Using the MQTT Protocol in Real Time for Synchronizing IoT Device State

Adnan Shaout and Brennan Crispin
Electrical and Computer Engineering Department, University of Michigan, Dearborn

Abstract: *This paper will present a design and implementation for an embedded system to connect to a Machine to Machine (M2M) broker. The proposed system will use the cloud server to communicate with other embedded systems. The system will be configurable from a cloud-based web service. The paper also will explore previous research on M2M protocols such as Message Queueing Telemetry Transport (MQTT) and Advanced Messaging Queuing Protocol (AMQP). The paper will present and demonstrate an MQTT based system for synchronizing IoT device state across multiple client nodes. The objective of the system is for state changes to be registered and distributed throughout the system in under 1 second; and initial registration of a new node should occur in under 30 seconds.*

Keywords: *MQTT, synchronizing, machine to machine, real time systems, IoT, cloud computing.*

Received February 16, 2018; accepted April 11, 2018

1. Introduction

With the growing number of IoT devices, there is an increasing need for the ability to synchronize state and data across multiple IoT devices. As many IoT devices will be operating in constrained environments with unstable network connections, understanding the tradeoffs of various protocols is critical. Standards to develop specific protocols for IoT are needed [13].

The Internet of Things (IoT) increasingly covers a wide array of applications and use cases, and is a large current field of study and innovation. For example, in smart cities IoT is the key features and the driver technologies and the physical digital integration within city systems [31].

One of the major issues for IoT devices is how to efficiently communicate and synchronize between each other, a process referred as Machine to Machine (M2M) communication. The IoT emerging network topologies and communication technologies is another area of IoT research [5]. The connectivity technologies and tools and their contributions for setting up and sustaining smarter environments is another major issue for IoT devices [26]. Since operating remotely and in low-power devices is critical to the success of IoT devices, nodes must often be able to operate in constrained environments with unreliable network access. Several protocols have been explored for Machine to Machine communication of IoT devices, including Message Queueing Telemetry Transport (MQTT) [23], the Advanced Messaging Queuing Protocol (AMQP) [1], Constrained Application Protocol (CoAP) [7], Data Distribution Service (DDS) [10], Web sockets [17], the Extensible Messaging and Presence Protocol (XMPP) [34], and HTTP based protocols [16]. Synchronization abstractions have been suggested to be used to tie together the interactions between 'things' in an IoT environment [21, 28]. The IBM Bluemix was also used to connect an IOT device to a cloud server [30]. An IoT protocol stack, which is an extension of the TCP/IP layered protocol model was also proposed by Rayes and Salam [27].

The paper will present a design and implementation of an IoT system that can detect local state changes and synchronize those state changes with other remote nodes as shown in Figure 1.

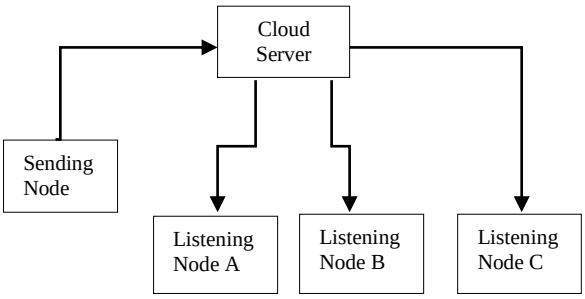

Figure 1. Overview of system flow-sending node sends an update which is received by all nodes registered as listeners.

The primary objective of this system will be for an embedded system to connect to a M2M broker, use the cloud server to communicate with other embedded systems, and be configurable from a cloud-based web service. The system will register state changes then distributed them throughout in under 1 second, and initial registration of a new node should occur in under 30 seconds.

Additionally, no state change data should be lost. For the purposes of this research in this paper the MQTT protocol was chosen for appearing to best meet the requirements set forward.

