A Constrained Monitoring Protocol for the Internet of Things

do Rosario, Yago Fontoura ; Fafoutis, Xenofon

Published in:
Journal of Signal Processing Systems

Link to article, DOI:
10.1007/s11265-021-01658-y
https://rdcu.be/ciFVj

Publication date:
2021

Document Version
Peer reviewed version

Noname manuscript No.
(will be inserted by the editor)

A Constrained Monitoring Protocol for the Internet of Things

Yago Fontoura do Rosário · Xenofon Fafoutis

Received: date / Accepted: date

Abstract Network monitoring has been traditionally conducted using Simple Network Management Protocol (SNMP): a network monitoring protocol that allows network administrators to keep track of every node in the network and ensure that it behaves correctly. This paper presents the Constrained Monitored Protocol (CoMP): a lightweight resource-efficient alternative to SNMP that targets the low-end devices of the Internet of Things (IoT). These devices are characterised by severe resource constraints in terms of memory, processing power, and bandwidth. Moreover, they are often energy-constrained as well, powered either by small batteries or energy harvesting. While SNMP does work with these devices, it has an unnecessary overhead resulting in a waste of resources that could otherwise be used for some other task, or to save energy. Furthermore, this paper proposes a cross-protocol CoMP-SNMP proxy that operates at the border router of the resource-constrained network and enables the efficient monitoring of resource-constrained IoT devices using CoMP from existing SNMP-based network monitoring infrastructures.

Keywords Network Monitoring · Wireless Sensor Networks · Resource-Constrained Devices · Internet of Things

1 Introduction

With the emerging Internet of Things (IoT) billions connected devices support a wide variety applications in the industry and in the society in general. Managing such a high volume of devices is a challenge. To make matters worse,

This is a post-peer-review, pre-copyedit version of an article published in Journal of Signal Processing Systems. The final authenticated version is available online at: `http://dx.doi.org/10.1007/s11265-021-01658-y`.

Yago Fontoura do Rosário · Xenofon Fafoutis
Technical University of Denmark, Denmark
E-mail: yago.rosario@hotmail.com.br, xefa@dtu.dk

the edge of the IoT is composed of networks of severely resource-constrained sensors, namely Wireless Sensor Networks (WSN). Monitoring these resource-constrained IoT device is of critical importance, particularly when the network supports safety critical applications, such as, for example, a train scheduling and signaling system. A simple failure in such networks can cause anything from mild degradation in user experience (e.g. train delay) to catastrophic outcomes (e.g. train crash).

The standard protocol for network monitoring is the Simple Network Management Protocol (SNMP), which was developed in the 1980s. The main issue with SNMP is that it was not designed for resource-constrained networks, such as the low-end wireless sensor networks of the IoT. Whilst SNMP does work on resource-constrained IoT devices, it imposes unnecessary overhead that degrades their performance. For example, it uses a relatively old and heavy encoding technique where to encode the number 4 would be 02_{16}, 01_{16}, 04_{16}. It also reserves an entire 32-bit integer to address the nodes, which is not sensible in WSN. Instead, there is a need for a lightweight alternative to SNMP that is suitable for low-power IoT networks.

To that end, we introduce the Constrained Monitoring Protocol (CoMP). CoMP provides the same functionality as the current state-of-the-art monitoring protocols, yet in a much more resource-efficient manner. We have implemented CoMP for the Contiki-NG[1] operating system and compared it against monitoring standards, including SNMP [9] and CoAP (Constrained Application Protocol) [35]. Specifically, we compare the protocols in terms of (i) code footprint, (ii) application-layer overhead, (iii) over-the-air bytes, (iv) radio energy usage, (v) delay, and (vi) jitter. An early version of CoMP appears in [13]. This paper extends [13] with a cross-protocol proxy for CoMP and SNMP that runs at the border router of the sensor network. This extension substantially improves the applicability of CoMP in existing networks. In practical terms, the proxy enables existing monitoring applications that use SNMP to efficiently collect data from constrained nodes that use CoMP. Moreover, it enables the monitoring of hybrid IoT networks that consist of resource-constrained 6LoWPAN networks and traditional unconstrained IPv4/IPv6 networks. Additionally, two new experiments were done to extend the evaluation of CoMP. In the first experiment, we count the transmitted frames and fragments and demonstrate the impact of 6LoWPAN fragmentation. In the second experiment, we measure the delay and jitter. These additional experiments confirm that CoMP uses less resources than SNMP and CoAP.

The rest of this paper is structured as follows. Section 2 presents some related work on monitoring WSNs. Section 3 briefly introduces SNMP. Section 4 presents our proposed monitoring protocol CoMP. Section 5 presents the CoMP-SNMP border router proxy. Section 6 provides experiments that evaluate and compare CoMP against SNMP and CoAP. The same section also evaluates the border router proxy. Finally, Section 7 concludes the paper.

[1] `www.contiki-ng.org`

2 Related Work

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 [29] a new protocol for monitoring IoT 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 IoT devices 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 report to a coordinator their 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 network. However, its table is available in a 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 proposed improvement is 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 CoMP, the OID system is kept but compression is applied to it. However, it does not address the Network Discovery problem because this can be solved using the Physical Topology MIB proposed on [4]. This MIB was designed to be used with the Link Layer Discovery Protocol (LLDP) which was proposed on [38]. But this can also be filled in with the RPL Neighbors Table information.

On [11], the SNMP 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 a 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 BER (Basic Encoding Rules) 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. CoMP uses a different encoding which targets both key requirements for constrained devices, namely the code footprint and and the encoded buffer size. Also, an OID compression technique is used.

On [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 system description (`sysDescr`). In this work, the Network Discovery problem is not tackled, but to solve the Monitoring problem better techniques to reduce the Network usage are used and the OID is kept, no new MIBs are introduced.

On [41], a new architecture is proposed using Blockchain and Fog Computing which will deliver a more stable and secure monitoring system. Defining an entire architecture allows for a more target solution where it is possible to define all the layers accordingly to the needs. It can be both good and bad since the solution will be designed to work in the use-cases it was thought for, however, an extremely optimized solution will make it less reusable. Additionally, this comes with a heavy price. The first downside is the high complexity since it requires sensor nodes, gateway nodes, aggregator nodes, sink nodes, and the Blockchain network as main components. Another caveat is that this solution is an entire architecture that imposes the challenge of an entirely new infrastructure being required. CoMP doesn't propose a new architecture instead it's designed to work with the current architectures available at the moment. It's planned to work in any IP-based network, 6LoWPAN or not. The outline targets IoT networks but it also carefully ensures seamless integration with already deployed monitoring systems to reduce cost and complexity.

On [42], another protocol for monitoring IoT devices was proposed and also based on SNMP. However, it was done as a simple prototype to show that it's possible to use the same model used by SNMP to also monitor constrained devices. It did not define key elements of the proposed protocol such as packet structure, encoding, resource identification, monitored values. It did show that

by changing the SNMP and making it more compact yields less traffic, CPU and memory usage. CoMP is proposed in more details where all elements were defined and the design decisions were justified. Additionally, a proxy between SNMP and CoMP is also introduced which has the goal of allowing a seamless integration of CoMP into current monitoring system that support SNMP.

On [24], a completely different solution was proposed where the resources that are allocated and are not used to it's fully extension are used to transmit monitoring data. An In-Band Network Telemetry (INT) proposal designed for WSN is described. It leverages the 6TiSCH architectural flexibility. It delivered very good results in term of performance since it uses a resource that would we lost. It enabled the network to be used to it's truly full extent. *However, It can be considered a layer violation because the MAC layer is being used to control the transmission of application data.* Additionally, If an application is using all or almost all network resources, INT will deliver poor results because it relies on non used resources. Furthermore, in it's current state, it's a TSCH tailed solution which can make integration with current monitoring solutions harder.

On [18], multiple use cases of constrained devices networks monitoring are presented. Two examples proves how different these networks can be: First, it can be used for environmental monitoring, which due to it's deployment location and conditions are expected to have small failures. Therefore, in some cases, hours of failure are required in order to trigger a maintenance. Secondly, it can be used for medical monitoring, which can be a simple routine checkup metrics or critical metrics. In the first case, failures can be accepted and several minutes of failure would require intervention. However, on the second case, failures are likely not acceptable and a few seconds of non operation are enough to trigger maintenance.

