M.Sc. Thesis
Master of Science in Engineering

DTU Compute
Department of Applied Mathematics and Computer Science

Efficient Protocol for Monitoring Embedded Devices

Yago Fontoura do Rosário (s180136)

Kongens Lyngby 2020

DTU Compute
Department of Applied Mathematics and Computer Science
Technical University of Denmark

Richard Petersens Plads
Building 322
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary

Network monitoring has always being a challenge for network administrators. Their tasks are to keep track of every node in a network and ensure that they behave correctly. This is done by checking the network, CPU, memory or energy usage for anomalous behavior. The *Simple Network Management Protocol* (SNMP)[Pre02] was designed to aid network administrators in their everyday tasks. This protocol was designed for computers where resources like memory, CPU, energy, and others are not that constrained or for network devices, such as Routers, that can have constrained resources but have a very simple main task which doesn't utilize a lot of resources.

A new wave of devices has been coming in the last years where small embedded devices are also being connected to the network. These new devices are part of the *Internet of Things* (IoT)[1]. A good example is all the sensors that are spread around a city to analyze the car traffic and report it to the main station, a gateway. These devices compose *Wireless Sensor Network* (WSN) that according to [DP10] differ in many aspects from traditional networks. However, the main issue with these devices is the hardware constraints where they only have a very small RAM, ROM, and CPU, but the most challenging is the energy limitation. They are usually powered by batteries and can be deployed in harsh environments.

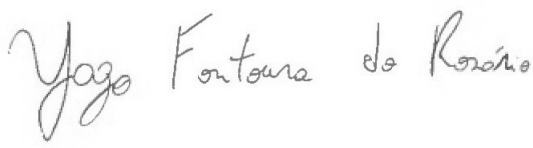
The current monitoring protocols were designed without taking the constraints from WSN nodes into account. One doesn't want to use many resources to monitor constrained devices. While the current protocols do work with these devices they have an unnecessary overhead resulting in a waste of resources that could otherwise be used for some other task, or to save power. This work proposes a new protocol designed for embedded devices and evaluates it against the state of the art.

[1] `http://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx`

Preface

This M.Sc. thesis was prepared at the Embedded Systems Engineering (ESE) section of the Department of Applied Mathematics and Computer Science of the Technical University of Denmark in fulfillment of the requirements for acquiring an M.Sc. degree in Computer Science.

Kongens Lyngby, January 2, 2020



Yago Fontoura do Rosário (s180136)

Acknowledgements

I would like to thank my supervisor Xenofon (Fontas) Fafoutis for his guidance and help during this thesis. Also, all of those that helped me in one way or another, directly or indirectly during this project.

Last but not least I would like to thank my family, especially my grandmother Alzira, for all their support not just during this project but throughout my entire life If I managed to acquire this title it is thanks to them.

Contents

List of Figures

List of Tables

List of Abbreviations

6LoWPAN IPv6 over Low-Power Wireless Personal Area Networks xi, 2, 9, 11, 12, 15, 18, 19, 35, 67, 75

BER Basic Encoding Rules xi, 18, 23, 32, 39

CBOR Concise Binary Object Representation xi, 31–33, 52, 70, 75
CLI Command Line Interface 2, 14–16
CoAP Constrained Application Protocol xi, xii, xv, 2, 3, 17, 19, 25–28, 31, 33, 35, 43–47, 49, 50, 55, 56, 58, 60, 61, 63, 65, 67–69, 75, 76
CoMP Constrained Monitoring Protocol xi–xiii, xv, 3, 19, 29–34, 36, 51–56, 58, 60, 61, 63, 65, 67, 69, 71–73, 75, 76
CSMA Carrier Sense Multiple Access 9

DODAG Destination Oriented Directed Acyclic Graph xii, 12, 37, 38

HTTP HyperText Transfer Protocol xii, 6, 17, 25, 28, 67–69, 71, 75

IAB Internet Architecture Board 16
IETF Internet Engineering Task Force 16
IoT Internet of Things i, 1, 5, 6, 17
ISP Internet Service Provider 19

JSON JavaScript Object Notation 25, 44, 46

KPI Key Performance Indicator 1, 25, 28, 35, 36, 44, 46, 71

LLDP Link Layer Discovery Protocol 18

MAC Medium Access Control xi, 2, 8–11, 36, 37
MIB Management Information Base 18, 19, 23, 24, 28, 34, 40, 52, 70, 75, 76
MQTT Message Queuing Telemetry Transport 17

NETCONF Network Configuration Protocol xi, 2, 16

OID Object Identifier 18, 19, 23, 24, 30–33, 51, 70, 71, 75, 76

PDR Packet Delivery Rate 36

PDU Protocol Data Unit xi, 21, 22, 29–31, 39
PRR Packet Reception Rate 36

QoS Quality of Service 1

RAM Random Access Memory xv, 5, 6, 14, 16, 17, 35, 39, 43, 51, 55
REST Representational State Transfer 68, 75, 76
ROM Read Only Memory xv, 5, 6, 14, 17, 35, 39, 43, 51, 55
RPL Routing Protocol for Low-Power and Lossy Networks 12, 18, 35, 36

SNMP Simple Network Management Protocol i, xi–xiii, xv, 2, 3, 13, 16, 18, 19, 21–24, 26, 28, 29, 32–34, 39–43, 55, 56, 67, 69–73, 75–82
SSH Secure Shell 14, 16

TCP Transmission Control Protocol 15–17, 68
TLS Transport Layer Security 6
TSCH Time Slotted Channel Hopping 9, 10, 36, 83

UDP User Datagram Protocol 15–17, 33, 68
URI Uniform Resource Identifier 17, 27, 44, 75

WSN Wireless Sensor Network i, xii, 1–3, 5, 7, 8, 12–14, 18, 19, 25, 29, 33, 35–37, 58, 67–69, 71, 72, 75, 76

XML Extensible Markup Language 6, 16

CHAPTER 1

Introduction

1.1 Motivation

According to [Hun] by 2020 there will be 4 connected devices for each human, this means that it is expected that there will 20 billion of these gadgets. These numbers show that *Internet of Things* (IoT) devices will be everywhere in society. Managing this amount of devices will not be an easy task. Other than the huge amount of pieces, they are also heavily constrained which makes everything more complicated. Although not every device has to be monitored, there will be key gadgets that cannot fail. Consider a train scheduling and signaling system that relies heavily on a *Wireless Sensor Network* (WSN) due to the low operational cost over a conventional wired network. However, the trade-off is the additional network administrators that have to be employed to monitor the behavior and correct operation of such a critical network. A simple fail in this network can cause anything from a simple delay to a severe train crash.

On [AlS+17] it is shown that the important *Key Performance Indicators* (KPIs) depend on which stakeholder will receive this information. Two key metrics described are the Energy and Power Metrics. As described this metric can help mitigate the environmental impact, boost the working lifetime of the device and, most importantly, reduce the costs. Additionally, the *Quality of Service* (QoS) is another very important metric, in the case of WSNs, the latency is a QoS key performance indicator since many WSN applications depend on the delay. This means that there is a maximum delay for a piece of information to be delivered from the sensor to the sink. Moreover, there are the reliability and resilience metrics that deliverers key information like Network Utilization, which can show how busy the network is and how close it is to be in its maximum capacity.

The current monitoring protocols were designed without taking the constraints from WSN nodes into account. One does not want to use many resources to monitor constrained devices. While the current protocols do work with these devices they have an unnecessary overhead in those. A new protocol designed for embedded devices is proposed in this paper which is evaluated against the state of the art.

1.2 Goals

The goal of this project is to analyze and evaluate the current solutions for network monitoring in the WSN context. The base hypothesis in this work is that the current technologies use too many resources and they are too heavy for constrained devices. Such an assumptions stems from the fact that the state-of-art protocol for network monitoring was developed in the 1980s. Therefore, a new protocol would be necessary to fulfill this task. This protocol has to perform the same actions as the current state-of-art protocol. It must additionally perform much better to be worth its implementation, usage and integration. The goal is to have a protocol that is capable of saving as much as possible of the microcontroller's resources. Since monitoring is not seen as a must by most, it is very important to reduce the overhead when introducing such functionality.

The evaluation of application protocols will be done in four steps:

1. Code footprint.
2. Application layer overhead.
3. Over-the-air bytes.
4. Energy usage.

Assuming that the new protocol performance is much better than the current state of the art. A cross-protocol proxying to seamlessly integrate both protocols is necessary to remove the integration overhead from the equation, leaving only the implementation and usage.

1.3 Structure

Chapter 2 introduces all the background knowledge necessary to understand what will be discusesed in this project. It starts by explaining what is an embedded device, Section 2.1. Additionally the wireless network concepts are explained, Section 2.2. It explains what is a WSN, Section 2.2.1, followed by the IEEE 802.15.4, Section 2.2.2 which covers the Physical Layer, Section 2.2.2.1, and the *Medium Access Control* (MAC) Layer, Section 2.2.2.2. Furthermore, the *IPv6 over Low-Power Wireless Personal Area Networks* (6LoWPAN) is defined, Section 2.2.3 and the Routing Protocol used is described, Section 2.2.4. It furthermore introduces some basic monitoring protocols and techniques, Section 2.3, *Simple Network Management Protocol* (SNMP), Section 2.3.1, *Command Line Interface* (CLI), Section 2.3.2, *Network Configuration Protocol* (NETCONF), Section 2.3.3 and *Constrained Application Protocol* (CoAP), Section 2.3.4. Lastly, it presents some related works, Section 2.4, where the issue

of monitoring WSN was addressed by explaining what was done and showing what could be done better.

Chapter 3 describes the current state of art when it comes to monitoring protocols and techniques, Sections 3.1 and 3.2. It describes the packet structure, Sections 3.1.1 and 3.2.1, the encoding, Sections 3.1.2 and 3.2.2, the messaging pattern, Sections 3.1.3 and 3.2.3, the resource identification, Sections 3.1.4 and 3.2.4, the monitoring values, Sections 3.1.5 and 3.2.5, and the operations, Sections 3.1.6 and 3.2.6.

Chapter 4 presents a new monitoring protocol that is designed to work better in the WSN context. It likewise presents the packet structure, Section 4.1, the encoding, Section 4.2, the messaging pattern, Section 4.3, the resource identification, Section 4.4, the monitoring values, Section 4.5, and the operations, Section 4.6.

Chapter 5 provides some experiments that evaluates and compare all three protocol presented, SNMP, CoAP and *Constrained Monitoring Protocol* (CoMP). The goal is to define which protocol is best in terms of resource usage. It starts by presenting the experiments setup, Section 5.1. Afterward, the results are shown, Section 5.2. First the SNMP results, Section 5.2.1, second the CoAP, Section 5.2.2, third the CoMP, Section 5.2.3, and lastly the comparison between all three protocols, Section 5.2.4, with an evaluation of which is the best protocol in each scenario.

Chapter 6 suggests a Cross-Protocol Proxy between the newly proposed CoMP and the already consolidated SNMP. It starts with the motivation behind this proposal, Section 6.1. A design is presented, Section 6.2, with the implementation details, Section 6.2.1. The issue regarding the types mapping is addressed, Section 6.2.2. A caching solution to reduce the network overhead in the WSN is conferred, Section 6.2.3.

CHAPTER 2

Background

In this chapter, some key concepts regarding embedded devices are introduced. These concepts are necessary to understand all its main characteristics and limitations. Furthermore, the way they communicate with each other is explained in detail. This is also necessary to perceive how different these networks are from the common ones. Along with, the state-of-the-art protocols designed for network monitoring or that can be used for this purpose. Lastly, some works regarding network monitoring in the WSN context are presented.

2.1 Embedded Devices

Embedded devices usually fall into the category of constrained devices since they frequently have some limitations, mostly because of physical conditions such as weight, size, and power/energy along with cost constraints. Furthermore, these limitations heavily affect the space available for code, *Read Only Memory* (ROM), and space available for variables and functions during code execution, *Random Access Memory* (RAM).

In [BEK14] these devices were classified by constrains. Firstly they were categorized in *Classes* when it comes to RAM and ROM sizes. Class 0 (C0) consists of gadgets with << 10.24 kB of RAM and << 102.4 kB of ROM. Furthermore Class 1 (C1) dwell those with ~10.24 kB of RAM and ~102.4 kB of ROM. As well as in Class 2 subsist those with ~51.2 kB of RAM and ~256 kB of ROM. These gadgets are additionally classified by energy limitation. Class E0 covers devices where its energy is limited by events. Furthermore, Class E1 comprise those with a time limited energy source that can be recharged or replaced. Whereas Class E2 encompass those with a lifetime energy source therefore when the energy source is drained out it cannot be reused. Finally, Class E9 where its members have no limitation regarding available energy. These classes are summarized on Tables 2.1 and 2.2.

As described in [BCS12], Class 0 devices impose limitations that make them unlikely to be part of the IoT and they'll form harmonious exchange with other devices with fewer limitations to participate in communications. Correspondingly those in Class 1

Class	RAM	ROM
C0	<< 10.24 kB	<< 102.4 kB
C1	~10.24 kB	~102.4 kB
C2	~51.2 kB	~256 kB

Table 2.1: ROM and RAM Classes.

Class	Limitation
E0	Events
E1	Periodic
E2	Lifetime
E9	No Limitation

Table 2.2: Energy Classes.

have enough resources to engage in purposeful conversations even though those cannot be done using *Transport Layer Security* (TLS), *Extensible Markup Language* (XML) or *HyperText Transfer Protocol* (HTTP). Additionally, Class 2 gadgets have enough capability to use the same protocols that are used in normal computers, however, it is possible to assume that they are not in Class E9 when it comes to energy limitation. Therefore, they can furthermore benefit from constrained protocols since they will reduce energy and network usage.

In this project the focus will be onto Class 1 devices, Figure 2.1, since they are relatively cheap and can perform most of the tasks of Class 2. Consequently, they are the most cost-efficient gadgets to be used in the IoT environment.

2.2 Network

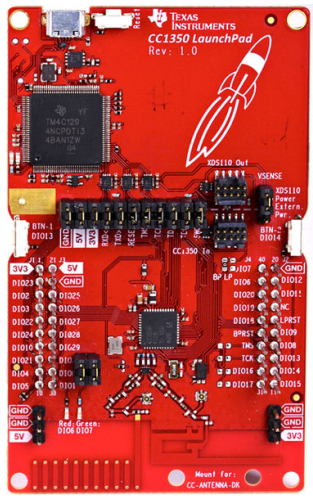


Figure 2.1: Class 1 - Device - SimpleLink™ Dual-Band CC1350 Wireless MCU LaunchPad Development Kit.

2.2.1 *Wireless Sensor Network* (WSN)

According to [DP10], sensors are the channel between the physical and digital world because they are capable of seizing real-world events and converting them into a format that computers can process, store and analyze it. These gadgets can help avoid catastrophic infrastructure failures. As an example, the dam burst that happened in Brazil [Car16] could have been avoided if sensors had been used to constantly evaluate the dam structure. Moreover, an alarm system should have been used in the case of a burst, this could easily have been done with a WSN where the sensors would transmit the information of an accident to a sink node that could trigger an alarm in the cities that were far from the dam but would likewise be affected by this calamity. Additionally, a WSN can help monitor the destruction of natural resources, like the melting of the polar ice cap, boost productivity, reduce traffic jams, etc.

Still according to [DP10], even though these networks have a lot of similarities with ordinary distributed systems, they are subject to several new challenges. The most common set back when it comes down to WSN is the energy constraint. Usually, the nodes in such networks run on a limited, some times very limited, energy source. Commonly these gadgets are battery powered but some are even disposable, meaning that when they ran out of energy they are discarded. Additionally, there is the management challenge, many of these networks need to be deployed in harsh and remote environments making maintenance and repair unrealistic. A common way to deploy these systems is the *Ad Hoc* approach since these devices sometimes need to be in areas that are hard to access. For instance, in case of a disaster, natural or

human, a WSN can be used to monitor these areas. Since the area might not be human-friendly the nodes need to be deployed from an airplane and dropped over the area. Some problems during the landing can occur and the node might become nonoperational, therefore the network needs to be able to fix itself. This shows that these networks must work autonomously being capable of adapting, repairing and configuring themselves. Moreover, there is the wireless networking challenge, attenuation affects the radio signals reducing its intensity as it travels through a medium. For a signal to be transmitted for larger distances it requires more power, however, the sensors in these networks already have the energy restriction. It is more efficient, energy-wise, to break a larger distance into several smaller distances, introducing the need for multi-hop networking, Figure 2.2. Similarly, these systems cannot rely on centralized management due to all already mentioned constraints, but mainly the energy one, it is expected constant topology changes which would introduce a huge overhead for a centralized routing algorithm. Instead, WSN relies on decentralized routing algorithms, which might lead to a non-optimal route but the management overhead is reduced drastically.

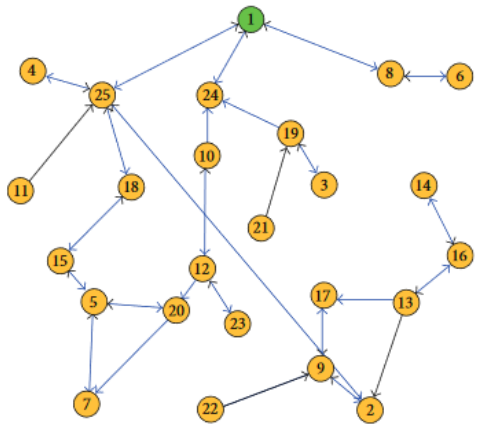


Figure 2.2: Multi-Hop Network [WRV13].

In this project, WSN simulations and real physical networks will be used to evaluate the protocols that can be used to monitor embedded devices in a network and some scenarios where having a good monitoring protocol can help troubleshoot a problem, making possible to take actions to solve the issue easier and more efficient.

2.2.2 IEEE 802.15.4

As stated in [16], the IEEE 802.15.4 focuses on the physical and the MAC layers, which are the first and second layer in the OSI model. The other layers are left

open, there are several implementations of these such as: Zigbee[1], Thread[2], Z-Wave[3], 6LoWPAN, etc.

2.2.2.1 Physical Layer

As stated in [16], the IEEE 802.15.4 defines the frequencies, number of channels and data rate that can be used at the physical layer. The 868 MHz frequency supports data rates of 20 kbit/s, 100 kbit/s and 250 kbit/s and it has one channel. Furthermore, 915 MHz frequency supports data rates of 40 kbit/s and 250 kbit/s and has 10 channels. Additionally, the 2.4 GHz frequency supports data rate of 250 kbit/s, it has 16 channels. Along with, the packet structure for this layer is also defined, Figure 2.3.

Many other physical specifications are defined in the IEEE 802.15.4. However, the physical layer is not the focus of this project since the protocols that will be studied are in the application layer. Therefore, this layer will not be analyzed in depth.

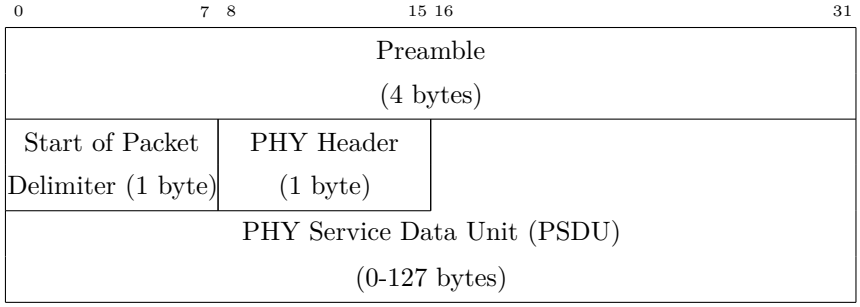


Figure 2.3: IEEE 802.15.4 Physical Packet Structure.

2.2.2.2 *Medium Access Control* (MAC) Layer

As reported in [16], the IEEE 802.15.4 MAC has a very low cost, extremely low power consumption, simple structure and reliable data transfer. It can use 64-bit IEEE addresses when the big network size is needed or a 16-bit addressing can be used in small networks or local. This is important because it can reduce the bytes transmitted significantly. It additionally allows AES-128 security if needed. Addtionally, it supports contention-based protocols, like *Carrier Sense Multiple Access* (CSMA), and scheduled-based protocols, like *Time Slotted Channel Hopping* (TSCH).

The basic MAC packet structure, Figure 2.4, uses 25 bytes. In case security is enabled an extra 21 bytes are used, in the worst-case scenario.

[1] `https://www.zigbee.org/`
[2] `https://www.threadgroup.org/`
[3] `https://www.z-wave.com/`

In this paper, during the simulations and test scenarios the TSCH MAC protocol will be used. According to [WPG15], TSCH does not modify the physical layer, therefore, it works in any hardware that is compliant with the IEEE 802.15.4. This protocol offers high reliability and can work well in time-critical environments, like critical industries. The time-slotted approach can reduce significantly the number of collisions.

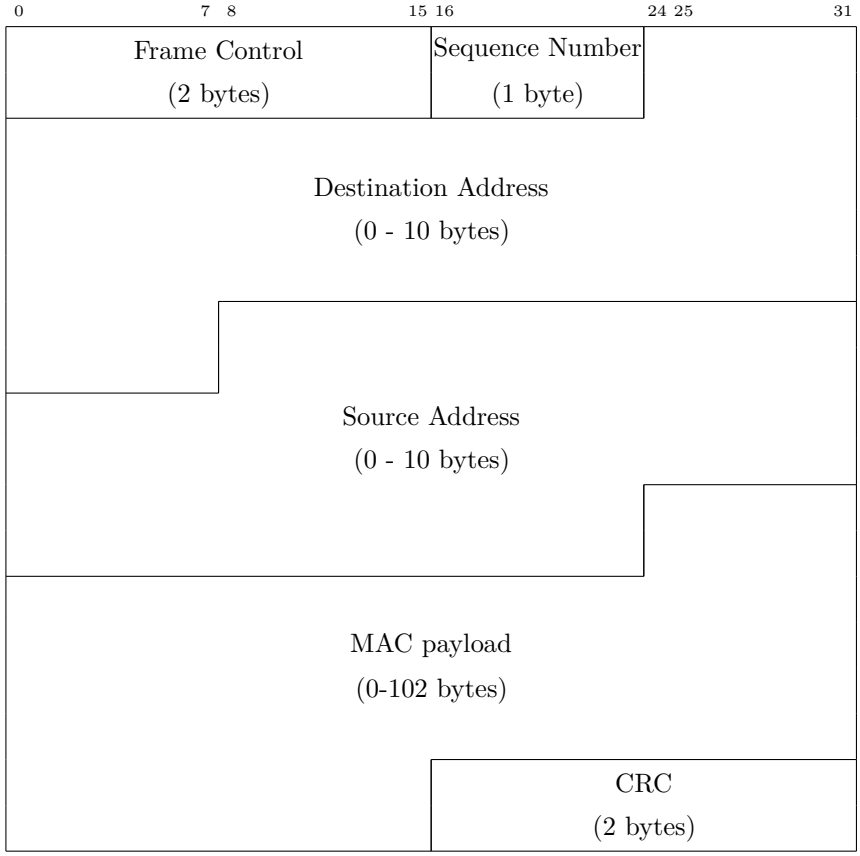


Figure 2.4: IEEE 802.15.4 MAC Packet Structure.

2.2.3 *IPv6 over Low-Power Wireless Personal Area Networks* (6LoWPAN)

On [KMS07] an overview of what needs to be done to be able to transmit IP over IEEE 802.15.4 networks is introduced. There are several characteristics of these networks that must be taken into consideration when designing a solution to this problem. The first challenge is the small packet size, as mentioned in Section 2.2.2.1, the maxi-

mum physical service data unit size is 127 bytes. Furthermore as mentioned in Section 2.2.2.2 the maximum frame size left for data on the MAC layer is 102 bytes on the best case, without security, and on the worst case, with AES-128 security, only 81 bytes are left for data. Another problem is the MAC addressing, as mentioned in Section 2.2.2.2 it supports 64-bit IEEE addresses or 16-bit addresses. Thirdly, as mentioned in Section 2.2.2.1, these networks have low bandwidth. Lastly, these networks are mainly composed of constrained devices, as mentioned in Section 2.2.1. All these characteristics have to be taken into consideration when solving this problem.

Likewise, on [KMS07] the assumptions that were made when designing this adaptation layer are mentioned. It is assumed that applications will have a small volume of data transmission, but the protocols do not restrain bulk transfers. Also, the use of IP technology provides several benefits. The IP networks are widely spread and this will permit the use of the current infrastructure. These techniques are proven to work and are known. There are several tools and techniques for diagnosing and fixing IP networks.

On [Mon+07] to allow IP packets to be transmitted over IEEE 802.15.4 networks the header encoding and compression methods are introduced. Using the HC1 header compression a 40 bytes header can be compressed into 2 bytes when the two nodes are inside the same network. It can be compacted to 12 bytes when the nodes are in different networks but the prefix of the external network is known. Lastly, in the worst case, it can be compressed to 20 bytes in a scenario similar to the previous one but without knowing the external network prefix. In Figure 2.5 these compression scenarios are shown with examples.

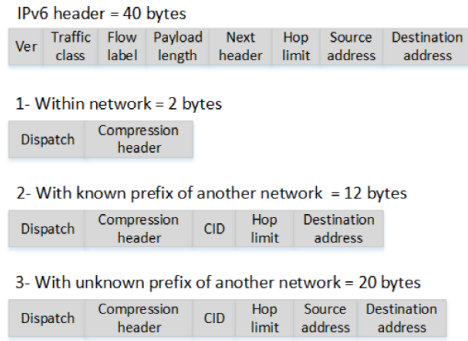


Figure 2.5: 6LoWPAN IPv6 Header Compression [Ayo+19].

Furthermore as mentioned on [Mon+07], the 6LoWPAN is an adaptation layer between the MAC layer, and the IP layer, as can be seen in Figure 2.6.

In this project, the 6LoWPAN will be used because it supports the current Application

Figure 2.6: 6LoWPAN Stack.

Layers Protocols that are based on the IP Layer. To make a fair comparison, the newly proposed protocol will again be an Application Layer Protocol running on top of the IP Layer.

2.2.4 Routing Protocol

Due to all the constraints present in the devices that are part of a WSN, a new Routing Protocol was proposed in [Win+12]. This protocol tries to provide an energy-efficient and a reliable routing mechanism. The *Routing Protocol for Low-Power and Lossy Networks* (RPL) builds a *Destination Oriented Directed Acyclic Graph* (DODAG) using an objective function that can use different metrics which is determined by the application needs.

According to [GK12], these DODAGs have as roots the most used sinks or the gateway. A RPL instance, identified by a unique identifier, is composed of one or multiple DODAGs. Each instance is logically independent of each other but together they can be on the same network. A device can be in several instances but it can only be in one DODAG in each. In Figure 2.7, it is possible to see how a RPL network looks like.

2.3 Monitoring Protocols

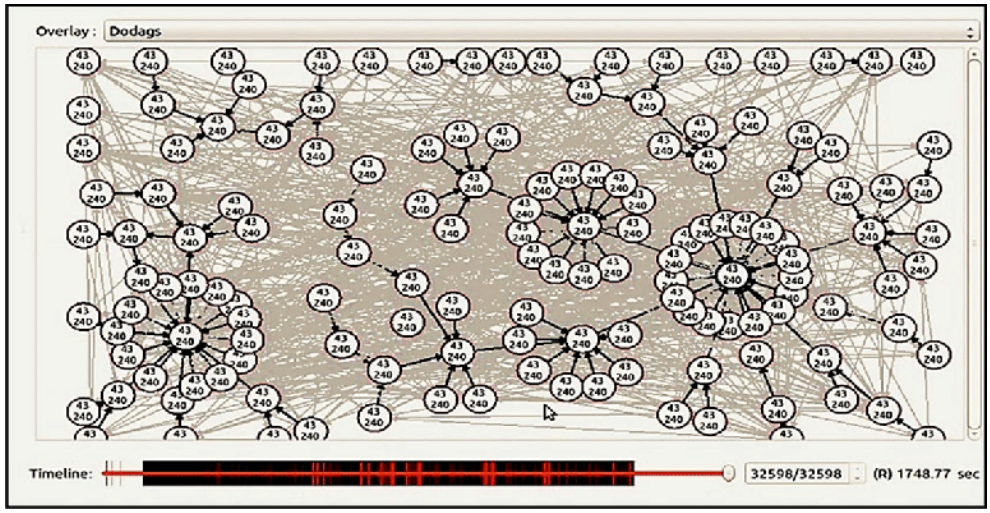


Figure 2.7: RPL DODAG Simulation [AHK19].

