Practical Comparison between COAP and MQTT - Sensor to Server level

Henri W. van der Westhuizen and Gerhard P. Hancke
Department of Electrical, Electronic and Computer Engineering
University of Pretoria
South Africa

***Abstract*—Sensor nodes need lightweight communication protocols that enable connectivity to a network from which commands can be retrieved and data can be uploaded. These systems need to ensure that constrained devices will function just as well as non-constrained devices. The Constrained Application Protocol (COAP) and Message Queuing Telemetry Transport (MQTT) are the two most common protocols. Two systems were built to test these with regards to communication delay and network traffic. It was found that COAP has on average smaller packet sizes and no keepalive messages. MQTT has lower communication delay and is simpler to implement. COAP is well designed to interface with systems that work on the HyperText Transfer Protocol (HTTP) while MQTT is designed to work well with automatically forward messages to multiple clients without any additional configuration. MQTT also has Quality of Service (QoS) built into the design to ensure message delivery.**

***Index Terms*—COAP, MQTT, Internet of Things, Sensor Systems**

I. Introduction

One of the problems with IoT sensors, is dealing with heterogeneous systems. These systems may have different architectures, resource limits and interfaces. Systems acting independently of the sensors become very useful [1]. These systems include MQTT (Message Queue Telemetry Transport) and COAP (Constrained Application Protocol). MQTT and COAP work in very different ways and have both advantages and disadvantages. One of the most important aspects regarding these protocols is that they are lightweight, making them ideal for resource-constrained devices. These systems need to provide a communication stack that is able to provide a power-efficient, reliable internet connection for resource-constrained devices. While maintaining a low packet size and complexity, it must still be able to provide end-to-end security for the system [2], [3].

Abstracting the data between business systems and sensors is another set of challenges faced. Business systems should not be interfacing directly with sensors. This limits the calls made to the sensors. This is important as the more calls the sensor needs to process the more resources are used to process these requests. The most critical is the power supply. For each message and reply to the message, the Network Interface Device needs to be used, which may very quickly drain the power supply. An example of this abstraction could be a system that queries the temperature data from a sensor every 10 seconds. This does not seem like a huge amount but if there are thousands of sensors and multiple systems, the data flow on the sensor network drastically increases. If the temperature remains constant for an extended time period, querying the sensor every few seconds results in a waste of resources. An improved infrastructure would be to have a middleman system that only updates stored values when the sensor interrupts the middleman system with new data. When the system requests data, the middleman system then sends the stored data. A middleman system could also trigger actions in the system if there is a drastic change in sensor data / receiving data falling outside a set of limits, which the system needs to be made aware of. The middleman can then make the system aware of the breaching of limits.

Another improvement drive is being made to the Virtualization of network interface, or more specifically sensor node interfaces, as well as virtualizing the entire sensor node. This allows the sensor to only interface with its virtual counterpart while all other systems interface with the virtual system and not the sensor directly. This may also have a major impact on data aggregation and distribution [4], [5].

It is a very simple system, however, it is clear that middleman systems such as MQTT and COAP can be beneficial to the total system.

In this paper, MQTT and COAP will be compared with regards to sensor network systems. The main comparison areas will be network delay and amount of network traffic.

II. Theoretical / Conceptual Formulation

A. Background

Many of the IoT systems use a centralized controller to manage all the IoT devices. This works well as it provides a central point for data capture and distribution, as well as a central point from where to provide security, if done correctly. These networks consist of two forms of devices: unconstrained and constrained. The standard unconstrained system uses TCP (Transmission Control Protocol) and HTTP (HyperText Transfer Protocol). Constrained systems, which are designed for resource-constrained devices, use UDP (User Datagram Protocol) and COAP or MQTT [6]. Many of the protocols still support IPv4, however, there is more support and a bigger move towards IPv6 and for sensor nodes 6LoWPAN [3], [7].