MQTT is an open-source TCP based protocol based on publish-subscription architecture. In publish-subscribe ("pub-sub"), clients connect to a central broker, and can "subscribe" to interested topics. When a client "publishes" to that topic, any client nodes that have subscribed will receive the published message. Being a TCP based protocol, MQTT has relatively high overhead, but also a high guaranteed Quality of Service (QoS), and also supports one-to-one and one-to-many messages. MQTT has numerous open source libraries and a robust support community, making it relatively easy to use for applications. A sample MQTT network is shown in Figure 2 [20].

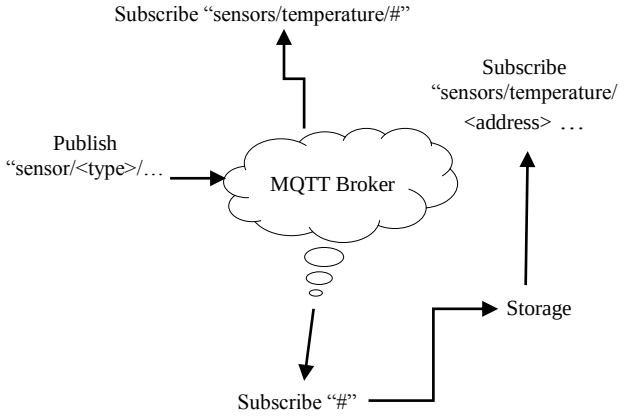

Figure 2. Sample MQTT system, several clients subscribe for updates from a temperature sensor.

This paper is further organized as follows: section 2, Prior research on common M2M protocols is summarized and used to provide a basis for using MQTT in this paper. In section 3, the implementation of the proposed design is discussed, both in terms of design and requirements testing. In section 4, testing results and how the system did meet the requirements set forward will be discussed; and section 5, conclusions and future work will be presented.

2. Related Work

In order to determine the best protocol for this project, research on the relative performance of several M2M communication protocols was examined and the results compared to the requirements for the project. A number of studies, [3, 4, 22, 29] have examined M2M protocols for networked devices in the qualitative sense, however, quantitative studies were required to determine which protocol would best meet the requirements for this project.

Yokotani and Sasaki [35] compared HTTP and MQTT protocol network requirements in a variety of network conditions, finding that MQTT is a more efficient protocol for connecting IOT devices than HTTP. This is largely due to the additional data overhead involved in HTTP compared to the MQTT protocol. In [9], Daud and Suhaili compared performance for Constrained Application Protocol (CoAP) and Hypertext Transfer Protocol (HTPP) protocols, finding that CoAP has a substantially lighter memory and processing footprint, but that HTTP has substantial security improvements due to build in TLS support. The authors recommended that CoAP be implemented with DTLS to improve communication security. In [14] HTTP and AMQP are compared in the case of RESTful web services; running several tests over a variety of client applications. After comparing the average number of messages sent and received, the authors conclude that AMQP allows a greater number of messages to be supported.

Since many IoT devices will be required to perform in unstable network environments, Luzuriaga *et al.* [19] compared the performance of AMQP and MQTT protocols in poor quality and unstable network environments. They found that AMQP is more reliable and stable, but that MQTT, being the lighter-weight service, is more appropriate for constrained edge nodes. Thangavel *et al.* [33] compared CoAP and MQTT bandwidth usage and delay time for varying packet loss constraints. They found that MQTT, being TCP based, had high delivery percentages at high packet loss, but longer delays, while CoAP, being UDP based, lost substantial numbers of packets at a lower delay time. Similarly, Sutaria and Govindachari [32] ran further comparisons of MQTT and CoAP, finding similar results in data loss and latency.

Due to the frequency of constrained networks in IoT applications, Chen and Kunz [8] compared several protocols under constrained wireless environments: MQTT, CoAP, DDS, and XMPP. Using the example of a medical device transmitting data to a care provider, they examined a number of factors such as latency and packet loss under progressively constrained environments. They found that both MQTT and DDS have zero packet loss in high latency environments, but that DDS is superior in terms of latency-but substantially higher bandwidth requirements. CoAP and XMPP both experienced substantial packet losses and higher bandwidth consumption.