2.3.1 *Simple Network Management Protocol* (SNMP)

SNMP dates back to the 1980s, it was proposed to be a very lightweight management protocol. Back at that time, computers were limited in capability which is why in [Sat90], it states that adding network management to managed nodes must have minimal effect on these. Even though the devices were likewise more constrained back in the day, the nodes that compose a WSN are way more limited.

On [KS11] an experiment was performed to see the resources requirements of running a SNMP agent on these devices. This analysis was done on the Contiki-OS[DGV04]. On Tables 2.3 and 2.4 the results of this research can be seen. However, they used dynamic memory allocation which is not the best approach for constrained devices[Das+15]. Because this allocation is known to have high overhead in embedded systems, it is highly recommended being avoided.

In this project a new implementation of SNMP will be done since Contiki-OS is no longer under development the new version will be used, Contiki-NG[4]. For this reason, a fresh implementation is necessary since a fair comparison starts by using the same operating system and libraries.

[4] `http://contiki-ng.org/`

Module	ROM	RAM (static)
snmpd.c	172	2
dispatch.c	1076	26
msg-proc-v1.c	634	6
msg-proc-v3.c	1184	30
cmd-responder.c	302	0
mib.c	1996	6
ber.c	4264	3
usm.c	1160	122
aes_cfb.c	9752	40
md5.c	10264	0
utils.c	416	0

Table 2.3: ROM and RAM Usage in Bytes (Contiki-OS) [KS11].

Version	Security level	Max. stack size
v1	-	688
v3	noAuthNoPriv	708
v3	authNoPriv	1140
v3	authPriv	1144

Table 2.4: Maximum Runtime RAM Usage in Bytes (Contiki-OS) [KS11].

2.3.2 *Command Line Interface* (CLI)

It is very normal for operating system to have local diagnostic binaries like *vmstat*, *lsblk*, *free* and the */proc* files, like the */proc/meminfo* file. As a result, another way to monitor a device performance or status is by using the CLI. This can easily be done by using some remote access protocol, like Telnet[PR83] or *Secure Shell* (SSH)[YL06].

This idea is very simple and generic, but this can introduce several security issues. Since the process that is collecting the information has to remotely connect to the device to get the information very good access rights and user permissions must be in place. If a misconfiguration happens a collector with a bug can crash the device from inside. Another risk is in case the login credentials are leaked, an evil person can remotely access the machine and gain complete root access by exploring some exploit that might be available.

In the WSN scenario, not every device has the shell enabled once it is deployed in production. This is done because production devices are heavily optimized for one task only and the shell is mainly used for debugging purposes on these. As a result, by removing the shell process the operating system footprint can be heavily reduced. Constrained devices furthermore rarely have a remote access process in place for the

```
rosario@dtu-5cg7284zs7v:~$ telnet 10.0.2.15
Trying 10.0.2.15...
Connected to 10.0.2.15.
Escape character is '^]'.
Ubuntu 18.04.2 LTS
dtu-5cg7284zs7v login: monitor
Password:
Last login: Tue Jul 16 14:28:13 CEST 2019 from dtu-5cg7284zs7v.win.dtu.dk on pts/0
Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-54-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

$ free -m
              total        used        free      shared  buff/cache   available
Mem:           2012          36        1434           0         540        1756
Swap:          1897           0        1897
$ lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda      8:0    0   40G  0 disk
└─sda1   8:1    0   40G  0 part /
sr0     11:0    1 1024M  0 rom
$ exit
Connection closed by foreign host.
rosario@dtu-5cg7284zs7v:~$ _
```

Figure 2.8: Telnet Monitoring Example.

same reason, code footprint. Additionally, since they usually do not have a shell process, a remote access process would be useless.

As shown in [TTC04], the Telnet approach works well for embedded systems. However, this prerogative is only valid when it comes down to cabled embedded devices and in the paper's case, devices above class-2 were used. When it comes down to class-1 gadgets, the telnet overhead is significant. Starting for the fact that telnet is *Transmission Control Protocol* (TCP) based, therefore it requires an open connection. This is already a huge overhead for constrained devices.

In this project, a CLI approach will not be taken into consideration. Even though the implementation of telnet would be fairly simple, the TCP overhead is already enough to remove this approach from the possibilities. Only *User Datagram Protocol* (UDP) protocols will be used, considering that they are simpler, connectionless and that there is a 6LoWPAN Header Compression designed for UDP.

2.3.3 *Network Configuration Protocol* (NETCONF)

According to [VM19], the main functionality desired for the SNMP while it was being designed was to able to remotely configure networks devices. However, it failed to do so and it is widely used only to monitor network devices. In 2002 the *Internet Engineering Task Force* (IETF), *Internet Architecture Board* (IAB) and the main network operators held a meeting in which the goal was to understand the current demand for network management. As per the meeting, it was clear that most were using the CLI for monitoring, the main reason for this is that this approach is human-readable and very straight forward.

Following this demand, the NETCONF was defined on [Enn+11]. It aims to solve all the management issues faced by the SNMP. The main advantage is that this protocol is human readable since it is encoded with XML. Even though it became easier for humans to understand and use it, this also introduces overhead for constrained devices. Since XML is text-based it requires a big amount of bandwidth for transmission additionally this means that the RAM usage on the device will additionally be big because it needs to build the message to send it. In Figure 2.9, a get operation done with NETCONF is shown.

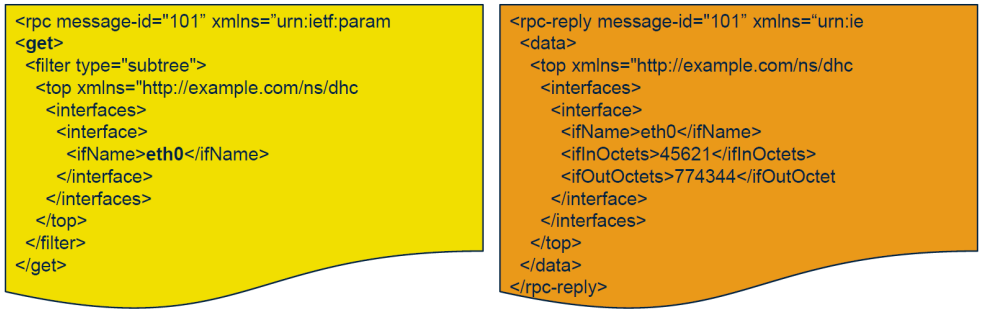


Figure 2.9: Example of NETCONF Get Operation [Tea16].

Another overhead from the NETCONF is the fact that it runs over SSH which uses TCP as transport. Protocols for Embedded Devices should use UDP as much as possible, due to the resource limitations in them.

In this project NETCONF likewise will not be evaluated, for the same reasons as the CLI. It is a heavy protocol running over TCP. It is even heavier because it runs on SSH which is an encrypted protocol. Even though security is very important, embedded devices are bound to weak security due to resource limitations.

2.3.4 *Constrained Application Protocol* (CoAP)

On [SHB14] the CoAP was proposed. Its goal is to replace the HTTP on constrained devices. The model proposed for CoAP is very similar to the HTTP model. It too uses *Uniform Resource Identifier* (URI), Content-Formats, Method Codes, Response Codes. The first difference is that CoAP uses UDP instead of TCP. To overcome the fact that UDP is connectionless, CoAP introduces the Reliable Message Transmission and the Unreliable Message Transmission. A Reliable Message waits for an Acknowledgment reply for the Server, while the Unreliable just sends the message. In Figure 2.10 a Reliable Message Transmission example is shown.

As shown in [Kod+19], CoAP can be used for many practical purposes in the IoT world. In paper a simple home automation implementation was proposed and done. A comparison with the *Message Queuing Telemetry Transport* (MQTT), which is another protocol for constrained devices, was done and made clear that CoAP was a better choice.

On [Maz+13], a monitoring approach using CoAP was proposed. However, no code footprint, ROM and/or RAM, analysis was done, only a power usage analysis was performed in this research. Additionally, it was not compared to any other protocol available for this task. Therefore, this publication showed that it is possible to use CoAP for monitoring IoT devices, but it did not show if this is the best option.

In this project, a similar experiment will be performed. Since Contiki-NG already has the CoAP implemented, the only necessary development will be to implement the CoAP endpoints and performing the performance analysis.

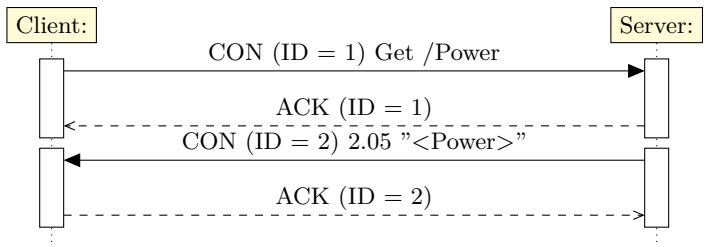


Figure 2.10: Simple Sequence Diagram of a CoAP Get Request.

2.4 Related Works

Since it is clear that there has not been defined a standard way of monitoring resources on embedded systems, several papers have proposed either new protocols or adaptation of the current protocols to have a good way to perform this action.

On [Ham+08] a new protocol for monitoring WSN devices is proposed. The LoWPAN Network Management Protocol (LNMP) targets two problems faced in these networks, Network Discovery and Device Monitoring. Since the deployment is usually done in an *ad hoc* fashion, a normal problem when monitoring devices on WSN is to keep track of which nodes are present on the network. This could be done manually, but in a very dense network, this would be a massive task and very prone to errors. To target this issue, a Network Discovery proposal was done. In this, the end devices should report to a coordinator its status periodically. This is done recursively until it reaches the Gateway. During normal operations only the changes on the state table are reported upwards, reducing the traffic. The gateway likewise has a state table, like any other device in the WSN. However, its table is available in an *Management Information Base* (MIB), this way management systems can be aware of the devices available in the network. To target the resource monitoring problem an adaptation layer is proposed. A conversion between the SNMP *Object Identifier* (OID) to a 6LoWPAN OID is the only modification proposed. Another improvement proposed is a caching in the Gateway where if a value is constant for the entire network the Gateway replies instantly without sending the request to the End Device. The Network Discovery solution is really solid since it targets a real issue in dense networks, however, the only compression proposed in the SNMP was a conversion between OIDs which is good, but not enough. In this work's protocol, the OID system will be kept but compression will be applied to it. However, it will not address the Network Discovery problem because this can be solved using the Physical Topology MIB proposed on [BJ00]. This MIB was designed to be used with the *Link Layer Discovery Protocol* (LLDP) which was proposed on [05]. But this can also be filled in with the RPL Neighbors Table information.

On [CKC09] the Simple Network Management Protocol for 6LoWPAN (6LoWPAN-SNMP) was proposed. The same principle behind the 6LoWPAN was used. The SNMP headers were analyzed and compressed as much as possible. Some fields are way too large, only a small fraction of it is usually used. For example, the header version field which is a 4-byte integer can easily be compressed into 3-bit since he proposes 4 new versions. This limits the number of new versions since all 3 bits are used. Other fields like the Request ID can be limited too, there is no need to use 4 bytes in total for this. The main reason for this identification is to avoid duplicate messages, but it is almost impossible to have 4294967296 messages being handled at the same time, especially in the WSN context. For this reason, it proposed to limit it to 255 which only requires 1 byte. No details on the encoding were given, so it is possible to assume that the same *Basic Encoding Rules* (BER) encoding was used. The OID system was kept and no compression was suggested for it. Lastly, the same proxy technique as before was used in the border gateway to convert the messages from SNMP to 6LoWPAN-SNMP and vice versa. In this work's protocol, a different encoding will be used which targets both key requirements for constrained devices, the code footprint, and the encoded buffer size. And an OID compression technique will be used.

On [Cha+10], the EmNetS Network Management Protocol (EMP) is proposed. Just like the LNMP, it tackles two problems of the WSN, Network Discovery, and Monitoring. To solve the first, it uses a similar approach. There are Coordinators and End Devices, the coordinators are responsible for keeping track of its end devices. This is done recursively until the Gateway is reached. On the gateway a table with all the devices and the last time its entry was updated are kept, this can be used to create several statistics. To solve the second problem and keep this solution SNMP compliant, a new MIB structure was proposed that will be maintained only for WSN devices. Currently, the MIB scope is huge, meaning that many of those entries are either deprecated, not used or will never be used in the WSN context. This way only the MIBs that are used in this context will be considered, reducing the depth of the tree. In the gateway, a conversion table is available where the OIDs can be converted into an equivalent EMP OIDs. Another trick used to reduce the bandwidth is to keep all constant values in a cache on the border gateway. This way the gateway can respond to the request without forwarding this request into the WSN. A good example of which variable is always constant in a system until it is updated is the sysDescr. In this work, the Network Discovery problem will not be tackled, but to solve the Monitoring problem better techniques to reduce the Network usage will be used and the OID will be kept, no new MIBs will be introduced.

These approaches tried to follow the same principle used by the 6LoWPAN when an adaptation layer was proposed. This ensures the interoperability between these protocols. This approach ensures seamless integration of WSN into existing ecosystems. It is fairly simple to use the same monitoring system, like PRTG[5], that currently monitors all networks of a *Internet Service Provider* (ISP) to additionally monitor the WSN that this company has. In this work a new protocol will be proposed, the CoMP, which will be a standalone protocol, like the CoAP. Additionally, an adaptation layer, cross-protocol proxying, will be proposed that will make sure this protocol is also SNMP compliant. Making sure that it can be seamlessly integrated into existing networks.

[5] `https://www.paessler.com/prtg`

CHAPTER 3

Current Technologies

In this chapter, the state-of-art protocols for network monitoring are shown and explained in depth. Its packet structure is described, its encoding is explained, its messaging patterns are characterized, its resource identification technique is elucidated, its monitoring values structure is described and its operations are studied.

3.1 *Simple Network Management Protocol* (SNMP)

One well-known technology for monitoring networked devices is SNMP. This protocol was proposed in 1988 on [Cas+88]. Its objective was to be a lightweight and simple protocol for network management. Due to not being human-friendly and not having enough security in its initial versions, it never really reached its full capacity, however, it is widely used for network monitoring.

3.1.1 Packet Structure

On Figure 3.1 the SNMPv1 and SNMPv2 message format is shown. The first data in the message is the SNMP version, followed by the community string subsequently there is the SNMP *Protocol Data Unit* (PDU). Additionally, the SNMPv3 has a completely different packet structure, Figure 3.2. It also starts with the version field however, it is followed by new fields. The ID, which is a unique identifier for this message, note that this ID is different from the Request ID from the SNMP PDU. The Max Size indicates the requester buffer max size. Flags represent the message security level. A Security Model that contains the model used to generate the message. The Engine fields that are the ID which represents the SNMP entity that is participating in the transaction, the Boots which contains the SNMP entity boots and the Time which has the Engine Time of the SNMP entity. These are followed by the User Name which represents the conceiver of the request. Additionally, there are the security parameters which has the parameters that the model depends, the context engine ID which identifies uniquely a SNMP entity and the context name which is likewise unique for a SNMP entity. In the end it contains the SNMP PDU, like SNMPv1 and SNMPv2.

On Figure 3.3 the SNMP PDU which is the same for SNMPv1, SNMPv2 and SNMPv3 is shown. The first information is the PDU type, which can be Get, Get-Next, Set, Response, Trap and Inform. Afterward, there is the Request ID which is the identifier to relate a request with a response. Additionally, there are two error fields which are only set in the response PDU. The error status contains an error code and the error index indicates in which varbind this error occurred. Lastly, there are the varbinds that associate an object to value, on the Get requests the values are ignored.

On Figure 3.4 the SNMP PDU for Get-Bulk requests can be seen. This type is only available on SNMPv2 and SNMPv3, the structure is almost the same except for the fact that there are no error fields. Instead, there are the Non-Repeaters which defines the number of objects that should be retrieved no more than once and the Max Repetitions which specifies the number of times that other variables should be retrieved.

SNMP Version	Community String	SNMP PDU

Figure 3.1: SNMPv1 & SNMPv2 Packet Structure.

SNMP Version	ID	Max Size	Flags	Security Model
Engine ID	Engine Boots	Engine Time		User Name
Security Parameters	Context Engine ID			Context Name
SNMP PDU				

Figure 3.2: SNMPv3 Packet Structure.

PDU Type	Request ID	Error Status	Error Index
Variable Binding 1	Variable Binding 2	... Variable Binding n	

Figure 3.3: SNMP PDU Structure.

PDU Type	Request ID	Non Repeaters	Max Repetitions
Variable Binding 1	Variable Binding 2	... Variable Binding n	

Figure 3.4: SNMP Get-Bulk PDU Structure.

3.1.2 Encoding

The SNMP only uses a subset of the ASN.1 which was proposed in 1987 on [87]. A defined encoding technique is used to make it easy to implement this protocol in any

language because an object will have the same encoding when it is sent over the wire. In figure 3.5 it is possible to see how an object can be encoded with this technique. The Content can be another object which is recursively encoded. This is exactly how the SNMP packet is encoded.

Identifier (Type)	Length (Length)	Contents (Value)

Figure 3.5: BER Encoding Structure.

3.1.3 Messaging Pattern

The SNMP's main operations work in the Request-reply pattern where some entity requests the server which receives it, processes it accordingly and replies.

SNMP additionally supports the publish-subscribe pattern, however, this is only possible in the TRAP. This type of request is mainly used when events occur on the network, such as a DSP card going up or down. No setup in which this was used to send scheduled monitoring data was found during this research.

3.1.4 Resource Identification

SNMP uses the OID, proposed in 1994 on [94], to identify the resources available. It is an array of numbers which are allocated hierarchically. For example, only the authority for the 1.2.3 can determine what 1.2.3.4 is.

This identification method requires a centralized entity that is responsible for allocating the resources since they are supposed to be globally unique and not only locally. This furthermore implies a huge overhead. For example, to request the system description, or sysDescr, from a device, the 1.3.6.1.2.1.1.1 OID has to be requested. But if the server only knows the sysDescr variable there is a tree of depth 8 with one node at each level.

3.1.5 Monitoring Values

The SNMP uses the MIB, proposed in 1988 on [MR88], as a database for its entities. This hierarchical organization is great for huge environments like the internet, as it can be seen in Figure 3.6. A network usually contains multiple heterogeneous resources, for example in an office's network, not all routers will be from the same brand. Using a MIB to find which resources a specific brand has made available for the system administrator is easier. Another plus is the fact that all devices from the

same brand, technically, replies to the same MIBs. Networks are in constant change, different devices are joining and leaving a network all the time.

3.1.6 Operations

SNMP supports several operations: Get, Get-Next, Get-Bulk, and Set. The Set is irrelevant for monitoring, it can be used to change the properties of a device. Therefore it is only relevant for managing. In the Get request, the client sends a single OID, or a list of OIDs, with the value field empty. The server only gets this request, sets the values accordingly and replies. Additionally in the Get-Next operation, the client sends an OID and the server replies with the OID that is after the requested OID in the tree. The entire MIB of the server can be walked interactively if the first Get-Next OID is the 1 and for each response, the next of that is requested again, this is known as SNMP Walk. Lastly, the Get-Bulk was introduced in the SNMPv2 which is a better version of the Get-Next. Instead of only returning the next OID it returns the next X OIDs. This amount of variables is defined either by the server or the client. The one that supports the fewer wins. In the same way that it is possible to traverse an entire MIB with the Get-Next, it is also possible with the Get-Bulk, with the advantage that fewer packets are exchanged since multiple variables are sent at once.

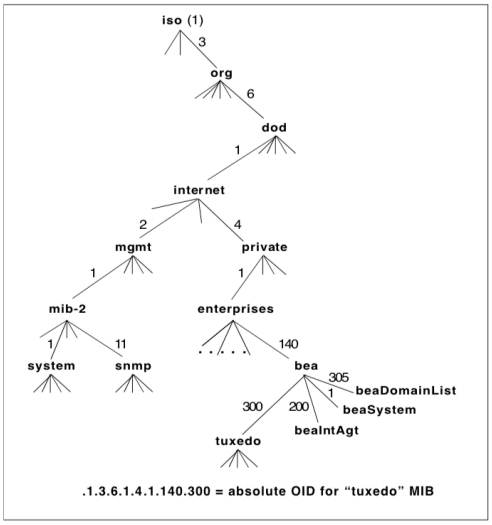


Figure 3.6: MIB Tree Example [BW11].

3.2 *Constrained Application Protocol* (CoAP)

CoAP proposed in 2014 on [SHB14] is a lightweight alternative to HTTP. The HTTP protocol is super human-friendly, meaning that it is mainly composed of strings, for example, the DELETE operation is represented by the DELETE string in the packet while in the CoAP it is represented by the code 4. Reducing the operation size on the packet from 6 bytes to 1 byte, this represents a reduction of 83%.

One generic solution for monitoring devices is to have a HTTP interface that responds with some KPI. On top of that, if a standardized data-interchange format is used, like *JavaScript Object Notation* (JSON), it makes the solution even more generic. In the WSN context, however, the HTTP is too heavy but it can easily be replaced by the CoAP.

In this project's approach, an interface that can easily be used by any service was implemented and is described in this Section.

3.2.1 Packet Structure

The CoAP packet, Figure 3.7, consists of a version field that is a 2 bit unsigned integer set to one by default. Followed by the Transaction Type field a 2-bit unsigned integer containing 0 if the message is confirmable, 1 if the message is non-confirmable, 2 if it is an acknowledgment or 3 in case of reset. Additionally, there is the Token Length field that is a 4-bit unsigned integer which indicates the length of the Token. Followed by the Code that is an 8-bit field which either contains the operation codes, 1 for GET, 2 for POST, 3 for PUT or 4 for DELETE, or it contains the response code. Additionally, it has the Message ID that is a 16-bit unsigned integer. This is used to match responses to requests and avoid message duplication. These fields are the minimal fields for a CoAP message. If a Token Length is given the first field after the ID is the token. After the token, there are the options, if any. To indicate the beginning of the payload and the end of the options there is an 8-bit marker with all bits set to one.

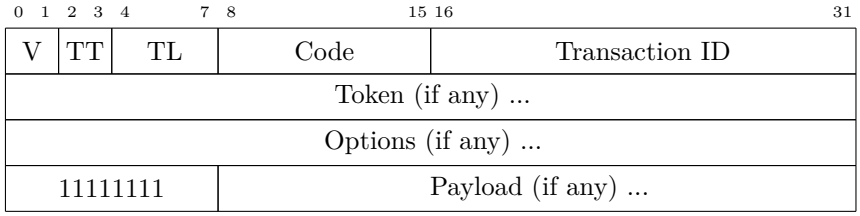


Figure 3.7: CoAP Packet Structure.

3.2.2 Encoding

The minimal header is encoded in a binary manner, therefore the binary value sent over the wire has to be interpreted as described in the protocol. This could create an endianness issue between architectures, but in the specification, it clearly states that: ”All multi-byte integers in this protocol are interpreted in network byte order”[SHB14]. The token, that could likewise be called request ID, is just an ID with a length determined in the header. It furthermore uses the multi-byte encoding. The options, Figure 3.8, are encoded in a different way. The first item is the Delta which indicates which option this is, all the values are defined in [SHB14]. This is followed by the length of the options in bytes. If either the delta or length does not fit in the minimal option, there are extensions for these fields which are optional. In the end, there is the Value of the option.

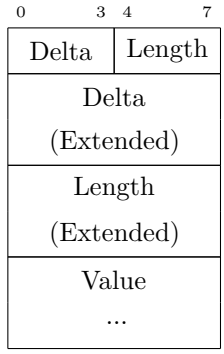


Figure 3.8: CoAP Option Format.

3.2.3 Messaging Pattern

Just like the SNMP, CoAP likewise supports the Request-reply pattern. However, as mentioned in Section 2.3.4, it has two models of Request-reply. One that is reliable, where after each message transmitted the sender waits for the receiver to send an acknowledgment, Figure 3.9. The second model is the unreliable one, where the sender sends a message and continues with its next operation, 3.10. The first model in case of no acknowledgment it will by default retransmit the message until it gets a response or it exhausts the number of retries. However the second one, if the client is making a GET request, for example, it can ask the server again if no response is received but if it just sends a message that does not need a response, like a POST, it will continue, hoping that the operation was successful. Additionally, CoAP has an extension called *Observe*. It works as the name indicates, the client observes the CoAP resource state. Therefore the server works as a notifier where it periodically checks its state and if it has changed, the clients can be notified. The same message ID that is sent from the client to the server when registering, is the same ID used

by the server to respond to the client. This message pattern works better for sensors reading or for events.

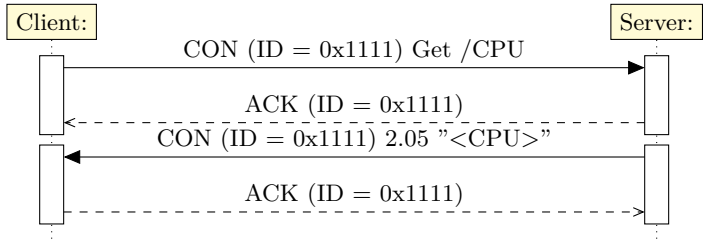


Figure 3.9: CoAP Reliable Transmission.

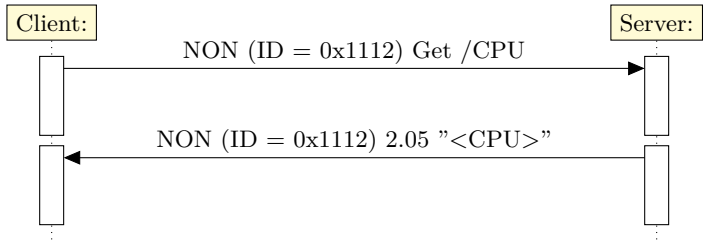


Figure 3.10: CoAP Unreliable Transmission.

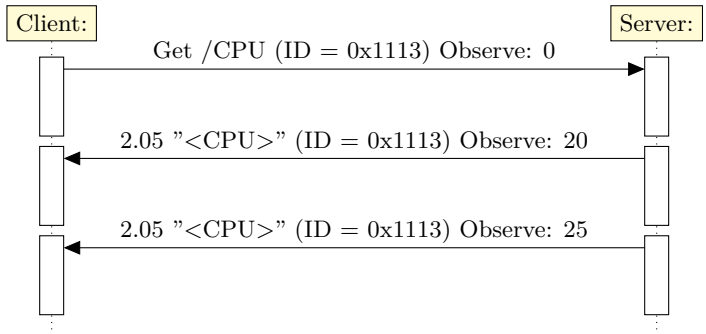


Figure 3.11: CoAP Observe Transmission.

3.2.4 Resource Identification

CoAP uses the URI, defined in 2005 on [BFM05], to identifies its resources. The key idea behind this idea is to have a generic syntax where first there is the scheme, *coap://*, followed by the authority, *fd00::202:2:2:2:5683*. It also has the path, *SNMPv2-MIB/system*, query, *info=sysDescr*, and the fragment, *#Article*.

This resource identification technique is mainly used in the HTTP context, hence why CoAP too uses it, because it is extremely human-friendly. However, this means strings, since this is the only thing humans can memorize easily, hence why no one memorizes IP address instead of the domain names [Moc83] are more used for the authority. Behind the hood, the domain name is translated into the IP address, but the path, query, and fragment are still sent as a text to the server.

3.2.5 Monitoring Values

To use the CoAP as a monitoring protocol the approach was to have the path to have the same SNMP MIB and to use a query named *info* with the value of the KPI, that was the same name used in the SNMP. If no query is sent to the endpoint a list with all the available KPIs are sent as a response, this method can be used to act similar to the SNMP Walk. This approach was done in the conventional Request-reply manner.

3.2.6 Operations

CoAP implements the same operations as HTTP, this means: GET, POST, PUT, DELETE and others. However, for monitoring purposes, the only operation that makes sense is GET. Which, as the name says, is used to retrieve data from the server.