There is an increased need for IoT mechanisms that minimize the need for human intervention when dealing with

configuration and maintenance of sensor systems. Here COAP and MQTT are ideal. COAP only requires the address of the server and the network connection to access data. MQTT takes this a step further. Similarly to COAP, MQTT can simply upload data by publishing to a topic. If the topic does not exist, it will create it, however, no other devices will be listening to this topic. Where it significantly differs from COAP, when subscribed to a topic, it will automatically receive any data being published to that topic. NO additional configuration needs to be done to receive / GET commands from the controller [8].

1) UDP and TCP: These are the two most common transport layers protocols. Each of these protocols has advantages and disadvantages. TCP allows for message acknowledgement. Message delivery is thus enforced by the protocols without any additional enforcement required from the system. This message delivery acknowledgement increases the complexity of the protocol, which in turn uses more resources. UDP is a lightweight protocol that allows messages to be transmitted similarly to TCP, however, message delivery is not ensured. This protocol works well for constrained devices. Most of the older Application protocols are based on TCP such as most REST architectures, HTTP and HTTPS. For the newer IoT application layer protocols such as MQTT and COAP, UDP is utilized [6], [9].

2) MQTT: MQTT is based on a publish and subscribe architecture. Clients that use these protocols will publish to the broker to upload data, and subscribe to the broker to download commands. MQTT will automatically push received messages to all subscribed clients. If multiple clients are subscribed to a topic, the broker will publish any received messages to all the clients. The advantages of this are that the system does not care how many and or if any clients are connected, it will still push any received messages to all the subscribed clients without any configuration. Another advantage is that when a new sensor is connected, it simply needs to connect to a network and subscribe to the topic to receive any commands. For publishing data, a sensor can easily just publish to the server. The other subscribed systems will then receive this new information without any need for configuration.

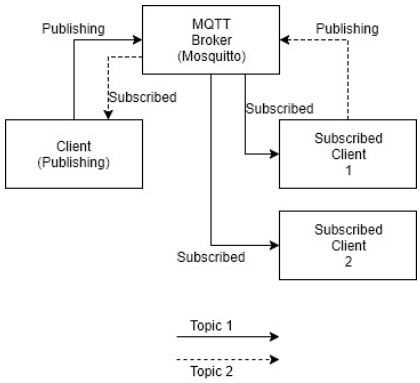

Fig. 1. MQTT model

3) COAP: COAP works very similarly to most standard servers using GET, PUT, POST and DELETE requests to access and manage data. COAP differs to the standard HTTP in that it is a lightweight system, making it ideal for resource-constrained devices. This is a system that is able to be used by not just IOT devices but also older systems more designed for HTTP, as it maps to HTTP making it compatible with HTTP [10]. This is a protocol that is being standardized by the IETF.

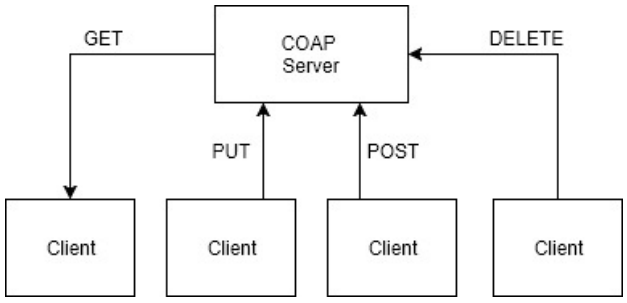

Fig. 2. COAP model

COAP uses DTLS (Datagram Transport Layer Security) to secure the connections to the server. DTLS was designed earlier for systems with considerably more resources compared to the general type of sensor node. Therefore, either the protocol had to be adjusted, or a different securing methodology was required. One such design was done making a lightweight DTLS system. This was shown to have some promising results by reducing the transmitted bytes of a DTLS message without compromising the end-to-end security of the system [11]. In addition to reducing the size, there are other designs as well. This includes a public key, extending DTLS for multicast communication and reducing size and complexity of implementations [12]. To ensure only the authorized nodes accessing the system, an authorization system needs to be implemented. It is important that this system has a low processing load, provides fine-grained customization of the security policies and is scalable for IoT [10].