On [37] and [36], the state-of-the-art protocols and frameworks that can be used for IoT monitoring were listed and compared. It also mentions several challenges that these face. First, taxonomy, there is no unified taxonomy therefore each protocol and framework defines its own. Additionally, the maturity is also another challenge, a newly proposed protocol needs to be deployed in real scenarios in order to mature and became more stable. Moreover, with maturity the tools and libraries will follow, which enables easy integration and deployment of those. The lack of agreement on the first, makes the second difficult because an agreed taxonomy will enable protocols and frameworks to target a specific groups which have similar requirements. Therefore, protocols can be proposed for a single group of IoT devices and not for all enabling a more targeted solution.

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 WSNs into existing ecosystems. It is fairly simple to use the same monitoring system, like PRTG[2], that currently monitors all networks of a ISP to additionally mon-

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

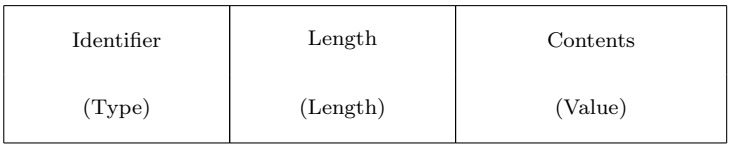


Fig. 1: BER Encoding Structure

itor the WSN that this company has. In this work a new protocol is proposed, CoMP, which is a standalone protocol, like CoAP.

3 Background: SNMP

3.1 Encoding

The SNMP only uses a subset of the ASN.1 which was proposed in 1987 on [22]. 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 1 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.

3.2 Messaging Pattern, Resource Identification and Monitored Values

SNMP's main operations follow a 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.

SNMP uses the OID, proposed in 1994 on [23], 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.

The SNMP uses the MIB, proposed in 1988 on [27], as a database for its entities. This hierarchical organization is great for huge environments like the internet. 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.

3.3 Operations

SNMP supports several operations: `Get`, `Get-Next`, `Get-Bulk`, and `Set`. The `Set` operation 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.

3.4 Packet Structure

On Figure 2 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 PDU. Additionally, the SNMPv3 has a completely different packet structure, Figure 3. 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 4 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`,

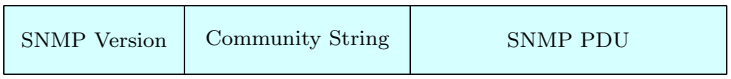


Fig. 2: SNMPv1 & SNMPv2 Packet Structure

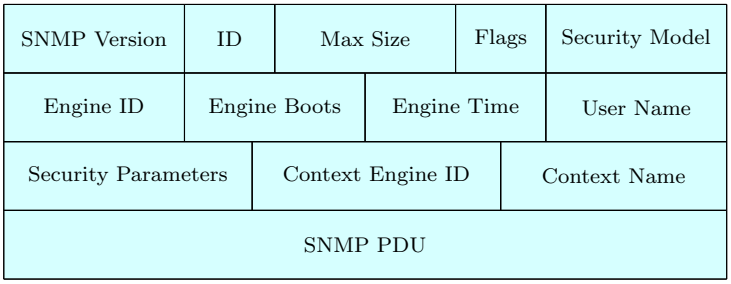


Fig. 3: SNMPv3 Packet Structure

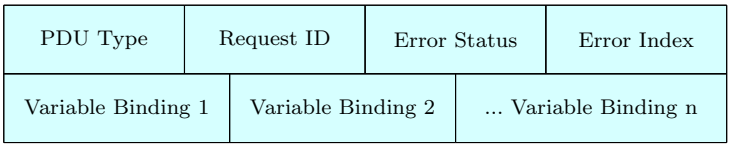


Fig. 4: SNMP PDU Structure

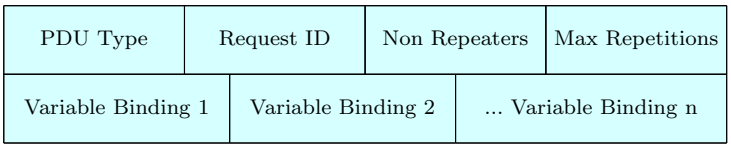


Fig. 5: SNMP `Get-Bulk` PDU Structure

`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 5 the SNMP PDU for `Get-Next` 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.

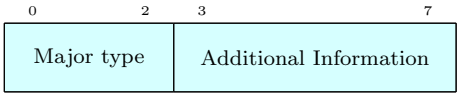


Fig. 6: CBOR tiny field encoding.

4 Constrained Monitoring Protocol (CoMP)

In this section, we introduce CoMP. Its goal is to be a very lightweight monitoring protocol to be used in a 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 (*i.e.* `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.

In this regard, CoMP offers resource-efficient equivalents to SNMPv1 and SNMPv2 [39], which are referred to as CoMPv1 and CoMPv2 respectively.

4.1 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 CBOR (Concise Binary Object Representation) encoding. Proposed on [7] 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, 0111_2. The additional information meaning varies, from 0, 00000_2, to 23, 10111_2, it is used as the byte count composing a short field encoding, Figure 7, where the byte count is used to know the length of the value. If 24, 11000_2, the next byte is an unsigned 8-bit integer with the byte count, thus composing a long field encoding, Figure 8. If 25, 11001_2, the next 2 bytes is an unsigned 16-bit integer with the byte count, also a long field encoding. If 26, 11010_2, the next 4 bytes is an unsigned 32-bit integer with the byte count, likewise a long field encoding. If 27, 11011_2, 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_2, it means that this is an indefinite string, each byte has to be read until a *Break*, 0xFF, 11111111_2, is encountered, therefore this composes a tiny field encoding, Figure 6.

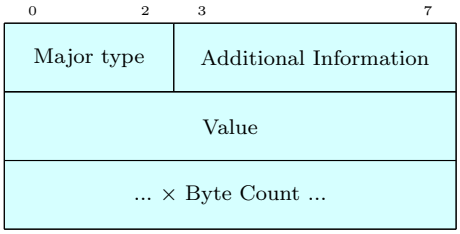


Fig. 7: CBOR short field encoding.

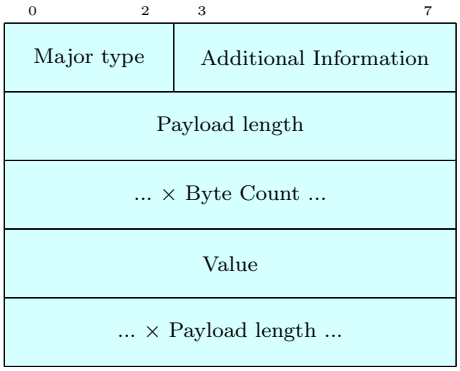


Fig. 8: CBOR long field encoding.

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 each item of the array can have a different type. In the OID scenario, all items inside the array have the same type. On [5] 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 [8] 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 [33] 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.

4.2 Messaging Pattern, Resource Identification and Monitored Values

Similarly to SNMP and CoAP, CoMP supports the Request–Response message pattern.

Since the CoMP is designed to be SNMP compliant, it employs the same resource identification technique. Even though this constitutes protocol over-

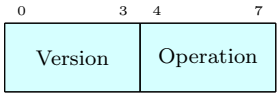


Fig. 9: CoMPv1 and CoMPv2 header structure.

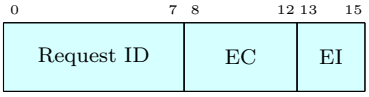


Fig. 10: CoMPv1 and CoMPv2, `Get` and `Get-Next` extra header.

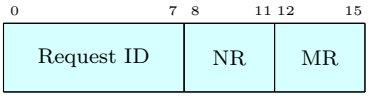


Fig. 11: CoMPv2, `Get-Bulk` extra header.

head, the OID is very good for large networks, 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 in WSN context.

The monitored 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.

4.3 Packet Structure

The CoMP packet is defined by a header, an extra header, and a PDU (Protocol Data Unit). The header is shared by CoMPv1 and CoMPv2, shown in Figure 9. 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.

If the operation is a `Get` or `Get-Next` the extra header will look like in Figure 10, which has a Request ID (unsigned 8-bit integer), followed by the Error Code (unsigned 5-bit integer), and lastly the Error Index (unsigned 3-bit integer).