CHAPTER 4

Constrained Monitoring Protocol (CoMP)

In this chapter, the CoMP is proposed. Its goals are to be a very lightweight monitoring protocol to be used in the WSN context. Additionally, it has the objective to be SNMP compliant, therefore it should be possible to convert SNMP requests into CoMP requests. However, in this paper, the focus area is the monitoring requests from SNMP, Get, Get-Next, Get-Bulk, and Response. The Set, Trap and Inform requests are not covered but the proposed structure should cover these requests out-of-the-box.

Moreover, to make CoMP fully SNMP compliant, it has to have the same versions as SNMP. Therefore, CoMP will have v1 and v2 described in the following sections. The version v3 is not covered, but the design supports its implementation.

4.1 Packet Structure

The CoMP packet is defined by a header, an extra header, and a PDU. The header is shared by CoMPv1 and v2, Figure 4.1. It is pretty simple, the first information in it is the protocol version which is an unsigned 4-bit integer and it is followed by the operation which is too an unsigned 4-bit integer.

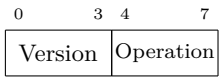


Figure 4.1: CoMPv1 and v2 Header Structure.

If the operation is a Get or Get-Next the extra header will look like Figure 4.2, which will first have a Request ID, that is an unsigned 8-bit integer, followed by the

Error Code, that is an unsigned 5-bit integer, and lastly the Error Index, which is an unsigned 3-bit integer.

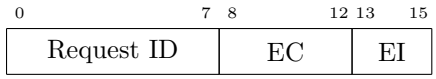


Figure 4.2: CoMPv1 and v2, Get and Get-Next Extra Header.

However, if the version is 2c and the operation is the Get-Bulk a different extra header is available, Figure 4.3. The first element is likewise a Request ID, which is an unsigned 8-bit integer, but it is followed by the Non-Repeaters, which is an unsigned 4-bit integer, and finally the Max Repeaters, which is too an unsigned 4-bit integer.

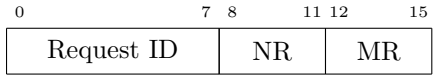


Figure 4.3: CoMPv2, Get-Bulk Extra Header.

Depending on the operation the CoMP PDU will carry different information. In any request, Get, Get-Next or Get-Bulk, the PDU will look like Figure 4.4. It will be a list of OIDs that the client is requesting the server.

Figure 4.4: CoMP PDU Variables on Request.

If the CoMP operation is a Get, the response will look like Figure 4.5. It is a list of Values, this list is indexed using the same position as the OID in the request. For example, if the client requested the following list, [1.3.6.1.2.1.1.1.0, 1.3.6.1.2.1.1.3.0], the server would reply [”sysDescr”, sysUpTime].

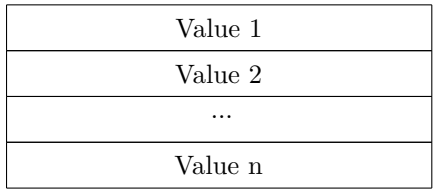


Figure 4.5: CoMP PDU Values on Response.

However, the same method to reduce the message size cannot be used in the Get-Next or Get-Bulk operation, since the server replies with the first OID after the requested one. The next OID is unknown to the client, so the server has to reply with the OID and the corresponding value. For this, a simple array is used where the first element is the OID and the second is the value.

[OID 1, Value 1]
[OID 2, Value 2]
...
[OID n, Value n]

Figure 4.6: CoMP PDU Variables/Value on Response.

4.2 Encoding

Following the CoAP's encoding proposal. All the CoMP headers are binary encoded and using the network byte order, however in the current proposal there is no 16-bit or higher integer, therefore, the byte order is irrelevant. All the PDU variables, values or the pair variable and value, were encoded using the *Concise Binary Object Representation* (CBOR) encoding. Proposed on [BH13] in 2013, it targets constrained devices since one of its goals is to allow a small code footprint, very small messages and to allow extensions without the need for new versions.

The CBOR encoding has a data item header that contains a major type and additional information. For each major type, the additional information contains different information that can be used to process the rest of the data. A good example is the text string major type. It is represented by the integer 3, 011 in binary. The additional information meaning varies, from 0, 00000, to 23, 10111, it is used as the byte count composing a short field encoding, Figure 4.8, where the byte count is used to know the length of the value. If 24, 11000, the next byte is an unsigned 8-bit integer with the byte count, thus composing a long field encoding, Figure 4.9. If 25, 11001, the next 2 bytes is an unsigned 16-bit integer with the byte count, also a long field encoding. If 26, 11010, the next 4 bytes is an unsigned 32-bit integer with the byte count, likewise a long field encoding. If 27, 11011, the next 8 bytes is an unsigned 64-bit integer with the byte count, once again a long field encoding. From 28 to 30 remains unsigned. If 31, 11111, it means that this is an indefinite string, each byte has to be read until a *Break*, 0xFF, 11111111, is encountered, therefore this composes a tiny field encoding, Figure 4.7.

One known issue in the CBOR encoding is the OID encoding. There no specific tag for this type, therefore, it is treated as a normal array of items, which implies that

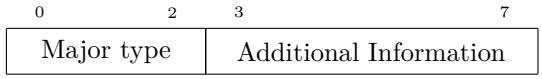


Figure 4.7: CBOR Tiny Field Encoding.

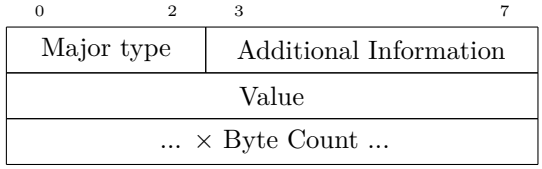


Figure 4.8: CBOR Short Field Encoding.

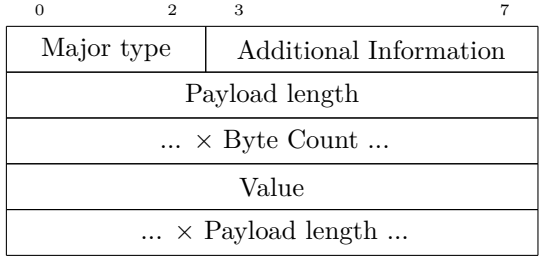


Figure 4.9: CBOR Long Field Encoding.

each item of the array can have a different type. In the OID scenario, all items inside the array have the same type. On [Bor19] a very generic approach for typed arrays is proposed. In this solution, the type of the array is determined by the tag, which implies a smaller overhead in each item. On [BL17] a more specific solution for OID encoding is proposed, which in its core is the same as the one before, but the author focus on the OID problem, however, the solution is to too have a typed array.

The OID compression from [Sch01] was used in all OIDs. It works in a very simple way, but efficient. The first OID in the list is used as the base and all the others are compressed using its common prefix with the base. This way most of the common prefixes are not repeated and little overhead is introduced in the protocol.

The main goal of this encoding technique is to allow future versions of the protocol, if necessary, to use a different encoding in its PDU data. Unlike the SNMP which had its entire application data encoded using the BER encoding, which makes it impossible to disassociate the protocol from the encoding technique. The only way to change a protocol without breaking its legacy is to use a version negotiation, but since in the SNMP, the version is again encoded using the BER the only way to read the version is to know how to decode and encode BER integers, at least. CoMP keeps

its header binary encoded, this format will never change, and leaves only the data encoded. This makes the protocol more flexible and more extendable.

A very concise implementation of the CBOR focused on constrained devices is the tinycbor[1]. It is available on GitHub, is maintained by Intel and is using the MIT license, so it can be used freely in open and closed source projects.

4.3 Messaging Pattern

Just like the SNMP and the CoAP, CoMP still supports the Request–reply message pattern. As described on [HW03] an ideal implementation of this pattern uses the asynchronous transmission to avoid a client being blocked by the server or vice versa, which implies a timeout mechanism. This message exchange should furthermore happen in different channels, to allow two-way messaging.

In the CoMP which uses the UDP transport layer no reliability is offered by it. However, in this messaging pattern, the server will always reply to the client's requests. It is up for the client to handle the unreliability of the UDP. If the client sends a message and it does not reach the server, no response will be sent, so the client will send the request again. If the server's response does not reach the client, the client will too resend the request. So it is a very simple way to implement reliability in such a message pattern.

On [Huh18], a more robust reliability method is proposed for UDP, this would be interesting for other operations, like the TRAP. Where it would be possible for the server to be sure that the client received the event notification. However, this is out of the scope of this project.

4.4 Resource Identification

Since the CoMP is designed to be SNMP compliant, they have to use the same resource identification technique. Since SNMP uses the OID, CoMP has to use the same. Even though this will continue to be an overhead in this protocol, the OID is very good for large scenarios like the network, but this implies a huge depth in the OID size and requires a centralized institution to manages this. Since the centralized institution never became a problem in all these years and the OIDs can be compressed, it is feasible to use this mechanism in the WSN context.

[1] `https://github.com/intel/tinycbor`

4.5 Monitoring Values

The monitoring values are organized the same way as the SNMP, again to be compliant. Therefore the MIB structure is used. Even though MIBs are defined using the ASN.1 specification this does not affect the protocol itself. This is only a standardized way to write MIBs and share them between systems or from a manufacturer to a client.

4.6 Operations

The CoMP's focus is on the monitoring aspect, therefore, it only implements the Get, Get-Next, Get-Bulk and Response operations. These operations are the ones used in the SNMP for monitoring. The Trap, Inform and Set operations are not monitoring focused therefore they are not covered by this CoMP's proposal. However, it is considered, so it would not be a problem to add these functionalities.

CHAPTER 5
Experiments & Results

In this chapter, some experimental scenarios are shown with the reason why this case is worth evaluating. Additionally, some experiments are run in these scenarios and the results are also presented. Subsequently, these outputs are analyzed and the best protocol, for each case, is shown.

5.1 Experiments

In this project, three experiments were conducted to evaluate the previously mentioned protocols in different scenarios and categories.

The first examination targets the code footprint, ROM, and RAM, of a specific protocol. A compilation of a server example for the cc26x0-cc13x0 target was done. The goal is to see how much resources a module uses on a device. Since the devices in a WSN are very constrained, these are key attributes.

The second examination assesses the network usage using the Cooja Simulator[Ost+06]. The Cooja simulations were used to analyze and evaluate many 6LoWPAN protocols like it was done for the RPL in [Acc+11] and for CoAP in [BSP16]. In Figures 5.1, 5.1(a), 5.1(b), 5.1(c) and 5.1(d) the topologies used are shown. For several reasons, it is very important to evaluate the network usage in these protocols. The obvious one is the resource limitation, the least data is transferred the better, but there are other reasons. The physical service data unit is limited to 127 bytes therefore, it is a good idea to have a protocol that fits inside of it, avoiding the expensive fragmentation. Hence the key value that has to be analyzed is the transferred data size. Anything under the application layer is out of the protocol's control because all the protocols used in this project are within this layer but it is furthermore necessary to analyze the bytes over the air because this determines the energy usage. In each scenario, four evaluation cases were studied. Every node, except the border router, was requested for each scenario by a client outside the WSN. The first one, Case 0, is a Walk where all the KPIs available in the server are asked for one by one. In the second one, Case 1, a Bulk Walk is performed where two KPIs are asked for at the same time. Only two were used because this increases the packet size and a balance has to be found

between these two attributes. Additionally in the third case, Case 2, a system description is asked, in this case, it is the operating system name and version tag. Lastly in the fourth case, Case 3, the system uptime is asked. The two last cases are used to analyze how much data does each protocol uses to transmit a string and an integer. To capture the data traffic the TCPDump[1] was used. To read the network dump the wireshark[2] and its terminal version the tshark[3] were used. Since the CoMP is a newly proposed protocol, Wireshark could not dissect it. Therefore a CoMP dissector[4] was written as a Wireshark plugin. All these cases are summarized in Table 5.1.

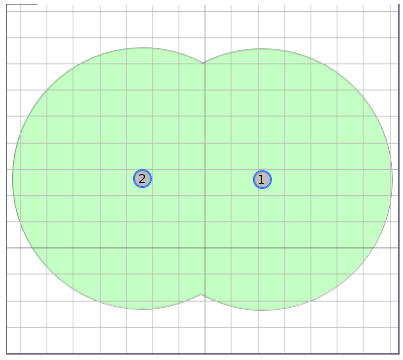
Case	Description
0	A Walk where all the KPIs available in the server are asked for one by one.
1	A Bulk Walk is performed where two KPIs are asked for at the same time.
2	A Get where a system description is asked
3	A Get where a system uptime is asked

Table 5.1: Second and Third Experiments - Cases Description.

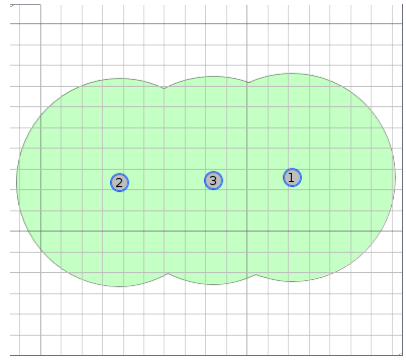
The third experiment evaluates the radio activity time of each protocol. The TSCH MAC protocol was used because it is asynchronous and as described in [Duq+18] it has a good overall performance in terms of energy when compared to other MAC protocols. The scheduler used was the Orchestra presented in [Duq+15] because it is proven to be a good scheduler with high *Packet Delivery Rate* (PDR) and *Packet Reception Rate* (PRR) by keeping the duty cycle to a minimum. The RPL's and TSCH's probing interval was tunned using the Five nines reliability proposed in [DEV17], which reduces the impact of the beacons in the experiments. It was run in the FIT IoT-LAB[Adj+15] located in Grenoble, France. It has 384 M3 open nodes, of which 6 are mobile on robots, and 256 A8 nodes with 32 GPS. In this paper, the M3 nodes were used since they represent the state-of-the-art WSN devices. This testbed was used in several papers to evaluate protocols in real-life scenarios like in [MH17] and in [Sed+18]. To estimate the radio activity time the Energest Module was used which is a software-based online mechanism for energy estimation. It simply counts the number of ticks spent in different states. In this evaluation, only the Radio TX and Radio RX states are considered important. The goal of this experiment is to prove that by reducing the bytes over the air in a protocol will impact positively the energy consumption which is still the biggest constraint in the WSN environment. The topology used in this experiment can be seen in Figure 5.2. In these tests it is interesting to look into the radio activity time during transmission (TX) because the receiving (RX) is controlled by the MAC protocol, therefore the only radio activity time where the protocol can influence positively or negatively is the TX cycle. Additionally, the

[1] `https://www.tcpdump.org/`
[2] `https://www.wireshark.org/`
[3] `https://www.wireshark.org/docs/man-pages/tshark.html`
[4] `https://github.com/Yagoor/wireshark/tree/CoMP`

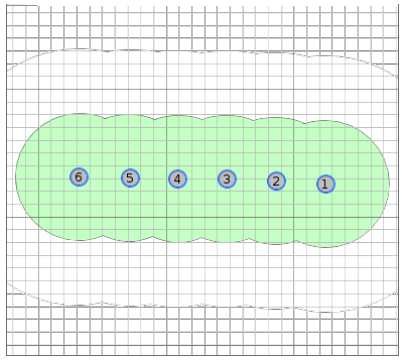
DODAG was built in a way that the border router has a direct route to every server, which in this scenario is good because to analyze the radio activity time of an application layer protocol is necessary to run the same experiment multiple times and get the smallest value generated by it. This is necessary because a synchronization beacon from the MAC protocol can be sent during the execution of the protocol and it will taint the evaluation. Another unwanted behavior is retransmission that can be caused by some external interference like a packet collision. In this experiment the medium is totally out of control and collisions are likely to happen, this would too impact negatively the evaluation. Instead of running the same experiment multiple times to get the smallest value, which is the one with the least probability of external interference, it is possible to get the smallest duty cycle from the four nodes either in transmission and reception. Therefore, in this experiment, every node, except the border router, is requested for each scenario from a client outside the WSN.



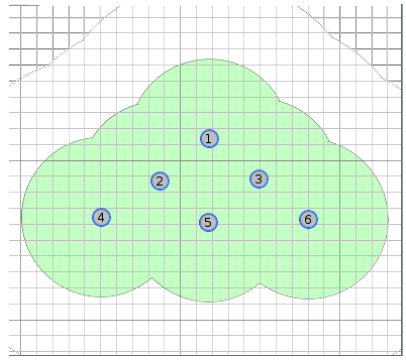
(a) One server (2) and one border router (1) (Scenario 01)



(b) Two servers (2-3) and one border router (1) (Scenario 02)



(c) Five servers (2-6) and one border router (1) in line shape (Scenario 05)



(d) Five servers (2-6) and one border router (1) in tree shape (Scenario Tree)

Figure 5.1: Cooja Simulations.

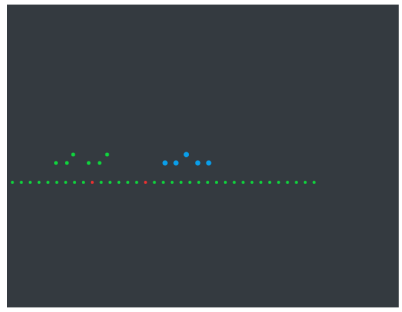


Figure 5.2: FIT IoT-LAB Topology. Four Servers (Blue Dots in Line) and One Border Router (Blue Dot Above Blue Line). The DODAG Was Built With a Directly Link From the Border Router to the Servers.

5.2 Results

5.2.1 *Simple Network Management Protocol* (SNMP)

In code footprint SNMP's performance was really good as shown in Table 5.2 and Figure 5.3. It uses 3140 bytes of ROM and 1120 of RAM. The most RAM expensive object is the snmp.o, this happens because it has the packet buffer, which by default uses 512 bytes. In terms of ROM, the most expensive object is snmp-message.o. Encoding and Decoding BER encoded buffers is not optimal for Constrained Devices, even though only the basic functionalities were implemented the code footprint was affected.

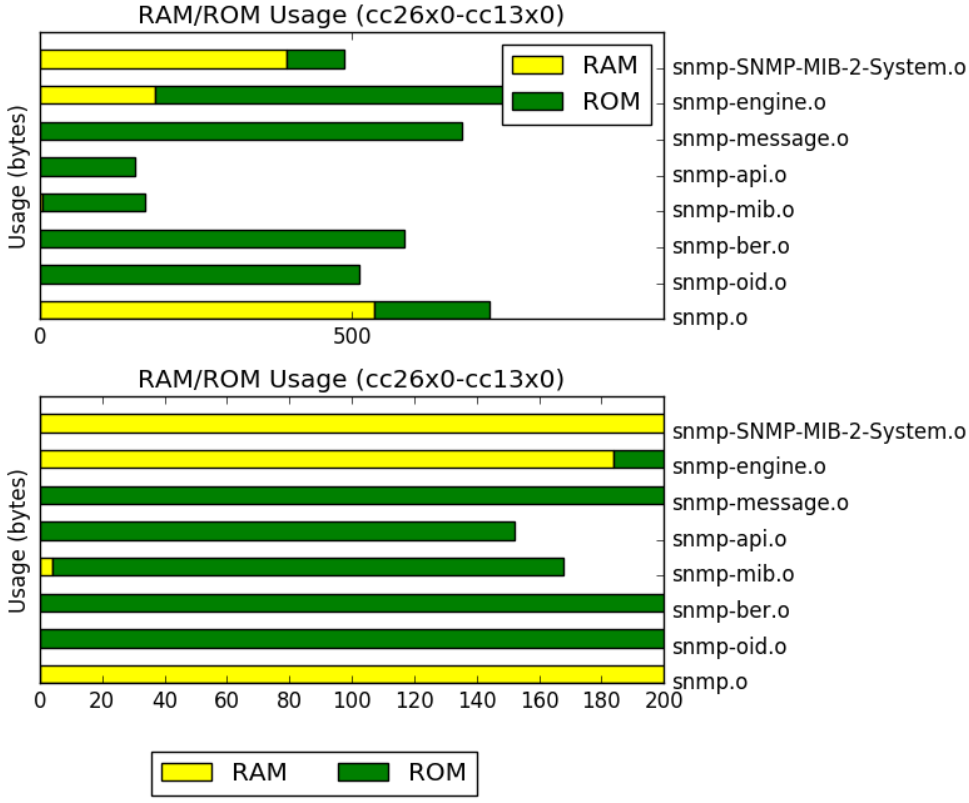


Figure 5.3: SNMP Code Footprint in the cc26x0-cc13x0 Target.

SNMP's request is analyzed in-depth in Figures 5.4. The header and everything before the variable-bindings in the SNMP PDU use 27 bytes, as illustrated in Figure 5.4(b). Each variable-binding has the Object Name and the Value which its minimum size is 7 bytes, where the Object Name is the 0.1 or itu-t.1 and the value is NULL, as indicated in Figure 5.4(d). However, a variable-binding is always encoded as a

File	RAM	ROM	Total
snmp.o	536	184	720
snmp-oid.o	0	512	512
snmp-ber.o	0	584	584
snmp-mib.o	4	164	168
snmp-api.o	0	152	152
snmp-message.o	0	676	676
snmp-engine.o	184	776	960
snmp-SNMP-MIB-2-System.o	396	92	488
Total	1120	3140	4260

Table 5.2: SNMP Code Footprint in the cc26x0-cc13x0 Target.

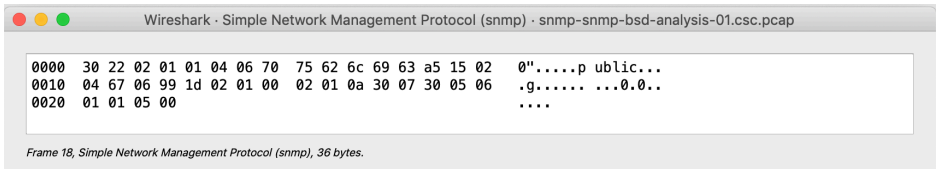
sequence of, therefore, even if only one varbind is present, it is still encoded as a sequence with only one content, as can be seen in Figure 5.4(c). The minimal possible packet for a SNMP request is a GetBulkRequest in the MIB root, itu-t.1, as described in Figures 5.4(a) and 5.4(e).

In Tables 5.3 and 5.4 it is possible to see that SNMP's performance was really good. In the individual requests, Case 2 and 3, it managed to keep the bytes below 127 bytes but only in the Case 3 response, it was kept below this limit. Since the response size is out of the protocols hand because the server can reply with whatever it wants, SNMP's performance was good overall.

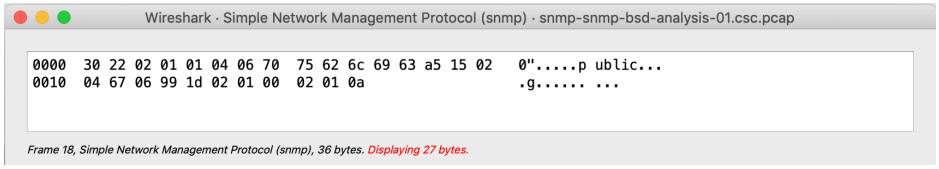
Simulation	RX		TX	
Case	Frames	Bytes	Frames	Bytes
0	8	468	8	337
1	4	352	4	165
2	1	85	1	43
3	1	45	1	43

Table 5.3: SNMP Application Layer Network Usage.

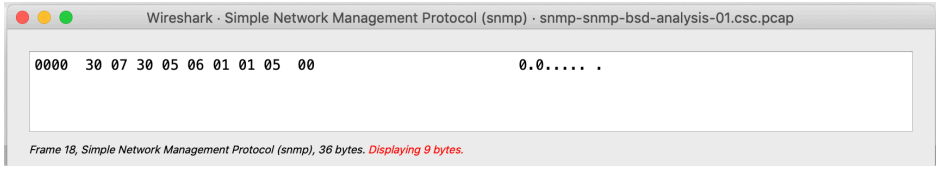
Furthermore, in Figure 5.5 and Table 5.5 the radio activity time of the protocol can be seen for each sink, which is the device being monitored. The more bytes are transmitted the more time it spends in TX. If the edge cases are removed, where most likely retransmissions occurred or beacons were transmitted during the protocol execution, a simple summary can be obtained. In case 0, 41,93 milliseconds are used, in case 1 27,80 milliseconds are used, in case 2 10,19 milliseconds are used and in case 3, 7,69 milliseconds are used.



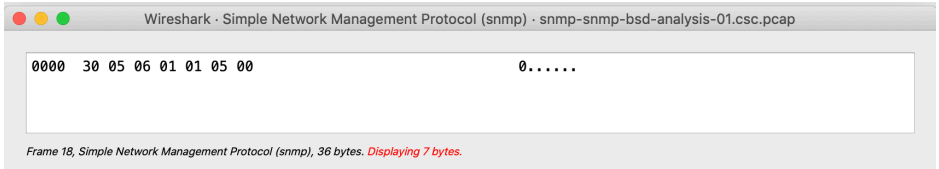
(a) SNMP Packet - Hex Dump (36 bytes)



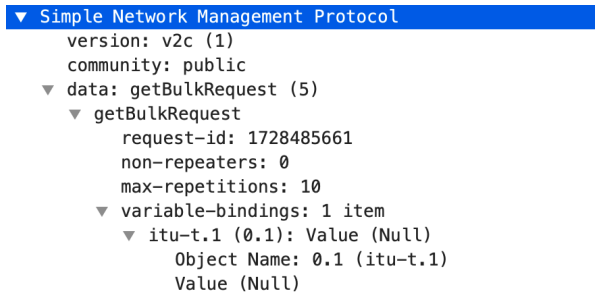
(b) SNMP Header - Hex Dump (27 bytes)



(c) SNMP Varbinds - Hex Dump (9 bytes)



(d) SNMP Varbind - Hex Dump (7 bytes)



(e) SNMP Packet - Visualization

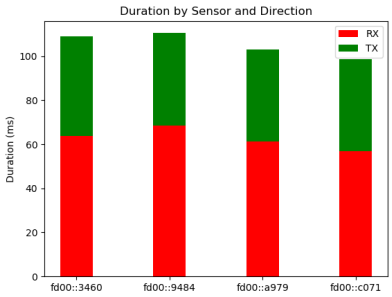
Figure 5.4: SNMP Wireshark.

Simulation		RX		TX	
Scenario	**Case**	**Frames**	**Bytes**	**Frames**	**Bytes**
01	0	8	874	8	761
01	1	4	562	4	377
01	2	1	141	1	96
01	3	1	94	1	96
02	0	24	2676	24	2411
02	1	12	1702	12	1195
02	2	3	423	3	304
02	3	3	291	3	304
05	0	120	13650	120	14080
05	1	60	8590	60	6980
05	2	15	2115	15	1775
05	3	15	1500	15	1775
tree	0	64	7154	64	6472
tree	1	32	4544	32	3208
tree	2	8	1128	8	816
tree	3	8	779	8	816

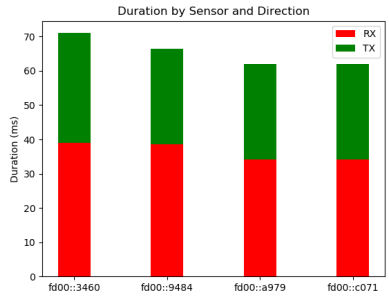
Table 5.4: SNMP Over the Air Bytes.

Case	Sink	TX (ms)	RX (ms)
0	fd00::3460	45,22705078	63,81225586
0	fd00::9484	41,93115234	68,57299805
0	fd00::a979	41,93115234	61,21826172
0	fd00::c071	41,93115234	56,79321289
1	fd00::3460	32,10449219	38,94042969
1	fd00::9484	27,80151367	38,60473633
1	fd00::a979	27,80151367	34,1796875
1	fd00::c071	27,80151367	34,21020508
2	fd00::3460	10,19287109	12,87841797
2	fd00::9484	10,19287109	14,03808594
2	fd00::a979	10,19287109	11,68823242
2	fd00::c071	10,19287109	11,65771484
3	fd00::3460	7,690429688	11,77978516
3	fd00::9484	7,690429688	14,09912109
3	fd00::a979	7,690429688	10,55908203
3	fd00::c071	7,690429688	11,77978516

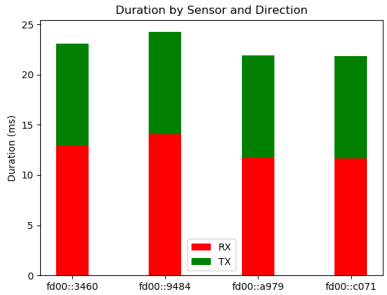
Table 5.5: SNMP Radio Activity Time.



