JJEE
Volume 6, Number 1, 2020
Pages 63-77
Jordan Journal of Electrical Engineering
ISSN (Print): 2409-9600, ISSN (Online): 2409-9619

Secure IoT Resources with Access Control over RESTful Web Services

Khalid Aloufi[1*], Omar Alhazmi[2]

[1, 2] College of Computer Science and Engineering, Taibah University, Madinah, Saudi Arabia
E-mail: koufi@taibahu.edu.sa

Received: November 27, 2019 *Revised: December 19, 2019* *Accepted: December 28, 2019*

***Abstract*—** With the Internet of Things (IoT), the number of connected devices on the internet has increased to billions, and this number is expected to grow exponentially in the coming years. Services and applications based on the IoT are expected to expand to cover more areas in the near future. Performance, connectivity and security are very important aspects of such an expansive environment. In order to enhance the performance and security of the IoT, a secure and cost-effective model for IoT applications that is based on Message Queue Telemetry Transport (MQTT) is proposed. This model addresses the IoT security challenges primarily by moving access control and data management from the MQTT broker to a fog server. The performance of the proposed model is validated by multiple metrics; and the obtained results show that it can be deployed successfully in the implementation of IoT applications to enhance both the IoT's security and performance.

***Keywords*—** Internet of Things; Message Queue Telemetry Transport; Access control; Security; Web service; Fog server

1. INTRODUCTION

Internet of Things (IoT) is essential for future internet integration into different services in smart homes or smart cities. With the advent of IoT, various new internet services became available. However, one of the main challenges of IoT services is security [1], for example, means of protecting home or personal data [2]. Therefore, access control is required for these data, and users and systems should have regulated data storage, processing and accessibility. Another challenge for IoT applications is that IoT devices are constrained, not high-end devices; therefore, application development should implement this security requirement over non high end devices. Using typical methods of controlling access to IoT devices will increase the overhead of traffic, and solutions will be required to adapt to the IoT requirements, such as the energy of processing power. When using the web; representational state transfer (REST) is one of the requirements for effective transactions. However, it is not suitable for the IoT because of the high overhead as a result of using web technologies. Therefore, for large IoT projects, there should be a middle layer between the IoT and the web.

Currently, IoT exchanges data with application protocols such as message queue telemetry transport (MQTT) using REST over the hypertext transfer protocol (HTTP) in its services [3, 4]. Crus et al. recommend the use of MQTT with user-managed access control (UMA) for IoT applications [5]. In MQTT, the connection with the broker can be secured with the CONNECT message. MQTT is a publish/subscribe application protocol in its simplest form. As mentioned, it can be secured with the CONNECT message. The basic sequence diagram in Fig. 1 shows the functionality of MQTT.

* Corresponding author
Article's DOI: 10.5455/jjee.204-1581015531

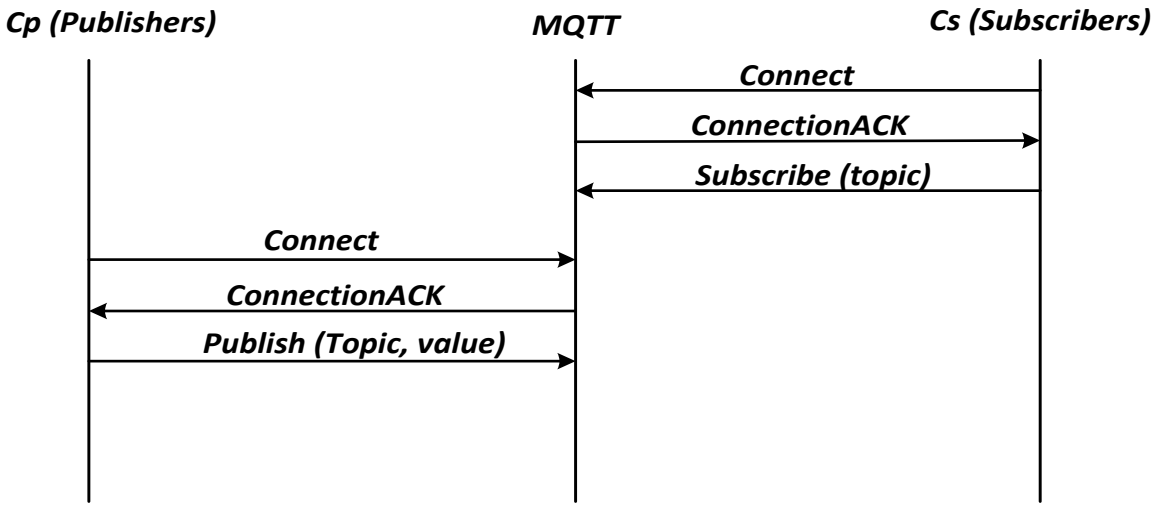

Fig. 1. The publish and subscribe transactions of MQTT.

Different methodologies are used to enhance the security of MQTT; such as proper use of the CONNECT message or the model developed by Crus et al., which used a security layer over MQTT using UMA. This work is based on the mechanism of MQTT since it is the most popular IoT application protocol, as shown in Fig. 2 [6].

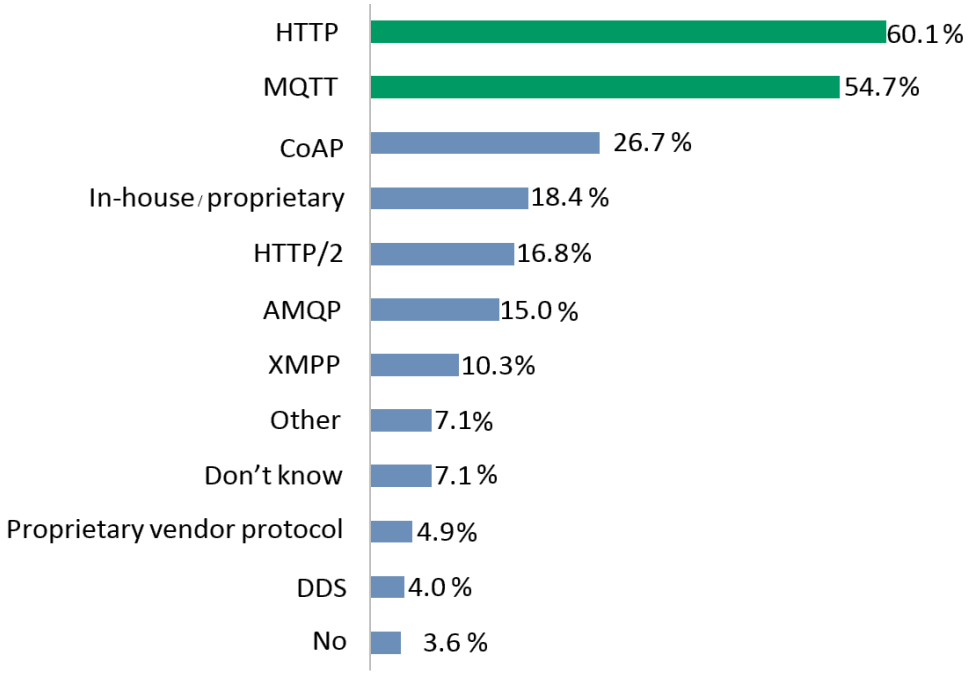

Fig. 2. The popularity of messaging protocols [6].