However, if the version is CoMPv2 and the operation is the `Get-Bulk` a different extra header is used (Figure 11). The first element is a Request ID (unsigned 8-bit integer), yet it is followed by the Non-Repeaters (unsigned 4-bit integer), and finally the Max Repeaters (unsigned 4-bit integer).

Depending on the operation the CoMP PDU carries different information. In any request, `Get`, `Get-Next` or `Get-Bulk`, the PDU is a list of OIDs that the client is requesting the server, as shown in Figure 12.

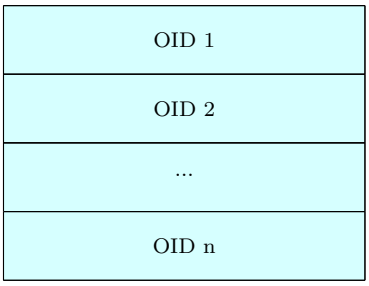


Fig. 12: CoMP PDU variables on request packet.

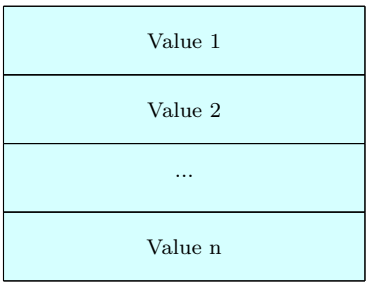


Fig. 13: CoMP PDU values on response packet.

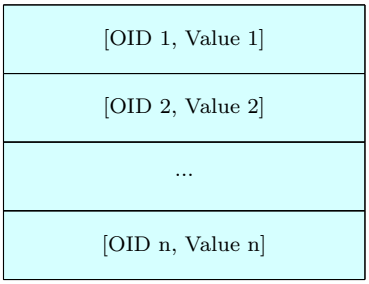


Fig. 14: CoMP PDU variable names and values on response packet.

If the CoMP operation is a `Get`, the response is a list of Values, as shown in Figure 13. This list is indexed using the same position as the OID in the request. For example, if the client requests the 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]`.

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, as shown in Figure 14.

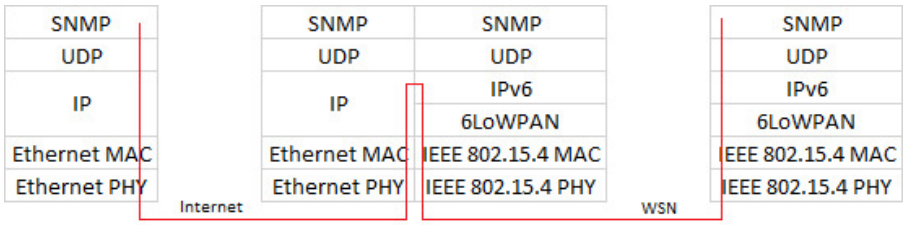


Fig. 15: Normal Border Router Protocol Stack

5 Border Router Proxy

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 [26]. The opposite is not always possible, since CoAP has the Observer extension, for example. As proposed in [12], 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 [21], common border router implementations only open the packet until the Network Layer to convert it, Figure 15. 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.

A border router that can unpack the packet until its last layer (Application layer) enables the application protocol to be converted 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. According to [25] 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

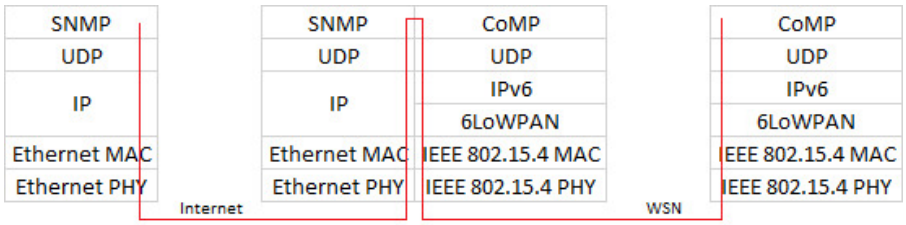


Fig. 16: Application Border Router Protocol Stack

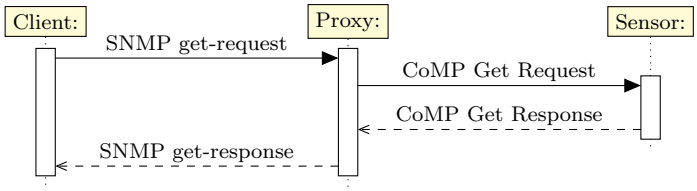


Fig. 17: Sequence diagram of SNMP to CoMP Cross-Protocol Proxy

protocol itself, the CoAP was remodeled to have a more concise header. In [26] it is explained that CoAP contains a subset of the HTTP methods, the ones that are compliant with the 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.

5.1 A CoMP-SNMP Cross-Protocol Proxy

The same concept of cross-protocol proxying proposed for CoAP and HTTP can be applied for CoMP and SNMP, Figure 16. 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 (*i.e.* a general failure occurred), when an operation that cannot be proxied is sent to a 6LoWPAN node.

This technique would allow several applications that already use SNMP to collect monitoring data, 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. In Figure 17 a sequence diagram better illustrates the idea.

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
IpAddress	cbor-network
Counter	Unsigned Integer
Gauge	Unsigned Integer
TimeTicks	Unsigned Integer
Opaque	Map of Pairs of Data Items

Table 1: SNMP to CoMP Types

5.2 Types

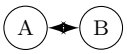
In this proxying all the types defined in the RFC1155 [31] are handled as shown in Table 1. All types have a corresponding type defined in the RFC8949 [7]. The only exceptions are the `NetworkAddress` and `IpAddress`, however, there is a reserved CBOR Tag[6] that covers these types.

The challenge faced here is the ambiguity, which can be seen in Table 1 where multiple SNMP types are mapped to the same CoMP type. An SNMP request has an 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. For example, an SNMP `Get` request on the `sysUpTime` OID will contain the OID and the value `Null`. The proxy will forward the OID only and will receive an unsigned integer in the response. Now the proxy does not know which SNMP type is the correct, there are multiple mappings (`Counter`, `Gauge`, `TimeTicks`). 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.

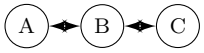
5.3 Caching

SNMP does not provide a standardized caching mechanism therefore the same process responsible for proxying SNMP-CoMP can also perform caching. In the initial proposal only caching static was considered, values which should already reduce the network bandwidth. Some monitored values are static during a life cycle, they can only change in a system reboot. A good example is the `sysDescr`[3], `1.3.6.1.2.1.1.1`, which is a textual description of the network entity. 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

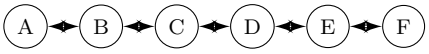
[3] `http://oidref.com/1.3.6.1.2.1.1.1`



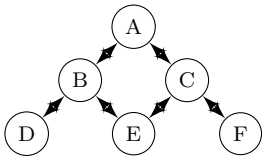
(a) **Topology 1:** One server (B) and one border router (A).



(b) **Topology 2:** Two servers (B, C) and one border router (A).



(c) **Topology 3:** Five servers (B-F) and one border router (A) in a line topology.



(d) **Topology 4:** Five servers (B-F) and one border router (A) in a tree topology.

Fig. 18: Topologies simulated in Cooja.

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 [20] 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, further study is necessary to determine which caching mechanism works better in the use cases of a WSN. If CoMP is used to extract live sensor readings, a caching mechanism could affect the metrics quality.

6 Evaluation

6.1 Experiments

We conduct three experiments to evaluate CoMP and compare it against SNMP and CoAP. The experiments are based on a proof-of-concept implementation of CoMP for Contiki-NG[4]. For the purposes of this work, we also de-

[4] `https://github.com/Yagoor/contiki-ng/tree/comp`

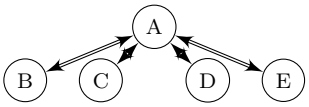


Fig. 19: FIT IoT-LAB Topology: four servers (B-E) and one border router (A). The DODAG was built with a directly link from the border router to the servers

	RAM	**ROM**	**Total**
SNMP	1120 (+1.82%)	3140 (-2.67%)	4260 (-1.29%)
CoAP	1986 (+80.55%)	9724 (+202%)	11710 (+173%)
CoMP	1100	3216	4316

Table 2: Code Footprint on cc26x0-cc13x0 [Bytes]

veloped and contributed to the Contiki-NG source code a standard-compliant implementation of SNMPv1 and SNMPv2[5].