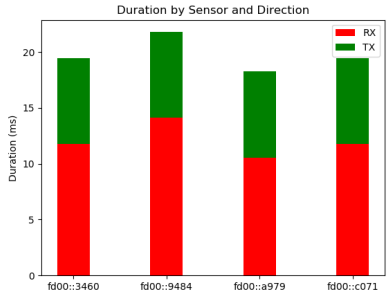
(a) SNMP Radio Activity Time - Case 0



(b) SNMP Radio Activity Time - Case 1



(c) SNMP Radio Activity Time - Case 2



(d) SNMP Radio Activity Time - Case 3

Figure 5.5: SNMP Radio Activity Time.

5.2.2 *Constrained Application Protocol* (CoAP)

Since the idea behind using the CoAP to also transmit monitoring data is to reuse a module that most likely is already being used by the device. Therefore, no optimization on the module was done, all minimal features were kept. However, the max chunk size was doubled since some strings can be quite large in this scenario. It uses 9724 bytes of ROM and 1986 of RAM. In Figure 5.6 and Table 5.6 the code footprint for the CoAP module can be seen. The object with the biggest ROM footprint is the coap.o which has the biggest number of functions and the biggest RAM footprint is in the coap-transactions.o which has a memory block, used for pseudo-dynamic allocation in the Contiki-NG environment, to store the transactions.

In the script's side, the CoAPthon presented in [TVM15] was used. In the CoAP's implementation there is no walk operation available, therefore the analysis will be done in both the request and response. In the request as shown in Figures 5.7(a) and 5.7(d), the CoAP header uses 6 bytes, as described in Figure 5.7(b). It is followed by the

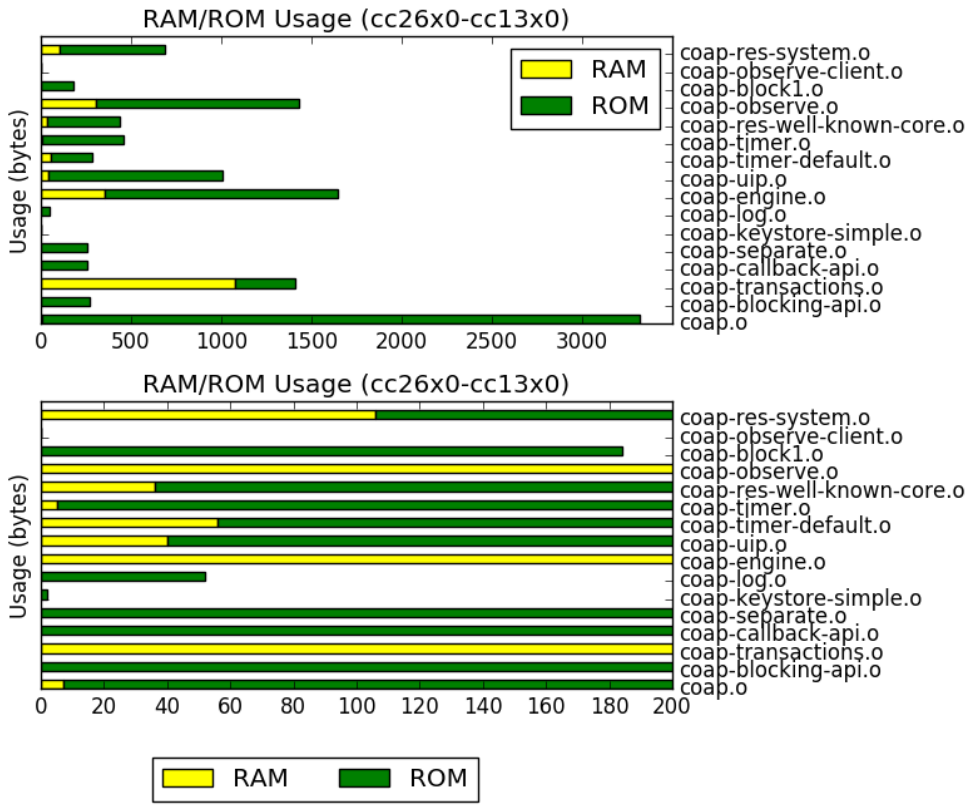
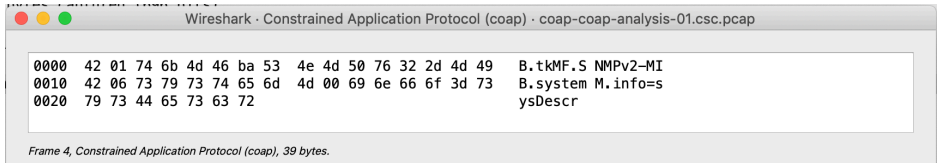


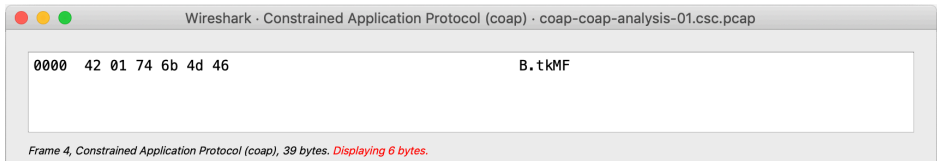
Figure 5.6: CoAP Code Footprint in the cc26x0-cc13x0 Target.

options that use 33 bytes, as evidenced in Figure 5.7(c) and in this scenario contains three values, the first URI Path which is the MIB name, in this case, SNMPv2-MIB, which is followed by the system tree as a URI Path too. To allow multiple KPIs in the same request a URI Query was used instead of a URI Path, in this case, the key info, and the sysDescr value.

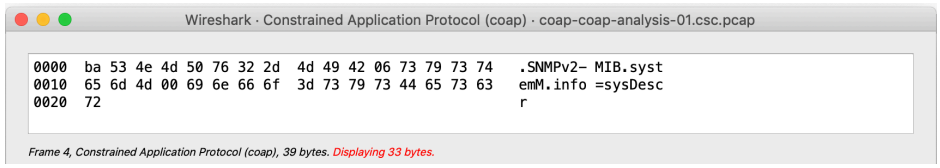
In the response as shown in Figures 5.8(a) and 5.8(e), the CoAP header uses 6 bytes, as illustrated in Figure 5.8(b). It is followed by the options that use 5 bytes, as indicated in Figure 5.8(c) and in this scenario contains two values, the Etag which is a resource local identifier and the Content-Format which shows how the payload should be interpreted, in this case, the payload is an application/json. Lastly, there is the Payload, as can be seen in Figure 5.8(d) which in this case uses 51 bytes. However, the JSON object size varies. Each one has a key which is surrounded by



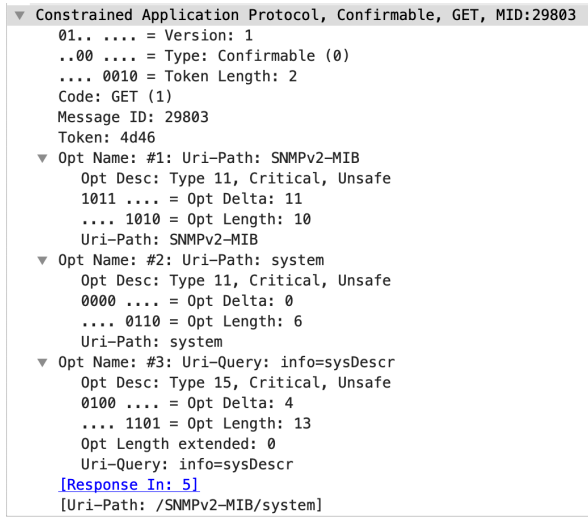
(a) CoAP Request Packet - Hex Dump (39 bytes)



(b) CoAP Request Header - Hex Dump (6 bytes)



(c) CoAP Request Options - Hex Dump (33 bytes)



(d) CoAP Request Packet - Visualization

Figure 5.7: CoAP Request Wireshark.

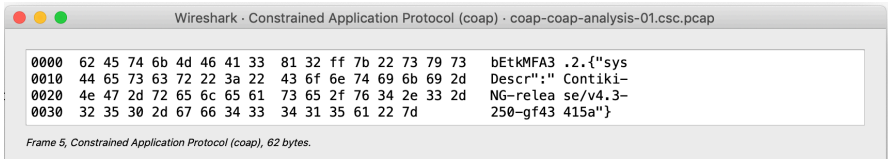
File	RAM	ROM	Total
coap.o	7	3316	3323
coap-blocking-api.o	0	268	268
coap-transactions.o	1076	334	1410
coap-callback-api.o	0	260	260
coap-separate.o	0	258	258
coap-keystore-simple.o	0	2	2
coap-log.o	0	52	52
coap-engine.o	353	1292	1645
coap-uip.o	40	968	1008
coap-timer-default.o	56	232	288
coap-timer.o	5	456	461
coap-res-well-known-core.o	36	400	436
coap-observe.o	307	1122	1429
coap-block1.o	0	184	184
coap-observe-client.o	0	0	0
coap-res-system.o	106	580	686
Total	1986	9724	11710

Table 5.6: CoAP Code Footprint in the cc26x0-cc13x0 Target.

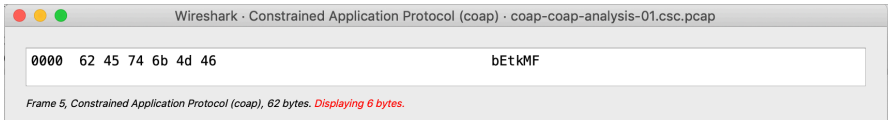
quotes and value. These are separated by a colon and an object starts and ends with curly brackets. A minimum JSON is this setup is {"" :} which uses 5 bytes. Each character in either the key or the value uses 1 byte.

In Tables 5.7 and 5.8 the CoAP network usage in all scenarios and cases are shown. The biggest downside of this implementation is in Case 0 and 1 where it is first necessary to request a list of all KPIs that are available before starting the bulk request. This disadvantage cannot be seen in Case 0 because on a traditional walk the client requests the next KPI after the one he is sending, therefore it does not know beforehand which is the last one available. It keeps asking until it is replied with the End of View message. In the CoAP implementation it asks for all KPIs available, so it knows the last one beforehand. The number of requests is the same because both have an extra request.

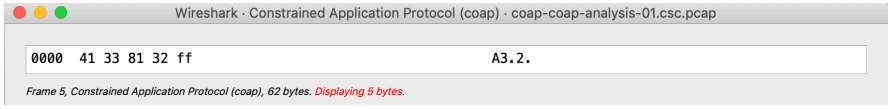
In addition, in Figure 5.9 and Table 5.9 the radio activity time of the protocol can be seen for each sink, which is the device being monitored. The more bytes are transmitted the more time it spends in TX. If the edge cases are removed, where most likely retransmissions occurred or beacons were transmitted during the protocol execution, a simple summary can be obtained. In case 0, 40,49 milliseconds are used, in case 1, 32,80 milliseconds are used, in case 2, 8,42 milliseconds are used and in case 3, 7,20 milliseconds are used.



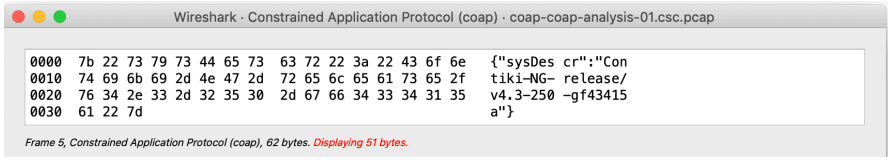
(a) CoAP Response Packet - Hex Dump (62 bytes)



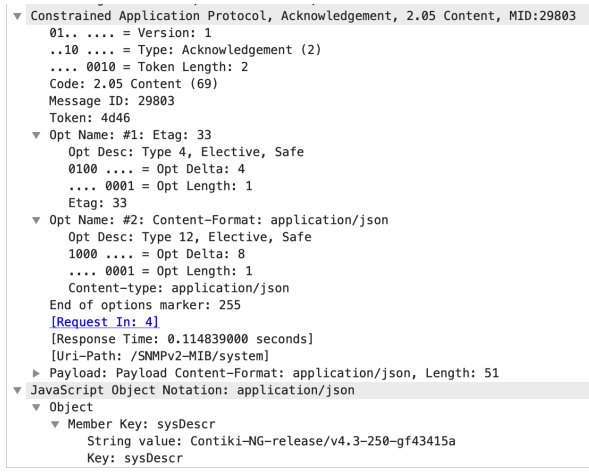
(b) CoAP Response Header - Hex Dump (6 bytes)



(c) CoAP Response Options - Hex Dump (5 bytes)



(d) CoAP Response Payload - Hex Dump (51 bytes)



(e) CoAP Packet - Visualization

Figure 5.8: CoAP Response Wireshark.

Simulation	RX		TX	
Case	**Frames**	**Bytes**	**Frames**	**Bytes**
0	8	407	8	307
1	5	371	5	218
2	1	68	1	39
3	1	29	1	40

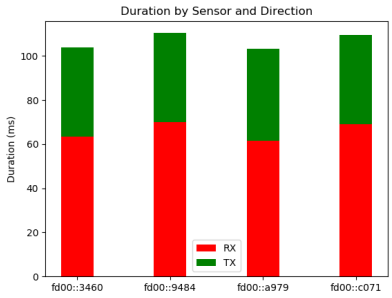
Table 5.7: CoAP Application Layer Network Usage.

Simulation		RX		TX	
Scenario	**Case**	**Frames**	**Bytes**	**Frames**	**Bytes**
01	0	8	813	8	731
01	1	5	637	5	483
01	2	1	117	1	92
01	3	1	78	1	93
02	0	24	2499	24	2321
02	1	15	1935	15	1529
02	2	3	360	3	292
02	3	3	246	3	295
05	0	120	12765	120	13525
05	1	75	9765	75	8914
05	2	15	1845	15	1700
05	3	15	1275	15	1715
tree	0	64	6666	64	6232
tree	1	40	5158	40	4104
tree	2	8	963	8	784
tree	3	8	659	8	792

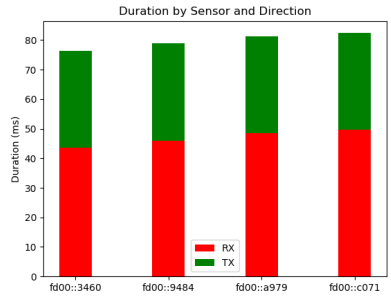
Table 5.8: CoAP Over the Air Bytes.

Case	Sink	TX (ms)	RX (ms)
0	fd00::3460	40,49682617	63,35449219
0	fd00::9484	40,49682617	69,94628906
0	fd00::a979	41,90063477	61,49291992
0	fd00::c071	40,52734375	69,03076172
1	fd00::3460	32,80639648	43,51806641
1	fd00::9484	32,80639648	46,02050781
1	fd00::a979	32,80639648	48,5534668
1	fd00::c071	32,80639648	49,59106445
2	fd00::3460	8,422851563	13,82446289
2	fd00::9484	8,422851563	15,13671875
2	fd00::a979	8,422851563	13,88549805
2	fd00::c071	9,826660156	13,85498047
3	fd00::3460	7,202148438	14,00756836
3	fd00::9484	10,68115234	17,45605469
3	fd00::a979	7,202148438	15,31982422
3	fd00::c071	7,202148438	16,51000977

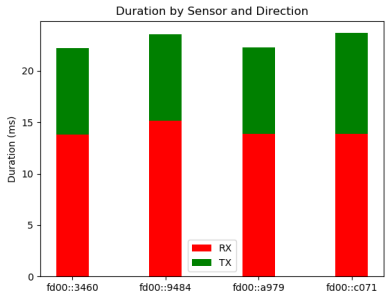
Table 5.9: CoAP Radio Activity Time.



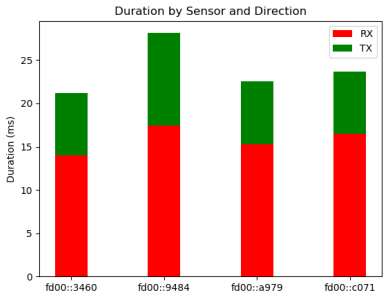
(a) CoAP Radio Activity Time - Case 0



(b) CoAP Radio Activity Time - Case 1



(c) CoAP Radio Activity Time - Case 2



(d) CoAP Radio Activity Time - Case 3

Figure 5.9: CoAP Radio Activity Time.

5.2.3 *Constrained Monitoring Protocol* (CoMP)

In code footprint, CoMP's performance was extremely well as shown in Table 5.10 and Figure 5.10. It uses 3216 bytes of ROM and 1100 of RAM. The most RAM expensive object is the comp.o, this happens because it has the packet buffer, which by default uses 512 bytes. In terms of ROM, the most expensive object is the comp-engine.o which handles all the requests. The code was optimized to reduce the code footprint as much as possible, therefore it might not be too human-friendly.

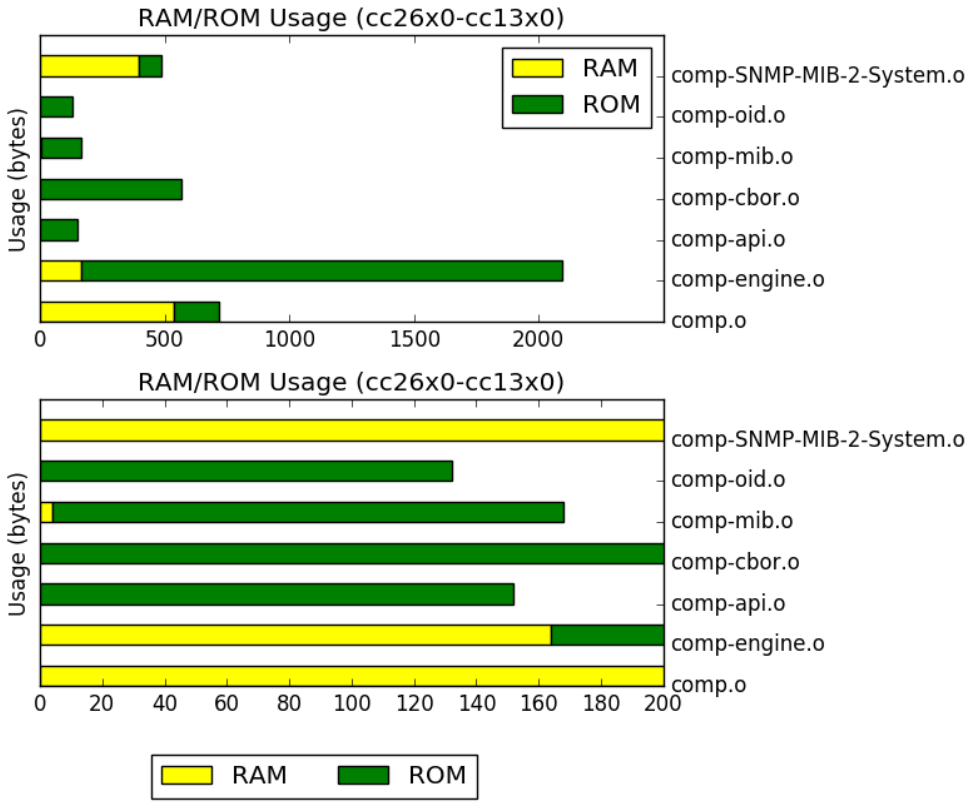


Figure 5.10: CoMP Code Footprint in the cc26x0-cc13x0 Target.

CoMP's request is analysed in depth in Figure 5.11. The header uses 3 bytes, as demonstrated in Figure 5.11(b). The variable content can vary, in this example the smallest possible OID in a request is used, 1, therefore it only uses 2 bytes, as shown in Figure 5.11(d), however, a variable-binding is always encoded as an array, therefore even if only one varbind is present, it is still encoded as an array with only one

File	RAM	ROM	Total
comp.o	536	180	716
comp-engine.o	164	1930	2094
comp-api.o	0	152	152
comp-cbor.o	0	566	566
comp-mib.o	4	164	168
comp-oid.o	0	132	132
comp-SNMP-MIB-2-System.o	396	92	488
Total	1100	3216	4316

Table 5.10: CoMP Code Footprint in the cc26x0-cc13x0 Target.

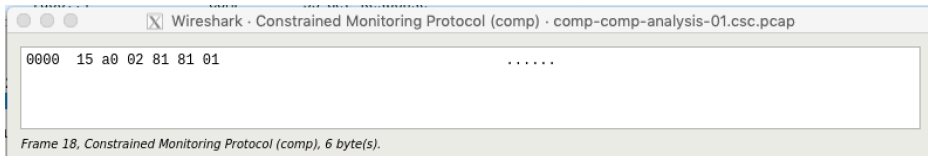
content, as can be seen in Figure 5.11(c). The minimal packet for a CoMP request is a Get-Bulk Request in the MIB root, 1, as described in Figures 5.11(a) and 5.11(e).

In Tables 5.11 and 5.12 it is possible to see that CoMP's performance was outstanding. Its header introduces almost no overhead, only 3 bytes. Therefore, the remaining bytes are filled with meaningful information and because of the CBOR encoding, it is concise.

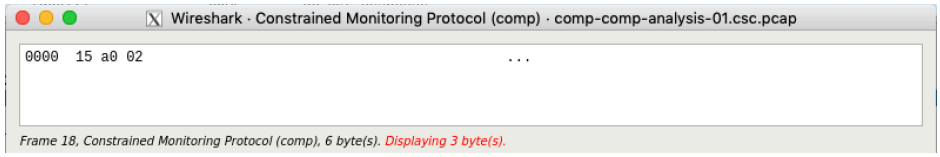
Simulation	RX		TX	
Case	Frames	Bytes	Frames	Bytes
0	8	258	8	104
1	4	216	4	48
2	1	48	1	14
3	1	7	1	14

Table 5.11: CoMP Application Layer Network Usage.

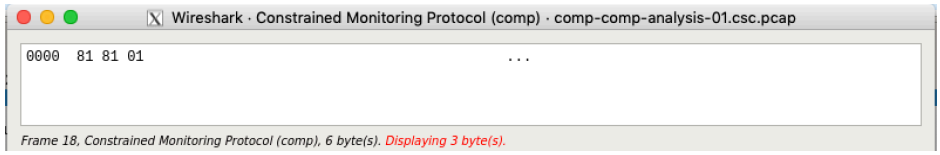
Equally important, in Figure 5.12 and Table 5.13 the radio activity time of the protocol can be seen for each sink, which is the device being monitored. The more bytes are transmitted the more time it spends in TX. If the edge cases are removed, where most likely retransmissions occurred or beacons were transmitted during the protocol execution, a simple summary can be obtained. In case 0, 32,74 milliseconds are used, in case 1, 20,99 milliseconds are used, in case 2, 7,78 milliseconds are used and in case 3, 6,49 milliseconds are used.



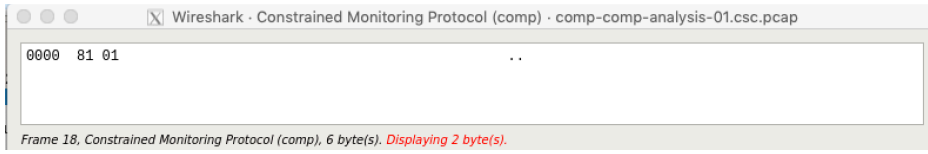
(a) CoMP Packet - Hex Dump (6 bytes)



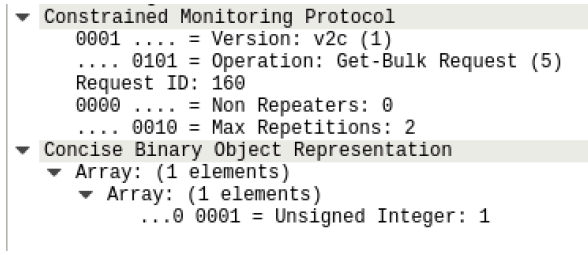
(b) CoMP Header - Hex Dump (3 bytes)



(c) CoMP Varbinds - Hex Dump (3 bytes)



(d) CoMP Varbind - Hex Dump (2 bytes)



(e) CoMP Packet - Visualization

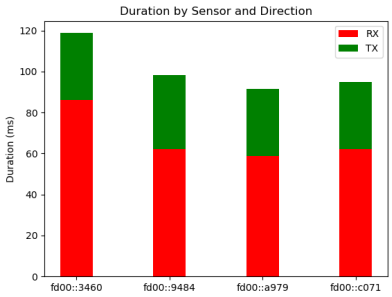
Figure 5.11: CoMP Wireshark.

Simulation		RX		TX	
Scenario	**Case**	**Frames**	**Bytes**	**Frames**	**Bytes**
01	0	8	650	8	528
01	1	4	412	4	260
01	2	1	97	1	67
01	3	1	56	1	67
02	0	24	2022	24	1712
02	1	12	1268	12	844
02	2	3	300	3	217
02	3	3	177	3	217
05	0	120	10470	120	10480
05	1	60	6500	60	5180
05	2	15	1545	15	1325
05	3	15	930	15	1325
tree	0	64	5416	64	4608
tree	1	32	3392	32	2272
tree	2	8	803	8	584
tree	3	8	475	8	584

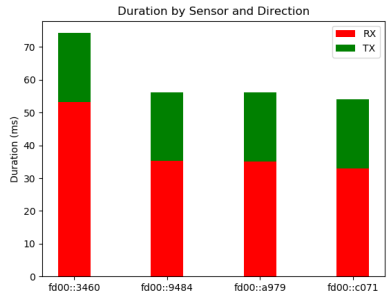
Table 5.12: CoMP Over the Air Bytes.

Case	Sink	TX (ms)	RX (ms)
0	fd00::3460	32,74536133	86,09008789
0	fd00::9484	36,04125977	62,34741211
0	fd00::a979	32,74536133	58,83789063
0	fd00::c071	32,74536133	62,31689453
1	fd00::3460	20,99609375	53,28369141
1	fd00::9484	20,99609375	35,18676758
1	fd00::a979	20,99609375	35,15625
1	fd00::c071	20,99609375	32,98950195
2	fd00::3460	7,781982422	20,20263672
2	fd00::9484	7,781982422	14,3737793
2	fd00::a979	7,781982422	10,89477539
2	fd00::c071	7,781982422	10,77270508
3	fd00::3460	6,439208984	18,92089844
3	fd00::9484	6,469726563	14,22119141
3	fd00::a979	6,469726563	10,77270508
3	fd00::c071	6,469726563	10,80322266

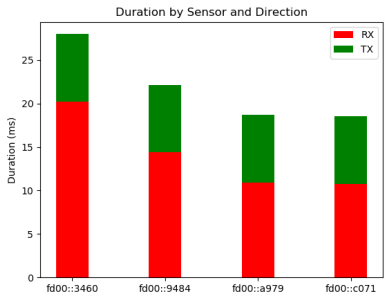
Table 5.13: CoMP Radio Activity Time.



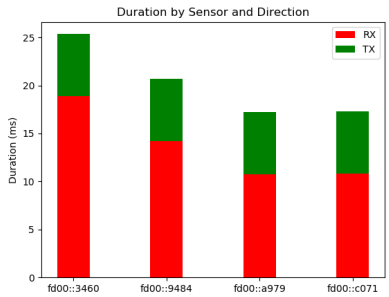
(a) CoMP Radio Activity Time - Case 0



(b) CoMP Radio Activity Time - Case 1



(c) CoMP Radio Activity Time - Case 2



(d) CoMP Radio Activity Time - Case 3

Figure 5.12: CoMP Radio Activity Time.

5.2.4 Cross Comparison

In Table 5.14 these protocol's code footprint information is summarized. It is evident that CoAP has the worst and that SNMP has a better performance than CoMP on ROM usage and CoMP is better on RAM usage. In Figure 5.13 the relative overhead from the smallest value is presented. It shows that the difference is indeed small, 1,82% for RAM and 2,42% for ROM.