From the research that has already been conducted, it was determined that the MQTT protocol would best meet the requirements set out previously for latency, memory and power, publish/subscribe architecture. MQTT has successfully been used to implement a wide variety of data collection services [6, 11, 12, 15, 18], although it has not yet been used to synchronize device state across multiple IoT nodes.

3. Implementation

This section discusses the hardware setup and software design and implementation of the system.

3.1. Hardware Setup

The hardware configuration is shown in Figure 3.
The MQTT client node consists of:

- Arduino duo revision 3.
- Arduino ethernet shield.
- LED and button input for Arduino.

The MQTT broker and server was hosted on a:

- MacBook Pro 2014.
- Linksys network router used for connection between client node and server.

Figure 3. Arduino uno and ethernet controller.

The LED and button input and output are directly attached to the Arduino's Pin 12 and Pin 2, respectably.

3.2. Software Implementation

For the complete system, three separate subsystems were required to be designed and coded: the embedded client node, the MQTT broker, and the control server.

For the MQTT broker, we took advantage of a widely used MQTT library, Mosquitto MQTT [29], provided by the Eclipse foundation. The broker is run on the MacBook server and was run with TCP and SSL ports open. The client node was implemented using Arduino C, and an open source library, mqtt PubSubClient [25], was used to implement connecting the device to the MQTT broker. Finally, the control server was implemented in Angular [2], a Javascript framework, using the built-in Javascript MQTT library to connect to the broker.

3.3. ClientDesign and Implementation

The client node was implemented using foreground-background architecture. During the background loop, the Software checks the Ethernet module for any newly received packets. The system parses the packet and takes one of three possible actions depending on the message topic should any packets be available from the MQTT broker. Figure 4 shows the main background loop. The following are the message possible topics:

1. *State*: The client node reads the state data and synchronizes its state to that state provided. The client will maintain this state until an interrupt overrides the state with new state information or a new, updated state arrives from the posting node.
2. *Subscribe/Unsubscribe*: The client node reads the subscription address from the message and subscribes to state updates for the given posting node.
3. *Query*: The query message comes from the server on initialization, and asks that all nodes update their register information on the server and their state information. The node, upon receiving a query, will re-register with the server and post its current state.

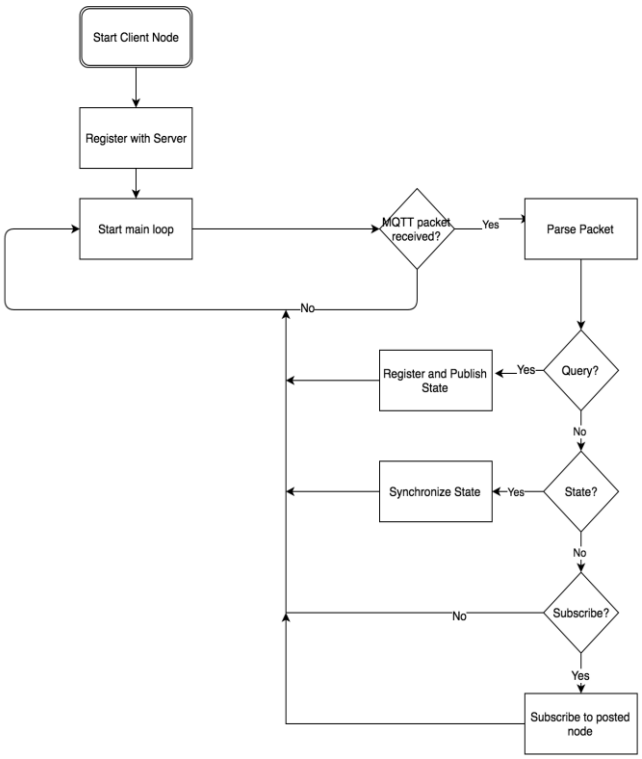

Figure 4. Overview of main background loop.

