security issues, and emerging trends. *Expert Systems*, 39(5):e12753.

Haxhibeqiri, J., De Poorter, E., Moerman, I., and Hoebeke, J. (2018). A survey of lorawan for iot: From technology to application. *Sensors*, 18(11):3995.

Hedengren, J. D., Shishavan, R. A., Powell, K. M., and Edgar, T. F. (2014). Nonlinear modeling, estimation and predictive control in apmonitor. *Computers & Chemical Engineering*, 70:133–148.

Hoebeke, J., Haxhibeqiri, J., Moons, B., Van Eeghem, M., Rossey, J., Karagaac, A., and Famaey, J. (2018). A cloud-based virtual network operator for managing multi-modal lpwa networks and devices. In *2018 3rd Cloudification of the Internet of Things (CIoT)*, pages 1–8. IEEE.

Huang, H., Qiao, D., and Gursoy, M. C. (2022). Age-energy tradeoff optimization for packet delivery in fading channels. *IEEE Transactions on Wireless Communications*, 21(1):179–190.

IoTivity (2022). `http://iotivity.org/`.

Kaul, S., Gruteser, M., Rai, V., and Kenney, J. (2011). Minimizing age of information in vehicular networks. In *2011 8th Annual IEEE communications society conference on sensor, mesh and ad hoc communications and networks*, pages 350–358. IEEE.

Kaul, S., Yates, R., and Gruteser, M. (2012). Real-time status: How often should one update? In *2012 Proceedings IEEE INFOCOM*, pages 2731–2735. IEEE.

Kim-Hung, L. and Le-Trung, Q. (2020). User-driven adaptive sampling for massive internet of things. *IEEE Access*, 8:135798–135810.

Kosta, A., Pappas, N., Angelakis, V., et al. (2017). Age of information: A new concept, metric, and tool. *Foundations and Trends® in Networking*, 12(3):162–259.

Lai, W.-K., Wang, Y.-C., and Lin, S.-Y. (2020). Efficient scheduling, caching, and merging of notifications to save message costs in iot networks using coap. *IEEE Internet of Things Journal*, 8(2):1016–1029.

Leshan (2022). `https://github.com/eclipse/leshan`.

Li, C., Li, S., and Hou, Y. T. (2019). A general model for minimizing age of information at network edge. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 118–126. IEEE.

LoRaWAN-Specification (2022). `https://resources.lora-alliance.org/technical-specifications/lorawan-specification-v1-1`.

Ludovici, A. and Calveras, A. (2015). A proxy design to leverage the interconnection of coap wireless sensor networks with web applications. *Sensors*, 15(1):1217–1244.

Mekki, K., Bajic, E., Chaxel, F., and Meyer, F. (2019). A comparative study of lpwan technologies for large-scale iot deployment. *ICT express*, 5(1):1–7.

Minaburo, A., Toutain, L., Gomez, C., Barthel, D., and Zúñiga, J.-C. (2020). Schc: Generic framework for static context header compression and fragmentation. *RFC*, 8724:1–71.

Mingozzi, E., Tanganelli, G., and Vallati, C. (2014). Coap proxy virtualization for the web of things. In *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, pages 577–582. IEEE.

NetWorld2020 (2020). "smart networks in the context of ngi".

Nishimura, R. and Suzuki, Y. (2020). Ml and em estimation of sampling intervals of sensor devices. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4915–4919. IEEE.

Niyato, D., Kim, D. I., Wang, P., and Song, L. (2016). A novel caching mechanism for internet of things (iot) sensing service with energy harvesting. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE.

OMA (2020a). Lightweight machine to machine technical specification: Core.

OMA (2020b). Lightweight machine to machine technical specification: Transport bindings.

OMA-SpecWorks (2022). `https://omaspecworks.org/what-is-oma-specworks/iot/lightweight-m2m-lwm2m/`.

Razzaque, M. A. and Dobson, S. (2014). Energy-efficient sensing in wireless sensor networks using compressed sensing. *Sensors*, 14(2):2822–2859.

Robles, M. I., D'Ambrosio, D., Bolonio, J. J., and Komu, M. (2016). Device group management in constrained networks. In *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, pages 1–6. IEEE.

Sanchez-Gomez, J., Gallego-Madrid, J., Sanchez-Iborra, R., Santa, J., and Skarmeta, A. F. (2020). Impact of schc compression and fragmentation in lpwan: A case study with lorawan. *Sensors*, 20(1):280.

Semtech (2022a). `https://www.thethingsnetwork.org/forum/uploads/default/original/1X/4fbda86583605f4aa24dcedaab874ca5a1572825.pdf`.

Semtech (2022b). `https://www.thethingsnetwork.org/forum/uploads/default/original/1X/555030509bdcdee51a0d3d87382a17dd6211b11c.pdf`.

Shelby, Z., Hartke, K., and Bormann, C. (2014). The Constrained Application Protocol (CoAP). RFC 7252.

Sinha, B. B. and Dhanalakshmi, R. (2022). Recent advancements and challenges of internet of things in smart agriculture: A survey. *Future Gener. Comput. Syst.*, 126(C):169–184.

Sisinni, E., Saifullah, A., Han, S., Jennehag, U., and Gidlund, M. (2018). Industrial internet of things: Challenges, opportunities, and directions. *IEEE Transactions on Industrial Informatics*, 14(11):4724–4734.

Taivalsaari, A. and Mikkonen, T. (2017). A roadmap to the programmable world: software challenges in the iot era. *IEEE software*, 34(1):72–80.

Thread (2022). `https://www.threadgroup.org/`.

Xu, C., Wang, X., Yang, H. H., Sun, H., and Quek, T. Q. (2020). Aoi and energy consumption oriented dynamic status updating in caching enabled iot networks. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 710–715. IEEE.

Yates, R. D., Sun, Y., Brown, D. R., Kaul, S. K., Modiano, E., and Ulukus, S. (2021). Age of information: An introduction and survey. *IEEE Journal on Selected Areas in Communications*, 39(5):1183–1210.

Zhang, S., Li, J., Luo, H., Gao, J., Zhao, L., and Shen, X. S. (2018). Towards fresh and low-latency content delivery in vehicular networks: An edge caching aspect. In *2018 10th International Conference on Wireless Communications and Signal Processing (WCSP)*, pages 1–6. IEEE.

Zhang, S., Li, J., Luo, H., Gao, J., Zhao, L., and Shen, X. S. (2020). Low-latency and fresh content provision in information-centric vehicular networks. *IEEE Transactions on Mobile Computing*.

Zhang, S., Wang, L., Luo, H., Ma, X., and Zhou, S. (2021). Aoi-delay tradeoff in mobile edge caching with freshness-aware content refreshing. *IEEE Transactions on Wireless Communications*, 20(8):5329–5342.

Zhong, J., Yates, R. D., and Soljanin, E. (2018). Two freshness metrics for local cache refresh. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 1924–1928. IEEE.

Zhou, B. and Saad, W. (2018). Optimal sampling and updating for minimizing age of information in the internet of things. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6.