	RAM	**ROM**	**Total**
SNMP	1120	3140	4260
CoAP	1986	9724	11710
CoMP	1100	3216	4316

Table 5.14: Protocols Code Footprint on cc26x0-cc13x0.

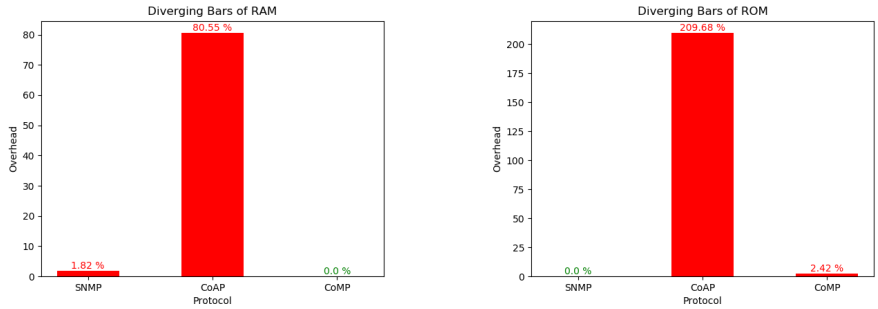


Figure 5.13: Protocols Code Footprint on cc26x0-cc13x0.

In Table 5.15 the protocol's application layer network usage information is summarized. It is clear that CoMP's gross numbers are the smallest ones. In Figures 5.14, 5.15, 5.16, 5.17 the relative overhead from the smallest value is presented for each case. In all cases, CoMP performed much better than the other protocols. In the worst-case SNMP received 542% more than CoMP and in the best case CoAP only received 41% more than CoMP.

	Case 0		**Case 1**		**Case 2**		**Case 3**	
	RX	**TX**	**RX**	**TX**	**RX**	**TX**	**RX**	**TX**
SNMP	468	337	352	165	85	43	45	43
CoAP	407	307	371	218	68	39	29	40
CoMP	258	104	216	48	48	14	7	14

Table 5.15: Protocols Application Layer Network Usage.

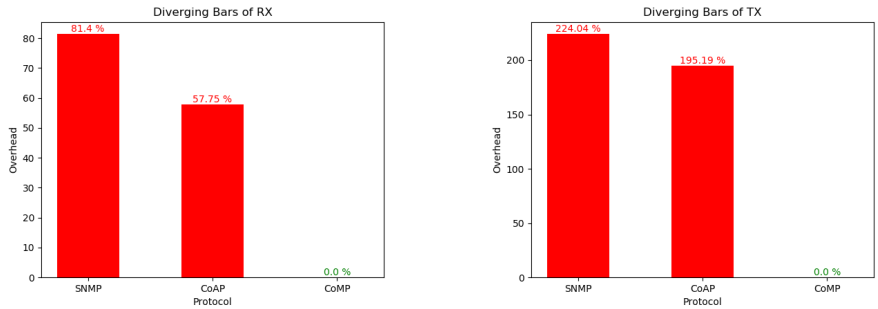


Figure 5.14: Protocols Application Layer Network Usage - Case 0.

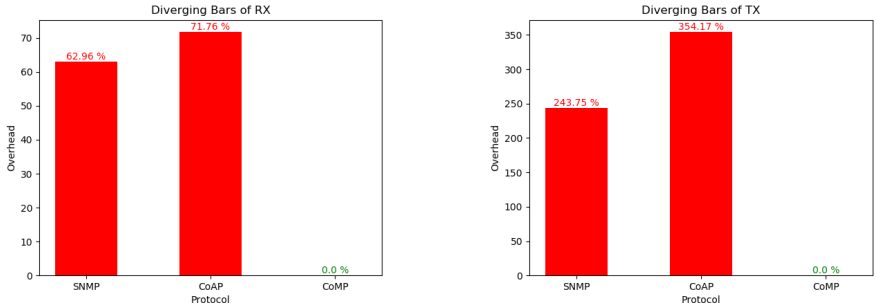


Figure 5.15: Protocols Application Layer Network Usage - Case 1.

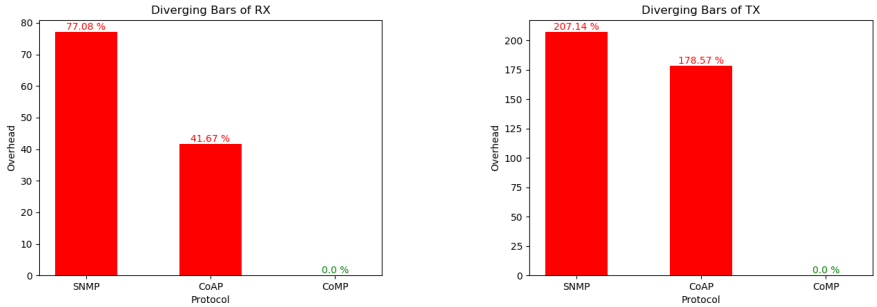


Figure 5.16: Protocols Application Layer Network Usage - Case 2.

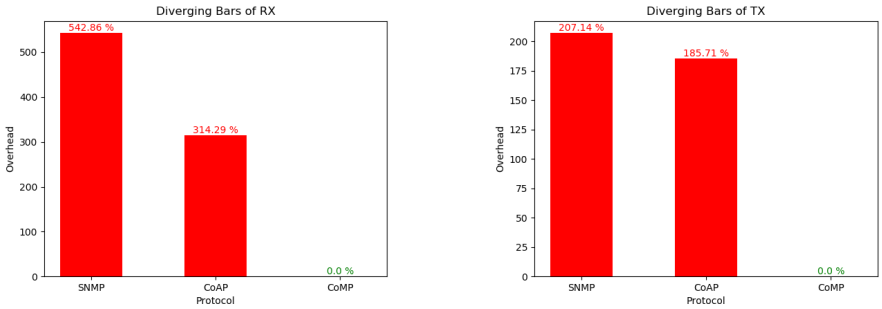


Figure 5.17: Protocols Application Layer Network Usage - Case 3.

However, in the WSN context, the number of bytes over the air matters an IoT. These networks are very energy-constrained therefore it is not enough to just analyze the bytes used in the application layer because each byte that is transmitted over the air increases energy consumption. Even though all the layers below the application one is out of the monitoring protocol control, such analysis is necessary to prove that by changing the protocol it is possible to reduce the energy consumption. Furthermore, network topology also affects energy consumption when monitoring WSNs. If the target is the last node in the network, all the nodes in between it and the server have to retransmit the request until it reaches the client and they again have to retransmit the response until it reaches the server. Therefore a more complete comparison of all the four topology bytes over the air was done to show the overall improvement.

In Table 5.16, the first topology bytes over the air for each protocol is summarized. It is again clear that CoMP's gross numbers are the smallest ones. In Figures 5.18, 5.19, 5.20 and 5.21 the relative overhead from the smallest value is presented for each case. In all cases, CoMP performed much better than the other protocols. In the worst-case CoAP transmitted 85% more than CoMP and in the best case CoAP only received 20% more than CoMP.

Scenario 01	**Case 0**		**Case 1**		**Case 2**		**Case 3**	
	RX	**TX**	**RX**	**TX**	**RX**	**TX**	**RX**	**TX**
SNMP	874	761	562	377	141	96	94	96
CoAP	813	731	637	483	117	92	78	93
CoMP	650	528	412	260	97	67	56	67

Table 5.16: Protocols Over the Air Bytes in Scenario 01.

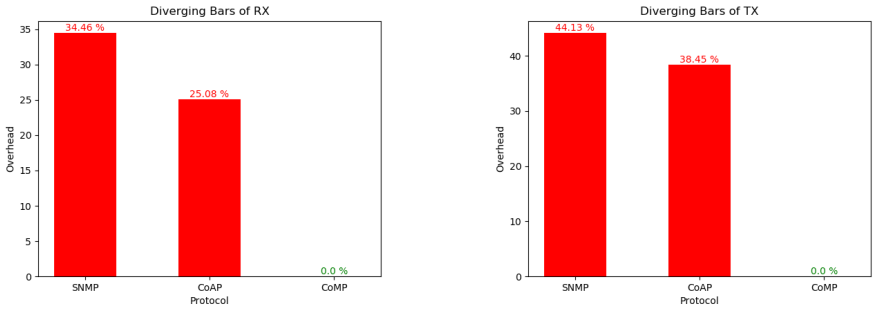


Figure 5.18: Protocols Over the Air Bytes in Scenario 01 - Case 0.

In Table 5.17, the second topology bytes over the air for each protocol is summarized. It is once more clear that CoMP's gross numbers are the smallest ones. In Figures

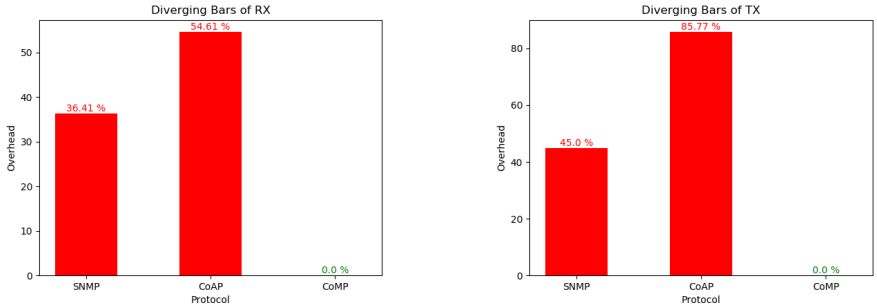


Figure 5.19: Protocols Over the Air Bytes in Scenario 01 - Case 1.

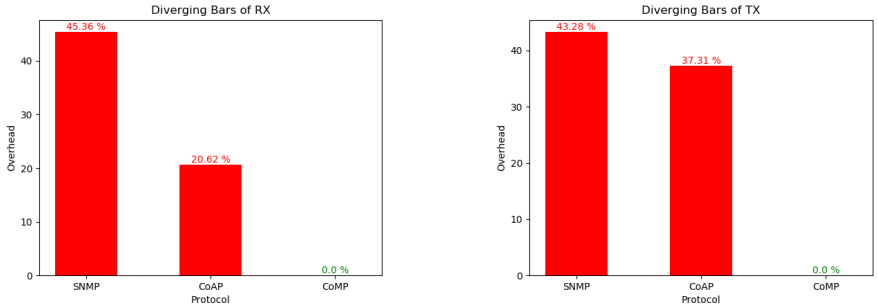


Figure 5.20: Protocols Over the Air Bytes in Scenario 01 - Case 2.

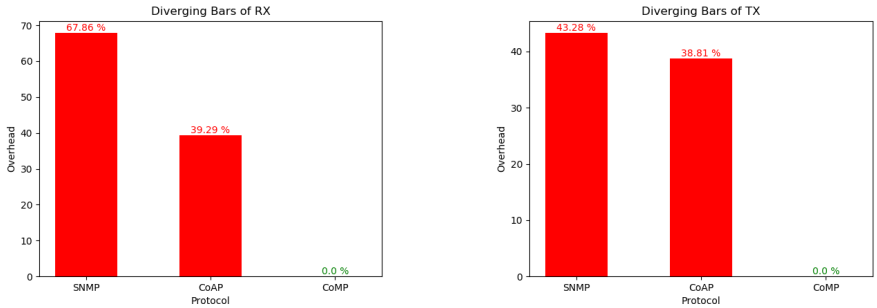


Figure 5.21: Protocols Over the Air Bytes in Scenario 01 - Case 3.

5.22, 5.23, 5.24 and 5.25 the relative overhead from the smallest value is presented for each case. In all cases, CoMP performed much better than the other protocols. In the worst-case CoAP transmitted 81% more than CoMP and in the best case CoAP only received 20% more than CoMP.

Scenario 02	Case 0		Case 1		Case 2		Case 3	
	RX	TX	RX	TX	RX	TX	RX	TX
SNMP	2676	2411	1702	1195	423	304	291	304
CoAP	2499	2321	1935	1529	360	292	246	295
CoMP	2022	1712	1268	844	300	217	177	217

Table 5.17: Protocols Over the Air Bytes in Scenario 02.

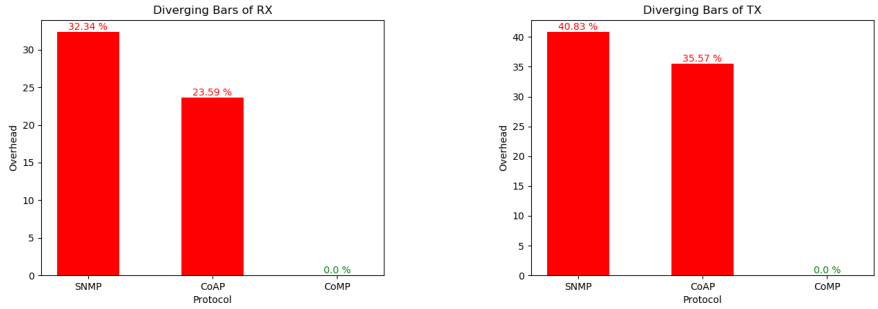


Figure 5.22: Protocols Over the Air Bytes in Scenario 02 - Case 0.

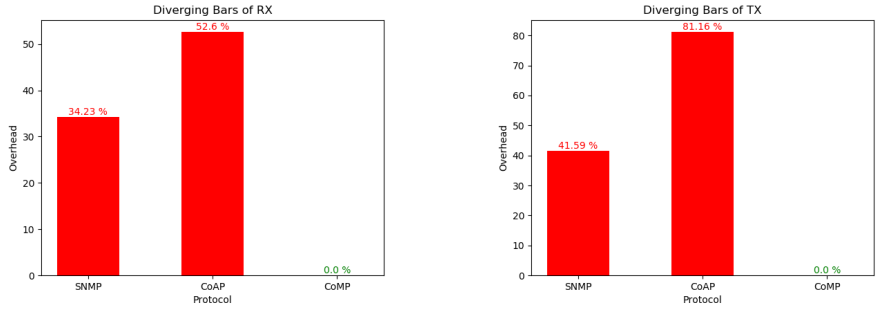


Figure 5.23: Protocols Over the Air Bytes in Scenario 02 - Case 1.

In Table 5.18, the second topology bytes over the air for each protocol is summarized. It is one more time clear that CoMP's gross numbers are the smallest ones. In Figures

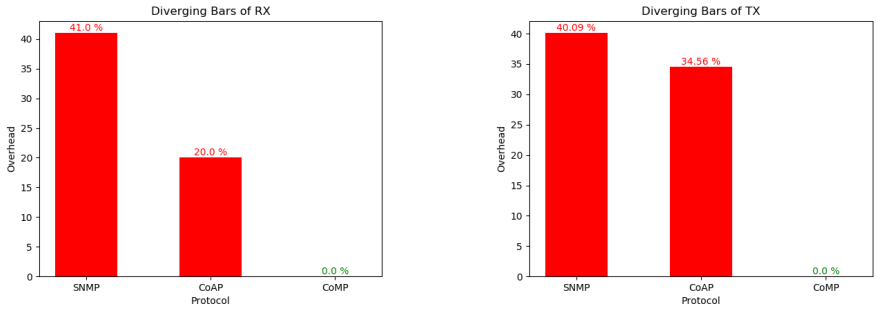


Figure 5.24: Protocols Over the Air Bytes in Scenario 02 - Case 2.

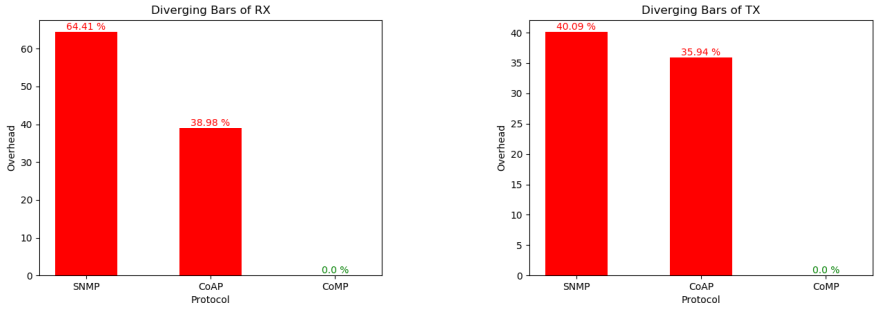


Figure 5.25: Protocols Over the Air Bytes in Scenario 02 - Case 3.

5.26, 5.27, 5.28 and 5.29 the relative overhead from the smallest value is presented for each case. In all cases, CoMP performed much better than the other protocols. In the worst-case CoAP transmitted 72% more than CoMP and in the best case CoAP only received 19% more than CoMP.

Scenario 05	**Case 0**		**Case 1**		**Case 2**		**Case 3**	
	RX	**TX**	**RX**	**TX**	**RX**	**TX**	**RX**	**TX**
SNMP	13650	14080	8590	6980	2115	1775	1500	1775
CoAP	12765	13525	9765	8914	1845	1700	1275	1715
CoMP	10470	10480	6500	5180	1545	1325	930	1325

Table 5.18: Protocols Over the Air Bytes in Scenario 05.

In Table 5.19, the second topology bytes over the air for each protocol is summarized.

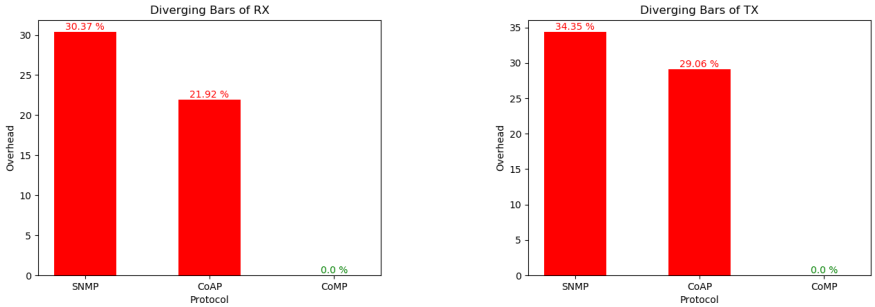


Figure 5.26: Protocols Over the Air Bytes in Scenario 05 - Case 0.

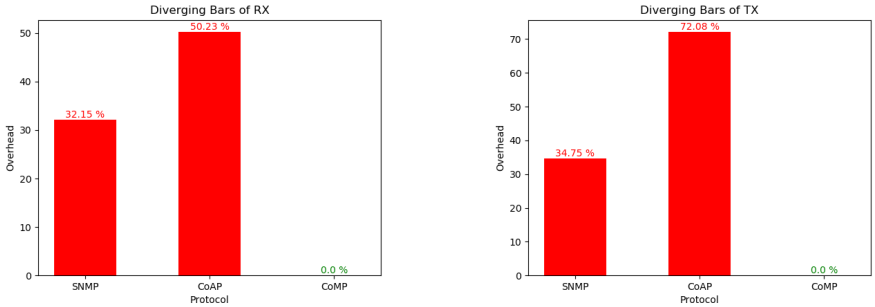


Figure 5.27: Protocols Over the Air Bytes in Scenario 05 - Case 1.

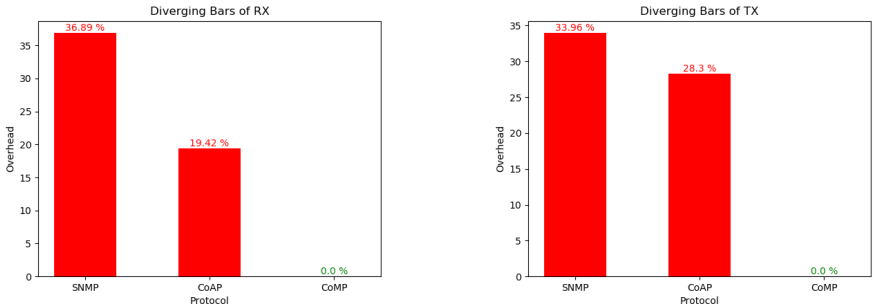


Figure 5.28: Protocols Over the Air Bytes in Scenario 05 - Case 2.

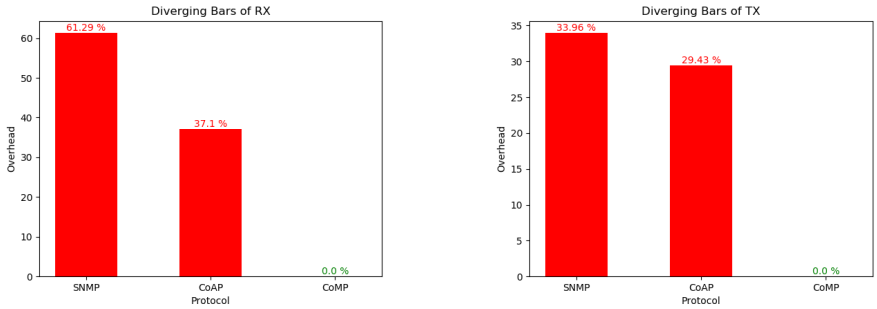


Figure 5.29: Protocols Over the Air Bytes in Scenario 05 - Case 3.

It is once more clear that CoMP's gross numbers are the smallest ones. In Figures 5.30, 5.31, 5.32 and 5.33 the relative overhead from the smallest value is presented for each case. In all cases, CoMP performed much better than the other protocols. In the worst-case CoAP transmitted 80% more than CoMP and in the best case CoAP only received 19% more than CoMP.

Scenario Tree	**Case 0**		**Case 1**		**Case 2**		**Case 3**	
	RX	**TX**	**RX**	**TX**	**RX**	**TX**	**RX**	**TX**
SNMP	7154	6472	4544	3208	1128	816	779	816
CoAP	6666	6232	5158	4104	963	784	659	792
CoMP	5416	4608	3392	2272	803	584	475	584

Table 5.19: Protocols Over the Air Bytes in Scenario Tree.

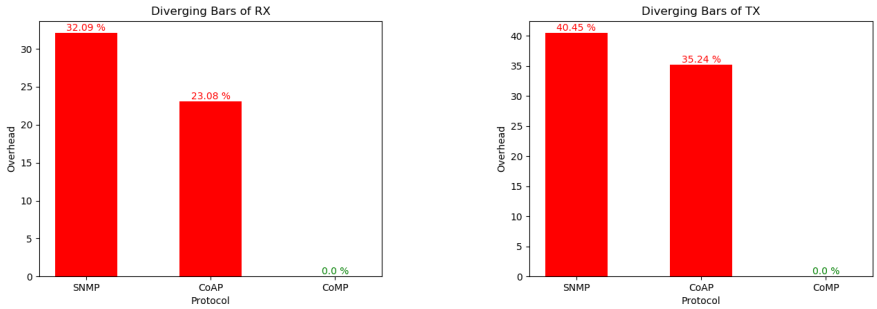


Figure 5.30: Protocols Over the Air Bytes in Scenario Tree - Case 0.

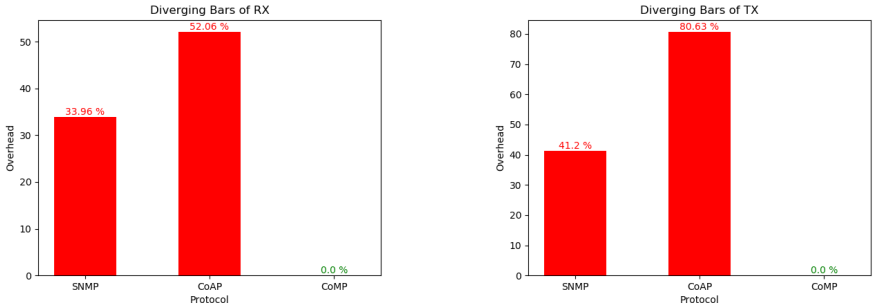


Figure 5.31: Protocols Over the Air Bytes in Scenario tree - Case 1.

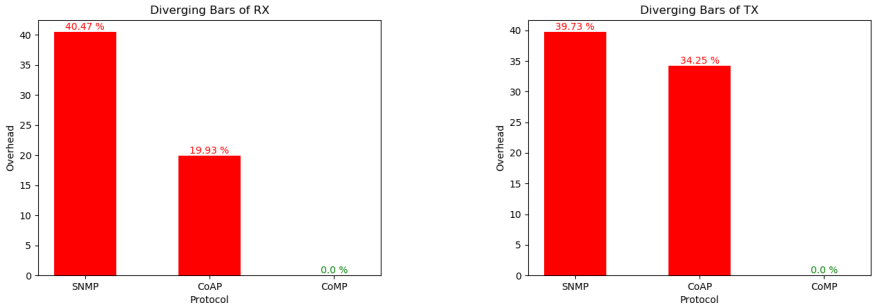


Figure 5.32: Protocols Over the Air Bytes in Scenario tree - Case 2.

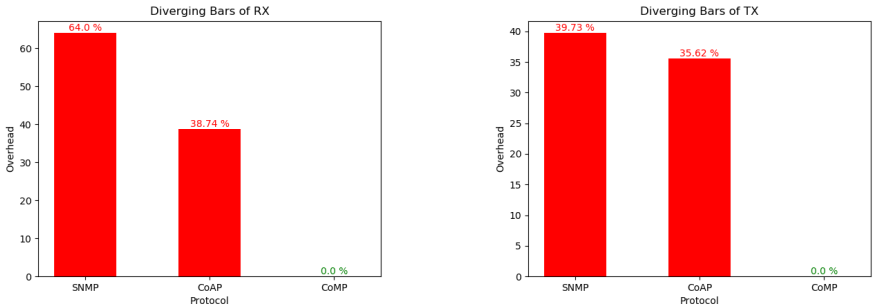
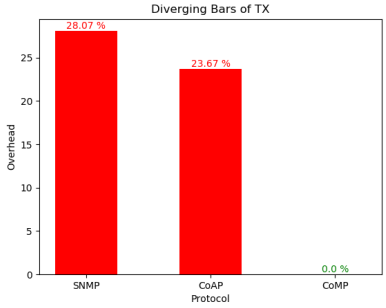


Figure 5.33: Protocols Over the Air Bytes in Scenario Tree - Case 3.

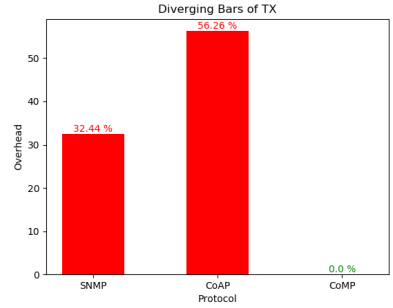
In Table 5.20, the radio activity time in the FIT/IoT Lab for each protocol is summarized. It is again clear that CoMP's gross numbers are the smallest ones. In Figure 5.34 the relative overhead from the smallest value is presented for each case. In all cases, CoMP performed much better than the other protocols. In the worst-case CoAP transmission took 56% more than CoMP and in the best case CoAP transmission took 8% more than CoMP.

	Case 0		**Case 1**		**Case 2**		**Case 3**	
	RX	**TX**	**RX**	**TX**	**RX**	**TX**	**RX**	**TX**
SNMP	56,79	41,93	34,17	27,80	11,65	10,19	10,55	7,69
CoAP	61,49	40,49	43,51	32,80	13,82	8,42	14,00	7,20
CoMP	58,83	32,74	32,98	20,99	10,77	7,78	10,77	6,43

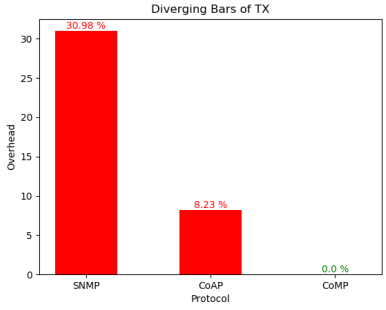
Table 5.20: Protocols Radio Activity Time FIT/IoT Lab.



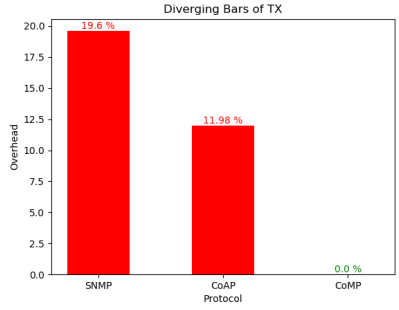
(a) Case 0



(b) Case 1



(c) Case 2



(d) Case 3

Figure 5.34: Protocols Radio Activity Time FIT/IoT Lab.