On button press, the controller receives an interrupt to read from the button pin. The client reads the button state, updates its internal and LED state to match, and then publishes its state for any subscribed nodes to synchronize with as shown in Figure 5. However, because the interrupt and the main loop both potentially need to use the Ethernet controller, access to the controller is protected by a semaphore to control sharing the resource (the Ethernet controller). If the background loop is current, then publishing the foreground interrupt will hold until that process has been completed.

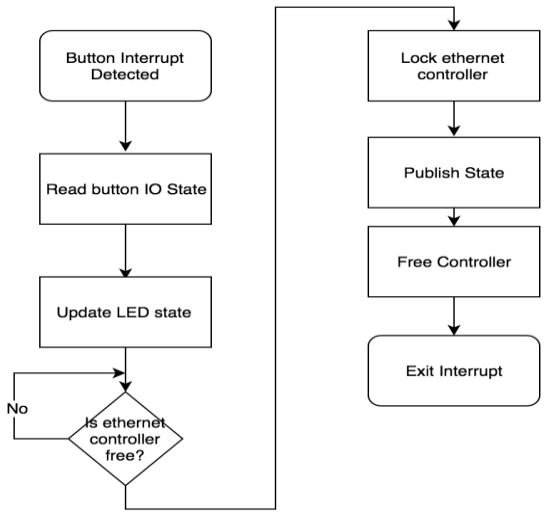

Figure 5. Button interrupt program flow.

3.4. Server Implementation

The server was implemented as a simple front-end interface to the overarching client nodes and message broker. State and data do not persist from instance to instance, so the server client first collects information on all active nodes by publishing a 'query' request and then collecting the 'register' posts from each client. When a new client node becomes active, it registers with the server and is added to the control list as shown in Figure 6.

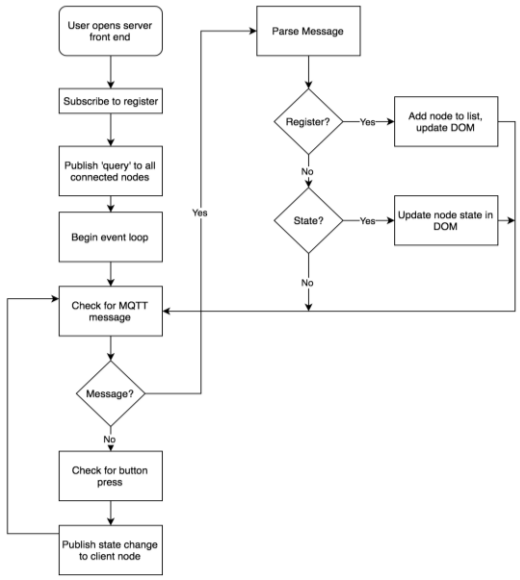

Figure 6. Server controller flow diagram.

From the registered client nodes, the server creates a list of available nodes, with the option to change any given node's state as well as to open a node's control page and select which other nodes that node is paired with. Paired nodes will listen for that other nodes state and update their internal state to match on change.

When a user selects an action for a given node, an MQTT publish action occurs that sends a message with the topic <ACTION>/<NODE ID> and the message for the node to follow to the MQTT broker, which then routes the message to the listening node.

Additionally, when the server receives a register message from a node, it subscribes to any state notifications from that node. When a state update is received, the server updates its display to reflect the new state of each node (Figure 6).

3.5. Experimental Setup

An important part of this project was ensuring that the system met certain time bounds. In order to test that these bounds were met, additional tests of the system were implemented.

Since many such IoT devices would need to operate in a constrained environment, NetEM [24] was used to test the system in varying levels of packet loss and latency.

