Proceedings

Performance Evaluation of CoAP and MQTT_SN in an IoT Environment †

Mónica Martí, Carlos Garcia-Rubio * and Celeste Campo

Department of Telematic Engineering, University Carlos III of Madrid, Avda. Universidad 30, E-28911 Leganés, Madrid, Spain; 100359402@alumnos.uc3m.es (M.M.); celeste@it.uc3m.es (C.C.)
* Correspondence: cgr@it.uc3m.es

† Presented at the 13th International Conference on Ubiquitous Computing and Ambient Intelligence UCAmI 2019, Toledo, Spain, 2–5 December 2019.

Published: 20 November 2019

Abstract: The fast growth of the Internet of Things (IoT) has made this technology one of the most promising paradigms of recent years. Wireless Sensor Networks (WSNs) are one of the most important challenges of the Internet of things. These networks are made up of devices with limited processing power, memory, and energy. The constrained nature of WSNs makes it necessary to have specific restricted protocols to work in these environments. In this paper, we present an energy consumption and network traffic study of the main IoT application layer protocols, the Constrained Application Protocol (CoAP), and the version of Message Queue Telemetry Transport (MQTT) for sensor networks (MQTT_SN). The simulations presented evaluate the performance of these protocols with different network configurations.

Keywords: Internet of Things; WSN; MQTT_SN; CoAP; Energy Consumption

1. Introduction

The Internet of Things (IoT) has become an emerging new technology. It is formed by a huge number of devices with sensing, actuating, processing, and communication capabilities, connected and exchanging data among themselves using various radio technologies, such as Zigbee (based on the IEEE 802.15.4 Standard), Wi-Fi (based on the IEEE 802.11 Standard), 6LowPAN over Zigbee (IPv6 over Low Power Personal Area Networks), or Bluetooth (based on the IEEE 802.15.1) [1].

New protocols have been developed in the protocol stack to facilitate communications in IoT environments [2]. The most relevant protocols for the IoT are classified into three categories: infrastructure protocols, service discovery protocols, and application layer protocols. The most relevant infrastructure protocols for the IoT are the Routing Protocol for Low-Power and Lossy Networks (RPL), considered the routing layer standard for the Internet of Things [3], 6LowPAN [4] for the network layer, and IEEE 802.15.4 [5], LTE-A [6], EPCglobal [7], and Z-Wave [8] for the data link and physical layers. The most popular service discovery protocols are multicast DNS (mDNS) and DNS Service Discovery (DNS-SD), which are aimed at discovering resources and services offered by IoT devices [2].

In the IoT application layer, several protocols designed specifically for constrained devices have been proposed. This paper addresses the two main ones; the Constrained Application Protocol (CoAP) and the Message Queue Telemetry Transport (MQTT), specifically regarding its version for sensor networks (MQTT_SN).

The Constrained Application Protocol (CoAP) [9] is a lightweight version of HTTP. CoAP was design by the Internet Engineering Task Force (IETF) to target constrained-resource devices using a subset of the HTTP methods, making it interoperable with HTTP [10]. It sends the messages compressed and over UDP, while HTTP uses TCP. Since it is similar to HTTP, it uses the same client-server architecture, providing resource-oriented interactions.

Message Queue Telemetry Transport (MQTT) [11] follows a publish/subscribe architecture and runs over TCP, as HTTP. MQTT_SN is the lightweight version of MQTT for sensor networks. Unlike MQTT, MQTT_SN runs over UDP.

This document discusses CoAP and MQTT_SN in terms of energy consumption and traffic generated, emphasizing the type and length of messages transmitted in each case. The paper is structured as follows: Section 2 provides a state of art of the protocols involved in the experiment, CoAP and MQTT_SN, attending to their main features, architecture, and type of messages. Section 3 reviews some related works. Section 4 presents the scenario in which the tests were developed and gives an overview of the simulations. Section 5 provides an analysis of the experimental results. Section 6 offers some lines to continue this work and the conclusions.

2. IoT Application Layer Protocols

Many application protocols have been proposed for the IoT, where devices have several constraints and are deployed in low power and lossy networks. We will next review the main two: CoAP and MQTT_SN.

2.1. Constrained Application Protocol (CoAP)

CoAP [9] was developed by the IETF Constrained RESTful Environments (CoRE) working group. It defines a web transfer protocol based on Representational State Transfer (REST) on top of HTTP functionalities [2]. Unlike HTTP, CoAP runs over UDP, removing all the TCP overhead, which reduces bandwidth requirements, provides more simplicity, and makes it more suitable for IoT applications. CoAP uses a request/response architecture as HTTP. Therefore, it shares the same methods as HTTP: GET, PUT, POST, and DELETE [12]. As opposed to HTTP, CoAP relies on a non-connection-oriented transport protocol (UDP), and it supports unicast as well as multicast. A typical CoAP environment is shown in Figure 1.

Since UDP is an unreliable transport protocol, CoAP implements some mechanisms at the application layer in order to achieve reliability. There are four types of CoAP messages (Figure 2):

- Confirmable (CON): CON messages require an acknowledgement (ACK), and the response could be sent in the same ACK message (synchronously) or in a separate message if more computational time is needed (asynchronously).
- Non-Confirmable (NON): a NON message does not need an ACK.
- Acknowledgement (ACK): sent in response to CON messages, to confirm their reception.
- Reset (RST): sent as a response of a message that could not be processed [1].