SNMP and CoMP have a well defined way to store the monitored values and a defined resource identification method. However, CoAP is more generic. In order to make it suitable for this experiment a standardized way of mapping CoAP resources to monitored values was defined. 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 element, that was the same name used in the SNMP. If no query is sent to the endpoint a list with all the available monitored values are sent as a response, this method can be used to act similar to the SNMP Walk. Since the key argumentation to use CoAP as a monitoring protocol is because it is human-friendly, a very descriptive method was utilized and the JSON encoding was used.

The first experiment evaluates the code footprint (ROM and RAM) of each protocol. To this end, a compilation of a Contiki-NG server example for the *cc26x0-cc13x0* target was done. The goal is to see how much memory resources a module uses on a typical IoT device.

The second series of experiments assesses the network usage using the Cooja Simulator[30]. The Cooja simulations were used to analyze and evaluate many 6LoWPAN protocols like it was done for RPL [1] and for CoAP [3]. Figure 18 shows the four topologies that we considered. For the remainder of this section we refer to these four topologies as Topology 1-4 respectively. We evaluate the network usage in these scenarios for various reasons. The first one is the resource limitation, the least data is transferred the better. Moreover, 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

[5] `https://github.com/contiki-ng/contiki-ng/pull/1020`

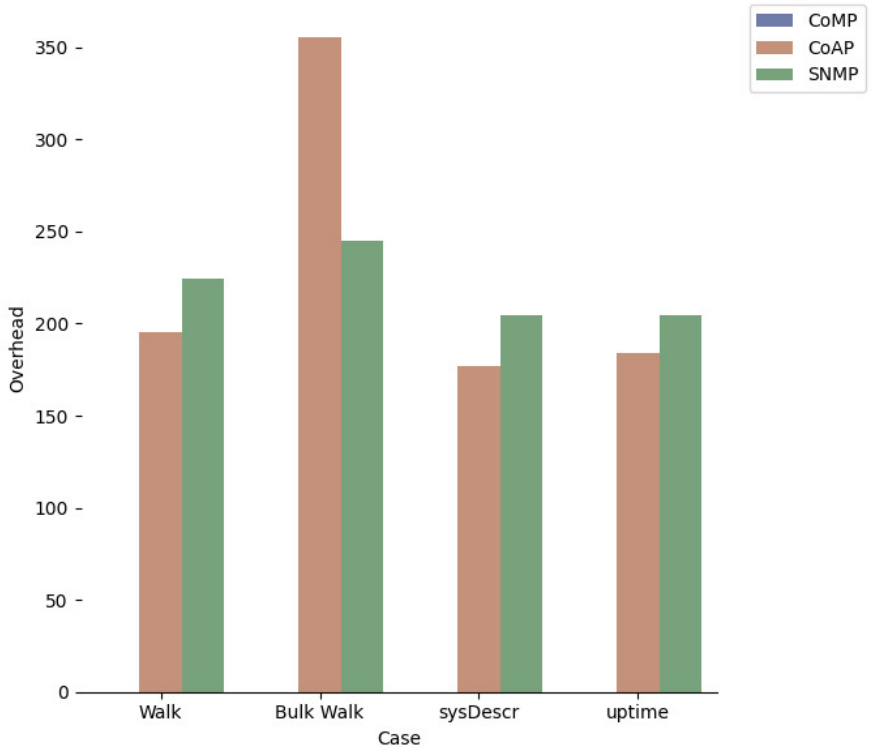


Fig. 20: Relative application-layer overhead of request packets relative to CoMP.

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 topology, four use cases are studied. Every node, except the border router, is requested by a client outside the WSN. First, a Walk where all the monitored values available on the server are requested one by one. Secondly, a Bulk Walk where all the monitored values are requests, two at a time. Only two were used because this increases the packet size and a balance has to be found between these two attributes. Thirdly, a system description (`sysDescr`) is asked, in this case, it is the operating system name and version tag. Lastly, the system `uptime` is asked. The two last cases are used to analyze how much data each protocol uses to transmit a string and an integer.

The third experiment evaluates the radio activity time of each protocol in a test-bed environment. For these experiments, the TSCH (Time-Slotted, Channel Hopping) protocol is used because it has a good overall performance in terms of energy and reliability [16,17]. The scheduler used is the Orchestra [14]. The RPL's and TSCH's probing interval was tuned using the five

Table 3: Application Layer Network Usage [Bytes]

	Walk		Bulk Walk		sysDescr		uptime	
	RES	**REQ**	**RES**	**REQ**	**RES**	**REQ**	**RES**	**REQ**
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

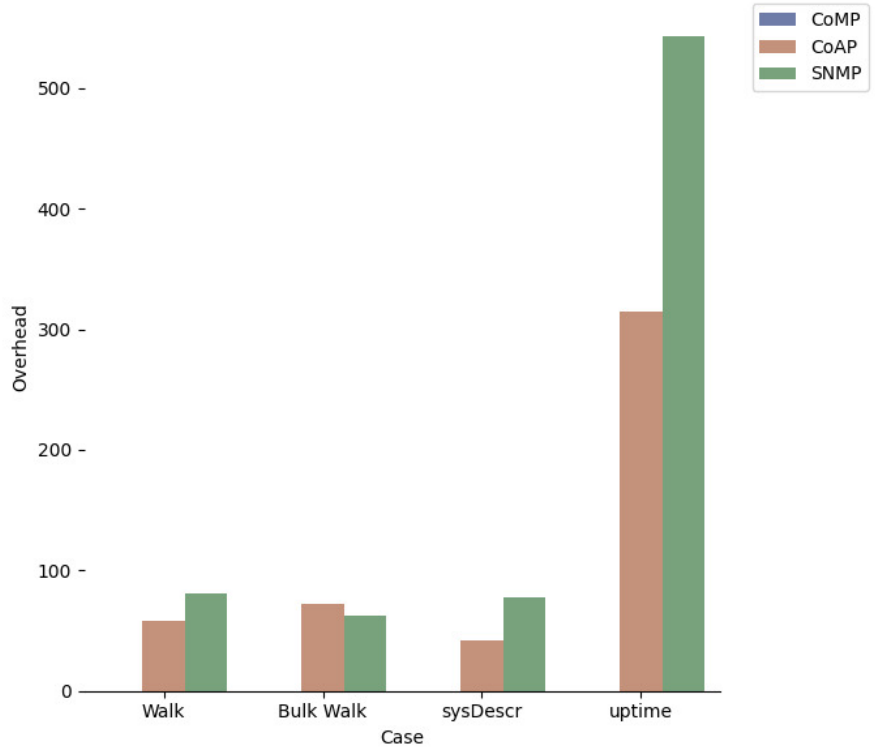


Fig. 21: Relative application-layer overhead of response packets relative to CoMP.

nines reliability proposed on [15], which reduces the impact of the beacons on the experiments. The experiment was conducted on the FIT IoT-LAB[2] located in Grenoble, France. This laboratory is composed of several micro-controllers of different architectures. 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 on [28] and [34]. 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. Energest has been used to track the energy consumption of IoT devices in real-world deployments [19]. In this evaluation, we focus on the radio duty cycle (Radio TX and Radio RX states). The decision to use of Energest instead of the FIT IoT-LAB current measurement was based on the high accuracy that Energest has [32]. It also provides empirical values that can be used to calculate current measurements in other boards, not only the M3 nodes.

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 19. In these tests it is mostly interesting to look into the Radio Activity Time during transmission (TX) because the receiving (RX) is controlled by the scheduler of the MAC protocol, therefore the only radio activity time where the application-layer protocol can influence positively or negatively is the TX cycle. Additionally, the 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 affect 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 activity time from the four nodes either in transmission and reception. Therefore in this experiment, every node, except the border router, is requested by a client outside the WSN.

6.2 Results

Table 2 summarizes the code footprint information of the three protocols. It is evident that CoAP performs the worst, whilst SNMP has a slightly better performance than CoMP on ROM usage. CoMP is slightly better than SNMP on RAM usage. The table also shows the overhead relative to CoMP. SNMP is a really simple protocol in terms of functionality which explains its extremely low ROM and RAM usage. In terms of functionality, CoMP has the same with the addition of the OID compression which explains the more ROM usage because two extra functions are needed for compression and decompression. The real improvement is on the network aspects where CoMP focuses on the payload size more than SNMP.