MQTT works over transmission control protocol/internet protocol (TCP/IP) and is an application protocol for the IoT, as shown in Fig. 3 [7]. MQTT is considered as a lightweight IoT protocol for smart city applications [8]. Smart city applications are expected to have great size of data sharing and MQTT is coping with such requirement by having low data overhead which is more adapted towards constrained devices and networks. Table 1 shows the IoT stack model using MQTT as the application server.

 © 2020 Jordan Journal of Electrical Engineering. All rights reserved - Volume 6, Number 1

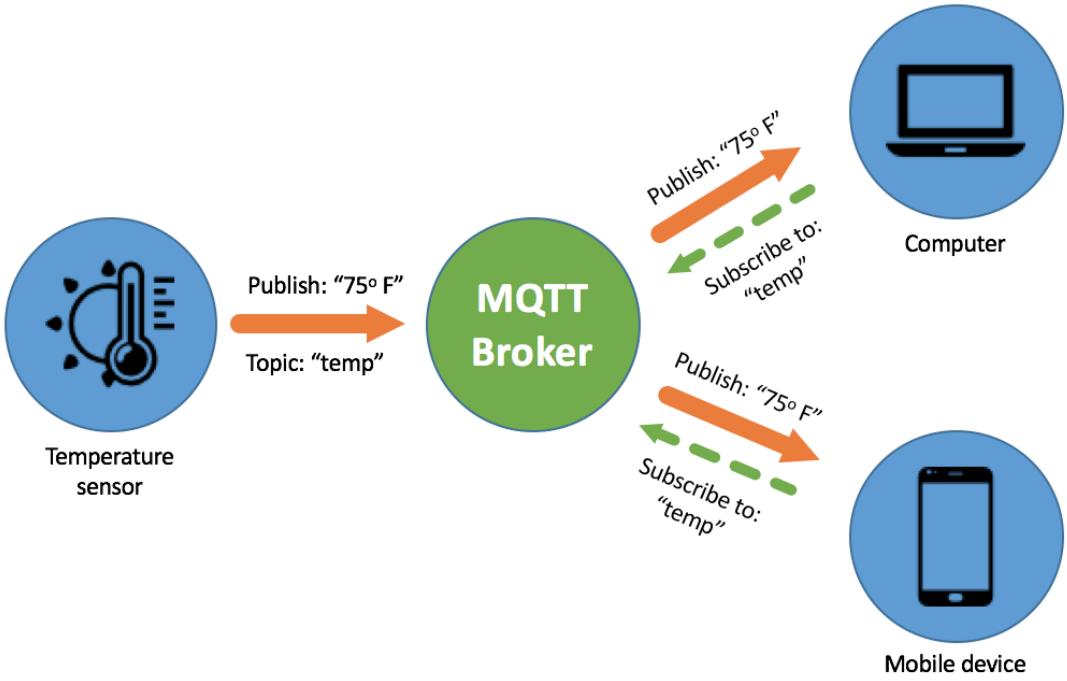

Fig. 3. The MQTT broker publishing and subscribing clients [7].

Table 1. IoT stack model.

Layer	OSI layers	TCP/IP	IoT (MQTT)
7	Application	Application	MQTT
6	Presentation		
5	Session		
4	Transport	Transport	TCP
3	Network	Internet	IP
2	Data link	Network access	IEEE 802.15.4 MAC
1	Physical		IEEE 802.15.4 PYS

This work presents a model to address IoT security challenges primarily by moving access control and data management from the MQTT broker to a fog server. To achieve this goal; a new model of the IoT is presented. The proposed model uses web technologies to authenticate users to access IoT resources by adding a fog server layer with RESTful web services, saving the IoT power and bandwidth and increasing the security level. Web security, which is mature enough to be trusted in current internet best practices, is adapted in the IoT design model to enhance security. Using web technologies will allow the connection of different devices from different vendors. In addition, database server is used in the fog server for data management of publish/subscribe messages. This model is validated by multiple metrics such as: the security level; the extra overhead of the performance added to the model; and the energy consumption level which became lower because the system has moved the overhead from the IoT model to the fog server, as will be shown later.

The main objectives of this work are:

- Designing and implementing IoT application based on MQTT application protocol
- Moving the authentication of users, subject management, including subscribing and publishing, from the MQTT broker to the fog server
- Increasing the functionality of the IoT system to serve more subscribers and publishers by moving the subscribing and publishing mechanism from the MQTT broker to the fog server which has more features compared to the MQTT broker.

2. RELATED WORK

Yokotani and Sasaki [9, 10] discussed the high overhead of using the HTTP and the performance advantage of MQTT and proposed compression to enhance MQTT for some topic lengths. Naik discussed four of the IoT systems protocols; MQTT, constrained application protocol (CoAP), advanced message queuing protocol (AMQP) and HTTP and presented a detailed analysis of the main characteristics of each protocol in terms of reliability, security and performance [11]. Sueda, Sato and Hasuike evaluated Web Socket, HTTP, and MQTT in terms of quality of service [12]. Alhazmi and Aloufi identified the main advantages of implementing the IoT for the fog-cloud rather than the cloud only; moreover, they previewed CoAP and MQTT protocols [13]. Aloufi presented the constrained devicesand network features and their requirements for effective design and implementation and recommended moving some functions from IoT devices, which are constrained, to an edge server and showed how the IoT could work in different network typologies [14]. Luis Cruz-Piris et al. presented an access control mechanism for IoT devices running over MQTT using UMA for managing the services of the IoT resources, controlled with communication procedures and presented a model based on the resources of the communication protocols [5]. They indicated that there are different access control mechanisms relevant for the IoT, such as UMA that is integrated with the web access mechanism. The discussed IoT models focused, in general, on lightweight and simple design models as their primary ground, which slows the development of other important required features, such as security [5].

IoT has different messaging application protocols, such as MQTT, CoAP and many more. One of the known messaging protocols is HTTP, which is used to manipulate and present the structure of web pages [15]. HTTP is widely used protocol. However, as mentioned earlier, it has high overhead for tiny devices; therefore, other protocols were developed, such as MQTT and CoAP.