III. Design

To compare COAP and MQTT, two systems were designed to do two different tests. The following sections give a visual representation and an explanation of the design of the systems. The tests compared the communication delay and network traffic for the two core communication protocols [13].

A. MQTT

1) MQTT Broker: The MQTT broker was implemented using the Mosquitto MQTT broker [14]. This is a broker which is regularly used as a broker and interfaces easily with Eclipse Paho enable clients.

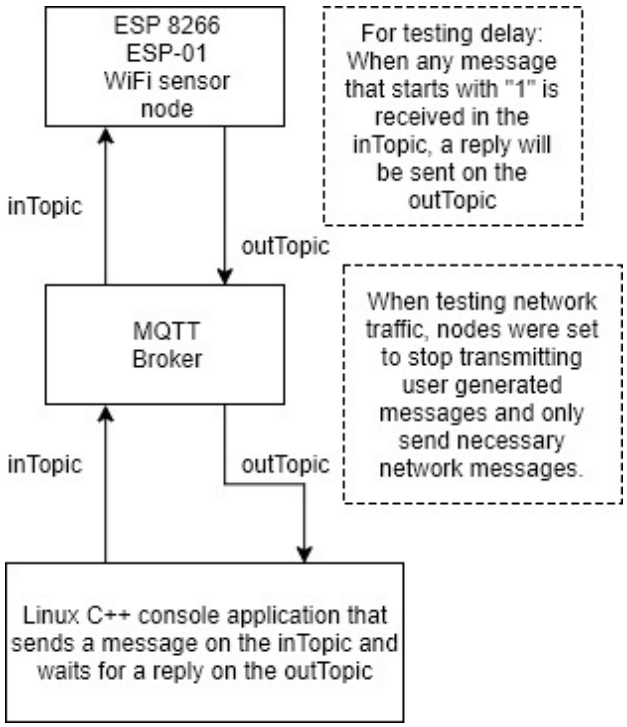

Fig. 3. MQTT Broker model

As seen in fig. 3, the MQTT broker is responsible for sending the messages received from the Linux application on the inTopic, to the ESP sensor node. When the ESP sensor node then replies on the outTopic, these messages are forwarded to the Linux application. If there were more ESP sensor nodes / Linux applications, the broker would replicate the message to the subscribed devices as well [13].

2) MQTT Clients: Eclipse Paho MQTT clients where used. This is an open source library that has resources for Android, Arduino, Windows and Linux. The wide variety made it possible to test with multiple types of clients. Four clients were used to test connectivity to the broker, but extensive testing was done using two clients. The two clients were an ESP 8266 ESP-02 WiFi board and an oDroid (Linux Lubuntu v14.04) [13].

A browser-based MQTT client called MQTTLens was used to test whether connections could be made to the MQTT broker. It is a simple to use visual output to test sending and receiving data to and from other clients. An android application was also used to test if the connection could be made to the broker. The two tested clients were chosen as these had simple setups and allowed for easy and effective testing of the broker.

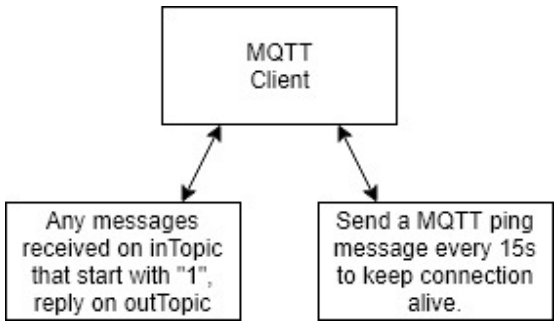

Fig. 4. MQTT Client model

In fig. 4 it can be seen that the ESP sensor node will reply to any messages received on the ”inTopic” Topic, if that message starts with a ”1”. Irrespective of when the last message was received/sent, the PubSubClient Arduino library will send a Ping message and wait for a reply from the broker. This is a keep-alive message to ensure the connection between the Broker and client is kept alive. For different clients, this timer may differ. For example, the MQTTLens browser application has a keep alive Ping timer of 120s.