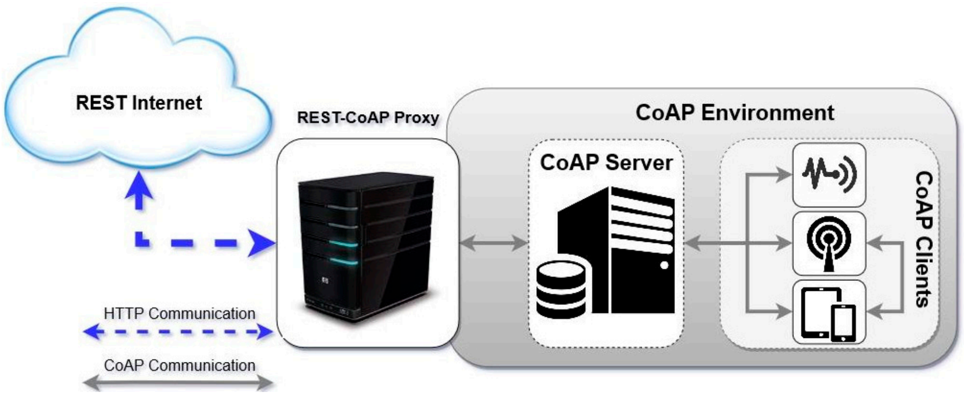

Figure 1. Client/Server model protocol in a Constrained Application Protocol (CoAP) environment.

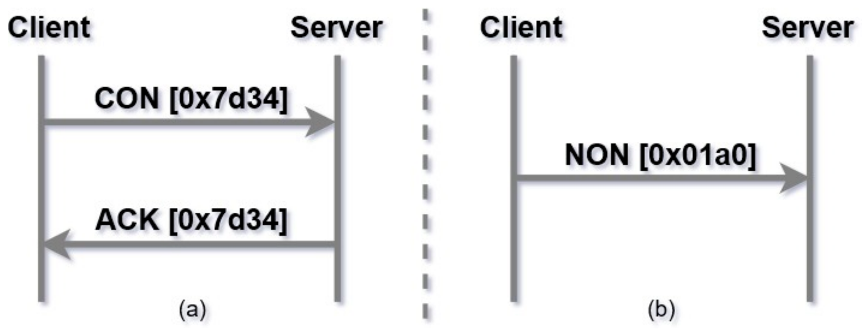

Figure 2. Messages types exchanged in CoAP: (**a**) Confirmable Messages; (**b**) Non-Confirmable Messages.

CoAP does not include any built-in security features. However, as HTTP is secured using Transport Layer Security (TLS) over TCP, CoAP can use the Datagram Transport Layer Security (DTLS) over UDP, providing confidentially, integrity, authentication, and non-repudiation [9]. However, using DTLS implies more overheads, bandwidth, and energy consumption in the communication.

2.2. Message Queue Telemetry Transport for Sensor Networks (MQTT_SN)

MQTT [13] is an application layer protocol designed for constraints devices, developed by IBM and standardized at OASIS. It follows an asynchronous publish/subscribe protocol that runs over TCP. This message pattern involves a publisher, a subscriber, and a broker that manages and controls the exchanged packets between publishers and subscribers, (Figure 3).

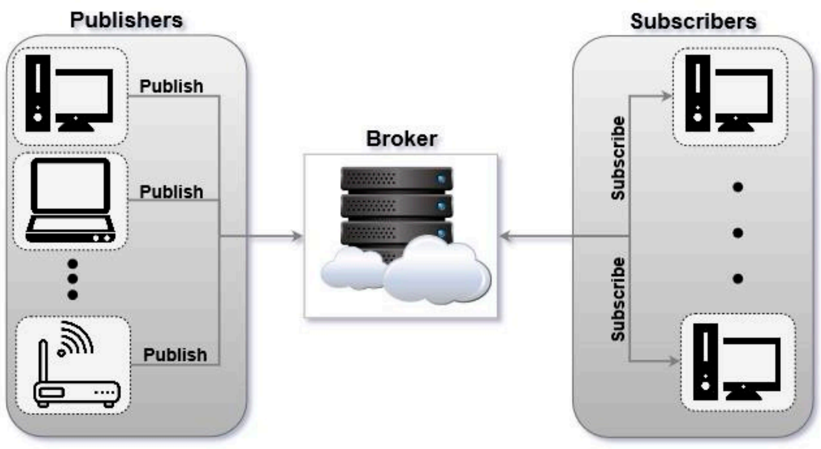

Figure 3. Publish/Subscribe pattern followed by the Message Queue Telemetry Transport (MQTT) protocol.

For instance, a client (subscriber) registers to different topics of interest in the broker. The sensors (publishers) send to the broker all recollected data, and then the broker forwards it to the clients subscribed to the topics related to that publish message, as shown in Figure 4.

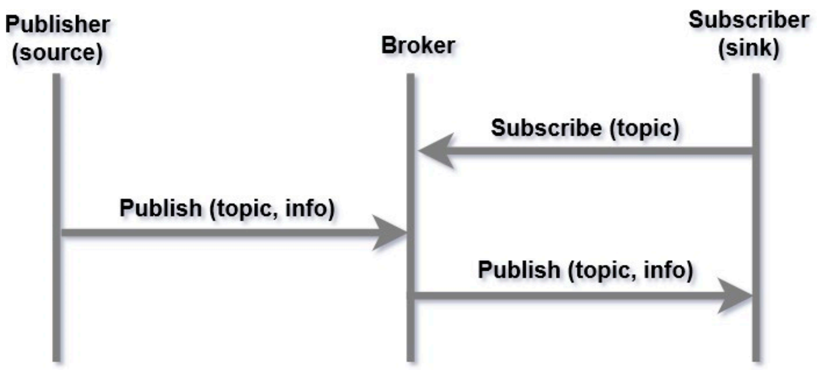

Figure 4. Messages exchanged by MQTT protocol.