In order to measure latency time from sending to receiving, the client node was modified to subscribe to its own updates, time them from publish to reception, and output the results to an attached serial port. In this manner, a total roundtrip time could be measured from interrupt detection to reception of the state packet. By measuring the response time across varying network and system conditions, a map of system performance in constrained environments would be created.

During latency tests, packet losses were additionally measured. Any state change that featured a posted time of greater than 5 seconds was qualified as a 'lost message' and counted in the data. Finally, the Arduino IDE provides a useful tracking of CPU utilization during operation. During the above tests, average CPU utilization was tracked. Additionally, a Python script was written to spam the system with state updates, simulating a 'high use' operation and CPU utilization was further measured.

4. Result

Three metrics were used to determine whether the design and system meet the requirements established for the project: latency time under various network constraints, lost messages under packet loss, and CPU utilization under expected and extreme loads.

4.1. Message Latency

Using the NetEM package to vary the loss of different percentages of packets on the network, the resultant latency of packets was measured and is shown in Figure 7. The Figure shows the number of packets verses the latency. Since MQTT is a TCP protocol, additional packets are expected to be sent as packet loss increases, leading to an increased latency time. With a requirement that the total routing time for any message be less than 1 second, we can reasonably expect MQTT to be appropriate for network conditions

in which less than 15% of the packets are lost. However, at higher loss rates the response time will be greater than 1 second, which is unacceptable for the project requirements in this paper.

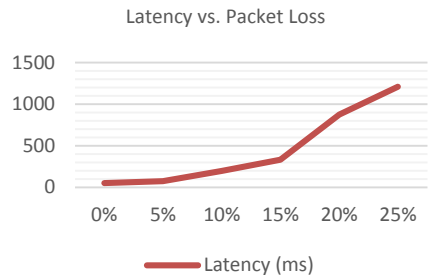

Figure 7. Experienced latency vs. packet loss in network.

4.2. Lost Messages

Again, using the NetEM package, the loss of messages during varying packet loss scenarios was measured. The results are summarized in Figure 8. MQTT, being a TCP based library, experienced zero losses in total messages.

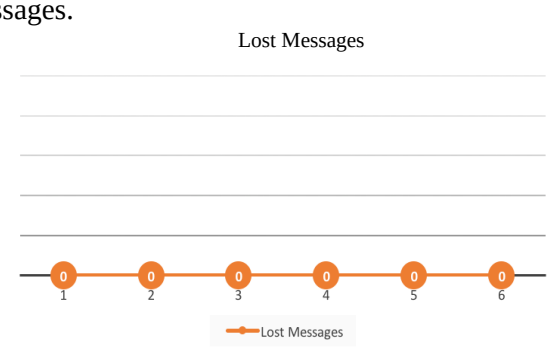

Figure 8. Total lost messages vs. packet loss in network.

Note that zero losses has been experienced because MQTT is a TCP based algorithm.

4.3. CPU Utilization

CPU utilization under various load conditions is shown in Table 1. In general, the CPU utilization of the Arduino board remained under 50%, even for high load conditions, indicating that MQTT is a suitably lightweight protocol for low power IoT applications and even lower power devices will support its implementation.

Table 1. CPU utilization under varying load conditions.

CPU Utilization for Various Use Cases	
Use Case	**CPU Utilization**
0% Data Loss, normal press	26%
25% Data Loss, normal press	28%
0% Data Loss, Spam Press	35%
25% Data Loss, Spam Press	41%
Spammed Notifications	39%

5. Conclusions

The requirements imposed on the design of the system in this paper have shown that MQTT is an appropriate protocol for connecting low power devices in constrained environments. MQTT provides a simple, flexible architecture that allows for the easy synchronization of device states with minimum overhead, and a selection of open source libraries make its implementation comparatively simple.