CHAPTER 6

Border Router

In this chapter, an interoperability model between the SNMP and the CoMP is presented. This would allow a seamless integration of WSN into SNMP monitoring systems[1,2,3]. A border router experiment will also be done in order to strengthen this case and the results are shown and analyzed.

6.1 Motivation

The IEEE 802.15.4 networks are always separated from the IP networks by a border gateway. The key reason for this gateway is to translate IP packets from and to these networks, however, the payload of the IP packet is kept intact, only the header is compressed. But some protocols are too troublesome, with overheads in the application layer. One good example is the HTTP, this is why the CoAP was proposed. They have the same characteristics and basic functions. Meaning that it is possible to map every HTTP request into a CoAP, as shown in [LH14]. The opposite is not always possible, since CoAP has the Observer extension, for example. As proposed in [Cru+19], to make the interoperability easier it is possible to have the border gateway to translate the protocol from the IP networks that have a matching protocol in the IEEE 802.15.4 networks. This makes implementation easier because it is not necessary to change all HTTP requests into CoAP ones, on the client level. It does not solve the problem of having different standards for these networks, but sometimes it is necessary to create a new protocol that is more efficient for the WSN context.

As explained in [Ge+15], a common border router implementations only open the packet until the Network Layer to convert it, Figure 6.1. By doing so, it allows the communication between an IP network with a 6LoWPAN network, by adding the adaptation layer between the Network Layer and the Data Link Layer. It likewise needs to change the Data Link Layer and the Physical Layer from the Ethernet standard, for example, to the IEEE 802.15.4 standard.

[1] `https://www.solarwinds.com/`
[2] `https://www.cacti.net/`
[3] `https://www.paessler.com/prtg`

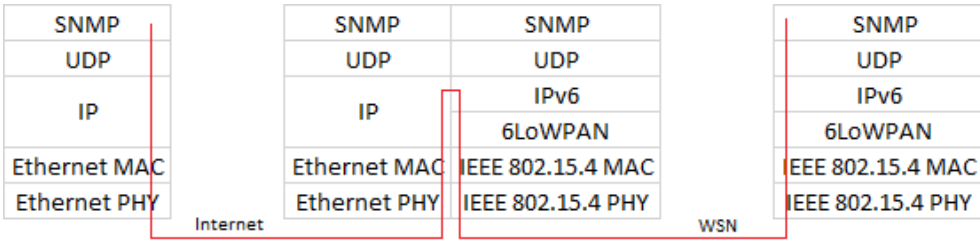


Figure 6.1: Normal Border Router Protocol Stack.

If the border router opens the packet until the Application Layer is possible to convert application protocols too. This would allow interoperability between different protocols, increasing the WSN performance. A good example is the possibility to map HTTP requests into CoAP ones. The performance increase can be seen in Figure 6.2. According to [LMT18] the throughput can be increased up to 55% in the same situation when comparing HTTP with CoAP and the latency is reduced up to 55% too in the same comparison. Assuming that the physical conditions were the same in all the experiments, the only variables that justify the better performance from CoAP over HTTP is the change in the transport protocol from TCP to UDP which reduces the transport reliability but it too reduces the complexity and size in this layer. The other change is in the application protocol itself, the CoAP was remodeled to have a more concise header. In [LH14] it is explained that CoAP contains a subset of the HTTP methods, the ones that are compliant with the *Representational State Transfer* (REST) architectural style. Therefore, it is possible to match some HTTP methods into CoAP doing what they called a cross-protocol proxying. This allows seamless integration of WSN networks into an already working network that uses HTTP to perform some task.

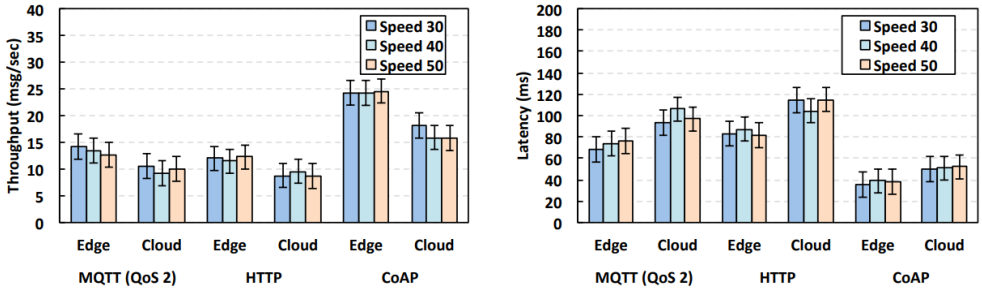


Figure 6.2: CoAP vs HTTP[LMT18].

6.2 Design

6.2.1 Implementation

The same concept of cross-protocol proxying proposed for CoAP and HTTP can be applied for CoMP and SNMP, Figure 6.3. CoMP implements a subset of SNMP's operations, Get, Get-Next, Get-Bulk, and Response, therefore it is possible to proxy these operations. Since SNMP does not have an unimplemented error, the only workaround is to send a *genError* error, when an operation that cannot be proxied is sent to a 6LoWPAN node.

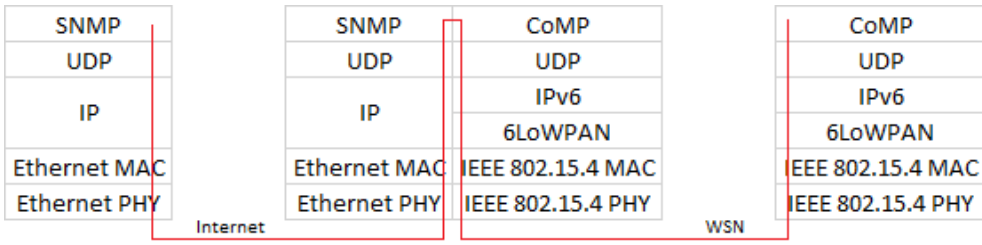


Figure 6.3: Application Border Router Protocol Stack.

This technique would allow several applications that already use SNMP to collect monitoring data like PRTG, as shown in Figure 6.4, to work on constrained nodes inside a WSN. This would make integration seamless because of the monitoring server it will be monitoring using SNMP, no change in the server has to be done. And the same features available for SNMP devices will be available for the ones with CoMP inside the WSN.

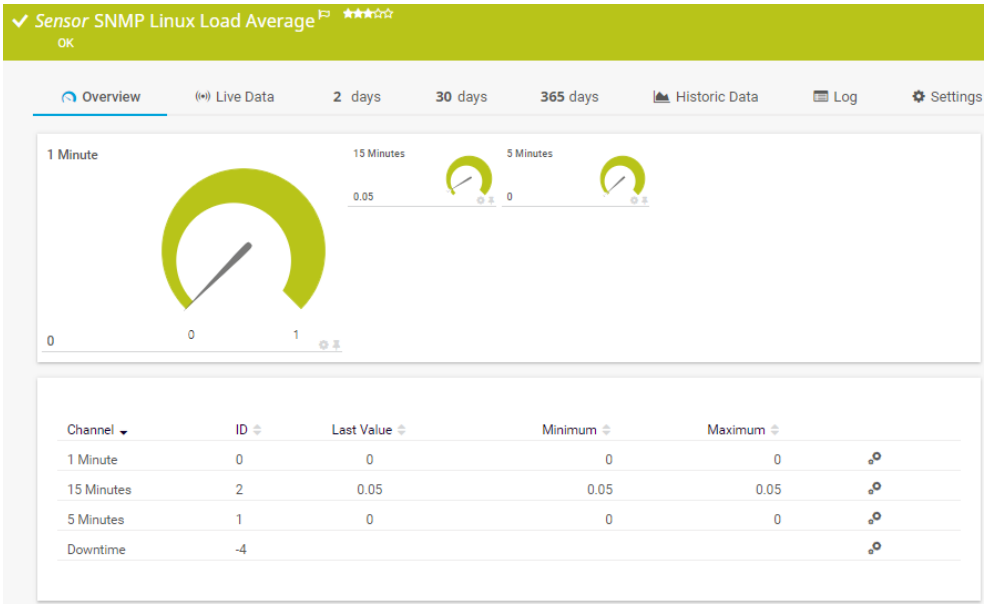


Figure 6.4: SNMP Data Visualization.

6.2.2 Types

In this proxying all the types defined in the RFC1155[RM90] are handled as shown in Table 6.1. All types have a corresponding type defined in the RFC7049[BH13]. The only exceptions are the NetworkAddress and IpAddress, however, there is a reserved CBOR Tag that covers these types.

The challenge faced here is the ambiguity. An SNMP request has a OID and the Null value. Therefore the request does not contain the response data type so the proxy cannot use it to know which data type the SNMP response should contain. One way to solve this challenge is to load all the MIBs that the proxy will be capable of handling. This approach will solve the problem by introducing a limited scope of OIDs.

[4] `http://www.employees.org/~ravir/cbor-network.txt`
[5] `http://www.employees.org/~ravir/cbor-network.txt`

SNMP	CoMP
INTEGER	Unsigned/Negative Integer
OCTET STRING	Text String
OBJECT IDENTIFIER	Array of Data Items
NULL	Null
SEQUENCE	Array of Data Items
SEQUENCE OF	Array of Data Items
NetworkAddress	cbor-network[4]
IpAddress	cbor-network[5]
Counter	Unsigned Integer
Gauge	Unsigned Integer
TimeTicks	Unsigned Integer
Opaque	Map of Pairs of Data Items

Table 6.1: SNMP to CoMP Types.

6.2.3 Caching

The same proxy can be used to reduce the number of messages being forwarded to the WSN. Some KPIs are static during a life cycle, they can only change in a system reboot. A good example is the sysDescr[6], 1.3.6.1.2.1.1.1, which it is a textual description of the network entity, like in Figure 6.5. This information will only be changed when the System is updated, therefore there is no point in forwarding these OID's request to the constrained node every time a request comes into the proxy. However, this is not a simple task because SNMP and CoMP have the Get-Next operation where the OID requested is not the one that will be responded, therefore the proxy needs to know which OID comes before the static OID to cache it. The normal Get request caching is straight forward, however, it is possible to request multiple OIDs in the same Get request, therefore, the OIDs that are not cached have to be requested to the WSN node, but this already reduces the network transmission. In a simple implementation, all statics OIDs are defined on compile time.

There are other caching mechanisms, in [Eri+15] time to live cache is analyzed, this caching technique is widely used on HTTP requests especially of resources like style sheets, JavaScript files or images. When data is transmitted through the proxy, it is cached for a certain amount of time. If this caching approach is used in the proxy, it would reduce a lot the network overhead if multiple servers are requesting the same OID in the same device. However it would affect live monitoring of these nodes, but the feasibility of live monitoring constrained devices is another challenge.

[6] `http://oidref.com/1.3.6.1.2.1.1.1`

```
% snmpwalk -v 2c -c public snmp.live.gambitcommunications.com system
SNMPv2-MIB::sysDescr.0 = STRING: Cisco Internetwork Operating System Software ..IOS (tm) RSP Software (RSP-JSV56I-M), Version 12.1(7), RELEASE SOFTWARE
(fc1)..Copyright (c) 1986-2001 by cisco Systems, Inc...Compiled Fri 23-Feb-01 05:14 by kellythw
```

Figure 6.5: sysDescr Example.

6.3 Evaluation

In Figure 6.6 a simple WSN setup is shown. To simulate this scenario application level and to evaluate the viability of this cross-protocol proxying three python scripts were used. One performs the same operations as a SNMP client would and it represents the Host Controller. The next script performs the proxying where the SNMP request is converted into a CoMP requests and the CoMP response is converted into a SNMP response and it represents the WSN Gateway. The last script simulates a CoMP server where a request is received, processed and the response is created, and it represents the WSN Measurement Nodes. This evaluation was done with python scripts for simplicity since the goal was to analyze if the protocols are compliant with each other and not to evaluate the performance.

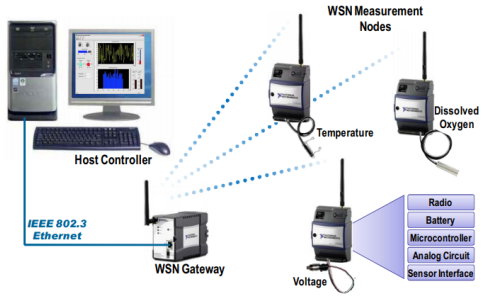
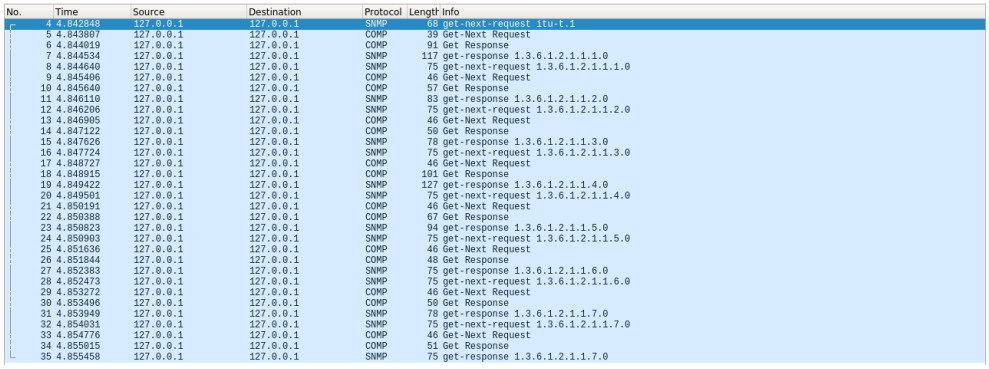


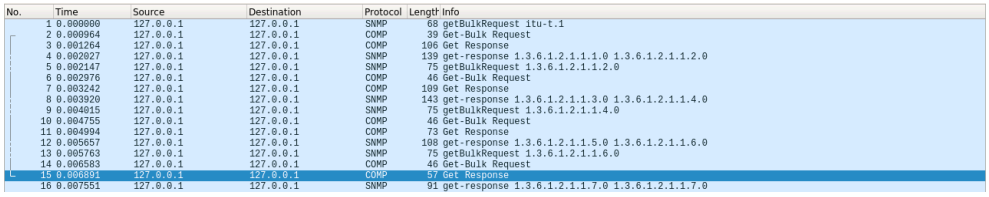
Figure 6.6: Example of WSN Setup [Han+13].

In Figures 6.7, 6.8, 6.9 and 6.10 it is possible to see the Wireshark output in each case. This proves that cross-protocol proxying is possible using the same scenarios used to evaluate the performance of the protocols.



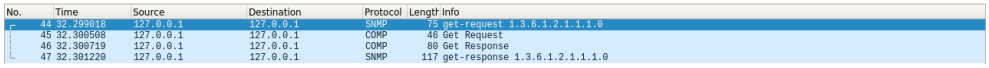
No.	Time	Source	Destination	Protocol	Length	Info
4	4.842848	127.0.0.1	127.0.0.1	SNMP	68	get-next-request itu-t.1
5	4.843807	127.0.0.1	127.0.0.1	COMP	39	Get-Next Request
6	4.844019	127.0.0.1	127.0.0.1	COMP	91	Get Response
7	4.844534	127.0.0.1	127.0.0.1	SNMP	117	get-response 1.3.6.1.2.1.1.1.0
8	4.844640	127.0.0.1	127.0.0.1	SNMP	75	get-next-request 1.3.6.1.2.1.1.1.0
9	4.845406	127.0.0.1	127.0.0.1	COMP	46	Get-Next Request
10	4.845640	127.0.0.1	127.0.0.1	COMP	57	Get Response
11	4.846110	127.0.0.1	127.0.0.1	SNMP	83	get-response 1.3.6.1.2.1.1.2.0
12	4.846206	127.0.0.1	127.0.0.1	SNMP	75	get-next-request 1.3.6.1.2.1.1.2.0
13	4.846905	127.0.0.1	127.0.0.1	COMP	46	Get-Next Request
14	4.847122	127.0.0.1	127.0.0.1	COMP	50	Get Response
15	4.847626	127.0.0.1	127.0.0.1	SNMP	78	get-response 1.3.6.1.2.1.1.3.0
16	4.847724	127.0.0.1	127.0.0.1	SNMP	75	get-next-request 1.3.6.1.2.1.1.3.0
17	4.848727	127.0.0.1	127.0.0.1	COMP	46	Get-Next Request
18	4.848915	127.0.0.1	127.0.0.1	COMP	101	Get Response
19	4.849422	127.0.0.1	127.0.0.1	SNMP	127	get-response 1.3.6.1.2.1.1.4.0
20	4.849501	127.0.0.1	127.0.0.1	SNMP	75	get-next-request 1.3.6.1.2.1.1.4.0
21	4.850191	127.0.0.1	127.0.0.1	COMP	46	Get-Next Request
22	4.850388	127.0.0.1	127.0.0.1	COMP	67	Get Response
23	4.850823	127.0.0.1	127.0.0.1	SNMP	94	get-response 1.3.6.1.2.1.1.5.0
24	4.850903	127.0.0.1	127.0.0.1	SNMP	75	get-next-request 1.3.6.1.2.1.1.5.0
25	4.851636	127.0.0.1	127.0.0.1	COMP	46	Get-Next Request
26	4.851844	127.0.0.1	127.0.0.1	COMP	48	Get Response
27	4.852383	127.0.0.1	127.0.0.1	SNMP	75	get-response 1.3.6.1.2.1.1.6.0
28	4.852473	127.0.0.1	127.0.0.1	SNMP	75	get-next-request 1.3.6.1.2.1.1.6.0
29	4.853272	127.0.0.1	127.0.0.1	COMP	46	Get-Next Request
30	4.853496	127.0.0.1	127.0.0.1	COMP	50	Get Response
31	4.853949	127.0.0.1	127.0.0.1	SNMP	78	get-response 1.3.6.1.2.1.1.7.0
32	4.854031	127.0.0.1	127.0.0.1	SNMP	75	get-next-request 1.3.6.1.2.1.1.7.0
33	4.854776	127.0.0.1	127.0.0.1	COMP	46	Get-Next Request
34	4.855015	127.0.0.1	127.0.0.1	COMP	51	Get Response
35	4.855458	127.0.0.1	127.0.0.1	SNMP	75	get-response 1.3.6.1.2.1.1.7.0

Figure 6.7: SNMP to CoMP Gateway - Case 0.



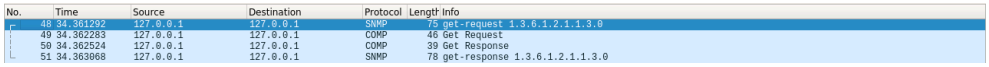
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	SNMP	68	getBulkRequest itu-t.1
2	0.000964	127.0.0.1	127.0.0.1	COMP	39	Get-Bulk Request
3	0.001264	127.0.0.1	127.0.0.1	COMP	106	Get Response
4	0.002027	127.0.0.1	127.0.0.1	SNMP	139	get-response 1.3.6.1.2.1.1.1.0 1.3.6.1.2.1.1.2.0
5	0.002147	127.0.0.1	127.0.0.1	SNMP	75	getBulkRequest 1.3.6.1.2.1.1.2.0
6	0.002976	127.0.0.1	127.0.0.1	COMP	46	Get-Bulk Request
7	0.003242	127.0.0.1	127.0.0.1	COMP	109	Get Response
8	0.003920	127.0.0.1	127.0.0.1	SNMP	143	get-response 1.3.6.1.2.1.1.3.0 1.3.6.1.2.1.1.4.0
9	0.004015	127.0.0.1	127.0.0.1	SNMP	75	getBulkRequest 1.3.6.1.2.1.1.4.0
10	0.004755	127.0.0.1	127.0.0.1	COMP	46	Get-Bulk Request
11	0.004994	127.0.0.1	127.0.0.1	COMP	73	Get Response
12	0.005657	127.0.0.1	127.0.0.1	SNMP	108	get-response 1.3.6.1.2.1.1.5.0 1.3.6.1.2.1.1.6.0
13	0.005763	127.0.0.1	127.0.0.1	SNMP	75	getBulkRequest 1.3.6.1.2.1.1.6.0
14	0.006583	127.0.0.1	127.0.0.1	COMP	46	Get-Bulk Request
15	0.006891	127.0.0.1	127.0.0.1	COMP	57	Get Response
16	0.007551	127.0.0.1	127.0.0.1	SNMP	91	get-response 1.3.6.1.2.1.1.7.0 1.3.6.1.2.1.1.7.0

Figure 6.8: SNMP to CoMP Gateway - Case 1.



No.	Time	Source	Destination	Protocol	Length	Info
44	32.299018	127.0.0.1	127.0.0.1	SNMP	75	get-request 1.3.6.1.2.1.1.1.0
45	32.300508	127.0.0.1	127.0.0.1	COMP	46	Get Request
46	32.300719	127.0.0.1	127.0.0.1	COMP	80	Get Response
47	32.301220	127.0.0.1	127.0.0.1	SNMP	117	get-response 1.3.6.1.2.1.1.1.0

Figure 6.9: SNMP to CoMP Gateway - Case 2.



No.	Time	Source	Destination	Protocol	Length	Info
48	34.361292	127.0.0.1	127.0.0.1	SNMP	75	get-request 1.3.6.1.2.1.1.3.0
49	34.362283	127.0.0.1	127.0.0.1	COMP	46	Get Request
50	34.362524	127.0.0.1	127.0.0.1	COMP	39	Get Response
51	34.363068	127.0.0.1	127.0.0.1	SNMP	78	get-response 1.3.6.1.2.1.1.3.0

Figure 6.10: SNMP to CoMP Gateway - Case 3.

CHAPTER 7

Conclusion

The major focus in this project was to analyze the SNMP and the CoAP and to propose a new protocol that offers the same functionality as the former but that has a constrained design like the latter.

SNMP proved to be extremely structured when used for network monitoring, hence why it is still the most used protocol for this purpose since the 1980s. It also has a small code footprint due to its simplicity. However, the amount of bytes transmitted in the application layer is massive, due to its encoding technique and because the request is like a questionnaire that the server fills in with the answers.

CoAP is very human-friendly in its design mainly because it was designed to be the HTTP for constrained devices. It has a very small header which considerably reduces the number of bytes in the application layer but it uses the URI for identification which means strings and consequentially a significant amount of bytes for resource identification. Additionally, it implements all the REST functionalities which are not necessary for network monitoring since the only method necessary would be the GET. This increases the code footprint significantly.

CoMP was designed to have the best of both worlds. It incorporates the messaging pattern, Request-Reply, resource identification, OID, monitoring values, MIB, and operation, Get, Get-Next and Get-Bulk, from SNMP. It uses the binary encoding for the header and the CBOR encoding for the body just like it can be done with CoAP. This design proved to have a really good code footprint, it was not better than SNMP. The difference was minimum and it was in favor of SNMP because CoMP implements an OID compression which increases the code footprint but it reduces the bytes transmitted when multiple OIDs are present. The bytes transmitted in the application layer were reduced significantly, up to 85% in some cases. This reduction reflected in the bytes over the air which affects the radio duty cycle that is directly tied to energy consumption.

Lastly, a cross-protocol proxying was proposed to allow the seamless integration of 6LoWPAN networks with CoMP with IP networks with SNMP. This makes easier to monitor WSN nodes with the same applications currently used to monitor networks

nodes.

7.1 Future Work

In this project, three proposals for network monitoring in the WSN context were shown. The newly introduced protocol, CoMP, had the best overall performance.

One question that can be raised from this is. Would another use case for CoMP be to export sensor readings? Since it uses the OID structure to organize its resources and the MIB as a database for its entities, it would be extremely simple to use it as a default way to export sensor readings. Currently, the standard way is to use the CoAP, but if the intention is to only export values and not receive any information, the latter feels like overkill for the problem since it has the entire REST functionality.

Another factor that was overlooked in this paper was the security perspective. SNMPv1 and v2 uses the community string as authentication, which its efficiency is very questionable. SNMPv3 uses username and password with several options for encryption. Since SNMP supports the Set operation, that can change information in the server, some authentication is highly necessary. However, CoMP only implement Get operations, making authentication not very important. Data sensitivity can be questioned in this moment but wireless networks are extremely vulnerable to sniffing. IEEE 802.15.4 offers access control, message integrity, confidentiality and replay protection. Does IEEE 802.15.4 offers all security necessary or an application layer security is likewise necessary?

APPENDIX A

Simple Network Management Protocol (SNMP) Implementation

In this project, the Contiki-NG operating system was used. However, it did not have the SNMP implemented. Since this is a standardized protocol, the implementation that was done for this project was merged into the operating system's source code. On Figures A.1, A.2, A.3, A.4 and A.5 it is possible to see all the discussions in the Contiki-NG repository on Github done during the SNMP implementation.

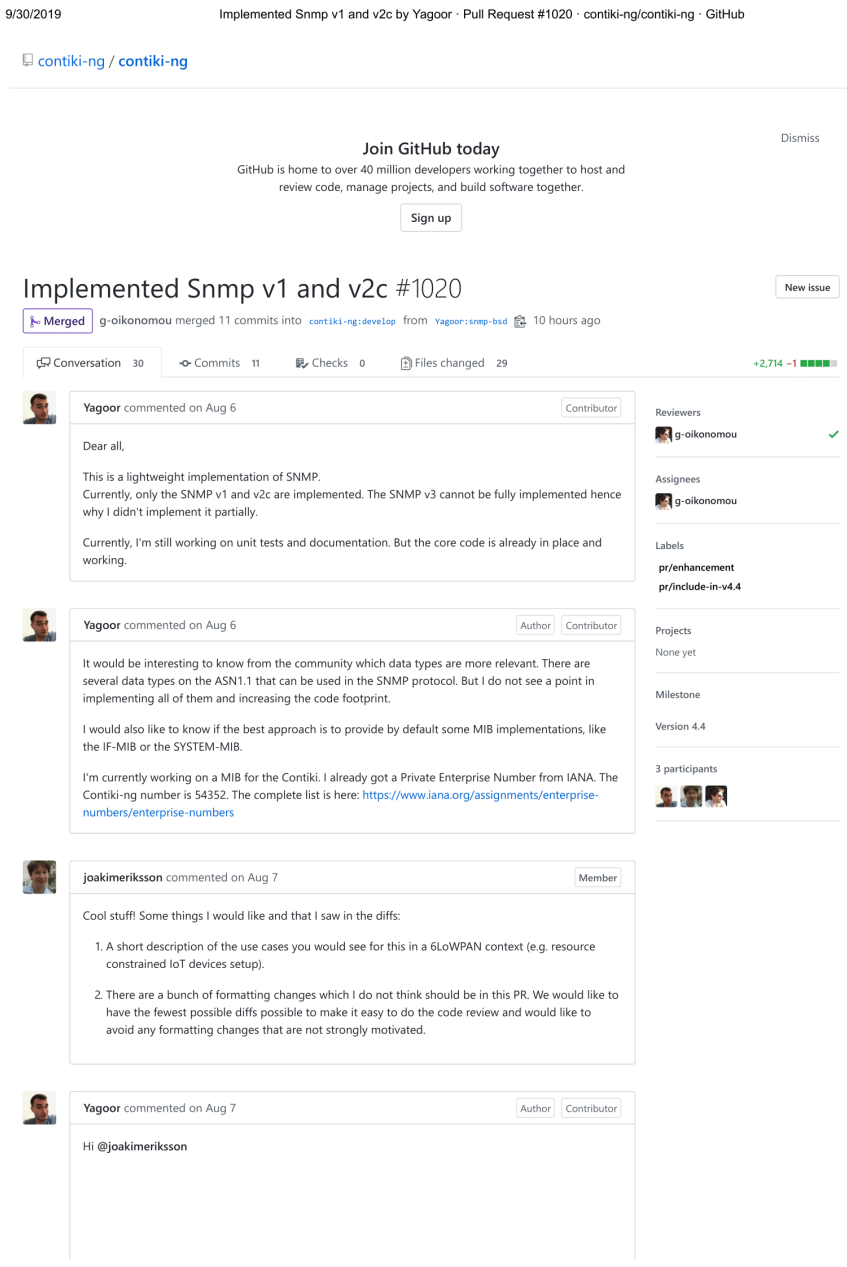


Figure A.1: SNMP Implementation - Page 1.