B. COAP

1) COAP Server: For the COAP server, The open source library Californium was used. This is a commonly used server and provides capabilities for many platforms. In the Firefox browser the Copper plug-in app could be used. Libraries are provided for Android, embedded systems such as Arduino and Linux.

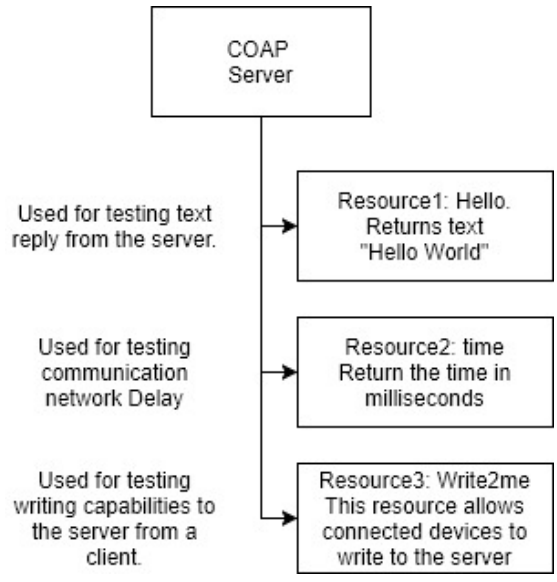

Fig. 5. COAP Server model

The COAP server consists of 3 unique resources used for testing which is shown in fig. 5. Resource 1 was used primarily for two purposes. First, to test whether a connection can be made to the server and data can be retrieved. The second was

to test whether the sensor node can receive and decode text-based data. Resource 2 was used for communication delay testing. Resource 2 returns the time in milliseconds as an integer, allowing for the testing of delays with a resolution of 1ms. Finally, Resource 3 was used to test writing to the server. The sensor nodes could write the calculated delay to the server to keep a record if necessary.

2) COAP Clients: For the COAP clients, a browser-based Firefox plug-in called Copper was used to test connections to the COAP server as well as an Android client. For the majority of the testing, Linux and Embedded C libraries were used. This was implemented on an oDroid running Lubuntu 14.04, and an ESP 8266 ESP-01 WiFi board.

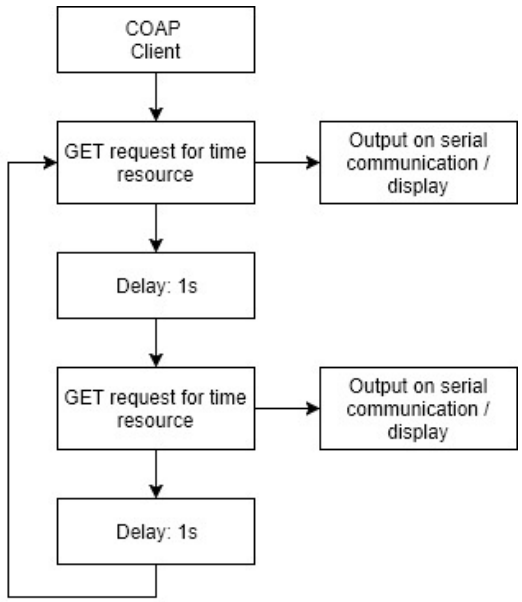

Fig. 6. COAP Client model

The COAP clients tested the communication delay by querying the time resource. It would then return the time in milliseconds. The client would then wait for 1s. This is done because the function that returns the time in milliseconds needs some time before it can be called again. Constant querying (querying with less than 50ms delay) also resulted in some packet errors, as it caused multiple packets to be handled in the same callback function on the client. The combination of packets in the callback function was a problem as it only displayed the data from one of the received packets.