MQTT, on the other hand, is developed specifically for IoT devices, for small sized overhead messages and processing requirements. MQTT has two implementations: one as a broker and the other as either a publisher or a subscriber as depicted in Fig. 3.

In contrast to MQTT, CoAP is an IoT protocol that is deigned to have enhanced features that are not available in MQTT [16]. While MQTT works over TCP, CoAP works over user datagram protocol (UDP), which decreases the transactions overhead. CoAP overcomes the reliability issues by integrating reliability small transaction at the application layer. Furthermore, CoAP has enhanced functionality, such as discovery and observation of resources using a pull push mechanism of messages.

CoAP is designed to work over communication standards, such as IEEE 802.15.4, that has a transmission rate of 250 kbps and the maximum message size is 127 B. An adaptation

layer is added to CoAP in case it is used over IPv6 in low-power wireless personal area networks (6LoWPAN) to support IPv6 [14].

Extensible Messaging and Presence Protocol (XMPP) is based on extensible markup language (XML) and widely known in different applications including IoT application based in the publish/subscribe mechanism [17]. However, the simplicity of using MQTT - compared to MQTT – for IoT application stems from the fact that XMPP is designed for publish/subscribe web applications; while MQTT is designed for publish/subscribe IoT applications.

AMQP is suitable for IoT applications; however, it has more overhead for the constrained IoT devices making MQTT a clear suitable solution [18]. Using data-local reconstruction layer (DLRL) and data-centric public-subscribe (DCPS) layer, data distribution service (DDS) transmits information between publishers and subscribers directly without a broker [19]. While earlier messaging protocols are application layer protocols, DDS is a session layer protocol. Also, DDS is decentralised service, where subscribers are subscribing to the publisher without a broker in the middle as in MQTT. Despite the number of features of using DDS, because of the the setup and configuration of the system, compared to other protocols, DDS is used by specific kind of Machine to Machine (M2M) applications.

MQTT has a fixed header of only 2B for the different commands, such as the CONNECT command which is used to establish connection from the MQTT client to the MQTT broker; which replays by the CONNACK. There are other commands as well, such as PUBLISH, which is used by The MQTT client to publish a subject. The client requests subscription using the command SUBSCRIBE; and the broker confirms subscription by the command SUBACK. Also, the client unsubscribes using the command UNSUBSCRIBE; and the broker replays with the command UNSUBACK. The client can cancel the connection using the command DISCONNECT.

3. SYSTEM MODEL

Each publishing client (C_P) has S_1 to S_n sensors as resources. The user can access the resources through the access control (AC) mechanism developed for this system. The AC will use the available protocols and web technologies to simplify the system's implementation and design. The sequence diagram for a simple system design is shown in Fig. 4, which also shows the system design model. The user utilizes one of the subscribed clients to access the resources through the system. The MQTT client subscriber (C_S) contacts the fog server, which in turn authenticate the request using the AC mechanism shown later. The request is granted access, and then; the MQTT broker receives the request and obtains the MQTT client publisher (C_P) data as resources. One of the authentication methods for resources is AC for web pages using the HTTP protocol. Following the client-server model, the request is received by the web server that responds appropriately.

In Fig. 4, the fog server's main job is to authenticate and authorize the user if it has the right privileges of using the web server. It also manages subscribe and publish of topics using a database server. The access to the system in MQTT is secured using secure session access. The direction of the publish messages is from left to right where the publishers (C_P) send messages to the MQTT broker, which in its turn moves the messages to the fog server.

The latter updates the database for the subject and its value. When there is a subscriber to a subject the fog server transmits the new messages to its subscribers. On other hand, the subscribers (C_S) send subscribe messages to the fog server to subscribe to a specific subject.

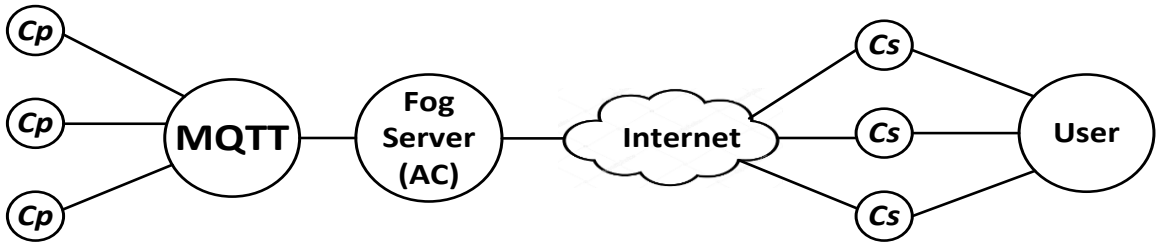

Fig. 4. System model.

The system in Fig. 5 shows a typical MQTT system without a fog server. Subscribers contact the MQTT broker directly over the cloud. This increases vulnerability to attack and increases the load above the limited abilities of the broker. Therefore, a fog server is required between the broker and the subscriber connections as shown in Fig. 6. The fog server in the model contains a web server and a database server; it acts as a subscriber for all the resources connected to the MQTT broker. The fog server is recommended as a middle-ware service between the IoT devices and cloud services, and it is accessed by users over the internet [7].

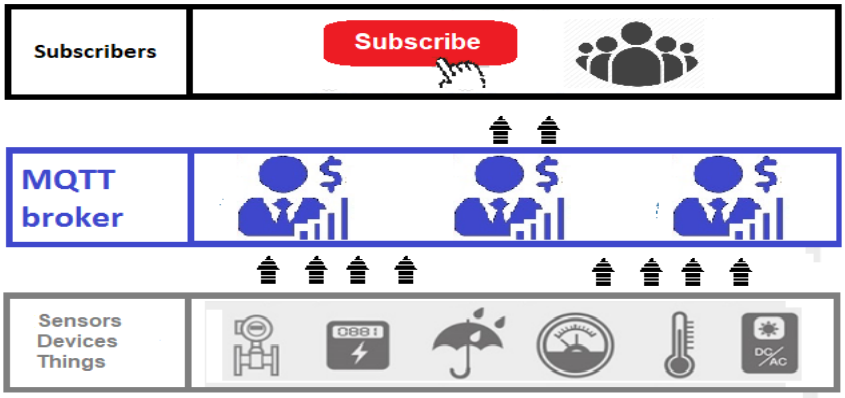

Fig. 5. Typical MQTT.

