3rd International Engineering Conference (IEC 2019)
Federal University of Technology, Minna, Nigeria

TOWARDS A HYBRID MQTT-COAP PROTOCOL FOR DATA COMMUNICATIONS IN WIRELESS SENSOR NETWORKS

*Nwankwo, E. I[1], Onwuka, E. N[2] & Michael, D[3]
[1,2,3] Department of Telecommunications Engineering, School of Electrical Engineering and Technology, Federal University of Technology, Minna , Niger State, Nigeria.
*Corresponding author email: emmanueln_nike@hotmail.com, +2347035448942

ABSTRACT
Wireless Sensor Networks (WSNs) consist of sensor nodes and gateways which are resource constrained devices. Lightweight communication protocols for WSNs are emerging for Machine to Machine (M2M) communications and thus there is always going to be a possible conflict of interest on which protocol is best suited for any particular application. The architecture of these emerging protocols is mainly categorized into the publish-subscribe architecture and the request-response architecture. Although there are other protocols for data communication in WSNs, the two protocols of interest in this study are the Message Queuing Telemetry Transport (MQTT) protocol based on the publish-subscribe architecture and the Constrained Application protocol (CoAP) based on the request-response architecture. Studies have shown that the performance of these different protocols are dependent on different network conditions and they have complimentary advantages and disadvantages. MQTT messages experience lower delays than CoAP for lower packet loss and higher delays than CoAP for higher packet loss. MQTT is also considered very scalable since message subscribers do not need to know about the publishers and vice-versa as opposed to CoAP where the client has to explicitly know the server address to be able to make requests. In this paper we propose a hybrid MQTT-CoAP protocol technique for data communication in wireless sensor networks that harnesses the advantages of both MQTT and CoAP protocol.

Keywords: *CoAP, Hybrid protocol, Machine to Machine Communication, MQTT, Wireless Sensor Network*

1 INTRODUCTION

In recent years, introduction of smart sensors in the market place has given rise to considerable advancements in the development of wireless sensor networks (WSNs). Smart sensor nodes are low power devices equipped with one or more sensors, possibly with an actuator, a processor unit, memory/storage unit, a power supply and a wireless communication radio (Akyildiz, Weilian Su, Sankarasubramaniam, & Cayirci, 2002).

As the Internet of Things (IoT) expands to numerous applications through the increasing minimization of hardware, availability of versatile sensors, and "smart objects" (Giusto, Iera, Morabito, & Atzori, 2010), many potential protocols are emerging for M2M communications thus, the question of which protocol to use for the Internet of Things becomes a topic of high interest. From an end-to-end perspective, a WSN can be viewed as comprising of two subnets; a subnet connecting sensor nodes and one or more gateway nodes in which sensor nodes route data until it reaches one of the gateways using WSN protocols (e.g., Collection Tree Protocol (Gnawali, Fonseca, Jamieson, Moss, & Levis, 2009)), and another subnet connecting the gateway and a back-end server or broker. The typical communication architecture for WSNs is shown in Figure 1 below. Sensor data generated by sensor nodes are delivered to the server through the gateway. Meanwhile, clients that are interested to receive sensor data connect to the server to obtain the data. To transfer all the sensor data collected by a gateway node to a server, the former requires a protocol that is bandwidth-efficient, energy-efficient and capable of working with limited hardware resources. As a result, protocols such as Message Queue Telemetry Transport (MQTT) (MQTT, 2014) and Constrained Application Protocol (CoAP) (Shelby, 2013) have been proposed to specifically address the difficult requirements of real-world WSN deployment scenarios.

One way for wireless sensor networks to transfer data from a gateway to clients is the "request-response" also known as "client-server" architecture which is supported by CoAP. In the client-server architecture, request messages initiate a transaction with a server, which may send a response to the client with a matching transaction ID and this is based on a polling method (Davis, Calveras, & Demirkol, 2013). Another way is the "publish-subscribe" architecture (Eugster, Felber, Guerraoui, & Kermarrec, 2003). In this architecture, a client needing data (known as subscriber) registers its interests with a server (also known as broker). The client producing data (known as publisher) sends the data to a server and this server forwards the fresh data to the subscriber. One of the major advantages of this architecture is the decoupling of the clients needing data and the clients sending data. This decoupling enables the architecture to be highly scalable (Eugster et al., 2003). The "publish-subscribe" architecture is supported by MQTT and CoAP (Davis et al., 2013; Thangavel, Ma, Valera, Tan, & Tan, 2014). The publish-subscribe architecture, emerged out of the need to provide a distributed, asynchronous, loosely coupled communication between data generators and destinations. The solution appears today in the form of numerous publish-subscribe Message-Oriented