After the 1s delay, the client would send a request to the time resource again. Taking the second received time and subtracting the first, gave a result of 1s plus the communication time delay. This was then displayed on the serial output, connected to a computer which was used to program the ESP device and show the serial output visually.

IV. Results

Tables I,II and III contains the acquired results when testing the designed systems.

TABLE I
COAP AND MQTT COMMUNICATION DELAY RESULTS FOR BOTH LOCAL AND EXTERNAL NETWORKS

	Network	
	Local	**External**
	ms	ms
MQTT	1	4
COAP	1	6

Both MQTT and COAP had very similar response times when both the client and the Broker/server are on the same network. This is to be expected as the Access Point is just redirecting. When the client is connected to an external network and needs to access the broker/server through the Internet, there is a greater time delay. These packets are still very small and therefore the delay is small. It can be seen that MQTT does outperform COAP. With large packets, this would become more apparent.

TABLE II
FREQUENCY PACKETS ARE BEING SENT

	MQTT	**COAP**
Freq of packets	1 every 15s or 120s	0

Unlike MQTT, COAP does not have a keep alive timer. The client connects and disconnects when accessing data. The only time COAP therefore needs to send data is when accessing the data. MQTT, on the other hand, has a keepalive timer. This ensures that the client and broker connection is kept alive for data transmission. The ESP sensor node uses the PubSubClinet library which has a keep-alive timer of 15s and this cannot be changed. The MQTTLens browser application can have a variable keep alive timer, however, it was kept to the default 120s.

TABLE III
COAP AND MQTT RESULTS FOR NETWORK TRAFFIC

	To Server	**From Server**
	MQTT	
Subscribe	65	59
Publish	67	59
Ping	56	56
	COAP	
Discovery	63	206
GET	53	63
PUT	61	53

The results in table III show the packet sizes captured using Wireshark, MQTTLens and Copper. The last two mentioned are browser-based applications for MQTT and COAP. The MQTT subscribed to a topic called: ”test”. Publishing was done to the same topic and the text ”Hello” was used. As MQTT uses keep alive messages, the size of those messages has been captured as well. The Discovery COAP message is used to discover available resources. The reply is rather large in comparison to the other messages. This, however, only needs to be done once, and only if the sensor node would need to see if a particular resource has been created. The GET request requested the data for the ”time” resource. The reply from the server for the time resource was ”1512225590294”. For the PUT request, a request was done to put the following text: ”Hello” onto the ”Writeme!” resource.

V. Discussion

Both MQTT and COAP have been tested with regards to communication delay as well as network traffic. The tests were run on resource-constrained devices as well as non-constrained devices and both performed nearly identically. For COAP the open source library Californium was used to host a COAP server. The clients used parts of the same library such as the embedded C and Java library versions. For the MQTT, the mosquitto [14] open source libraries were used and eclipse PAHO libraries were used for the clients. In addition to the other clients, Browser-based applications such as MQTTLens in Chrome and Copper in Firefox were used to test the broker/server.

It was found that the MQTT had less communication delay than that of COAP. COAP had on average smaller packet sizes. If it is critical that resource usage such as battery power/data cellular usage must be kept to an absolute minimum, then COAP might be the better solution. This is due to of the smaller packet sizes and no keepalive messages. In another case, COAP might be the better solution if interfacing is required with an older system which was designed for / uses HTTP. COAP can be interfaced to the same systems supporting HTTP. When simplicity and lower communication delay is critical MQTT might be the better option. MQTT is certainly a more effective solution when multiple sensors are receiving the same commands. The functionality of MQTT to automatically forward messages to all subscribed sensors makes MQTT the better solution for this type of problem. This would be a common problem in smart homes. Sensors, such as smart lights, could subscribe to multiple topics, such as one for all lights and one for individual lights. Choosing which system to use is very dependant on the requirements of the system. In some cases, the solution may require a combination of the two. An MQTT system for the sensor network and a COAP system for interfacing with other systems such as Business Systems. MQTT also have QoS capabilities to ensure that messages are delivered [15], [16].