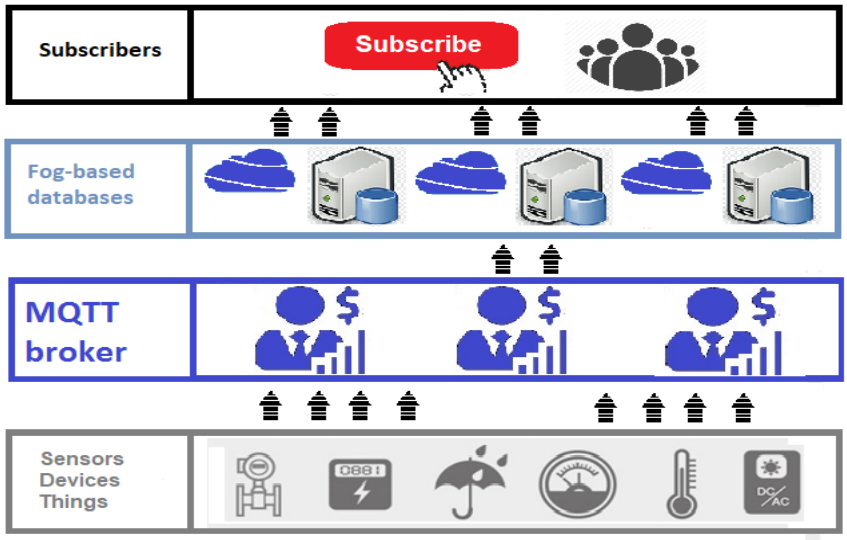

Fig. 6. MQTT with a fog-based database layer.

The other main part of the model is the MQTT broker, which can be considered as dynamic rather than a static implementation because the complexity of subscribers has moved to the fog server. The MQTT broker only passes published messages from C_P to the fog server. When the C_S connects to the fog server, the MQTT broker dynamic implementation has one code for each functionality for any IoT connected device. In regular implementations, the MQTT broker is static because each functionality requires specific code, which increases the complexity of the MQTT system implementation. MQTT broker is considered static when a code is required for every connected C_P or C_S. The main goal is to have the MQTT broker ready for plug and play. Dynamic services in IoT devices are very important for simple plug-and-play implementations, such as the Wi-Fi-manager network, although the Wi-Fi connection still requires a more high-performance simple connection [20-22].

In this model, there are many subjects from different publishers and one subscriber, which is the fog server. Therefore; the fog server is in the middle layer between the MQTT broker and its subscribers. The subscribers are connected to the fog server rather than connected to the MQTT broker directly. The fog server has a web server and a database of the subjects and their values. In this model, the MQTT system is connected to the internet through the fog server to help using web technologies on IoT devices.

An IoT device connects with the MQTT broker using a stateless connection [23] while having a stateful connection between the MQTT broker and the edge server. In a stateful connection between the edge server and the web users, either via a web browser or mobile applications, multiple sessions are considered the same. The cloud is used by users to access IoT resources after authentication and authorization with input-checking for enhanced security. The model between the user and the fog server follows the client-server model (C/S). The user is notified when any of the C_S gets any published data of any subject, to which it is subscribed. The MQTT broker is a publisher-subscriber application protocol with enhanced access control of resources managed by the fog server as a web agent. The resources - which are the publishers of sensor data - send information to the MQTT broker for each period, and their characteristics will depend on how the manufacturer has designed the device. The database contains the user credentials and the topic details as well as a list of the users (subscribers) and the devices (publishers).

The fog server works as an extension of the MQTT broker. Any actuator, such as a fan or LED light bulb, is a subscriber that is connected to one of two entities: the fog server or the MQTT broker. If it is connected to the MQTT broker directly, the broker rule - that has only one subscriber - is broken, and a customized and static configuration is required for the actuator. Therefore; the actuators are connected to the fog server directly as users and connected to the MQTT broker dynamically in accordance with the rule.

For the above reasons, the system is designed with two main functionalities: AC and data management in the fog server (see Fig. 4). The AC unit controls the C_S authentication and authorization using RESTful. The fog server takes the subscribe data management part from the MQTT broker. When the C_P publishes new data, the MQTT broker sends the new data to the fog server. Physically, the fog server is the only registered subscriber in the MQTT broker with all subscribers. Virtually, subscribers are connected to publishers through the MQTT broker over the fog server that saves the new data in the database server and

forwards it to the C_S. The fog server contains the database server to save the state of every publisher for any expected required processing, such as when a user requests the last ten messages from a specific publisher for a specific subject.

The fog server has a high-speed bandwidth connection with the internet to connect to the cloud. Additionally, it has a web server and database server for the subscriber and publisher. The MQTT in the model have a limited number of connections with the subscribers and the fog server. The fog server maintains a simple dynamic connection with different devices and users. When the C_P s publishes data to the MQTT broker, the MQTT broker sends the message one time to the fog server. The fog server then sends the message to several C_S. Therefore, for any published message, the IoT device sends the message one time to the MQTT broker, which also sends the message one time to the fog server. Therefore, the number of messages sent from the MQTT broker is kept minimal; one messages for any new subject value from the IoT devices. The fog server is a high end machine with high bandwidth connection, while the MQTT broker has limited energy and connection bandwidth. IoT devices have a limited bandwidth and a limited power, and therefore, any message transmission costs the IoT device some energy. Also, the security layer adds an extra overhead to messages. Therefore, keeping the messages' sizes - between the IoT device and the MQTT broker - to minimal saves the energy of the IoT devices and the MQTT broker. Similarly, the security layer in the fog server adds some overhead for messages; so, by moving the C_S management to the fog server, the bandwidth of the MQTT broker is saved and messages are kept for minimal size. The model using the fog server enhances the security between the MQTT broker and the world. The security between the IoT device and the MQTT broker and between the MQTT broker and the fog server is maintained by the embedded security in the MQTT.

Fig. 7 represents the sequence diagram for the subscription of users to a topic published by one of the devices connected to the system and shows the steps needed for the subscribed user to read a topic. It reveals that neither the publishers nor the MQTT broker has an action in this process. The fog server contains the web server, which controls the subscriptions of subscribers to the topics published by the IoT devices.