MQTT defines three levels of Quality of Service (QoS) that ensure the delivery of publish messages providing more reliability to the system:

- QoS level 0 (at most once): At this QoS level, delivery is not guaranteed. It is a best-effort delivery service level. The receiver does not acknowledge receipt of the message, and the message is not stored and re-transmitted by the sender. This QoS level is also known as "fire and forget" and is the simplest one, therefore, it has the lowest overheads [14].
- QoS level 1 (at least once): The sender stores and retransmits the message until it receives the acknowledgement from the receiver. This QoS level guarantees messages arrival, but duplicates can occur [14].
- QoS level 2 (exactly once): Where messages are assured to arrive exactly once. This level could be used, for instance, with billing systems where duplicate or lost messages cannot be afforded [14].

MQTT_SN is the version for sensor networks of MQTT. It was specifically designed to work in Wireless Sensor Networks (WSNs). It works over UDP, while the original MQTT works over TCP and TLS [15]. This protocol was designed to be as close as possible to MQTT, meaning that it can work with the same infrastructure as MQTT because it supports the same semantics. The only difference in the architecture is that MQTT_SN needs a new entity in the system, called a gateway, which has to translate all MQTT_SN messages over UDP to MQTT messages over TCP (Figure 5). Currently, there are many brokers that have this functionality integrated, therefore, all the complexity resides at the broker/gateway side, keeping the client side as simple as possible [16].

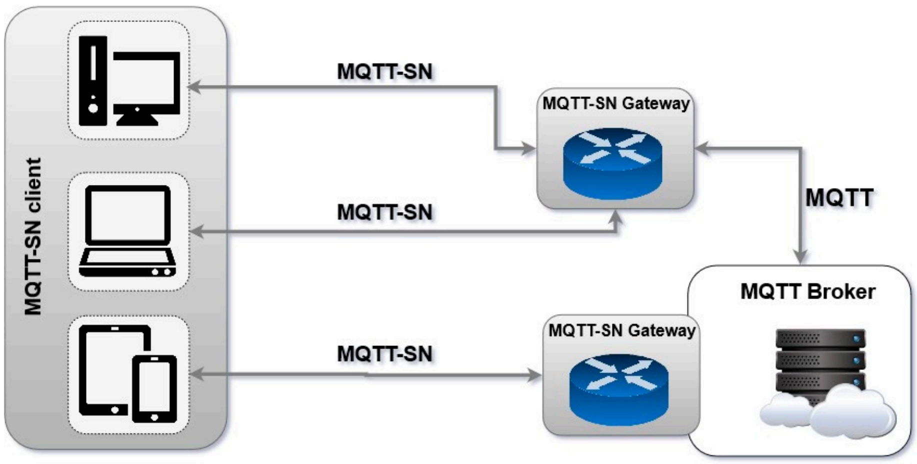

Figure 5. Publish/Subscribe pattern followed by the MQTT for sensor networks (MQTT_SN) protocol including the gateway functionality.

Compared to MQTT, MQTT_SN has the advantage of a new offline keep-alive procedure that is defined to support "sleeping clients". With this procedure, battery-operated devices can go to sleeping state; meanwhile, all the messages designated to them will be buffered at the server/gateway and delivered later when the nodes wake up.

MQTT_SN has several types of messages [17]. A typical exchange of MQTT_SN messages is shown in Figure 6.

MQTT_SN does not support DTLS for security because of the limitation of the payload size of the messages. However, it supports "sleeping clients" for energy efficiency.

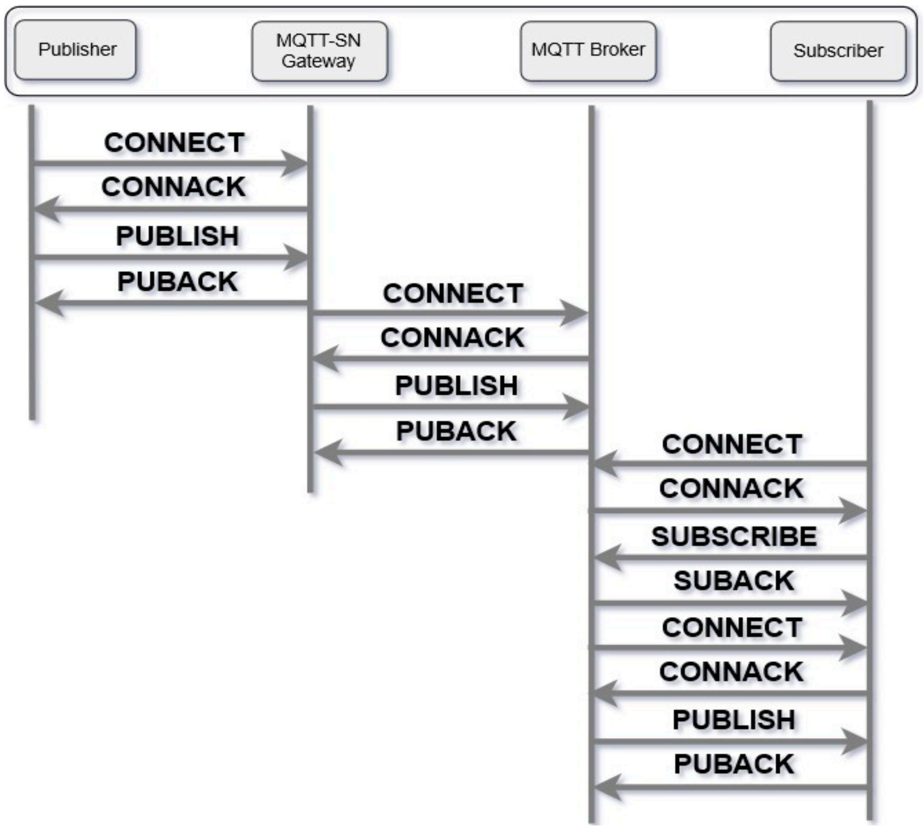

Figure 6. Messages Exchanged by MQTT_SN protocol.

3. Related Works

In the last five years, some works have studied and compared the performance of application layer protocols in IoT environments. We now review the most relevant of these works in chronological order.