Next, we present the results of the Cooja simulations. Table 3 summarizes the application-layer network usage information of the three protocols in all four cases. In particular, the table reports the total number of application-

Table 4: Bytes over the air in Topology 1

	Walk		**Bulk Walk**		**sysDescr**		**uptime**	
	RES	**REQ**	**RES**	**REQ**	**RES**	**REQ**	**RES**	**REQ**
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

layer bytes transmitted in requests packets (REQ) and response packets (RES) respectively. It is clear that CoMP offers a significant improvement. Figure 20 and Figure 21 present the relative overhead of the request and response packets with respect to CoMP for each case. In all cases, CoMP performed much better than the other protocols in all cases.

Table 4 shows the total bytes transmitted over the air in request (REQ) and response (RES) packets respectively for Topology 1. Similarly, Table 5, Table 6 and Table 7 summarize the total bytes over the air for Topology 2, Topology 3 and Topology 4 respectively. Table 8 shows the total Fragments/Frames transmitted and received for Topology 1. If more fragments than frames were transmitted, it implies that 6LowPAN fragmentation occurred. Similarly, Table 9, Table 10 and Table 11 summarize the fragments and frames for Topology 2, Topology 3 and Topology 4 respectively. Table 12 shows the Jitter and Delay for Topology 1. Similarly, Table 13, Table 14 and Table 15 summarize the Jitter and Delay for Topology 2, Topology 3 and Topology 4 respectively. In all topologies and cases, CoMP outperforms the other protocols in terms of bytes over the air and Fragments/Frames. In terms of Jitter and Delay, CoMP also deliver better overall results, however, these two metrics are heavily influenced by the MAC protocol.

The biggest downside of CoAP is in Walk and Bulk Walk where it is first necessary to request a list of all monitored values that are available before starting the walk request. This disadvantage cannot be seen in Walk because on a traditional walk the client requests the next value 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 CoAP it asks for all monitored values available, so it knows the last one beforehand. The number of requests is the same because both have an extra request. But in Bulk Walk, the SNMP sends the *End of View* message together with the last value eliminating the extra request.

Finally, we summarize the results of the test-bed experiments in the FIT IoT-Lab in Table 16. In particular, the table shows the total radio activity time (in ms) for transmitting request and response packets respectively. Similarly, Figure 22 and Figure 23 illustrate the relative overhead of transmitting request and response packets with respect to the best performing protocol respectively. In all cases, CoMP consumes less energy than the other protocols. More specifically, SNMP slightly outperforms CoMP in the response packets

Table 5: Bytes over the air bytes in Topology 2

	Walk		Bulk Walk		sysDescr		uptime	
	RES	**REQ**	**RES**	**REQ**	**RES**	**REQ**	**RES**	**REQ**
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 6: Bytes over the air bytes in Topology 3

	Walk		Bulk Walk		sysDescr		uptime	
	RES	**REQ**	**RES**	**REQ**	**RES**	**REQ**	**RES**	**REQ**
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 7: Bytes over the air in Topology 4

	Walk		Bulk Walk		sysDescr		uptime	
	RES	**REQ**	**RES**	**REQ**	**RES**	**REQ**	**RES**	**REQ**
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 8: Fragments/Frames transmitted in Topology 1

	Walk		Bulk Walk		sysDescr		uptime	
	Fragments	**Frames**	**Fragments**	**Frames**	**Fragments**	**Frames**	**Fragments**	**Frames**
SNMP	18	16	10	8	3	2	2	2
CoAP	18	16	13	8	2	2	2	2
CoMP	16	16	8	8	2	2	2	2

of Walk and `uptime`, yet the improvements in the request packets are much larger, making CoMP more energy-efficient in total. As mentioned before, the receiving (RX) cycle is controlled by the scheduler and the network is prone to external interference which may cause a different number of retransmissions in each case. Therefore, the small difference between SNMP and CoMP in the response packets are due to the MAC protocol and not because the application-layer protocol is better.

Table 9: Fragments/Frames transmitted in Topology 2

	Walk		Bulk Walk		sysDescr		uptime	
	Fragments	**Frames**	**Fragments**	**Frames**	**Fragments**	**Frames**	**Fragments**	**Frames**
SNMP	54	48	31	24	9	6	6	6
CoAP	54	48	39	30	6	6	6	6
CoMP	48	48	26	24	6	6	6	6

Table 10: Fragments/Frames transmitted in Topology 3

	Walk		Bulk Walk		sysDescr		uptime	
	Fragments	**Frames**	**Fragments**	**Frames**	**Fragments**	**Frames**	**Fragments**	**Frames**
SNMP	306	240	175	120	50	30	35	30
CoAP	270	240	218	150	30	30	30	30
CoMP	240	240	140	120	30	30	30	30

Table 11: Fragments/Frames transmitted in Topology 4

	Walk		Bulk Walk		sysDescr		uptime	
	Fragments	**Frames**	**Fragments**	**Frames**	**Fragments**	**Frames**	**Fragments**	**Frames**
SNMP	144	128	83	64	24	16	16	16
CoAP	144	128	104	80	16	16	16	16
CoMP	128	128	70	64	16	16	16	16

Table 12: Jitter and Delay in Topology 1 (in ms)

	Walk		Bulk Walk		sysDescr		uptime	
	Jitter	**Delay**	**Jitter**	**Delay**	**Jitter**	**Delay**	**Jitter**	**Delay**
SNMP	169	2540	169	1180	170	170	160	160
CoAP	169	2540	169	1520	160	160	160	160
CoMP	169	2540	169	1180	160	160	160	160

6.3 Evaluation of CoMP-SNMP Border Router Proxy

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.

In Figures 24, 25, 26 and 27 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.

Table 13: Jitter and Delay in Topology 2 (in ms)

	Walk		**Bulk Walk**		**sysDescr**		**uptime**	
	Jitter	**Delay**	**Jitter**	**Delay**	**Jitter**	**Delay**	**Jitter**	**Delay**
SNMP	147	3210	145	1510	135	245	128	235
CoAP	147	3210	146	1935	128	235	128	235
CoMP	147	3210	145	1510	128	235	128	235

Table 14: Jitter and Delay in Topology 3 (in ms)

	Walk		**Bulk Walk**		**sysDescr**		**uptime**	
	Jitter	**Delay**	**Jitter**	**Delay**	**Jitter**	**Delay**	**Jitter**	**Delay**
SNMP	141	9213	136	4325	139	1026	137	1016
CoAP	109	6705	136	5398	100	681	100	681
CoMP	109	6705	109	3305	100	681	100	681

Table 15: Jitter and Delay in Topology 4 (in ms)

	Walk		**Bulk Walk**		**sysDescr**		**uptime**	
	Jitter	**Delay**	**Jitter**	**Delay**	**Jitter**	**Delay**	**Jitter**	**Delay**
SNMP	132	3767	129	1769	118	313	114	303
CoAP	131	3724	143	2513	114	303	114	303
CoMP	131	3722	131	1778	113	301	113	301

Table 16: Radio Activity Time in FIT IoT-Lab [ms]

	Walk		**Bulk Walk**		**sysDescr**		**uptime**	
	RES	**REQ**	**RES**	**REQ**	**RES**	**REQ**	**RES**	**REQ**
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	7.20
CoMP	58.83	32.74	32.98	20.99	10.77	7.78	10.77	6.43

7 Conclusion

This paper analyzes SNMP and CoAP, and proposes a new protocol that offers the same functionality as the former whilst having 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.