Analysis of our design has shown that MQTT meets system response time requirements for mildly unstable networks, but will begin to fail as greater packet losses occur. MQTT also successfully delivered all packets in constrained environments, which may be more important depending on the exact requirements of a system. MQTT was relatively efficient, utilizing less than 40% of the CPU during any operations, indicating that it would be appropriate for even lower power devices. These tradeoffs will need to be considered for anyone attempting to design a practical IoT system with synchronized device states.

References

[1] AMQP protocol specification, (http://amqp.org), Last Visited, 2017.

[2] AngularJS Framework, (https://angularjs.org), Last Visited, 2017.

[3] Asensio A., Marco A., Blasco R., and Casas R., "Protocol and Architecture to Bring Things into Internet of Things," *International Journal of Distributed Sensor Networks*, vol. 2014, pp.1-18, 2014.

[4] Atzori L., Iera A., and Morabito G., "The Internet of Things: A Survey," *Computer Networks*, vol. 54, no. 15, pp. 2787-2805, 2010.

[5] Soundarabai P. and Chelliah P., *Connected Environments for the Internet of Things. Computer Communications and Networks*, Springer, 2017.

[6] Caro N., Colitti W., Steenhaut K., Mangino G., and Reali G., "Comparison of Two Lightweight Protocols for Smartphone-Based Sensing," *in Proceedings of IEEE 20th Symposium on Communications and Vehicular Technology in Benelux*, Namur, pp. 1-6, 2013.

[7] CoAP protocol specification, (http://coap.technology), Last Visited, 2017.

[8] Chen Y. and Kunz T., "Performance Evaluation of Iot Protocols Under A Constrained Wireless Access Network," *in Proceedings of International Conference on Selected Topics in Mobile and Wireless Networking*, Cairo, pp. 1-7, 2016.

[9] Daud M. and Suhaili W., "Internet of Things (IoT) with CoAP and HTTP Protocol: A Study on Which Protocol Suits IoT in Terms of Performance," *in Proceedings of the Computational Intelligence in Information Systems Conference*, pp. 165-174, Cham, 2017.

[10] DDS protocol specification,

(http://www.omg.org/spec/DDS), Last Visited, 2017.

[11] Dhar P. and Gupta P.,"Intelligent Parking Cloud Services Based on IoT using MQTT Protocol," *in Proceedings of International Conference on Automatic Control and Dynamic Optimization Techniques*, Pune, pp. 30-34, 2016.

[12] Ding Y., Binwen F., Xiaoming K., and Qianqian M.," Design and Implementation of Mobile Health Monitoring System Based on MQTT Protocol," *in Proceedings of IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference*, Xi'an, pp. 1679-1682, 2016.

[13] Diogo, P., Lopes N., and Reis L., Cluster Computing 20: 2193, https://doi.org/10.1007/s10586-017-0861-0, Last Visited, 2017.

[14] Fernandes J., Lopes I., Rodrigues J., and Ullah S., "Performance Evaluation Of Restful Web Services And AMQP Protocol," *in Proceedings of the 5th International Conference on Ubiquitous and Future Networks*, Da Nang, pp. 810-815, 2013.

[15] Hantrakul K., PramokchonP., KhoenkawP., Tantitharanukul N., and Osathanunkul K., "Automatic Faucet with Changeable Flow Based on MQTT Protocol," *in Proceedings of International Computer Science and Engineering Conference*, Chiang Mai, pp. 1-5, 2016.

[16] HTTP protocol specification, (https://www.w3.org/Protocols/rfc2616/rfc2616.html), Last Visited, 2017.

[17] Ian F. and Melnikov A., *The WebSocket Protocol*, Internet Engineering Task Force, 2011.

[18] Kang D., Park M., Kim H., Kim D., Kim S., Son H., and Lee S.," Room Temperature Control and Fire Alarm/Suppression IoT Service Using MQTT on AWS," *in Proceedings of International Conference on Platform Technology and Service (PlatCon)*, Busan, pp. 1-5, 2017.