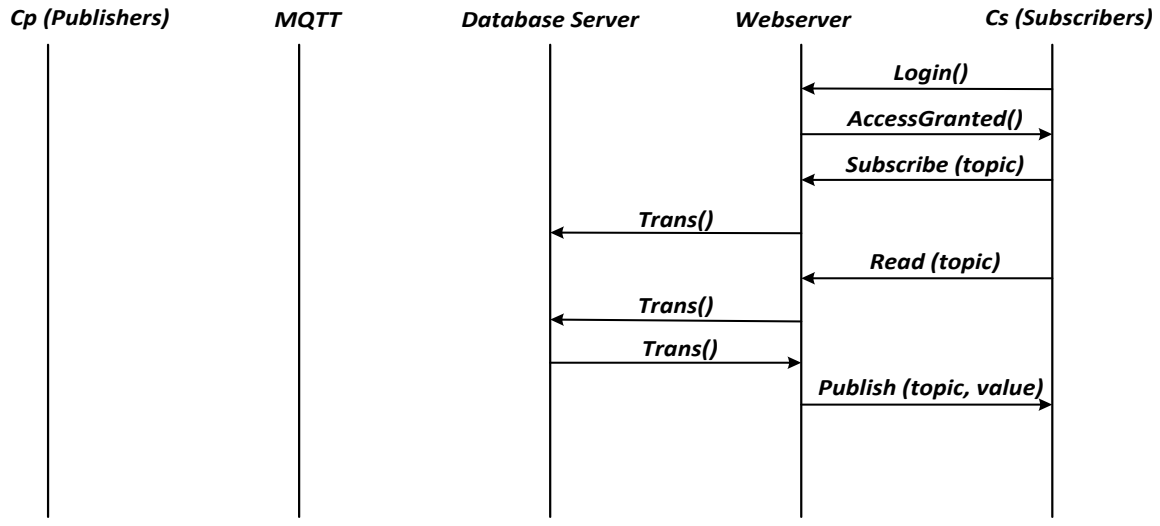

Fig. 7. Transactions in subscribing/reading.

The fog server – as shown in Fig. 7 - represents the front end of the MQTT network of devices and manages AC. The subscribers are logging in the system and, then, subscribe to a topic. The subscription information is saved in the database. The subscriber can read a topic anytime using the read command. The subscribers will need authentication when reading a topic data from the fog server. They get a publish message as a response to the read command. The read function is an added feature which is not available in the regular MQTT implementation.

In Fig. 8, the publishing steps are shown as a sequence diagram as well. The publishers will send the topics to the broker, which in its turn forwards it to the fog server that is considered the only subscriber of the broker. The fog server is a subscriber with the MQTT broker. Other subscribers are considered as subscribers with the fog server. The fog server is taking this task with the MQTT broker. However, since the fog server has more than one subscriber, a transaction is invoked and a database record is created, and the topic details are published and sent by the fog server to all subscribers. The fog server is a central control unit for the topics that works with the broker to extend the range of subscribers and applications. Therefore, the broker is required to access the fog server to publish topics to the subscribers through the fog server.

As shown in Fig. 8, when C_P has a topic and value to publish, the IoT device connects to the MQTT broker; then, it sends the publish command with the subject and the value. The publish messages is sent from the C_P to the MQTT broker. The MQTT broker does login to the fog server and, then, sends the publish message to the fog server. After that, a publish message is sent again from the fog server using the web server to all C_S. Then, at the same time, the web server sends the message to the *Cs* and to the database server to store the state of the subject. The subject in the database can be requested by any *Cs* later as depicted in Fig. 7.

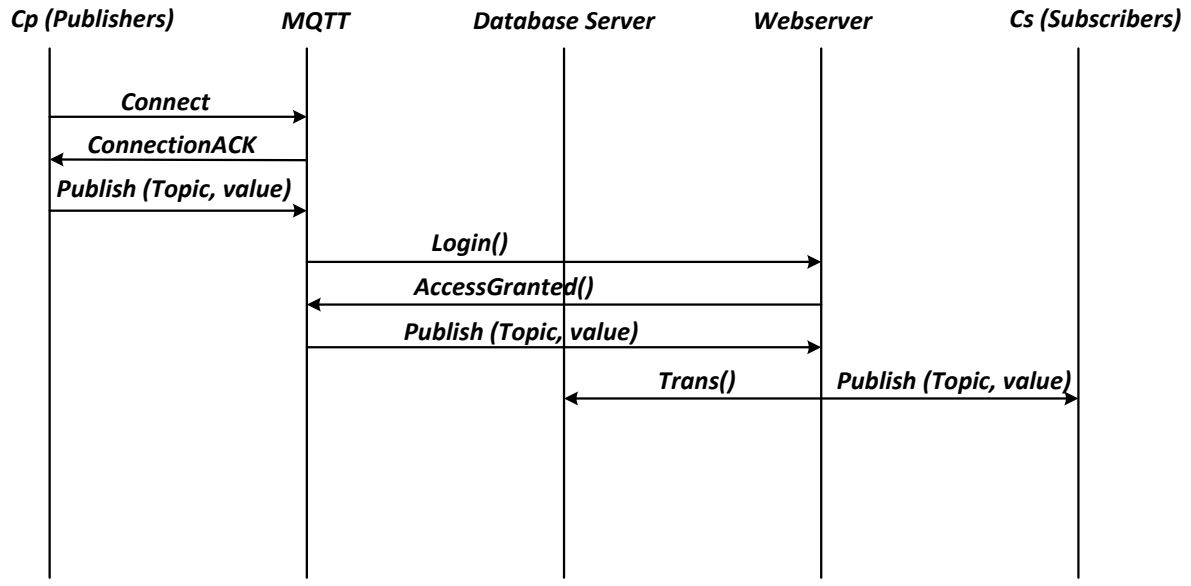

Fig. 8. Transactions in publishing.

4. RESULTS AND DISCUSSION

The proposed IoT model, shown on Fig. 4, contains one web server, one database server, one MQTT broker and a set of IoT devices with a Poisson arrival process with arrival rate λ and an interarrival time of 1 s. The service rate is fixed at each node. The web server and the database server compose the fog server that has a high speed connection with the cloud as well as the subscribers.

The simulation is run using Java in a system with 64 bit Windows 10, running on an Intel® Core™ i7-4770 processor with 16 GB of RAM. Configuration of the simulation model is as follows: the wireless connection between the fog server and the MQTT is 100 Mbps. The Wi-Fi connection between the MQTT broker and the IoT devices is 10 Mbps over the 802.11g protocol. However, the sending rate of each IoT device is much lower, allowing the broker to receive data from many IoT devices. Each IoT device is equipped with ZigBee, using an IEEE 802.15.4 antenna, with a rate of 250 kbps, and the maximum message size of 127 B according to the ZigBee specification [24]. The time required to access the database - tested with the PostgreSQL access time, obtained from the Java driver - is 130 ms. The time required to query the database is approximately 100 ms. The processing time at the broker, web server, cloud and subscriber are 100 ms each. The processing time at the IoT devices is 10 ms. The transmission time between the fog server and the broker, obtained by a ping to the MQTT broker at test.mosquitto.org and tested by Jaloudi [8], is 120 ms. The simulation model can process a maximum of two jobs at a time considering the different sub-processes of the main processes, which are the subscribing and the publishing process.