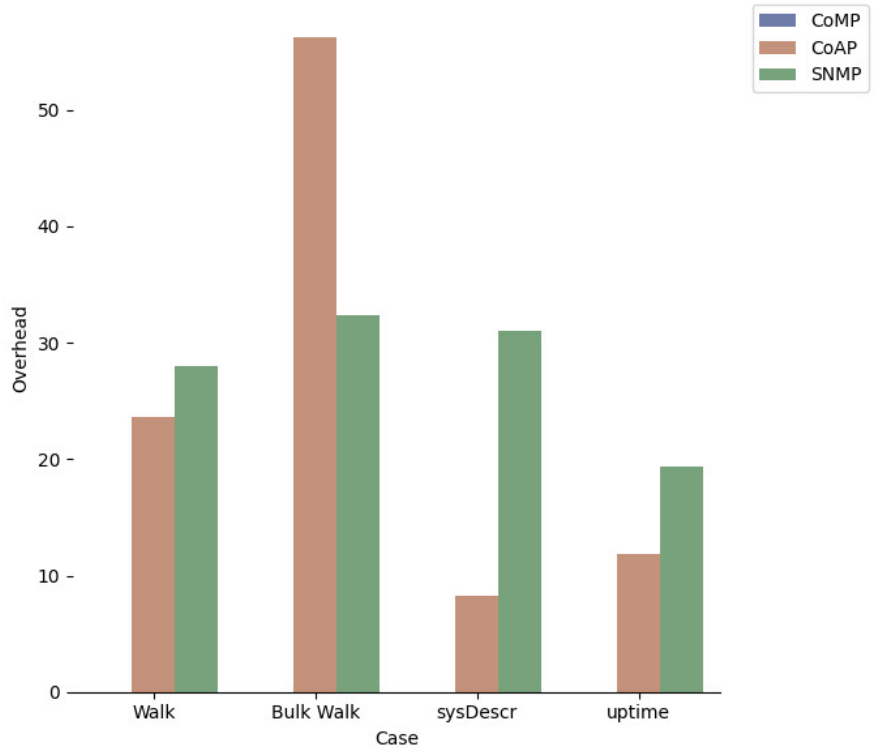


Fig. 22: Relative radio activity time (ms) for transmitting request packets with respect to CoMP in FIT IoT-Lab test-bed experiments.

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.

Instead, CoMP was designed to have the best of both worlds. It incorporates the messaging pattern, Request-Response, resource identification, OID, monitored values, MIB, and operations, `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, equivalent to 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

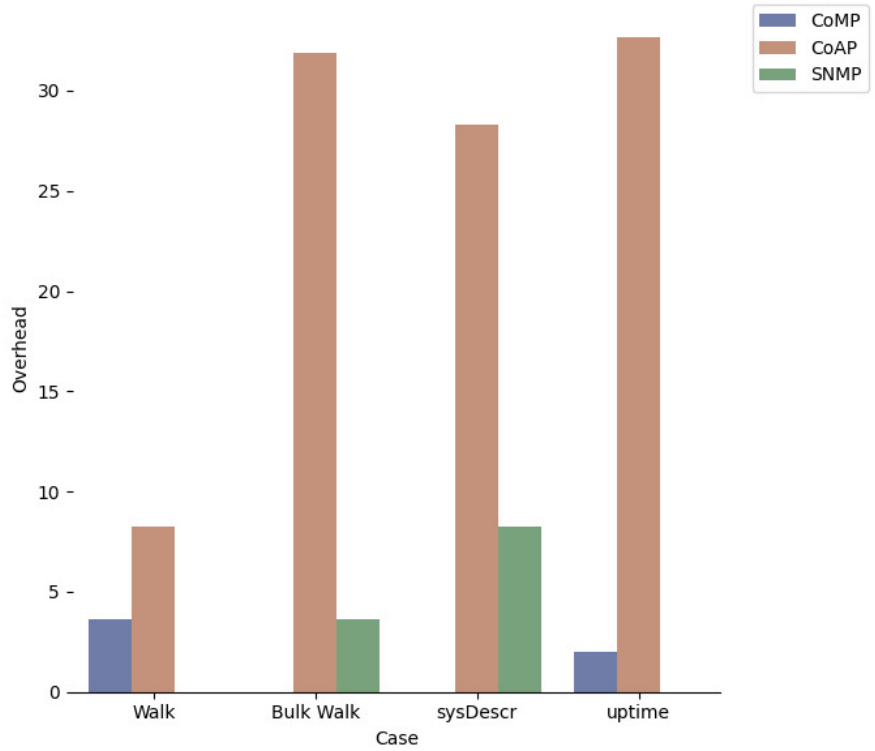


Fig. 23: Relative radio activity time (ms) for transmitting response packets with respect to the best performing protocol in FIT IoT-Lab test-bed experiments.

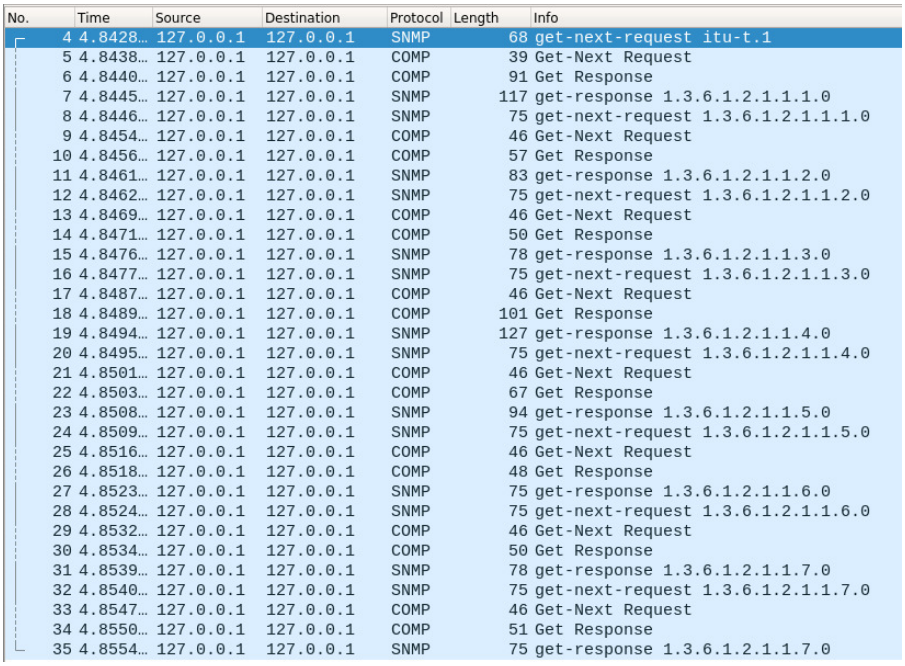
reduced significantly, up to 85% in some cases. This reduction reflected in the bytes over the air which, in turn, affects the radio duty cycle that is directly tied to energy consumption.

This paper also designed and evaluated a cross-protocol proxy that runs at the border router of the WSN and provides a translation layer between CoMP and SNMP. The experiments show that seamless cross-protocol proxying is possible, enabling the integration of CoMP to existing SNMP-based network monitoring infrastructures.

A future extension of CoMP could incorporate a security layer, adopting elements of SNMPv3 [40].

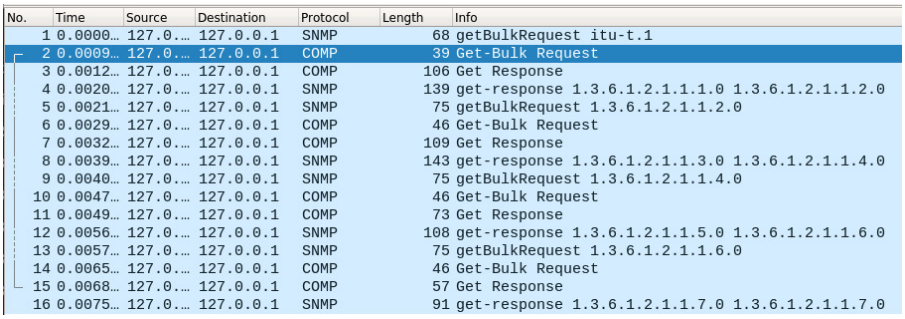
References

1. Accettura, N., Grieco, L.A., Boggia, G., Camarda, P.: Performance analysis of the RPL Routing Protocol. IEEE International Conference on Mechatronics, Icm 2011 - Proceedings pp. 5971218, 767–772 (2011). DOI 10.1109/ICMECH.2011.5971218