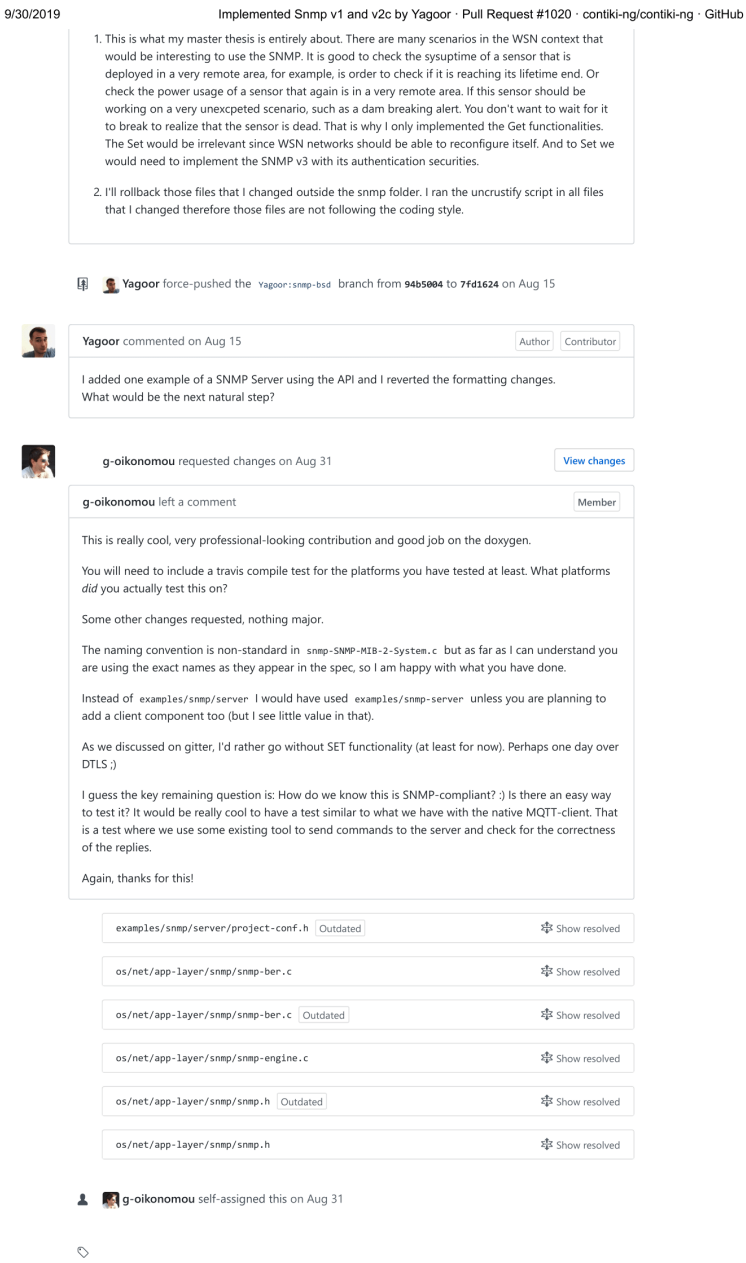


Figure A.2: SNMP Implementation - Page 2.

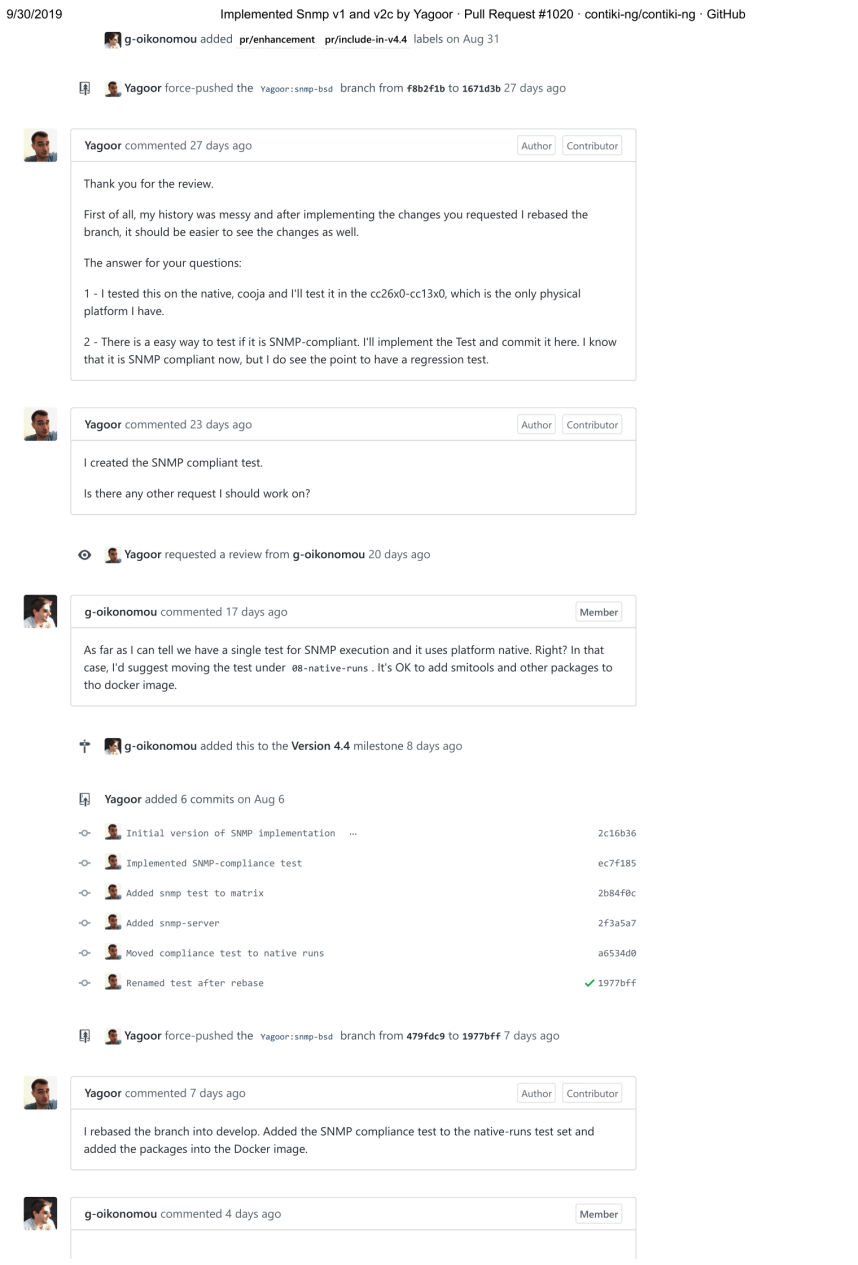


Figure A.3: SNMP Implementation - Page 3.

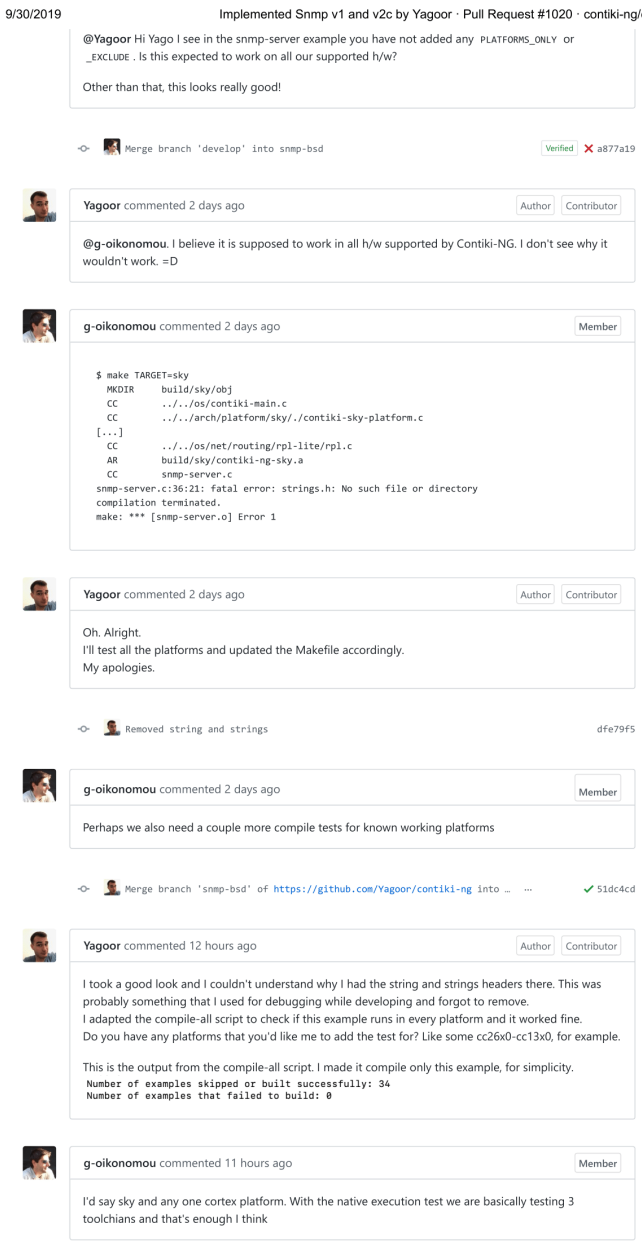


Figure A.4: SNMP Implementation - Page 4.

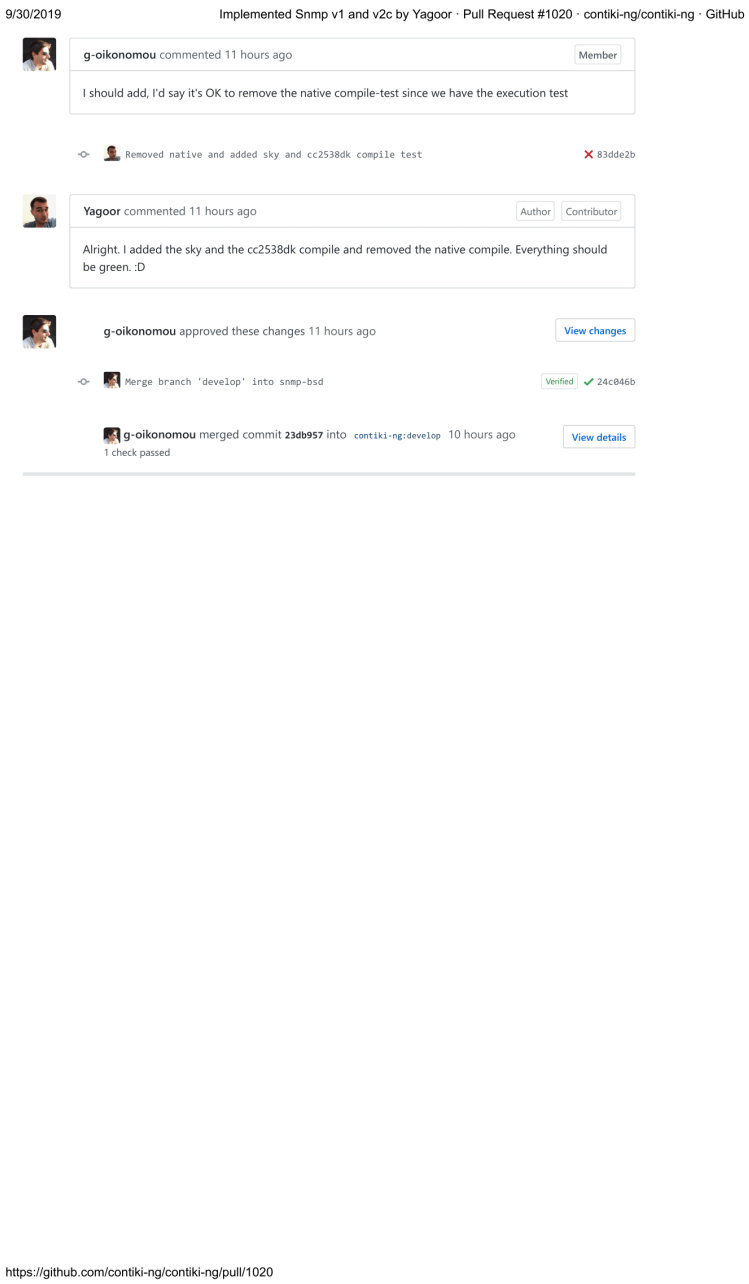


Figure A.5: SNMP Implementation - Page 5.

APPENDIX B

FIT/IoT Lab

While experimenting in the FIT/IoT Lab several small fixes were necessary to run the experiments.

The first issue encountered was the port would not compile when TSCH was enabled because there was a bug when expanding the macros, Figure B.1. Additionally, the ENERGEST_TYPE_IRQ macro, that did not exist in the latest Contiki-NG source code, Figure B.2. Either, due to version changes in the develop branch two version bumps of the submodule were necessary, Figures B.3 and B.8. Moreover, the platform_idle core function was not implemented which increased the energy usage, Figures B.4 and B.5. Furthermore, the energest was not implemented in the radio driver and to track the ticks spend in each radio duty cycle this is necessary, Figures B.6 and B.7. Also, during these fixes, continuous integration problems appeared and after discussing with the code responsible it was decided that some CI was necessary, Figures B.9 and B.10.

All these issues were fixed and Pull Requests were done to merge the fixes into the port source code. All were merged and are available.

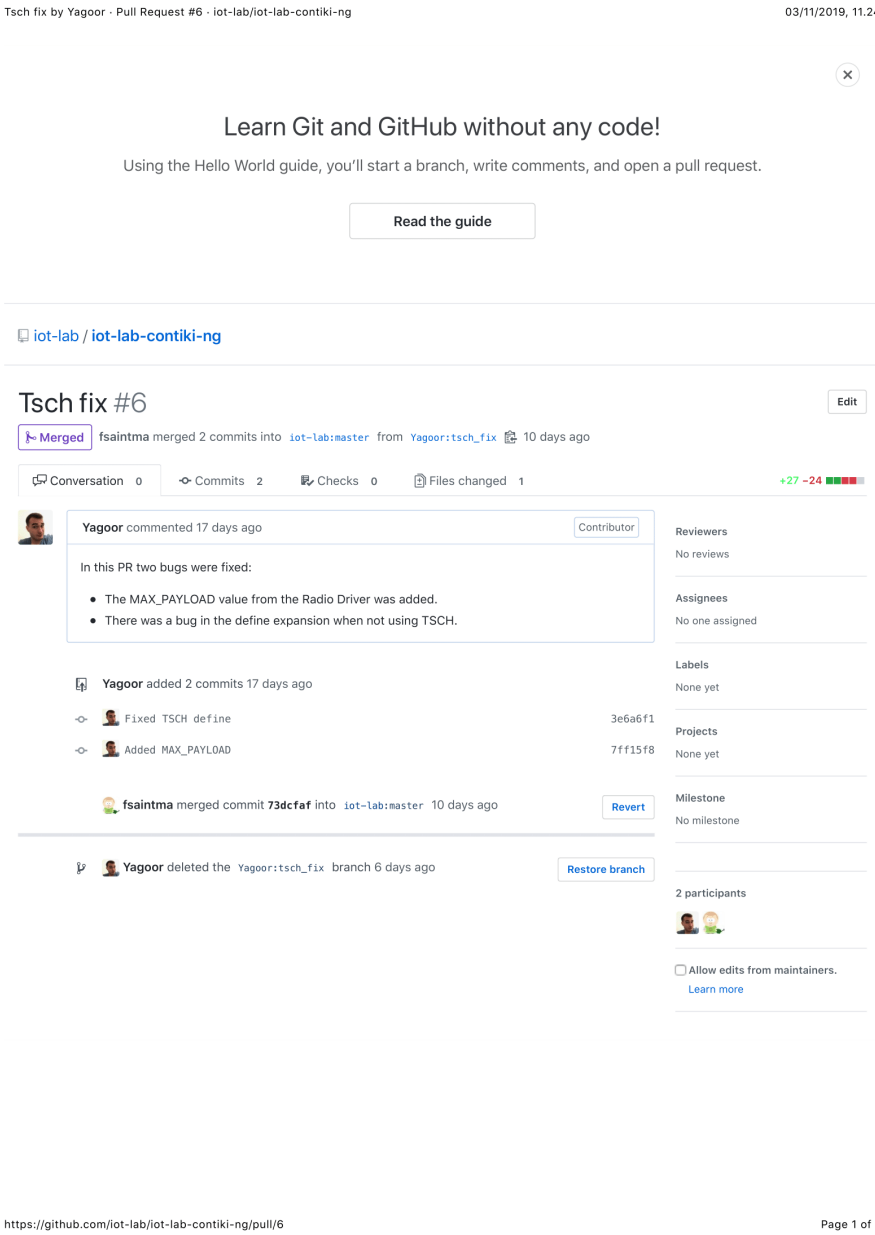


Figure B.1: TSCH Fix - Page 1.

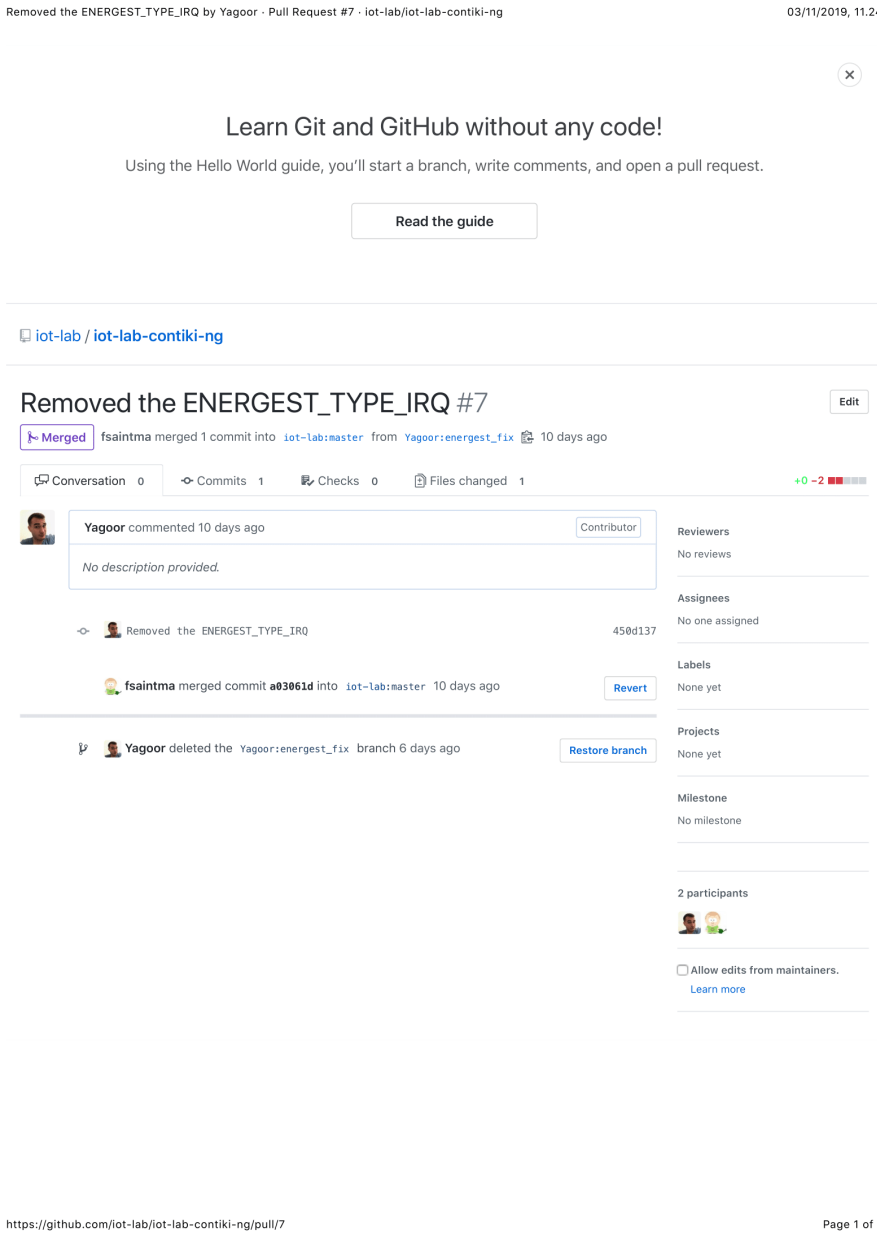


Figure B.2: Removed ENERGEST_TYPE_IRQ - Page 1.

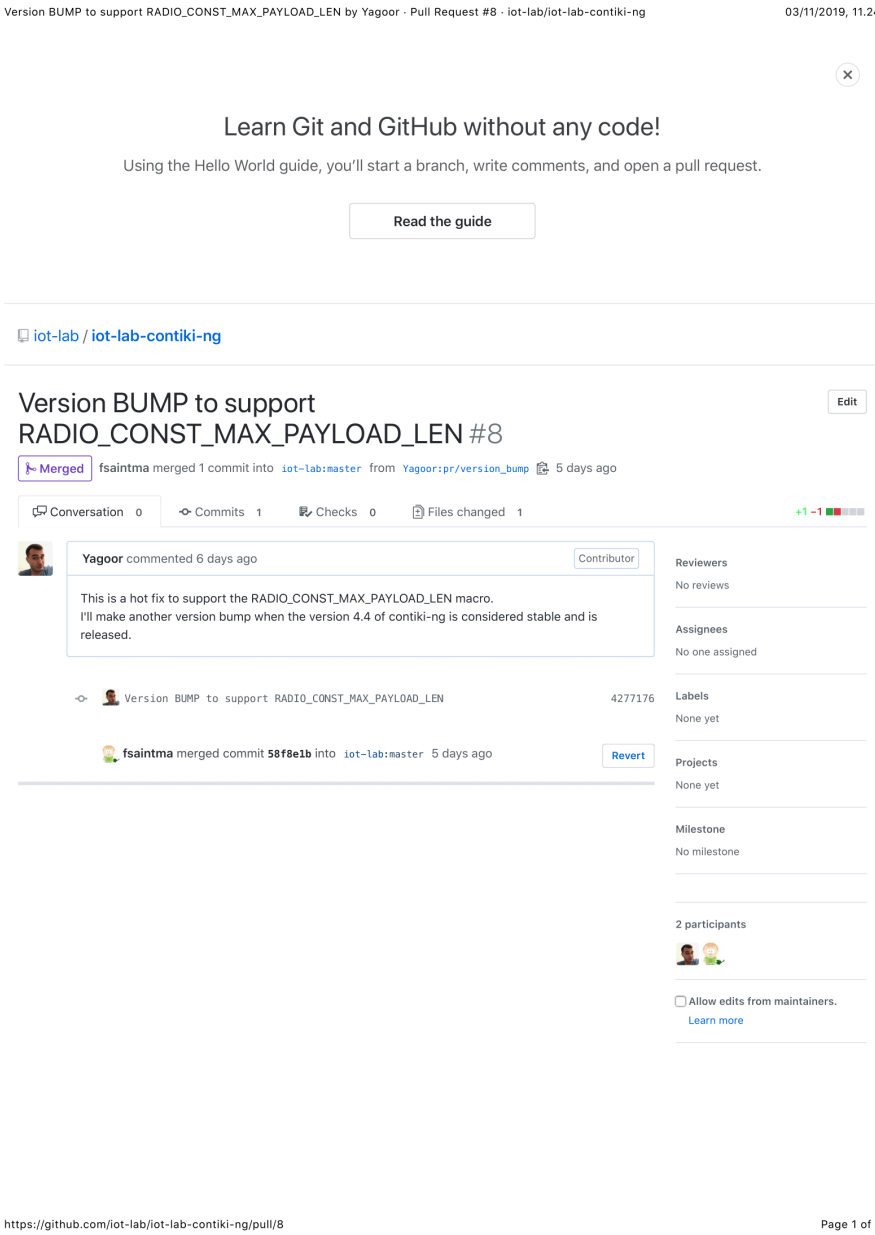


Figure B.3: Version Bump - Page 1.

iot-lab / **iot-lab-contiki-ng**

Implemented platform_idle #9

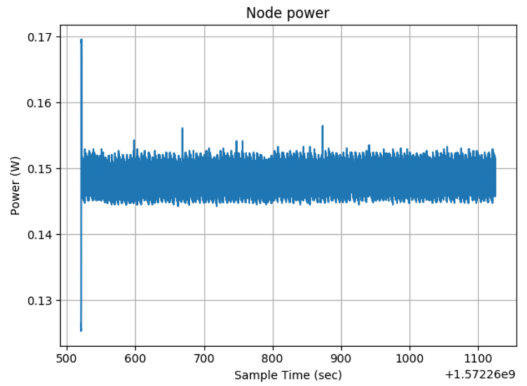
Edit

Merged fsaintma merged 2 commits into `iot-lab:master` from `Yagoor:pr/low_power` 1 hour ago

Conversation 0 Commits 2 Checks 0 Files changed 1 +7 −1

Yagoor commented 8 days ago Contributor

Added pwr_eneter_sleep usage



In this figure we have the same code running without the modification

Reviewers
No reviews

Assignees
No one assigned

Labels
None yet

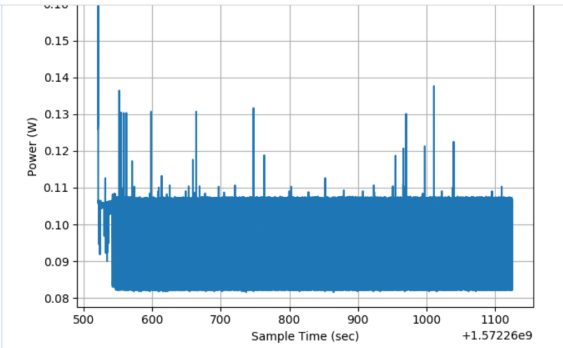
Projects
None yet

Milestone
No milestone

2 participants

Allow edits from maintainers.
Learn more

Figure B.4: Implemented platform_idle - Page 1.



In this figure we have the platform_idle implemented.

Implemented platform_idle fb4734c

Yagoor referenced this pull request 2 days ago

Added energest in radio driver #11 Merged

Added Energest ON to init d6dda8f

fsaintma merged commit 9bf9c02 into iot-lab:master 1 hour ago Revert

Figure B.5: Implemented platform_idle - Page 2.