In [18] the authors present an analysis of resource requirements for CoAP and MQTT protocols based on experimental results conducted using Intel X86 systems with Ubuntu, libcoap, and Mosquitto. They are compared based on their different protocol specific characteristics as well as usage of resources like bandwidth and energy. The analysis of the experimental results depicts that CoAP is most efficient in terms of energy consumption as well as bandwidth. For sending one message with reliability for both protocols, CoAP uses only 140 bytes total (including link, network, and transport layer header). MQTT with QoS-1 is second with 162 bytes, while MQTT with QoS-2 requires 318 bytes total. Generally, results show that CoAP and MQTT consume low bandwidth.

In [19], a qualitative and quantitative comparison is provided between MQTT and CoAP when used in a smartphone application. For the study they use a CoAP Server and Client based on a Java implementation called Californium. The MQTT Broker is based on Mosquitto. The MQTT Publisher and CoAP Server run on a smartphone with Android 4.2.2, and the MQTT Broker and CoAP Client run on a Windows 7 OS laptop. They measure bandwidth use and round-trip time, and conclude that CoAP can be a valid alternative to MQTT for certain application scenarios.

In [20], the authors analyze the costs of using CoAP and HTTP in IoT applications, based on a theoretical cost model. The analysis suggests that the simpler hardware requirements of CoAP smart objects, as well as the lower communication overheads of the protocol and the resulting reduced power consumption, lead to cost advantages in many application scenarios. CoAP only requires 40% of the power normally used by HTTP.

In [21], the authors evaluate CoAP and MQTT-SN for robot communications. For the experiments they use a Raspberry Pi B (ARMv6 processor) connected to the network through a Local Area Network Ethernet cable, using CON and QoS 1. They just measure transmission times, finding that MQTT- SN shows a 30% better transmission time over CoAP.

In [22], the author presents an empirical comparison of different open source CoAP implementations in terms of latency, memory, and CPU consumption in a real testbed consisting of two Raspberry Pi 3 model Bs connected via Wi-Fi through a 56 Mbps router. They measure latency with Tcpdump after 50 requests have been sent for each combination of CoAP client and server implementation.

In [23], the authors discuss and analyze the efficiency, usage, and requirements of MQTT and CoAP using a Raspberry-Pi with Raspbian OS and a temperature sensor in a simple experiment that contains one publisher, server, and broker. They conclude that as the size of the message being sent increases, CoAP handles more data than MQTT.

In [24], the authors compare the performance of CoAP, MQTT, and HTTP REST using a Raspberry Pi 3 and the aiocoap, Mosquitto, and Django implementations of the protocols. They compare the number of bytes consumed and the delay for each of the protocols under different network conditions. The results show that CoAP is most efficient in terms of time and bandwidth for smaller payloads, and its performance deteriorates as payload size increases.

In [25], the authors investigate and analyze four protocols, namely, CoAP, MQTT, MQTT-SN, and QUIC, to understand the overheads of obtaining data from an IoT device at a sink to potentially disseminate this data downstream. For evaluation, the experiments were performed in an emulated environment using VirtualBox VMs. A subset of the experiments was also run using Raspberry Pis. They measure delay and total packets sent. Results show that in terms of overheads, CoAP is the most efficient protocol. Another key finding from the experiments is that for IoT protocols that use a fire-and-forget paradigm, such as CoAP non-confirmable, MQTT QoS 0, and MQTT-SN QoS 0, the wait or keep alive timers play a crucial role in performance.

In [26], the authors look at the impact of the application protocol (CoAP and MQTT) on performance over two different wireless networks: Bluetooth Low Energy and Wi-Fi. The evaluation was performed using an Ericsson internal event-based radio network system simulator. The authors find that CoAP performs better both in terms of latency and power consumption over both wireless networks.

In [27] and [28], the authors compare CoAP and MQTT with regards to communication delay and network traffic. For the experiments they use Linux Lubuntu.

In [29], the author deploys NDN, an information-centric networking (ICN) protocol, with name-based routing and in-network caching, and the IP-based CoAP and MQTT-SN, on a large-scale IoT testbed in single- and multi-hop scenarios under varying traffic loads. The devices in the testbed use RIOT, a small, open source operating system with a focus on low-power wireless Internet of Things (IoT) devices. The authors find that NDN is more resource-friendly and robust in multi-hop scenarios, while CoAP and MQTT operate at less overhead and higher speed in single-hop deployments. Although the paper is focused on comparing ICN protocols against current, IP based, IoT protocols, the authors also point out that a holistic analysis of MQTT and CoAP in a consistent experimental setting, including low-end IoT devices, is missing in the literature.

Finally, in [30], the authors look at the impact of the protocol stack (specifically CoAP and MQTT) on performance over a 5G massive IoT realization (NB-IoT). As in [26], the author uses an Ericsson internal event-based radio network system simulator, and obtains the application level throughput. The simulation study confirms that MQTT as a TCP based system impacts negatively on the device's perceived throughput compared with CoAP.

From this state of the art review, we can conclude that most of the previous work on performance evaluation of IoT application layer protocols considered just CoAP, or CoAP and the not constrained version of MQTT (MQTT over TCP). We use the UDP-adapted version of MQTT, MQTT-SN, since TCP is inappropriate for the constrained IoT, as pointed out in [29]. From all the related works,only [21], [25], and [29] include in their study CoAP and MQTT-SN. In [21] and [25] the study is conducted using one Raspberry or VirtualBox VMs, without considering the effects of several devices and multi-hop scenarios, and they measure delay and packet transferred but not energy consumption, which is a key parameter in constrained environments. [29] is the most complete work to date, including multi-hop scenarios and energy consumption measurements, but its focus is different from ours, since it compares protocols based on ICN versus IP-based protocols, not IoT application layerprotocols.