Eq. (1) shows the complete processing time for one publishing transaction while Eq. (3) shows the average transaction time for one subscribing transaction. Eq. (2) shows that the processing time at the fog server is the sum of the processing times at the web server and the database server. The processing time at the IoT device is defined as T_{PA}. The transmission time between the IoT device and the MQTT broker is T_{XAB}. The processing time at the MQTT broker is defined as T_{PB}. The transmission time between the MQTT broker and the fog server is $3*T_{XBS}$, where S represents the fog server that contains the web server, C, and the database server, DB. Fig. 8 shows that there are three transactions involved in the publication between the MQTT broker and the fog server. Processing time at the webserver is defined as T_{PC}. Processing time at the database server is defined as T_{PD}. The transmission time between the fog server and the subscriber is T_{XSE}. Processing time at the subscriber is defined as T_{PE}. Eq. (2) shows the total processing time at the fog server as a sum of the processing time required by the web server and by the database server.

$$T_{AE} = T_{PA}+3*T_{XAB}+T_{PB}+3*T_{XBC}+T_{PC}+T_{PD}+T_{XCE}+T_{PE} \quad (1)$$

$$T_{PC} = T_{PS} + T_{PDB} \quad (2)$$

$$T_{AE} = T_{PC} + T_{PD}+5 * T_{XCE}+ T_{PE} \quad (3)$$

From the simulation experiments, the arrival rate of data from each of the IoT devices is 127 B/s, which is one reading of subject data, keeping the data size minimal and avoiding fragmentation. Figs. 9 and 10 show that the number of jobs in the system increases over time, and that a congestion-control or compression mechanism is essential to keep the number of

jobs steady or stable, and such a mechanism consequently decreases the system's waiting time.

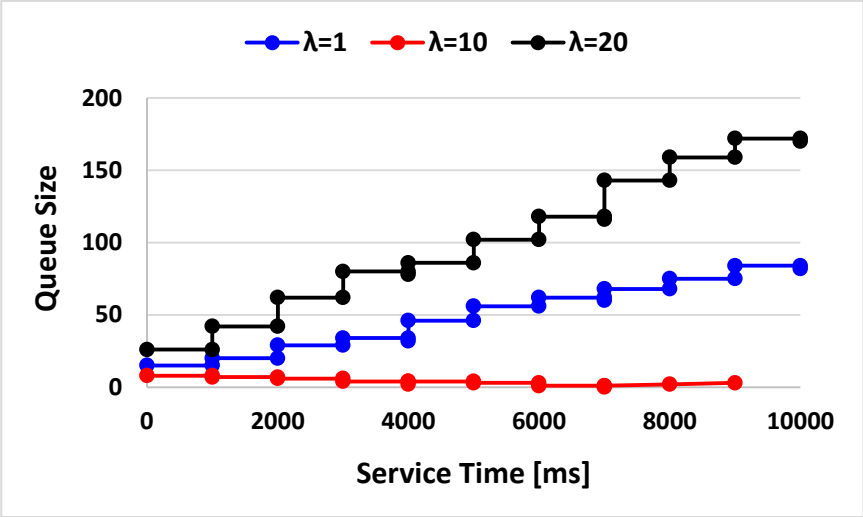

Fig. 9. Subscribing: queue size vs service time for different arrival rates.

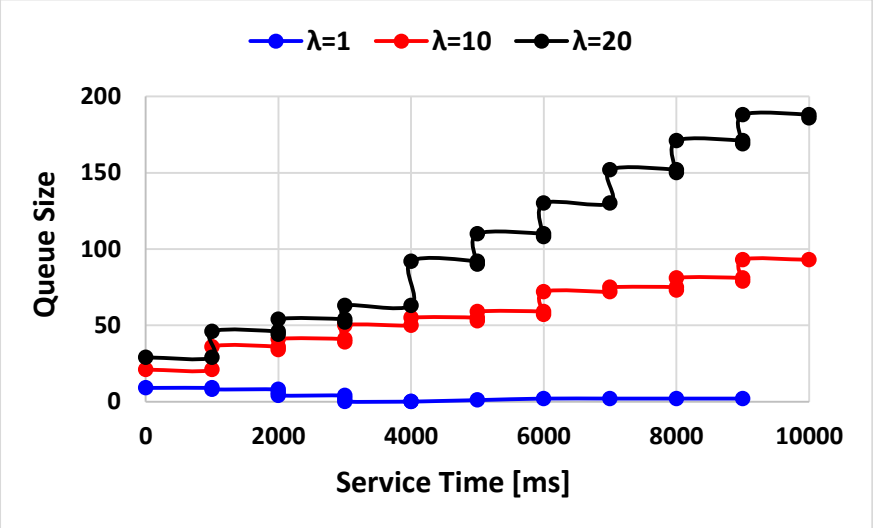

Fig. 10. Publishing: queue size vs service time for different arrival rates.

The simulation results show that publishing one message from the fog server to the subscriber, T_{CE}, takes approximately 0.6 s to 0.8 s (see Fig. 11) and that publishing one message from the publisher to the fog server, T_{AC}, takes from 0.755 s to .941 s (see Fig. 12.)

Figs. 11 and 12 also show the extra system time required for implementing the model. Subscribing requires approximately 0.1 s or 22% more time for subscribing while publishing requires 0.55 s more or 300% more publishing time. The proposed model constitutes proposed a data management model that increases the system's security, the processing requirements and the response time. However, the proposed model increases the capabilities of an IoT application since it has the ability to respond to greater number of Cs. In the proposed model, the subscribers Cs are connected to the fog serve rather than to the MQTT directly. Because the fog server has more processing capabilities and bandwidth, the fog server can server more subscribers Cs more efficiently and securely. In fact, more than one

MQTT broker can be connected to the fog server. Therefore, the fog server can serve more C_P than a single MQTT broker.

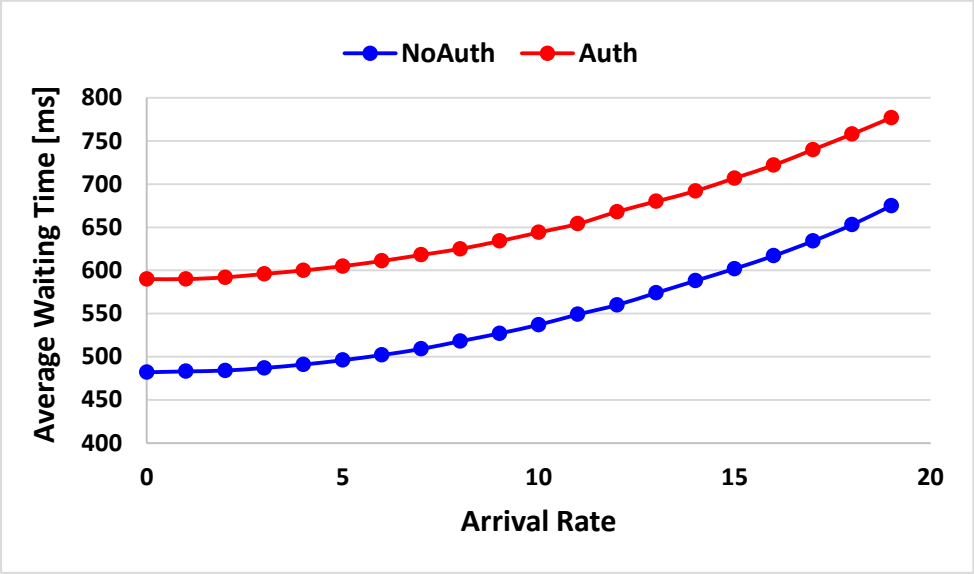

Fig. 11. Subscribing: average waiting time of the system.