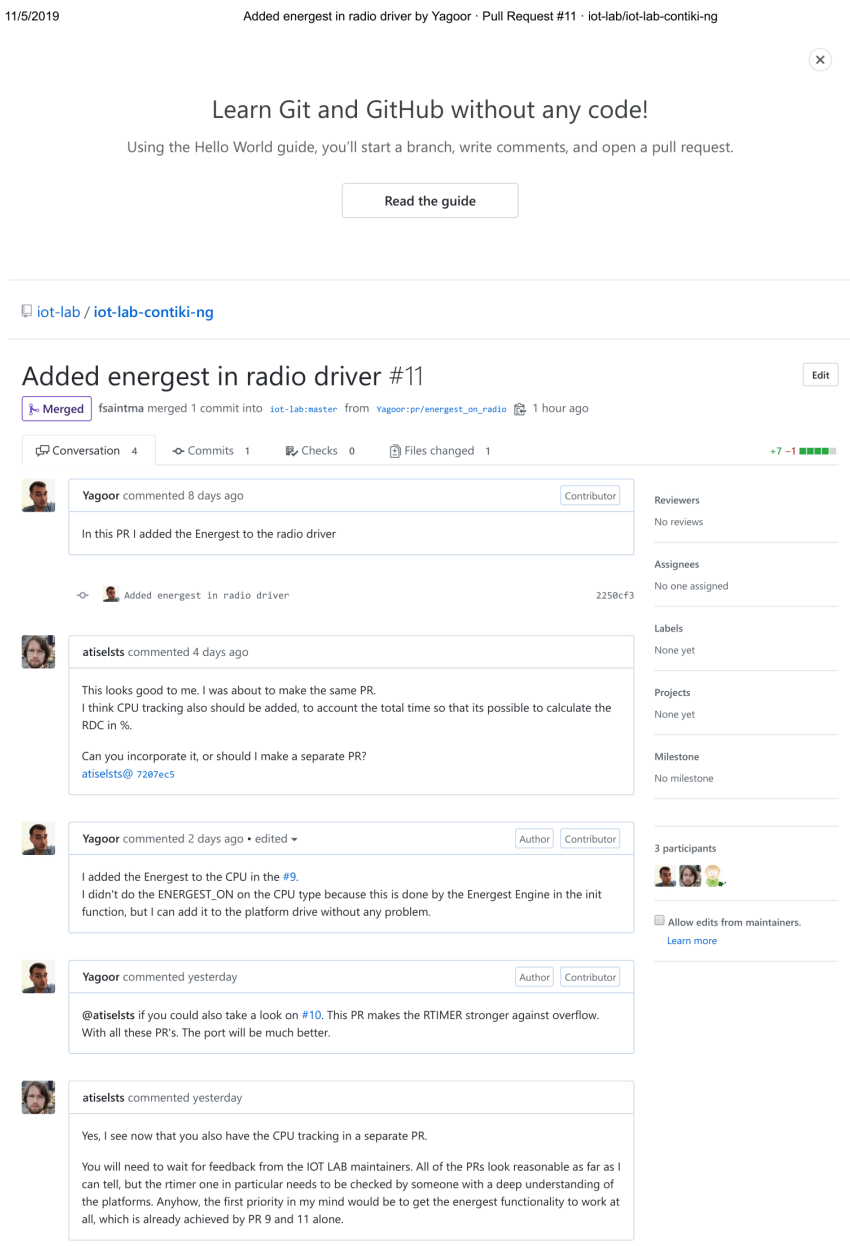
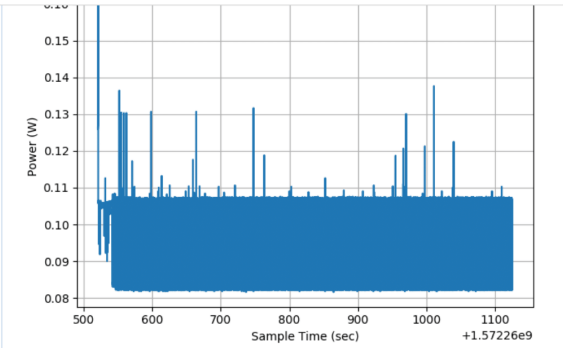


Figure B.6: Added energest in Radio Driver - Page 1.



In this figure we have the platform_idle implemented.

Implemented platform_idle fb4734c

Yagoor referenced this pull request 2 days ago

Added energest in radio driver #11 Merged

Added Energest ON to init d6dda8f

fsaintma merged commit 9bf9c02 into iot-lab:master 1 hour ago Revert

Figure B.7: Added energest in Radio Driver - Page 2.

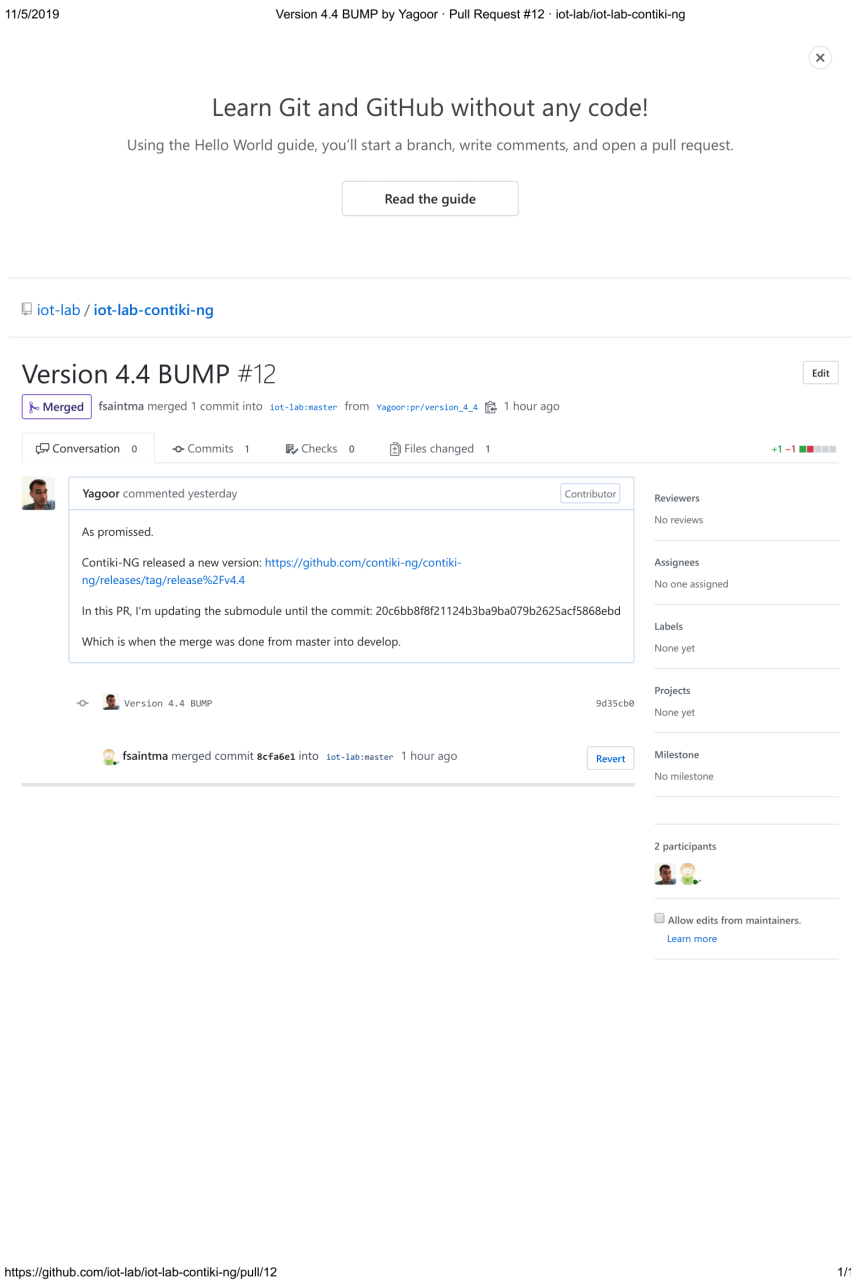


Figure B.8: Version 4.4 Bump - Page 1.

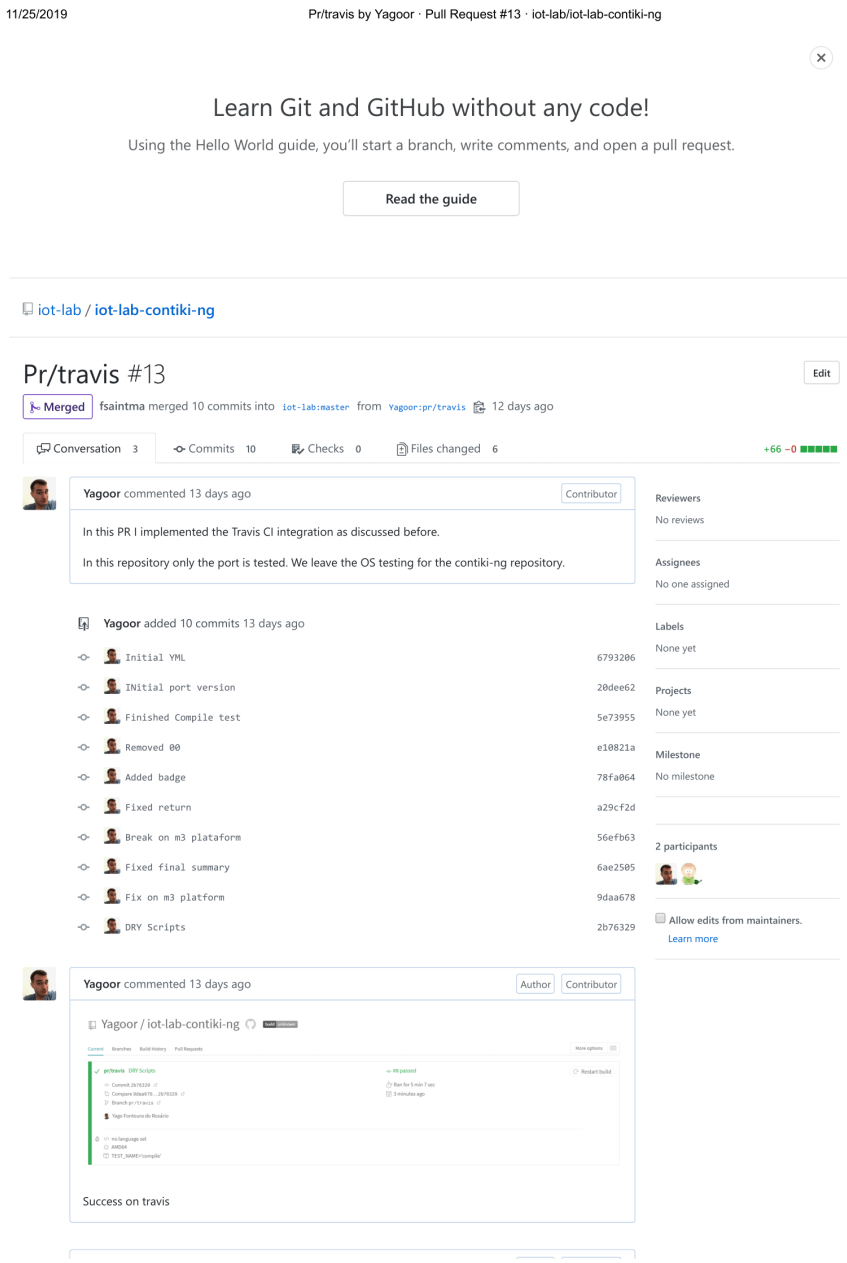


Figure B.9: Travis CI - Page 1.

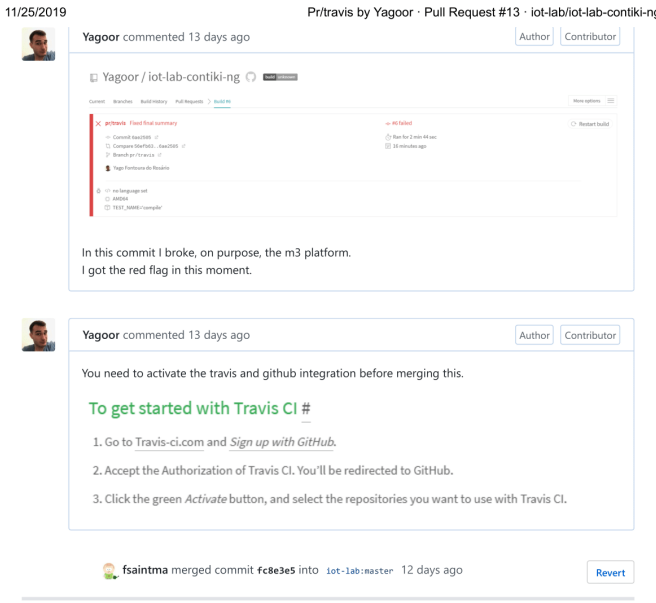


Figure B.10: Travis CI - Page 2.

Bibliography

[05] "IEEE Standard for Local and metropolitan area networks – Station and Media Access Control Connectivity Discovery". In: *IEEE Std 802.1AB-2005* (May 2005), pages 1–176. DOI: 10.1109/IEEESTD.2005.96285.

[16] "IEEE Standard for Low-Rate Wireless Networks". In: *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)* (April 2016), pages 1–709. DOI: 10.1109/IEEESTD.2016.7460875.

[87] *Information Processing Systems – Open Systems Interconnection – Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)*. Standard. International Organization for Standardization, October 1987.

[94] *Information Technology – Open Systems Interconnection – Presentation Service Definition*. Standard. International Organization for Standardization, October 1994.

[Acc+11] N. Accettura et al. "Performance analysis of the RPL Routing Protocol". eng. In: *2011 Ieee International Conference on Mechatronics, Icm 2011 - Proceedings* (2011), pages 5971218, 767–772. DOI: 10.1109/ICMECH.2011.5971218.

[Adj+15] Cedric Adjih et al. "FIT IoT-LAB: A large scale open experimental IoT testbed". eng. In: *Ieee World Forum on Internet of Things, Wf-iot 2015 - Proceedings* (2015), pages 7389098, 459–464. DOI: 10.1109/WF-IoT.2015.7389098.

[AHK19] Shimaa A. Abdel Hakeem, Anar A. Hady, and HyungWon Kim. "RPL Routing Protocol Performance in Smart Grid Applications Based Wireless Sensors: Experimental and Simulated Analysis". In: *Electronics* 8.2 (2019). ISSN: 2079-9292. DOI: 10.3390/electronics8020186. URL: https://www.mdpi.com/2079-9292/8/2/186.

[AlS+17] Salman M. Al-Shehri et al. "Common Metrics for Analyzing, Developing and Managing Telecommunication Networks". In: *CoRR* abs/1707.03290 (2017). arXiv: 1707.03290. URL: http://arxiv.org/abs/1707.03290.

[Ayo+19] Wael Ayoub et al. “Implementation of SCHC in NS-3 Simulator and Comparison with 6LoWPAN”. In: *26th International Conference on Telecommunications (ICT)*. HANOI, Vietnam, April 2019. URL: https://hal.archives-ouvertes.fr/hal-02051757.

[BCS12] C. Bormann, A. P. Castellani, and Z. Shelby. “CoAP: An Application Protocol for Billions of Tiny Internet Nodes”. In: *IEEE Internet Computing* 16.2 (March 2012), pages 62–67. ISSN: 1089-7801. DOI: 10.1109/MIC.2012.29.

[BEK14] C. Bormann, M. Ersue, and A. Keranen. *Terminology for Constrained-Node Networks*. RFC 7228. http://www.rfc-editor.org/rfc/rfc7228.txt. RFC Editor, May 2014. URL: http://www.rfc-editor.org/rfc/rfc7228.txt.

[BFM05] Tim Berners-Lee, Roy T. Fielding, and Larry Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. STD 66. http://www.rfc-editor.org/rfc/rfc3986.txt. RFC Editor, January 2005. URL: http://www.rfc-editor.org/rfc/rfc3986.txt.

[BH13] C. Bormann and P. Hoffman. *Concise Binary Object Representation (CBOR)*. RFC 7049. RFC Editor, October 2013.

[BJ00] A. Bierman and K. Jones. *Physical Topology MIB*. RFC 2922. http://www.rfc-editor.org/rfc/rfc2922.txt. RFC Editor, September 2000. URL: http://www.rfc-editor.org/rfc/rfc2922.txt.

[BL17] Carsten Bormann and Sean Leonard. *Concise Binary Object Representation (CBOR) Tags and Techniques for Object Identifiers, UUIDs, Enumerations, Binary Entities, Regular Expressions, and Sets*. Internet-Draft draft-bormann-cbor-tags-oid-06. Work in Progress. Internet Engineering Task Force, March 2017. 34 pages. URL: https://datatracker.ietf.org/doc/html/draft-bormann-cbor-tags-oid-06.

[Bor19] Carsten Bormann. *Concise Binary Object Representation (CBOR) Tags for Typed Arrays*. Internet-Draft draft-ietf-cbor-array-tags-08. Work in Progress. Internet Engineering Task Force, October 2019. 15 pages. URL: https://datatracker.ietf.org/doc/html/draft-ietf-cbor-array-tags-08.

[BSP16] Rahul Bhalerao, Sridhar Srinivasa Subramanian, and Joseph Pasquale. “An analysis and improvement of congestion control in the CoAP Internet-of-Things protocol”. eng. In: *2016 13th Ieee Annual Consumer Communications and Networking Conference, Ccnc 2016* (2016), pages 7444906, 889–894. ISSN: 23319860. DOI: 10.1109/CCNC.2016.7444906.

[BW11] For Bea Tuxedo and Bea Weblogic Enterprise. “BEA ™ SNMP Agent MIB Reference”. eng. In: (2011). DOI: 10.1.1.197.4199.

[Car16] Julia Dias Carneiro. “Brazil dam burst: Six months on, the marks left by sea of sludge”. en-GB. In: (May 2016). URL: https://www.bbc.com/news/world-latin-america-36230578.

[Cas+88] J. Case et al. *Simple Network Management Protocol*. RFC 1067. http://www.rfc-editor.org/rfc/rfc1067.txt. RFC Editor, August 1988. URL: http://www.rfc-editor.org/rfc/rfc1067.txt.

[Cha+10] S. A. Chaudhry et al. “EMP: A Network Management Protocol for IP-based Wireless Sensor Networks”. In: *2010 International Conference on Wireless and Ubiquitous Systems*. October 2010, pages 1–6. DOI: 10.1109/ICWUS.2010.5670440.

[CKC09] H. Choi, N. Kim, and H. Cha. “6LoWPAN-SNMP: Simple Network Management Protocol for 6LoWPAN”. In: *2009 11th IEEE International Conference on High Performance Computing and Communications*. June 2009, pages 305–313. DOI: 10.1109/HPCC.2009.49.

[Cru+19] Mauro A.A. da Cruz et al. “A proposal for bridging application layer protocols to HTTP on IoT solutions”. eng. In: *Future Generation Computer Systems* 97 (2019), pages 145–152. ISSN: 18727115, 0167739x. DOI: 10.1016/j.future.2019.02.009.

[Das+15] Supratim Das et al. “A Low Overhead Dynamic Memory Management System for Constrained Memory Embedded Systems”. eng. In: *2015 2nd International Conference on Computing for Sustainable Global Development (indiacom)* (2015), pages 809–815.

[DEV17] Simon Duquennoy, Joakim Eriksson, and Thiemo Voigt. “Five-Nines Reliable Downward Routing in RPL”. In: *CoRR* abs/1710.02324 (2017). arXiv: 1710.02324. URL: http://arxiv.org/abs/1710.02324.

[DGV04] A Dunkels, B Gronvall, and T Voigt. “Contiki - a lightweight and flexible operating system for tiny networked sensors”. eng. In: *Conference on Local Computer Networks* (2004), pages 455–462. ISSN: 07421303.

[DP10] Waltenegus Dargie and Christian Poellabauer. *Fundamentals of Wireless Sensor Networks: Theory and Practice*. English. 1 edition. Chichester, West Sussex, U.K. ; Hoboken, NJ: Wiley, August 2010. ISBN: 978-0-470-99765-9.

[Duq+15] Simon Duquennoy et al. “Orchestra: Robust mesh networks through autonomously scheduled TSCH”. eng. In: *Sensys 2015 - Proceedings of the 13th Acm Conference on Embedded Networked Sensor Systems* (2015), pages 337–350. DOI: 10.1145/2809695.2809714.

[Duq+18] Simon Duquennoy et al. “TSCH and 6TiSCH for contiki: Challenges, design and evaluation”. eng. In: *Proceedings - 2017 13th International Conference on Distributed Computing in Sensor Systems, Dcoss 2017* 2018- (2018), pages 11–18. ISSN: 23252936, 23252944. DOI: 10.1109/DCOSS.2017.29.

[Enn+11] R. Enns et al. *Network Configuration Protocol (NETCONF)*. RFC 6241. http://www.rfc-editor.org/rfc/rfc6241.txt. RFC Editor, June 2011. URL: http://www.rfc-editor.org/rfc/rfc6241.txt.

[Eri+15] Nicaise Eric et al. "Analysis of TTL-based Cache Networks". eng. In: (2015). DOI: 10.1.1.696.7279.

[Ge+15] Wei Ge et al. "Implementation of multiple border routers for 6LoWPAN with ContikiOS". eng. In: *2015 International Conference on Information and Communications Technologies (ict 2015)* (2015), pages 1–6, 1–6. DOI: 10.1049/cp.2015.0221.

[GK12] Olfa Gaddour and Anis Koubâa. "RPL in a nutshell: A survey". In: *Computer Networks* 56.14 (2012), pages 3163–3178. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2012.06.016. URL: http://www.sciencedirect.com/science/article/pii/S1389128612002423.

[Ham+08] Hamid Mukhtar et al. "LNMP- Management architecture for IPv6 based low-power wireless Personal Area Networks (6LoWPAN)". In: *NOMS 2008 - 2008 IEEE Network Operations and Management Symposium*. April 2008, pages 417–424. DOI: 10.1109/NOMS.2008.4575163.

[Han+13] Anca Hangan et al. "A prototype for the remote monitoring of water parameters". eng. In: *Proceedings - 19th International Conference on Control Systems and Computer Science, Cscs 2013* (2013), pages 6569331, 634–639. ISSN: 23790482. DOI: 10.1109/CSCS.2013.41.

[Huh18] Jun-Ho Huh. "Reliable User Datagram Protocol as a Solution to Latencies in Network Games". eng. In: *Electronics (basel)* (2018). ISSN: 20799292.

[Hun] Mark Hung. "Gartner Insights on How to Lead in a Connected World". en. In: (), page 29.

[HW03] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional, 2003. ISBN: 9780321200686. URL: https://www.amazon.com/Enterprise-Integration-Patterns-Designing-Deploying/dp/0321200683?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0321200683.

[KMS07] N. Kushalnagar, G. Montenegro, and C. Schumacher. *IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals*. RFC 4919. http://www.rfc-editor.org/rfc/rfc4919.txt. RFC Editor, August 2007. URL: http://www.rfc-editor.org/rfc/rfc4919.txt.

[Kod+19] Ravi Kishore Kodali et al. “Implementation of Home Automation Using CoAP”. eng. In: *Ieee Region 10 Annual International Conference, Proceedings/tencon* 2018- (2019), pages 8650135, 1214–1218. ISSN: 21593450, 21593442. DOI: 10.1109/TENCON.2018.8650135.

[KS11] Siarhei Kuryla and Jürgen Schönwälder. “Evaluation of the Resource Requirements of SNMP Agents on Constrained Devices”. In: *Managing the Dynamics of Networks and Services*. Edited by Isabelle Chrisment et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pages 100–111. ISBN: 978-3-642-21484-4.

[LH14] Myriam Leggieri and Michael Hausenblas. “Interoperability of two RESTful protocols: HTTP and CoAP”. eng. In: *Rest: Advanced Research Topics and Practical Applications* 9781461492993 (2014), pages 27–49. DOI: 10.1007/978-1-4614-9299-3_3.

[LMT18] Zakaria Laaroussi, Roberto Morabito, and Tarik Taleb. “Service Provisioning in Vehicular Networks through Edge and Cloud: an Empirical Analysis”. eng. In: *2018 Ieee Conference on Standards for Communications and Networking (ieee Cscn)* (2018).

[Maz+13] Oleksiy Mazhelis et al. “Retrieving Monitoring and Accounting Information from Constrained Devices in Internet-of-Things Applications”. In: *Emerging Management Mechanisms for the Future Internet*. Edited by Guillaume Doyen et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pages 136–147. ISBN: 978-3-642-38998-6.

[MH17] I. Muhic and M. Hodzic. “Verification and validation of wireless sensor network energy consumption in internet of things”. eng. In: *Icat 2017 - 26th International Conference on Information, Communication and Automation Technologies, Proceedings* 2017- (2017), pages 1–6. DOI: 10.1109/ICAT.2017.8171642.

[Moc83] P. Mockapetris. *Domain names: Concepts and facilities*. RFC 882. http://www.rfc-editor.org/rfc/rfc882.txt. RFC Editor, November 1983. URL: http://www.rfc-editor.org/rfc/rfc882.txt.

[Mon+07] G. Montenegro et al. *Transmission of IPv6 Packets over IEEE 802.15.4 Networks*. RFC 4944. http://www.rfc-editor.org/rfc/rfc4944.txt. RFC Editor, September 2007. URL: http://www.rfc-editor.org/rfc/rfc4944.txt.

[MR88] K. McCloghrie and M.T. Rose. *Structure and identification of management information for TCP/IP-based internets*. RFC 1065. RFC Editor, August 1988.

[Ost+06] Fredrik Osterlind et al. “Cross-level sensor network simulation with COOJA”. eng. In: *Conference on Local Computer Networks* (2006), pages 641–648. ISSN: 07421303.

[PR83] J. Postel and J. Reynolds. *Telnet Protocol Specification*. STD 8. http://www.rfc-editor.org/rfc/rfc854.txt. RFC Editor, May 1983. URL: http://www.rfc-editor.org/rfc/rfc854.txt.

[Pre02] R. Presuhn. *Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)*. STD 62. http://www.rfc-editor.org/rfc/rfc3416.txt. RFC Editor, December 2002. URL: http://www.rfc-editor.org/rfc/rfc3416.txt.

[RM90] Marshall T. Rose and Keith McCloghrie. *Structure and identification of management information for TCP/IP-based internets*. STD 16. http://www.rfc-editor.org/rfc/rfc1155.txt. RFC Editor, May 1990. URL: http://www.rfc-editor.org/rfc/rfc1155.txt.

[Sat90] Greg Satz. "The Simple Book: An Introduction to Management of TCP/IP-based Internets". In: *SIGCOMM Comput. Commun. Rev.* 20.5 (October 1990), pages 6–7. ISSN: 0146-4833. DOI: 10.1145/381906.381911. URL: http://doi.acm.org.proxy.findit.dtu.dk/10.1145/381906.381911.

[Sch01] Jürgen Schönwälder. *SNMP Payload Compression*. Internet-Draft draft-irtf-nmrg-snmp-compression-01. Work in Progress. Internet Engineering Task Force, April 2001. 15 pages. URL: https://datatracker.ietf.org/doc/html/draft-irtf-nmrg-snmp-compression-01.

[Sed+18] Mohamed-El-Amine Seddik et al. "From Outage Probability to ALOHA MAC Layer Performance Analysis in Distributed WSNs". eng. In: *2018 Ieee Wireless Communications and Networking Conference (wcnc)* (2018). ISSN: 15582612,

[SHB14] Z. Shelby, K. Hartke, and C. Bormann. *The Constrained Application Protocol (CoAP)*. RFC 7252. http://www.rfc-editor.org/rfc/rfc7252.txt. RFC Editor, June 2014. URL: http://www.rfc-editor.org/rfc/rfc7252.txt.

[Tea16] TeamNANOG. *Tutorial: Tutorial NETCONF and YANG*. June 2016. URL: https://www.youtube.com/watch?v=FsamNrl7GkY.

[TTC04] Jen Hao Teng, Chin Yuan Tseng, and Yu Hung Chen. "Integration of networked embedded systems into power equipment remote control and monitoring". eng. In: *Ieee Region 10 Annual International Conference, Proceedings/tencon* C (2004), pages C566–C569, C566–C569. DOI: 10.1109/TENCON.2004.1414834.

[TVM15] G. Tanganelli, C. Vallati, and E. Mingozzi. "CoAPthon: Easy Development of CoAP-based IoT Applications with Python". eng. In: *2015 Ieee 2nd World Forum on Internet of Things (wf-iot)* (2015), pages 63–68.

[VM19] D. Valencic and V. Mateljan. “Implementation of NETCONF Protocol”. eng. In: *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (mipro)* (2019), pages 421–430, 421–430. ISSN: 26238764. DOI: 10.23919/MIPRO.2019.8756925.

[Win+12] T. Winter et al. *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*. RFC 6550. http://www.rfc-editor.org/rfc/rfc6550.txt. RFC Editor, March 2012. URL: http://www.rfc-editor.org/rfc/rfc6550.txt.

[WPG15] T. Watteyne, M. Palattella, and L. Grieco. *Using IEEE 802.15.4e Time-Slotted Channel Hopping (TSCH) in the Internet of Things (IoT): Problem Statement*. RFC 7554. http://www.rfc-editor.org/rfc/rfc7554.txt. RFC Editor, May 2015. URL: http://www.rfc-editor.org/rfc/rfc7554.txt.

[WRV13] Linus Wallgren, Shahid Raza, and Thiemo Voigt. *Routing Attacks and Countermeasures in the RPL-Based Internet of Things*. 2013.

[YL06] T. Ylonen and C. Lonvick. *The Secure Shell (SSH) Protocol Architecture*. RFC 4251. http://www.rfc-editor.org/rfc/rfc4251.txt. RFC Editor, January 2006. URL: http://www.rfc-editor.org/rfc/rfc4251.txt.

{2.7182818284}ϑφερτυθιοπσδφγηξκλ