In this paper, we will present an energy consumption, message length, and type of message study of the two main IoT application layer protocols, CoAP and MQTT-SN, with constrained devices (Z1 motes) in a multi-hop environment simulated using the Cooja simulator, using the routing protocol for low-power and lossy networks defined in the IETF, RPL.

4. Simulation Tools and Scenarios

In order to test the performance of the main application protocols in terms of energy consumption and traffic generated, Contiki [31] was used, which is an open source operating system for the Internet of Things. This operating system connects tiny low-cost, low-power microcontrollers to the Internet and supports IPv6 and IPv4, along with IoT standards such as 6LowPAN, RPL, CoAP, and MQTT. It also supports traditional IP stack protocols such as UDP, TCP, and HTTP. All the applications in Contiki are written in C language programming, and it has a very powerful tool named Cooja, which is a simulator where networks can be tested before being printed into hardware. Cooja offers the advantage of seeing what happens in the simulated network at any time and makes it easy for developers to debug software for such networks, which is quite difficult[31].

Contiki is designed to operate with low-power devices, for instance, devices whose battery may last for years. Most of theContiki motes use MSP430 microcontrollers [32], which are designed to run with small amounts of memory. Contiki provides mechanisms for estimating the system power consumption and for understanding where the power was spent.

4.1. Test Bed

In order to test and compare the energy consumption of these IoT application protocols, the Cooja simulator was used, along with the CoAP and MQTT_SN APIs. Both protocols were simulated using MSP430 microcontrollers [33], CC2420 radio transceivers [34], and Zolertia motes, more specifically Z1 motes [35].

CoAP implementation in Contiki is based on Erbium (Er), a low power REST machine for Contiki. For the test bed, three kinds of elements were used: a border router, a CoAP server to develop server-side applications, and a CoAP client that polls the actuators/toggle resource every 10 s, sending a POST method with non-confirmable messages, which means that no ACK is needed, saving much more energy.

MQTT_SN implementation in Contiki was achieved by using a border router, a publisher, and a subscriber (client), which is programmed to send an MQTT_SN PINGREQ message every 10 s, as the CoAP clients, with QoS level 0. The broker used was the Really Small Message Broker (RSMB), which is a light-weight and low-overhead messaging broker. It allows messaging to and from tiny devices such as sensors and actuators over networks that are constrained in terms of bandwidth, processing capability, and reliability. The RSMB broker is pretty similar to traditional brokers used in MQTT, such as Mosquitto, but the main difference is that RSMB creates a “bridge” that allows connections with those traditional MQTT brokers.

4.2. Energy Consumption

Contiki OS has a powerful tool called an Energest module that provides a lightweight, software-based energy estimation for IoT devices by keeping track when components are on, off, or in different modes. The Energest module was applied to CoAP and MQTT_SN clients by adding the powertrace application [36] to the APIs. The powertrace application prints the time in ticks that the radio has been transmitting (tx) and receiving (rx) as well as how long the CPU has been in active (cpu) and idle mode (lpm). The duration of a tick is platform-dependent and it is defined thanks to a system constant (RTIMER_ARCH_SECOND), whose typical value is 32768 ticks per second.

5. Experimental Results

The main goal of the simulations is to analyze which application protocol is more efficient in terms of transmission energy consumption and number of messages exchanged. The simulation time is 30 min, a time long enough to analyze what happens with these two protocols.

The CoAP client starts making requests by sending POST methods. The length of CoAP messages exchanged are shown in Table 1. However, setting up the connection in MQTT_SN is a little bit different than CoAP because more messages need to be exchanged due to the fact that MQTT_SN clients need to register and subscribe to a topic before receiving any publish message from the broker. Table 2 shows the messages exchanged by the MQTT_SN protocol and their length

Table 1. CoAP messages exchanged and its length.

CoAP Type of Message	**Length (Bytes)**
CoAP Non-Confirmable POST	103

As explained before, the types of messages used are non-confirmable, thus, no responses (ACKs) are needed. Figure 7 shows the number of messages exchanged for a CoAP simulation.

Figure 8 shows that MQTT_SN needs more messages than CoAP for setting up the connection. However, once a MQTT_SN client is subscribed to the interest topics, it falls to "sleep mode", which means less energy consumption. In order to start receiving all the publish messages related to the subscribed topics, a MQTT_SN client sends PINGREQ to the broker, which is pretty similar to sending POST methods in CoAP. The only difference is the length, and, as shown before in Tables 1 and 2, the CoAP POST message length is greater than the MQTT_SN PINGREQ message length. Therefore, it is expected that CoAP transmission energy consumption will be greater than MQTT_SN transmission energy consumption.

Figure 9 shows the accumulative transmission energy consumption of these two protocols during the 30 min simulation in our test bed. Figure 9 proves the veracity of what was expected. As MQTT_SNs exchange more messages at the beginning, with the main goal of subscribing and registering topics, they waste more transmission energy than CoAP when the simulation starts. However, since CoAP message lengths are larger than MQTT_SN message lengths, CoAP transmission energy consumption goes up when the simulation is advanced.

The results show that the total energy consumption is almost the same, with MQTT_SN being slightly more efficient than CoAP. Moreover, as explained in the previous sections, all the complexity of MQTT_SN resides at the broker side, unlike CoAP in which clients are more complex. Therefore, MQTT_SN clients are more lightweight than CoAP clients. The use of CoAP and MQTT_SN protocols depends on the application and what the customer wants. For instance, if the application to develop does not require any security mechanism or QoS, and the main goal is to save energy, it is better to use MQTT_SN. Otherwise, CoAP is a good option, and it is also advisable to look at other application protocols for the IoT, such as MQTT, although it is not so well suited for constrained devices.