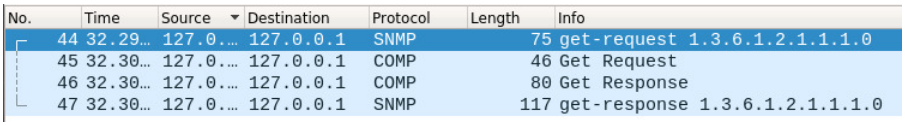
No.	Time	Source	Destination	Protocol	Length	Info
4	4.8428…	127.0.0.1	127.0.0.1	SNMP	68	get-next-request itu-t.1
5	4.8438…	127.0.0.1	127.0.0.1	COMP	39	Get-Next Request
6	4.8440…	127.0.0.1	127.0.0.1	COMP	91	Get Response
7	4.8445…	127.0.0.1	127.0.0.1	SNMP	117	get-response 1.3.6.1.2.1.1.1.0
8	4.8446…	127.0.0.1	127.0.0.1	SNMP	75	get-next-request 1.3.6.1.2.1.1.1.0
9	4.8454…	127.0.0.1	127.0.0.1	COMP	46	Get-Next Request
10	4.8456…	127.0.0.1	127.0.0.1	COMP	57	Get Response
11	4.8461…	127.0.0.1	127.0.0.1	SNMP	83	get-response 1.3.6.1.2.1.1.2.0
12	4.8462…	127.0.0.1	127.0.0.1	SNMP	75	get-next-request 1.3.6.1.2.1.1.2.0
13	4.8469…	127.0.0.1	127.0.0.1	COMP	46	Get-Next Request
14	4.8471…	127.0.0.1	127.0.0.1	COMP	50	Get Response
15	4.8476…	127.0.0.1	127.0.0.1	SNMP	78	get-response 1.3.6.1.2.1.1.3.0
16	4.8477…	127.0.0.1	127.0.0.1	SNMP	75	get-next-request 1.3.6.1.2.1.1.3.0
17	4.8487…	127.0.0.1	127.0.0.1	COMP	46	Get-Next Request
18	4.8489…	127.0.0.1	127.0.0.1	COMP	101	Get Response
19	4.8494…	127.0.0.1	127.0.0.1	SNMP	127	get-response 1.3.6.1.2.1.1.4.0
20	4.8495…	127.0.0.1	127.0.0.1	SNMP	75	get-next-request 1.3.6.1.2.1.1.4.0
21	4.8501…	127.0.0.1	127.0.0.1	COMP	46	Get-Next Request
22	4.8503…	127.0.0.1	127.0.0.1	COMP	67	Get Response
23	4.8508…	127.0.0.1	127.0.0.1	SNMP	94	get-response 1.3.6.1.2.1.1.5.0
24	4.8509…	127.0.0.1	127.0.0.1	SNMP	75	get-next-request 1.3.6.1.2.1.1.5.0
25	4.8516…	127.0.0.1	127.0.0.1	COMP	46	Get-Next Request
26	4.8518…	127.0.0.1	127.0.0.1	COMP	48	Get Response
27	4.8523…	127.0.0.1	127.0.0.1	SNMP	75	get-response 1.3.6.1.2.1.1.6.0
28	4.8524…	127.0.0.1	127.0.0.1	SNMP	75	get-next-request 1.3.6.1.2.1.1.6.0
29	4.8532…	127.0.0.1	127.0.0.1	COMP	46	Get-Next Request
30	4.8534…	127.0.0.1	127.0.0.1	COMP	50	Get Response
31	4.8539…	127.0.0.1	127.0.0.1	SNMP	78	get-response 1.3.6.1.2.1.1.7.0
32	4.8540…	127.0.0.1	127.0.0.1	SNMP	75	get-next-request 1.3.6.1.2.1.1.7.0
33	4.8547…	127.0.0.1	127.0.0.1	COMP	46	Get-Next Request
34	4.8550…	127.0.0.1	127.0.0.1	COMP	51	Get Response
35	4.8554…	127.0.0.1	127.0.0.1	SNMP	75	get-response 1.3.6.1.2.1.1.7.0

Fig. 24: SNMP to CoMP Gateway - Walk



No.	Time	Source	Destination	Protocol	Length	Info
1	0.0000…	127.0.…	127.0.0.1	SNMP	68	getBulkRequest itu-t.1
2	0.0009…	127.0.…	127.0.0.1	COMP	39	Get-Bulk Request
3	0.0012…	127.0.…	127.0.0.1	COMP	106	Get Response
4	0.0020…	127.0.…	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	0.0021…	127.0.…	127.0.0.1	SNMP	75	getBulkRequest 1.3.6.1.2.1.1.2.0
6	0.0029…	127.0.…	127.0.0.1	COMP	46	Get-Bulk Request
7	0.0032…	127.0.…	127.0.0.1	COMP	109	Get Response
8	0.0039…	127.0.…	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	0.0040…	127.0.…	127.0.0.1	SNMP	75	getBulkRequest 1.3.6.1.2.1.1.4.0
10	0.0047…	127.0.…	127.0.0.1	COMP	46	Get-Bulk Request
11	0.0049…	127.0.…	127.0.0.1	COMP	73	Get Response
12	0.0056…	127.0.…	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	0.0057…	127.0.…	127.0.0.1	SNMP	75	getBulkRequest 1.3.6.1.2.1.1.6.0
14	0.0065…	127.0.…	127.0.0.1	COMP	46	Get-Bulk Request
15	0.0068…	127.0.…	127.0.0.1	COMP	57	Get Response
16	0.0075…	127.0.…	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

Fig. 25: SNMP to CoMP Gateway - Bulk Walk



No.	Time	Source	Destination	Protocol	Length	Info
44	32.29…	127.0.…	127.0.0.1	SNMP	75	get-request 1.3.6.1.2.1.1.1.0
45	32.30…	127.0.…	127.0.0.1	COMP	46	Get Request
46	32.30…	127.0.…	127.0.0.1	COMP	80	Get Response
47	32.30…	127.0.…	127.0.0.1	SNMP	117	get-response 1.3.6.1.2.1.1.1.0

Fig. 26: SNMP to CoMP Gateway - sysDescr

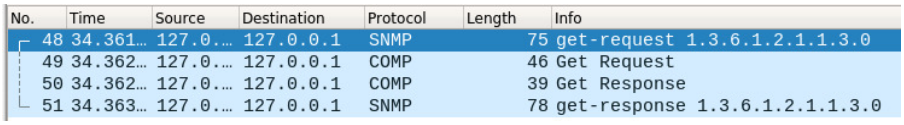


Fig. 27: SNMP to CoMP Gateway - uptime

2. Adjih, C., Baccelli, E., Fleury, E., Harter, G., Mitton, N., Noel, T., Pissard-Gibollet, R., Saint-Marcel, F., Schreiner, G., Vandaele, J., Watteyne, T.: FIT IoT-LAB: A large scale open experimental IoT testbed. IEEE World Forum on Internet of Things pp. 7389098, 459–464 (2015). DOI 10.1109/WF-IoT.2015.7389098
3. Bhalerao, R., Subramanian, S.S., Pasquale, J.: An analysis and improvement of congestion control in the CoAP Internet-of-Things protocol. 2016 13th IEEE Annual Consumer Communications and Networking Conference, CCNC 2016 pp. 7444906, 889–894 (2016). DOI 10.1109/CCNC.2016.7444906
4. Bierman, A., Jones, K.: Physical Topology MIB. RFC 2922, RFC Editor (2000). URL `http://www.rfc-editor.org/rfc/rfc2922.txt`
5. Bormann, C.: Concise Binary Object Representation (CBOR) Tags for Typed Arrays. RFC 8746, RFC Editor (2020). URL `http://www.rfc-editor.org/rfc/rfc8746.txt`
6. Bormann, C.: Notable CBOR Tags. Internet-Draft draft-bormann-cbor-notable-tags-03, Internet Engineering Task Force (2021). URL `https://datatracker.ietf.org/doc/html/draft-bormann-cbor-notable-tags-03`. Work in Progress
7. Bormann, C., Hoffman, P.: Concise Binary Object Representation (CBOR). STD 94, RFC Editor (2020). URL `http://www.rfc-editor.org/rfc/rfc8949.txt`
8. Bormann, C., Leonard, S.: Concise Binary Object Representation (CBOR) Tags for Object Identifiers. Internet-Draft draft-ietf-cbor-tags-oid-04, Internet Engineering Task Force (2021). URL `https://datatracker.ietf.org/doc/html/draft-ietf-cbor-tags-oid-04`. Work in Progress
9. Case, J., Fedor, M., Schoffstall, M., Davin, J.: Simple Network Management Protocol (SNMP). RFC 1157, RFC Editor (1990). URL `http://www.rfc-editor.org/rfc/rfc1157.txt`
10. Chaudhry, S.A., Boyle, G., Song, W., Sreenan, C.: EMP: A Network Management Protocol for IP-based Wireless Sensor Networks. In: 2010 International Conference on Wireless and Ubiquitous Systems, pp. 1–6 (2010). DOI 10.1109/ICWUS.2010.5670440
11. Choi, H., Kim, N., Cha, H.: 6LoWPAN-SNMP: Simple Network Management Protocol for 6LoWPAN. In: 2009 11th IEEE International Conference on High Performance Computing and Communications, pp. 305–313 (2009). DOI 10.1109/HPCC.2009.49
12. da Cruz, M.A., Rodrigues, J.J., Lorenz, P., Solic, P., Al-Muhtadi, J., Albuquerque, V.H.C.: A proposal for bridging application layer protocols to HTTP on IoT solutions. Future Generation Computer Systems **97**, 145–152 (2019). DOI 10.1016/j.future.2019.02.009
13. do Rosário, Y.F., Fafoutis, X.: CoMP: Efficient Monitoring for Constrained Embedded Devices. In: 2020 IEEE Symposium on Computers and Communications (ISCC), pp. 1–7 (2020). DOI 10.1109/ISCC50000.2020.9219715
14. Duquennoy, S., Al Nahas, B., Landsiedel, O., Watteyne, T.: Orchestra: Robust Mesh Networks Through Autonomously Scheduled TSCH. In: Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems, pp. 337–350. ACM, New York, NY, USA (2015)
15. Duquennoy, S., Eriksson, J., Voigt, T.: Five-Nines Reliable Downward Routing in RPL. CoRR **abs/1710.02324** (2017). URL `http://arxiv.org/abs/1710.02324`
16. Elsts, A., Fafoutis, X., Oikonomou, G., Piechocki, R., Craddock, I.: TSCH Networks for Health IoT: Design, Evaluation and Trials in the Wild. ACM Trans. Internet Things **1**(2) (2020)
17. Elsts, A., Fafoutis, X., Woznowski, P., Tonkin, E., Oikonomou, G., Piechocki, R., Craddock, I.: Enabling Healthcare in Smart Homes: The SPHERE IoT Network Infrastructure. IEEE Communications Magazine **56**(12), 164–170 (2018). URL `https://doi.org/10.1109/MCOM.2017.1700791`