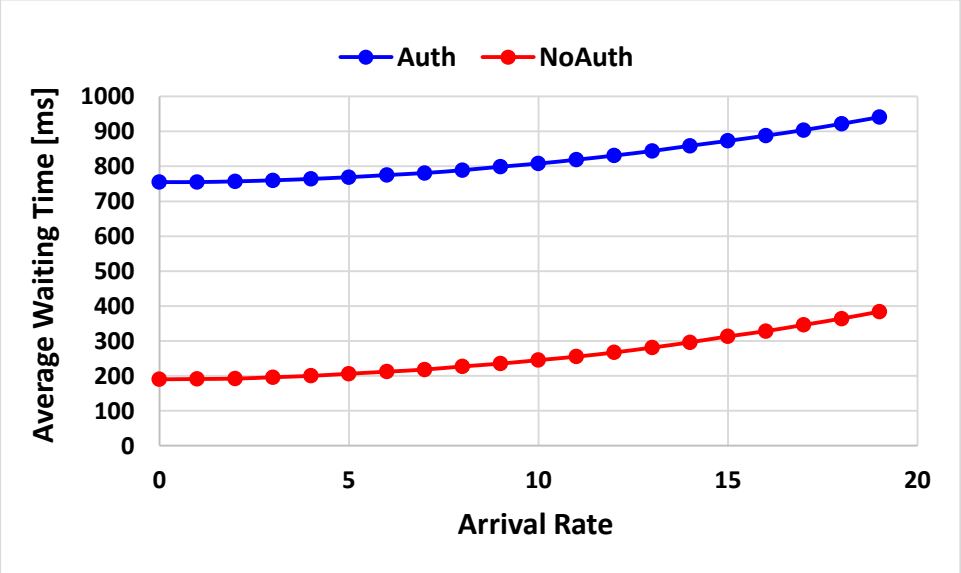

Fig. 12: Publishing: average waiting time of the system.

Results of comparing the proposed model with the regular model that has AC and data management are shown in Figs. 13 and 14. The fog server is 21 times faster according to the benchmark in [24] if it is equipped with a processer of Intel-core-i7-6700k and the MQTT broker, using Raspberry Pi, with process has cortex-A53 ARM-v8. Therefore, when implementing the model over the MQTT broker, publishing the model takes much more time (about 1782 ms or 300% more) as shown in Fig. 13. While for the subscribing, implementing the model over regular MQTT broker, the subscribing transaction requires about 1325 ms or 175% more time as revealed by Fig. 14. As a result, the regular model will consume more energy because of the added functionality. Therefore, to implement such a model, the layer, added in the model, is essential and will require a middle layer with high

processing power, such as the fog server in the model. The proposed model is outperforming the regular model when implementing AC and data management.

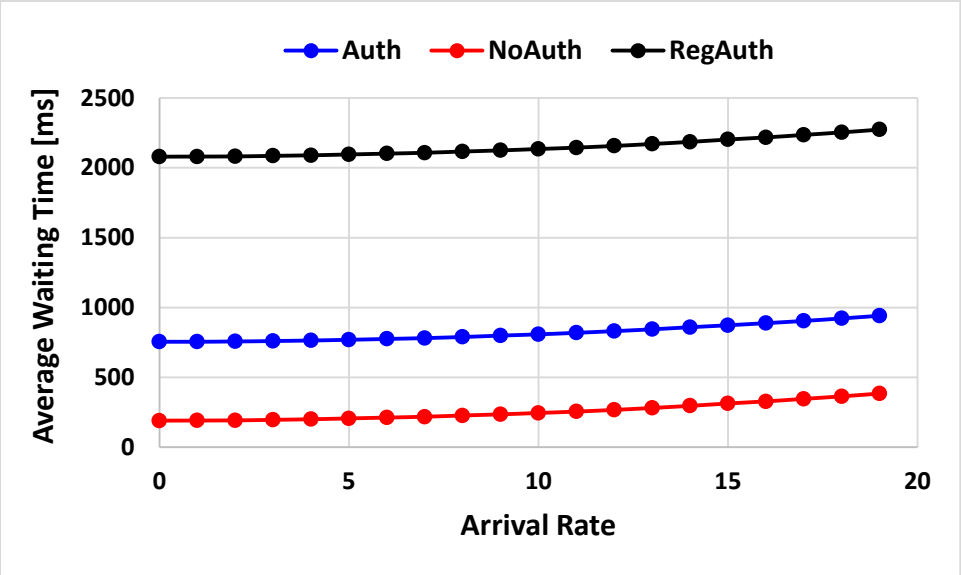

Fig. 13: Publishing: including the authentication with regular MQTT for the system's average waiting time.

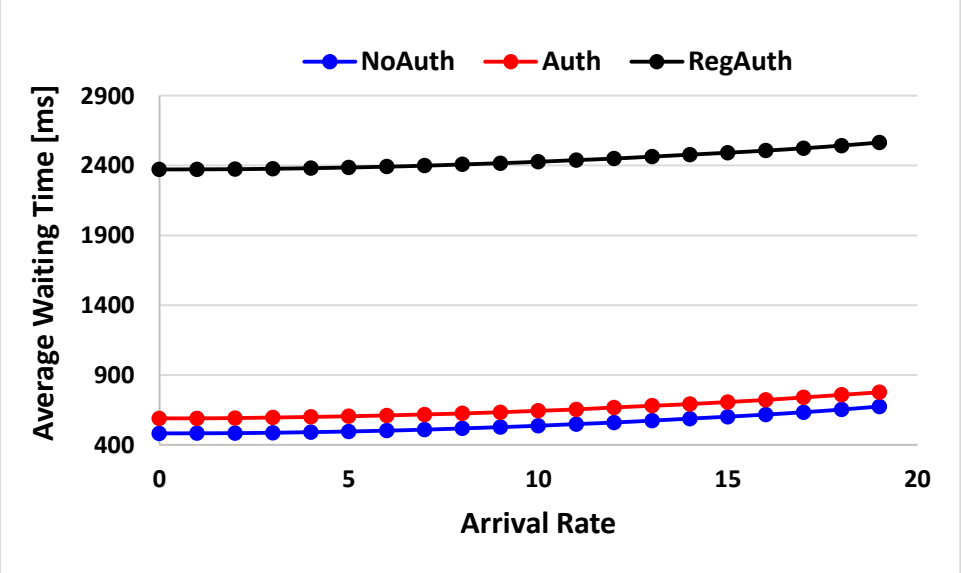

Fig. 14. Subscribing: including the authentication with regular MQTT for the system's average waiting time.