[19] Luzuriaga J., Perez M., Boronat P., Cano J., Calafate C., and Manzoni P.," A Comparative Evaluation of AMQP and MQTT Protocols over Unstable and Mobile Networks," *in Proceedings of the 12th Annual IEEE Consumer Communications and Networking Conference*, Las Vegas, pp. 931-936, 2015.

[20] Monitoring Your Devices with MQTT - Packt Publishing, https://www.packtpub.com/sites/default/files/downloads/1942OS_Chapter_9.pdf, Last Visited, 2018.

[21] Moreno M., Cerqueira R., and Colcher S., "Synchronization Abstractions and Separation of Concerns as Key Aspects to the Interoperability in IoT," *in Proceedings of Second International Conference, Inter IoT 2016 and Third International Conference SaSe IoT*, Paris, pp. 26-32, 2017.

[22] Mosquitto Project, (http://mosquitto.org), Last Visited, 2017.

[23] MQTT protocol specification, (http://mqtt.org), Last Visited, 2017.

[24] NetEM: Software Suite Provides Network Emulation Functionality, (https://wiki.linuxfoundation.org/start?do=search&id=NetEM), Last Visited, 2018.

[25] O'Leary N., Arduino Client for MQTT, (http://pubsubclient.knolleary.net/), Last Visited, 2017.

[26] Rajaraajeswari S., Selvarani R., and Raj P., *Connectivity Frameworks for Smart Devices*, Springer, 2016.

[27] Rayes A. and Salam S., *Internet of Things from Hype to Reality*, Springer, 2017.

[28] Sarwat A., Sundararajan A., Parvez I., Moghaddami M., and Moghadasi A., *Sustainable Interdependent Networks*, Springer, 2018.

[29] Sheng Z., Yang S., Yu Y., Vasilakos A., McCann J., and Leung K., "A Survey on the IETF Protocol Suite for the Internet of Things: Standards, Challenges, and Opportunities," *The IEEE Wireless Communications*, vol. 20, no. 6, pp. 91-98, 2013.

[30] Shovic J., *Raspberry Pi IoT Projects*, Springer, 2016.

[31] Suzuki L., *Smart City Networks*, Springer, 2017.

[32] Sutaria R. and Govindachari R., "Making Sense of Interoperability: Protocols and Standardization Initiatives in IOT," *in Proceedings of the2nd International Workshop on Computing and Networking for Internet of Things*, Hyderabad, 2013.

[33] Thangavel D., Ma X., Valera A., Tan H., and Tan C., "Performance Evaluation of MQTT and CoAP Via a Common Middleware," *in Proceedings of IEEE 9th International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, Singapore, pp.1-6, 2014.

[34] XMPP protocol specification. (http://xmpp.org), https://www.packtpub.com/sites/default/files/downloads/1942OS_Chapter_9.pdf -, Last Visited, 2017.

[35] Yokotani T. and Sasaki Y., "Comparison with HTTP and MQTT on Required Network Resources for IoT," *in Proceedings of International Conference on Control, Electronics, Renewable Energy and Communications)*, Bandung, pp. 1-6, 2016.

Adnan Shaout is a full professor and a Fulbright Scholar in the Computer Science Department at the Electrical and Computer Engineering Department at the University of Michigan-Dearborn. Dr. Shaout has more than 35 years of experience in teaching and conducting research in the Computer Science, Electrical and Computer Engineering fields at Syracuse University and the University of Michigan - Dearborn. Dr. Shaout has published over 230 papers in topics related to Computer Science, Electrical and Computer Engineering fields. Dr. Shaout has obtained his B.S.c, M.S. and Ph.D. in Computer Engineering from Syracuse University, Syracuse, NY, in 1982, 1983, 1987, respectively.

Brennan Crispin is currently, a graduate student at the University of Michigan-Dearborn and a Software Engineer at Ford Smart Mobility, Ford Motor Company.