18. Ersue, M., Romascanu, D., Schoenwaelder, J., Sehgal, A.: Management of networks with constrained devices: Use cases. RFC 7548, RFC Editor (2015). URL http://www.rfc-editor.org/rfc/rfc7548.txt
19. Fafoutis, X., Elsts, A., Vafeas, A., Oikonomou, G., Piechocki, R.: On Predicting the Battery Lifetime of IoT Devices: Experiences from the SPHERE Deployments. In: Proceedings of the 7th International Workshop on Real-World Embedded Wireless Systems and Networks, p. 7–12 (2018)
20. Fofack, N.C., Nain, P., Neglia, G., Towsley, D.: Analysis of TTL-based cache networks. In: 6th International ICST Conference on Performance Evaluation Methodologies and Tools, pp. 1–10 (2012)
21. Ge, W., Zheng, L., Luo, P., Liu, Z.: Implementation of multiple border routers for 6LoWPAN with ContikiOS. 2015 International Conference on Information and Communications Technologies (ict 2015) pp. 1–6, 1–6 (2015). DOI 10.1049/cp.2015.0221
22. ISO Central Secretary: Information Processing Systems – Open Systems Interconnection – Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1). Standard, International Organization for Standardization (1987)
23. ISO Central Secretary: Information Technology – Open Systems Interconnection – Presentation Service Definition. Standard, International Organization for Standardization (1994)
24. Karaagac, A., De Poorter, E., Hoebeke, J.: In-Band Network Telemetry in Industrial Wireless Sensor Networks. IEEE Transactions on Network and Service Management **17**(1), 517–531 (2020). DOI 10.1109/TNSM.2019.2949509
25. Laaroussi, Z., Morabito, R., Taleb, T.: Service Provisioning in Vehicular Networks through Edge and Cloud: an Empirical Analysis. 2018 IEEE Conference on Standards for Communications and Networking (IEEE CSCN) (2018)
26. Leggieri, M., Hausenblas, M.: Interoperability of two RESTful protocols: HTTP and CoAP. Rest: Advanced Research Topics and Practical Applications **9781461492993**, 27–49 (2014). DOI 10.1007/978-1-4614-9299-3_3
27. McCloghrie, K., Rose, M.: Structure and identification of management information for TCP/IP-based internets. RFC 1065, RFC Editor (1988). URL http://www.rfc-editor.org/rfc/rfc1065.txt
28. Muhic, I., Hodzic, M.: Verification and validation of wireless sensor network energy consumption in internet of things. ICAT 2017 - 26th International Conference on Information, Communication and Automation Technologies, Proceedings **2017-**, 1–6 (2017). DOI 10.1109/ICAT.2017.8171642
29. Mukhtar, H., Kang-Myo, K., Chaudhry, S.A., Akbar, A.H., Ki-Hyung, K., Seung-Wha Yoo: LNMP- Management architecture for IPv6 based low-power wireless Personal Area Networks (6LoWPAN). In: NOMS 2008 - 2008 IEEE Network Operations and Management Symposium, pp. 417–424 (2008). DOI 10.1109/NOMS.2008.4575163
30. Osterlind, F., Dunkels, A., Eriksson, J., Finne, N., Voigt, T.: Cross-level sensor network simulation with COOJA. Conference on Local Computer Networks pp. 641–648 (2006)
31. Rose, M., McCloghrie, K.: Structure and identification of management information for TCP/IP-based internets. RFC 1155, RFC Editor (1990). URL http://www.rfc-editor.org/rfc/rfc1155.txt
32. Sabovic, A., Delgado, C., Bauwens, J., De Poorter, E., Famaey, J.: Accurate Online Energy Consumption Estimation of IoT Devices Using Energest. In: L. Barolli, P. Hellinckx, T. Enokido (eds.) Advances on Broad-Band Wireless Computing, Communication and Applications, pp. 363–373. Springer International Publishing, Cham (2020)
33. Schönwälder, J.: SNMP Payload Compression. Internet-Draft draft-irtf-nmrg-snmp-compression-01, Internet Engineering Task Force (2001). URL https://datatracker.ietf.org/doc/html/draft-irtf-nmrg-snmp-compression-01. Work in Progress
34. Seddik, M.E.A., Toldov, V., Clavier, L., Mitton, N.: From Outage Probability to ALOHA MAC Layer Performance Analysis in Distributed WSNs. IEEE Wireless Commun. and Networking Conf. (2018)
35. Shelby, Z., Hartke, K., Bormann, C.: The Constrained Application Protocol (CoAP). RFC 7252, RFC Editor (2014). URL http://www.rfc-editor.org/rfc/rfc7252.txt

36. Sinche, S., Raposo, D., Armando, N., Rodrigues, A., Boavida, F., Pereira, V., Silva, J.S.: A Survey of IoT Management Protocols and Frameworks. IEEE Communications Surveys Tutorials **22**(2), 1168–1190 (2020). DOI 10.1109/COMST.2019.2943087
37. Sinche, S., Silva, J.S., Raposo, D., Rodrigues, A., Pereira, V., Boavida, F.: Towards Effective IoT Management. In: 2018 IEEE SENSORS, pp. 1–4 (2018). DOI 10.1109/ICSENS.2018.8589886
38. Society, I.C.: IEEE Standard for Local and metropolitan area networks – Station and Media Access Control Connectivity Discovery. IEEE Std 802.1AB-2005 pp. 1–176 (2005). DOI 10.1109/IEEESTD.2005.96285
39. Stallings, W.: SNMP and SNMPv2: the infrastructure for network management. IEEE Commun. Mag. **36**(3), 37–43 (1998)
40. Stallings, W.: SNMPv3: A security enhancement for SNMP. IEEE Communications Surveys **1**(1), 2–17 (1998)
41. Tahmasebi, S., Habibi, J., Shamsaie, A.: A Scalable Architecture for Monitoring IoT Devices Using Ethereum and Fog Computing. In: 2020 4th International Conference on Smart City, Internet of Things and Applications (SCIOT), pp. 66–76 (2020). DOI 10.1109/SCIOT50840.2020.9250193
42. Yahya, W., Basuki, A., Sakti, P.E., Fernanda, F.F.: Lightweight monitoring system for IOT devices. In: 2017 11th International Conference on Telecommunication Systems Services and Applications (TSSA), pp. 1–4 (2017). DOI 10.1109/TSSA.2017.8272897