5. CONCLUSIONS

In this work, an IoT model with enhanced security - achieved by moving the processing complexity from the MQTT broker to the fog server - was proposed. The model presented an added layer of access control to increase the security of an IoT system. But this layer invoked an overhead of additional processing time. However, this time can be managed and optimized to a required quality of service with different configurations of the system. While increasing the system's performance, for instance by decreasing waiting time, there is generally a requirement for congestion control, network management or data compression for the IoT connected devices. The performance of proposed model was

authenticated by multiple metrics; and the obtained results show that it can be deployed successfully in the implementation of IoT applications to enhance both the IoT's security and performance.

The proposed - in this work - model can be also extended by having more MQTT brokers connected to the fog server. Also, for load balancing, more fog servers can be added as a subscriber to the MQTT broker, which increases the system reliability.

In future work, it will be possible to control the amount of additional time overhead. In fact, future development and design of security systems should consider keeping the data stream in the real-time range. With such enhancement, the future systems can behave in real time with enhanced security and service of IoT applications without compromise.

REFERENCES

[1] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, "Internet of things (IoT): a vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645 – 1660, 2013.

[2] S. Vashi, J. Ram, J. Modi, S. Verma, C. Prakash, "Internet of things (IoT): a vision, architectural elements, and security issues," *Proceedings of IEEE International Conference on IoT in Social, Mobile, Analytics and Cloud, Palladam*, pp. 492-496, 2017.

[3] A. Banks, R. Gupta, *MQTT Version 3.1.1 Protocol Specification*, Oasis Standard, Oasis, Burlington, USA, 2014.

[4] L. Cruz-Piris, D. Rivera, G. Lopez-Civera, E. Da Ia Hoz, I. Marsa-Maestre, J. Velasco, "Protecting sensors in an IoT environment by modelling communications as resources," *Proceedings of the 5th International Symposium on Sensor Science*, Barcelona, Spain, 2017.

[5] L. Cruz-Piris, D. Rivera, I. Marsa-Maestre, E. De la Hoz, J. Velasco, "Access control mechanism for IoT environments based on modelling communication procedures as resources," *Sensors*, vol. 18, no. 3, pp. 1-21, 2018.

[6] Eclipse IoT Working Group, IEEE IoT, AGILE IoT, IoT Council, "IoT developer survey 2017," *Eclipse Foundation*, 2017. <https://www.eclipse.org/lists/iot-wg/pdf05OUMA7E1X.pdf>

[7] L. Brenman, "API builder and MQTT for IoT," *Axway*, 2018. <https://dzevblog.axway.com/apis/api-builder-and-mqtt-for-iot-part- 1/>

[8] S. Jaloudi, "MQTT for IoT-based applications in smart cities," *Palestinian Journal of Technology and Applied Sciences*, vol. 2, pp. 1–13, 2019.

[9] T. Yokotani, Y. Sasaki, "Comparison with HTTP and MQTT on required network resources for IoT," *Proceedings of IEEE International Conference on Control, Electronics, Renewable Energy and Communications*, Bandung, pp. 1-6, 2016.

[10] T. Yokotani, Y. Sasaki, "Transfer protocols of tiny data blocks in IoT and their performance evaluation," *Proceedings of IEEE 2016 3rd World Forum on Internet of Things*, Reston, VA, pp. 54-57, 2016.

[11] N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," *Proceedings of IEEE International Systems Engineering Symposium*, Vienna, pp. 1-7, 2017.

[12] Y. Sueda, M. Sato, K. Hasuike, "Evaluation of message protocols for IoT," *Proceedings of IEEE International Conference on Big Data, Cloud Computing, Data Science and Engineering*, Honolulu, HI, USA, pp. 172-175, 2019.

[13] O. Alhazmi, K. Aloufi, "Fog-based internet of things: a security scheme," *Proceedings of IEEE 2nd International Conference on Computer Applications & Information Security*, Riyadh, Saudi Arabia, pp. 1-6, 2019.

[14] K. Aloufi, "6LoWPAN stack model configuration for IoT streaming data transmission over coap," *International Journal of Communication Networks and Information Security*, vol. 11, no. 2, pp. 304–311, 2019.

[15] R. Fielding, J. Gettys, J. C. Mogul, H. Nielsen, L. Masinter, P. Leach, T. Berners-Lee, *Hypertext Transfer Protocol – HTTP/1.1. IETF*, Internet Engineering Task Force, RFC 2616, 1999.

[16] Tzapu, *Wifimanager*, GitHub, 2018. <https://github.com/tzapu/WiFiManager>

[17] M. Yuan, *Getting To Know MQTT*, IBM Developer, 2019.

[18] J. O'Hara, "Toward a commodity enterprise middleware," *ACM Queue*, vol. 5, no. 4, pp. 48-55, 2007.

[19] J. Schlesselman, G. Pardo-Castellote, B. Farabaugh, "OMG data-distribution service (DDS): architectural update," *Proceedings of IEEE MILCOM 2004. Military Communications Conference*, Monterey, CA, USA, vol. 2, pp. 961 – 967, 2004.

[20] M. Bassoli, V. Bianchi, I. De Munari, "A plug and play IoT Wi-Fi smart home system for human monitoring," *Electronics*, vol. 7, no. 9, pp. 1-13, 2018.

[21] C. Pei, Z. Wang, Y. Zhao, Z. Wang, Y. Meng, D. Pei, Y. Peng, W. Tang, X. Qu, "Why it takes so long to connect to a WiFi access point," *Proceedings of IEEE Conference on Computer Communications*, *IEEE INFOCOM 2017*, Atlanta, GA, pp. 1-9, 2017.

[22] S. Manandhar, *MQTT based communication in IoT*, Ph.D. thesis, Tampere University of Technology, Finland, 2017.

[23] M. Osipov, "Home automation with zigbee," *Next Generation Teletraffic and Wired/Wireless Advanced Networking*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 263–270, 2008.

[24] TrueBench, Toffee Project. <https://truebench.the-toffee-project.org/>