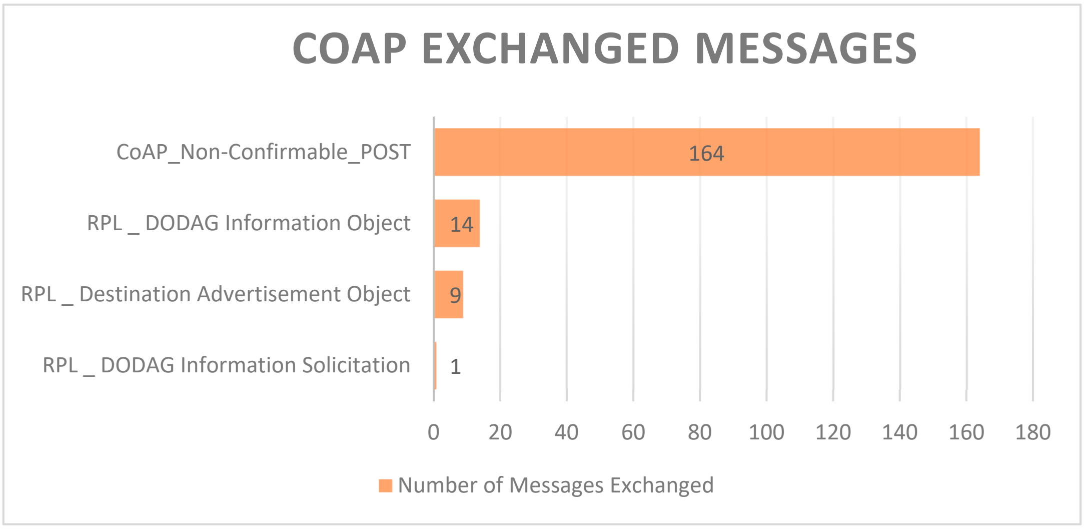

Figure 7. CoAP exchanged messages during the simulation.

Table 2. MQTT_SN messages exchanged and its length.

MQTT_SN Type of Message	Length (Bytes)
MQTT_SN Connect Command	96
MQTT_SN Register	88
MQTT_SN Subscribe Request	81
MQTT_SN Ping Request	92
MQTT_SN Publish Message	96

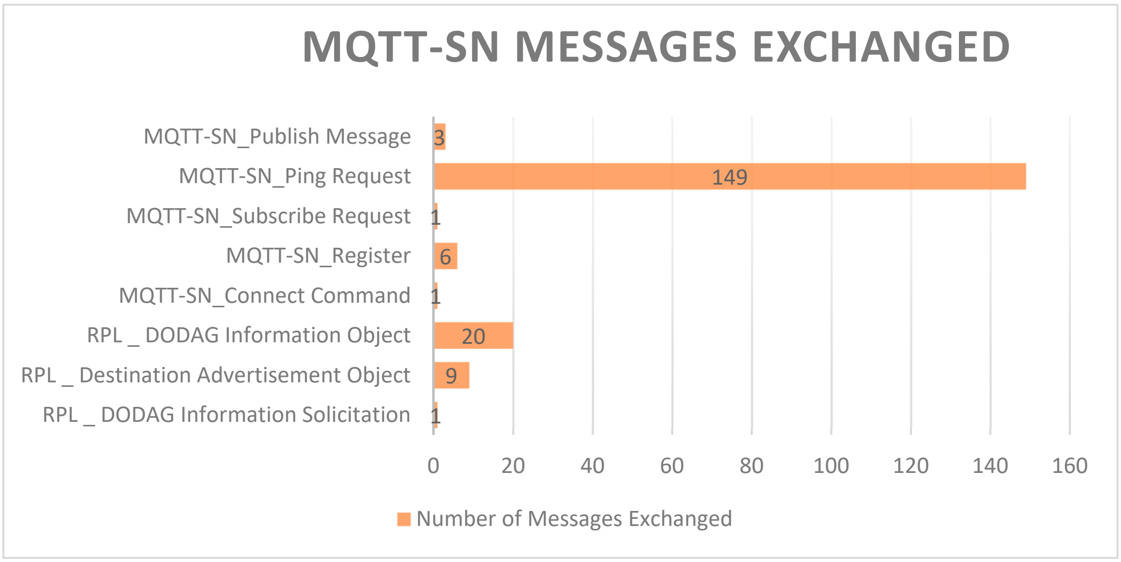

Figure 8. MQTT_SN exchanged messages during the simulation.

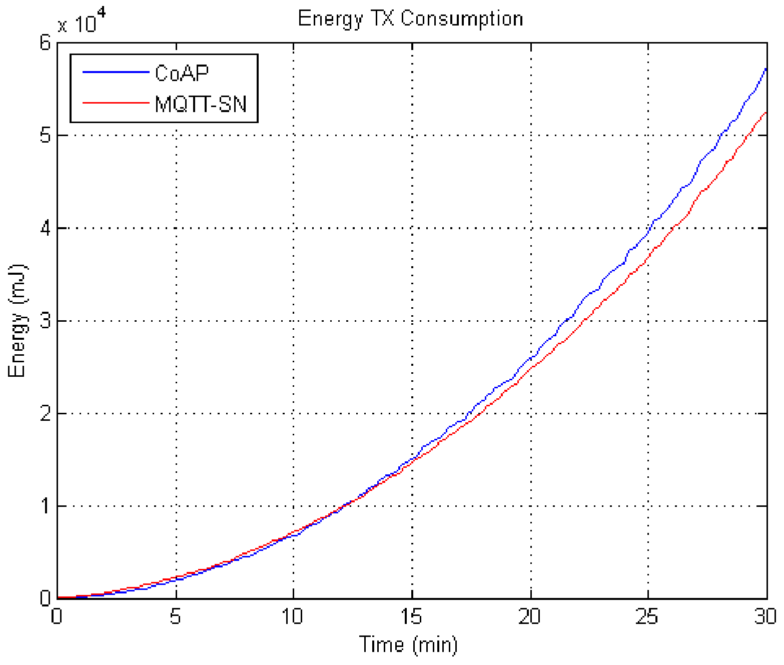

Figure 9. Energy Consumption comparison between CoAP and MQTT_SN.