For this paper only WiFi was used. This could however greatly be expanded into Bluetooth and other forms of internet/network connectivity [17], [18]. MQTT could still be used as long as the underlying system can send and receive messages [19]. There is still a very large amount of work and research necessary before IoT will become standard and a common place in our lives. A large part of this requires standardizing and implementing the required infrastructure [13].

Security is one of the main concerns in IoT. Many solutions will either use pre-shared keys or Public key encryption. This is still problematic. Using pre-shared keys is not scalable. If the key changes, it needs to be given to all the sensor nodes securely. Public key encryption may be too resource intensive for most small sensor nodes which are resource constrained. A security solution using symmetric keys was designed, proving to be lightweight enough to be used by constrained devices and is scalable to a large number of devices [20], [21].

VI. Practical Applications

IoT devices are making a huge impact in many different fields. These devices will become very common in almost all areas of life. This may include almost anything such as home automation, smart cars, digital health, industrial automation, electricity and remote monitoring [5], [6], [21], [22]. MQTT might be the optimal solution for home automation. Sensors can be added and removed as needed without any additional configuration to the system. The sensors can automatically connect to a network and publish/subscribe to the necessary topics [13].

There is also an opportunity to use both MQTT and COAP for connected cars to capture data from the car sensors and send it to cloud systems. From here it can be further accessed by other systems without a need to directly connect to the smart car.

VII. Conclusion

Both COAP and MQTT are designed for resource constrained devices but support a wide variety of clients including microprocessors, windows applications and browser-based applications. There are many open-source libraries available for these protocols, for this paper mosquitto [14] and PAHO libraries were used. For sensor network MQTT provides the simplest and most effective flow of data by using a publish and subscribe data flow. Should this data need to be accessed outside of the sensor network it should be combined with COAP for HTTP support. The smaller packet sizes and lack of keep-alive messages makes COAP an effective protocol for devices with strict power constraints. When receiving data or controlling actions of individual / groups of sensors, MQTT provides an effective solution which also provides a smaller network delay. For this system to become a viable system, security would still need to be added. This will have an impact on the packet sizes and processing time required to encrypt and decrypt the packets. This paper in conjunction with another paper looking at the server / broker to business level systems provides a good theoretical model for sensor to business level systems.

References

[1] C. P. Kruger, A. M. Abu-Mahfouz, and G. P. Hancke, "Rapid prototyping of a wireless sensor network gateway for the internet of things using off-the-shelf components," in *2015 IEEE International Conference on Industrial Technology (ICIT)*, March 2015, pp. 1926–1931.

[2] J. Granjal, E. Monteiro, and J. S. Silva, "Security for the internet of things: A survey of existing protocols and open research issues," *IEEE Communications Surveys Tutorials*, vol. 17, no. 3, pp. 1294–1312, thirdquarter 2015.

[3] M. R. Palattella, N. Accettura, X. Vilajosana, T. Watteyne, L. A. Grieco, G. Boggia, and M. Dohler, "Standardized protocol stack for the internet of (important) things," *IEEE Communications Surveys Tutorials*, vol. 15, no. 3, pp. 1389–1406, Third 2013.

[4] L. Mainetti, V. Mighali, and L. Patrono, "A software architecture enabling the web of things," *IEEE Internet of Things Journal*, vol. 2, no. 6, pp. 445–454, Dec 2015.

[5] B. Cheng, L. Cui, W. Jia, W. Zhao, and P. H. Gerhard, "Multiple region of interest coverage in camera sensor networks for tele-intensive care units," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 6, pp. 2331–2341, Dec 2016.

[6] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 22–32, Feb 2014.

[7] Z. Sheng, H. Wang, C. Yin, X. Hu, S. Yang, and V. C. M. Leung, "Lightweight management of resource-constrained sensor devices in internet of things," *IEEE Internet of Things Journal*, vol. 2, no. 5, pp. 402–411, Oct 2015.

[8] S. Cirani, L. Davoli, G. Ferrari, R. Lone, P. Medagliani, M. Picone, and L. Veltri, "A scalable and self-configuring architecture for service discovery in the internet of things," *IEEE Internet of Things Journal*, vol. 1, no. 5, pp. 508–521, Oct 2014.

[9] A. Betzler, C. Gomez, I. Demirkol, and J. Paradells, "Coap congestion control for the internet of things," *IEEE Communications Magazine*, vol. 54, no. 7, pp. 154–160, July 2016.

[10] S. Cirani, M. Picone, P. Gonizzi, L. Veltri, and G. Ferrari, "Iot-oas: An oauth-based authorization service architecture for secure services in iot scenarios," *IEEE Sensors Journal*, vol. 15, no. 2, pp. 1224–1234, Feb 2015.

[11] S. Raza, H. Shafagh, K. Hewage, R. Hummen, and T. Voigt, "Lithe: Lightweight secure coap for the internet of things," *IEEE Sensors Journal*, vol. 13, no. 10, pp. 3711–3720, Oct 2013.

[12] S. L. Keoh, S. S. Kumar, and H. Tschofenig, "Securing the internet of things: A standardization perspective," *IEEE Internet of Things Journal*, vol. 1, no. 3, pp. 265–275, June 2014.

[13] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2347–2376, Fourthquarter 2015.

[14] R. A. Light, "Mosquitto: server and client implementation of the MQTT protocol," *The Journal of Open Source Software*, vol. 2, no. 13, p. 265, may 2017. [Online]. Available: https://doi.org/10.21105/joss.00265

[15] A. Al-Fuqaha, A. Khreishah, M. Guizani, A. Rayes, and M. Mohammadi, "Toward better horizontal integration among iot services," *IEEE Communications Magazine*, vol. 53, no. 9, pp. 72–79, September 2015.

[16] Z. B. Babovic, J. Protic, and V. Milutinovic, "Web performance evaluation for internet of things applications," *IEEE Access*, vol. 4, pp. 6974–6992, 2016.

[17] C. A. Opperman and G. P. Hancke, "Using nfc-enabled phones for remote data acquisition and digital control," in *AFRICON, 2011*, Sept 2011, pp. 1–6.

[18] G. A. Akpakwu, B. J. Silva, G. P. Hancke, and A. M. Abu-Mahfouz, "A survey on 5g networks for the internet of things: Communication technologies and challenges," *IEEE Access*, vol. 6, pp. 3619–3647, 2018.

[19] D. C. Y. Vargas and C. E. P. Salvador, "Smart iot gateway for heterogeneous devices interoperability," *IEEE Latin America Transactions*, vol. 14, no. 8, pp. 3900–3906, Aug 2016.

[20] S. Raza, L. Seitz, D. Sitenkov, and G. Selander, "S3k: Scalable security with symmetric keys, dtls key establishment for the internet of things," *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 3, pp. 1270–1280, July 2016.

[21] B. Cheng, J. Zhang, G. P. Hancke, S. Karnouskos, and A. W. Colombo, "Industrial cyberphysical systems: Realizing cloud-based big data infrastructures," *IEEE Industrial Electronics Magazine*, vol. 12, no. 1, pp. 25–35, March 2018.

[22] C. H. Potter, G. P. Hancke, and B. J. Silva, "Machine-to-machine: Possible applications in industrial networks," in *2013 IEEE International Conference on Industrial Technology (ICIT)*, Feb 2013, pp. 1321–1326.