6. Conclusions and Future Work

In this paper, we have studied the performance of the two main application layer protocols for IoT, CoAP, and MQTT_SN in terms of transmission energy consumption while analyzing message length. Section 4 shows that the performance of these two light application protocols is pretty similar when the total transmission energy consumption is obtained. Nevertheless, MQTT_SN is more efficient than CoAP since its client nodes have less complexity than CoAP clients.

The use of these protocols depends on the application because MQTT_SN requires an infrastructure that implies having the broker system allocated. This demand is inconvenient when the network is already deployed and there is no availability to provide such requirements. However, the implementation of CoAP in terms of infrastructure is simpler because it can work without it, and this could be an advantage.

As mentioned previously, CoAP can run over UDP using DTLS. The use of this protocol, with or without security mechanisms, is up to the application. In future work, it would be interesting to analyze the performance of CoAP when using DTLS in terms of energy consumption and message length. It would also be interesting to take into account the delay and bandwidth measurements. Finally, this paper compares the CoAP and MQTT_SN protocols. There are other, less used, application layer protocols for constrained devices, such as XMPP [37], AMQP [38], and DDS [39]. Therefore, it would be interesting to evaluate the performance of these alternative application protocols in comparison to CoAP and MQTT_SN.

Author Contributions: C.G.-R. and C.C. participated in the conceptualization of this paper and the definition of the methodology. M.M. participated in the simulation tools and scenarios, test bed, and in obtaining the results. C.G.-R. and C.C. supervised this part of the work. M.M. participated in the writing of the original draft of the paper. C.G-R. contributed to reviewing and editing the final version of the paper. C.C. participated in funding acquisition.

Funding: This research was partially funded by the Ministry of Economy and Competitiveness through TEC2017-84197-C4-1-R, and TEC2014-54335-C4-2-R.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Karagiannis, V.; Chatzimisios, P.; Vazquez-Gallego, F.; Alonso-Zarate, J. A Survey on Application Layer Protocols for the Internet of Things. *Trans. IoT Cloud Comput.* **2015**, *3*, 11–17.
2. Al-Fuqaha, A.; Guizani, M.; Mohammadi, M.; Aledhari, M.; Ayyash, M. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 2347–2376.
3. Iova, O.; Picco, P.; Istomin, T.; Kiraly, C. RPL, the Routing Standard for the Internet of Things … Or Is It? *IEEE Commun. Mag.* **2016**, *54*, 16–22.
4. Shelby, Z.; Bormann, C. *6LoWPAN: The Wireless Embedded Internet,* 2nd ed.; John Wiley & Sons Ltd.: Chichester, UK, 2009.
5. Molisch, A.F.; Balakrishnan, K.; Chong, C.C.; Emami, S.; Fort, A.; Karedal, J.; Kunisch, J.; Schantz, H.; Schuster, U.; Siwiak, K. IEEE 802.15. 4a channel model-final report. *IEEE P802* **2004**, *15*, 1–36.
6. Ghosh, A.; Ratasuk, R.; Mondal, B.; Mangalvedhe, N.; Thomas, T. LTE-advanced: Next-generation wireless broadband technology [Invited Paper]. *IEEE Wirel. Commun.* **2010**, *17*, 10–22.
7. Kürschner, C.; Condea, C.; Kasten, O.; Thiesse, F. Discovery Service Design in the EPCglobal Network. *Internet Things* **2008**, *4952*, 19–34.
8. Gomez, C.; Paradells, J. Wireless home automation networks: A survey of architectures and technologies. *IEEE Commun. Mag.* **2010**, *48*, 92–101.
9. Shelby, Z.; Bormann, C.; Hartke, K. RFC7252: The Constrained Application Protocol (CoAP). Available online: https://tools.ietf.org/html/rfc7252 (accessed on 1 March 2019).
10. Castellani, A.P.; Gheda, M.; Bui, N.; Rossi, M.; Zorzi, M. Web Services for the Internet of Things through CoAP and EXI. In Proceedings of the IEEE International Conference on Communications Workshops (ICC), Kyoto, Japan, 5–9 June 2011; pp. 1–6.
11. Mq Telemetry Transport. Available online: http://mqtt.org/ (accessed on 1 March 2019).
12. Richardson, L.; Ruby, S. *RESTful Web Services*, 1st ed.; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2008.
13. MQ Telemetry Transport (MQTT) V3. 1 Protocol Specification. Available online: http://Www.Ibm.Com/Developerworks/Webservices/Library/Ws-Mqtt/Index.Html (accessed on 1 March 2019).
14. Portocarrero, T.; Rueda, J.M.; Smith, J. Similitudes y diferencias entre Redes de Sensores Inalámbricas e Internet de las Cosas: Hacia una postura clarificadora. *Rev. Colomb. Comput.* **2017**, *18*, 58–74.
15. Lesjak, C.; Hein, D.; Hofmann, M.; Maritsch, M.; Aldrian, A.; Priller, P.; Ebner, T.; Ruprechter, T.; Pregartner, G. Securing smart maintenance services: Hardware-security and TLS for MQTT. In Proceedings of the IEEE 13th International Conference on Industrial Informatics (INDIN), Cambridge, UK, 22–24 July 2015; pp. 1243–1250.
16. Cabrejas, I.F.; Vazquez, M.C.C. Performance Evaluation of Constrained Application Protocols in Wireless Sensor Networks. Master's Thesis, University Carlos III of Madrid, Madrid, Spain, 2018.
17. Stanford-Clark, A.; Troung, H.L. *MQTT for Sensor Networks (MQTT-SN) Protocol Specification*; Version 1.2; International Business Machines Corporation (IBM): Armonk, NY, USA, 2013.
18. Bandyopadhyay, S.; Bhattacharyya, A. Lightweight Internet protocols for web enablement of sensors using constrained gateway devices. In Proceedings of the 2013 International Conference on Computing, Networking and Communications (ICNC), San Diego, CA, USA, 28–31 January 2013; pp. 334–340, doi:10.1109/ICCNC.2013.6504105.
19. De Caro, N.; Colitti, M.; Steenhaut, K.; Mangino, G.; Reali, G. Comparison of two lightweight protocols for smartphone-based sensing. In Proceedings of the 2013 IEEE 20th Symposium on Communications and Vehicular Technology in the Benelux (SCVT), Namur, Belgium, 21 November 2013; pp. 1–6, doi:10.1109/SCVT.2013.6735994.
20. Levä, T.; Mazhelis, O.; Suomi, H. Comparing the cost-efficiency of CoAP and HTTP in Web of Things applications. *Decis. Support Syst.* **2014**, *63*, 23–38, doi:10.1016/j.dss.2013.09.009; ISSN 0167-9236.
21. Amaran, M.H.; Noh, N.A.M.; Rohmad, M.S.; Hashim, H. A Comparison of Lightweight Communication Protocols in Robotic Applications. *Procedia Comput. Sci.* **2015**, *76*, 400–405, doi:10.1016/j.procs.2015.12.318; ISSN 1877-0509.
22. Iglesias-Urkia, M.; Orive, A.; Urbieta, A. Analysis of CoAP Implementations for Industrial Internet of Things: A Survey. *Procedia Comput. Sci.* **2017**, *109*, 188–195, doi:10.1016/j.procs.2017.05.323.

23. Thota, P.; Kim, Y. Implementation and Comparison of M2M Protocols for Internet of Things. In Proceedings of the 2016 4th Intl Conf on Applied Computing and Information Technology/3rd Intl Conf on Computational Science/Intelligence and Applied Informatics/1st Intl Conf on Big Data, Cloud Computing, Data Science & Engineering (ACIT-CSII-BCD), Las Vegas, NV, USA, 12–14 December 2016; pp. 43–48, doi:10.1109/ACIT-CSII-BCD.2016.021.
24. Tandale, U.; Momin, B.; Seetharam, D.P. An empirical study of application layer protocols for IoT. In Proceedings of the 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS), Chennai, India, 1–2 August 2017; pp. 2447–2451, doi:10.1109/ICECDS.2017.8389890.
25. Liri, E.; Singh, P.K.; Rabiah, A.B.; Makhijani, K.K.K.; Ramakrishnan, K.K. Robustness of IoT Application Protocols to Network Impairments. In Proceedings of the 2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), Washington, DC, USA, 25–27 June 2018; pp. 97–103, doi:10.1109/LANMAN.2018.8475048.
26. Larmo, A.; Carpio, F.D.; Arvidson, P.; Chirikov, R. Comparison of CoAP and MQTT Performance over Capillary Radios. In Proceedings of the 2018 Global Internet of Things Summit (GIoTS), Bilbao, Spain, 4–7 June 2018; pp. 1–6, doi:10.1109/GIOTS.2018.8534576.
27. van der Westhuizen, H.W.; Hancke, G.P. Practical Comparison between COAP and MQTT—Sensor to Server level. In Proceedings of the 2018 Wireless Advanced (WiAd), London, UK, 26–28 June 2018; pp. 1–6, doi:10.1109/WIAD.2018.8588443.
28. van der Westhuizen, H.W.; Hancke, G.P. Comparison between COAP and MQTT—Server to Business System level. In Proceedings of the 2018 Wireless Advanced (WiAd), London, UK, 26–28 June 2018; pp. 1–5, doi:10.1109/WIAD.2018.8588445.
29. Gündoğran, C.; Kietzmann, P.; Lenders, M.; Petersen, H.; Schmidt, T.C.; Wählisch, M. NDN, CoAP, and MQTT: A comparative measurement study in the IoT. In Proceedings of the 5th ACM Conference on Information-Centric Networking (ICN '18). ACM, New York, NY, USA, 21–23 September 2018; pp. 159–171, doi:10.1145/3267955.3267967.
30. Larmo, A.; Ratilainen, A.; Saarinen, J. Impact of CoAP and MQTT on NB-IoT System Performance. *Sensors* **2019**, *19*, 7, doi:10.3390/s19010007.
31. Contiki: The Open Source OS for the Internet of Things. Available online: http://www.contiki-os.org/ (accessed on 1 March 2019).
32. Davies, J. *MSP430 Microcontroller Basics*, 1st ed.; Elsevier Ltd.: Linacre House, Jordan Hill, Oxford OX2 8DP, UK, 2008.
33. Texas Instruments. *MSP430x2xx Family User's Guide*; Texas Instruments: Dallas, Texas, 2013; pp. 23–27.
34. Chipcon Products. *CC2420: 2.4 GHz IEEE 802.15.4/ZigBee-Ready RF Transceiver*; Texas Instruments: Dallas, Texas, 2019; pp. 1–85.
35. ZOLERTIA™. *Z1 Datasheet*; v. 1.1, 2010. Available online: www.zolertia.com (accessed on 1 March 2019).
36. GitHub: Contiki-os/contiki/apps/powertrace**.** Available online: https://github.com/contiki- os/contiki/blob/master/apps/powertrace/powertrace.c (accessed on 1 March 2019).
37. Saint-Andre, P. *Extensible Messaging and Presence Protocol (XMPP): Core*; Internet Engineering Task Force (IETF): Fremont, CA, USA, 2011.
38. Oasis. *Advanced Message Queuing Protocol (AMQP)*; Version 1.0; Adv. Open Std. Inf. Soc. (OASIS): Burlington, MA, USA, 2012.
39. Data Distribution Services Specification. Available online: http://www.omg.org/spec/DDS/1.2/ (accessed on 1 March 2019).