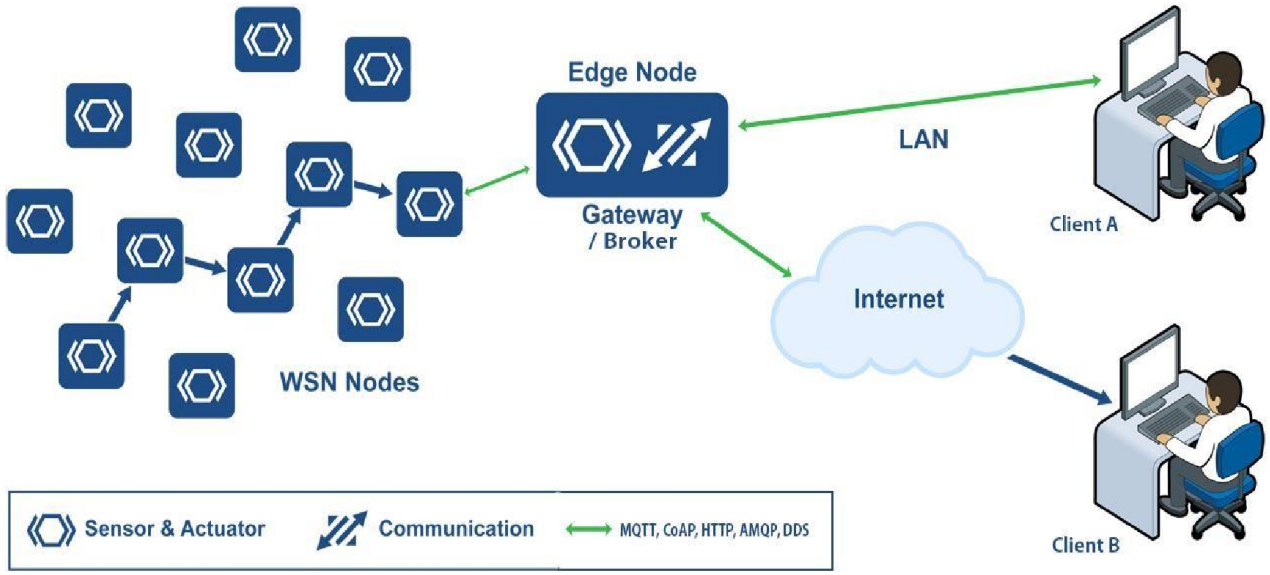

Figure 1: Typical Wireless Sensor Networks (WSNs) communication architecture

Middleware (MoM) (Jia, Bodanese, Phillips, Bigham, & Tao, 2014) and recently has been a subject of numerous research efforts (Chelloug & El-Zawawy, 2018; Hakiri et al., 2017; Veeramanikandan & Sankaranarayanan, 2017). Experimental studies as will be highlighted in this paper have shown that when CoAP is compared with MQTT, some of the disadvantages offered by one protocol is complemented by the other protocol and vice-versa, this study proposes a hybrid MQTT-CoAP protocol technique that brings the advantages of both protocols to be utilized in data communication.

2 MESSAGE QUEUE TELEMETRY TRANSPORT (MQTT) PROTOCOL

MQTT is a lightweight messaging protocol released by IBM that is based on the publish-subscribe paradigm. This makes it suitable for resource constrained devices and for non-ideal network connectivity conditions, such as with low bandwidth and high latency. The latest version used for IoT by the OASIS (Cohn, R., & Coppen, 2014) is MQTT v3.1. Because of its simplicity, and a very small message header compared with other messaging protocols, it is often recommended as the communication solution of choice in IoT. MQTT runs on top of the TCP transport protocol, which ensures its reliability. In comparison with other reliable protocols, such as HTTP, and thanks to its lighter header, MQTT comes with much lower power requirements, making it one of the most prominent protocol solutions in constrained environments. As illustrated in Figure 2 below, there are two communication parties in MQTT architecture that usually take the roles of publishers and subscribers, clients and servers/brokers. Clients are the devices that can publish messages, subscribe to receive messages, or both. The client must know about the broker that it connects to, and for its subscriber role it has to know the subject it is subscribing to. A client subscribes to a specific topic, in order to receive corresponding messages. However, other clients can also subscribe to the same topic and get the updates from the broker with the arrival of new messages. Broker serves as a central component that accepts messages published by clients and with the help of the topic and filtering delivers them to the subscribed clients. For a device to have a role of the broker, it is necessary to install MQTT broker library, for example Mosquitto broker (Eclipse, 2019), which is one of best-known open source MQTT brokers. It should be noted that there are various other MQTT protocol brokers that are open for use, which differ by way of implementation of the MQTT protocol. The clients are realized by installing MQTT client libraries. Topics in MQTT are treated as a hierarchy, with strings separated by slashes that indicate the topic level (Tantitharanukul, Osathanunkul, Hantrakul, Pramokchon, & Khoenkaw, 2017). One MQTT publisher can publish messages to defined set of topics. In this case client will publish the topic: topic/1. This information will be published to the broker which can temporally store it in a local database. The subscriber interested in this topic sends a subscribe message to a broker, specifying the same topic.

An important feature MQTT offers is the possibility to store some messages for new subscribers by setting a 'retain' flag in published messages. Brokers usually discard messages if there is nobody interested in a topic on which the publisher sends the updates. By setting a 'retain' flag to value: true, the broker is informed that it should store the published message, so it could be delivered to new subscribers. MQTT uses TCP which is quite critical for constrained devices and this has led to a proposed MQTT for Sensor Networks (MQTT-SN) which is an MQTT version that uses UDP and supports topic name indexing

(Govindan & Azad, 2015; Stanford-Clark & Truong, 2013). MQTT-SN added a feature which is the reduced size of the payloads by using numeric topic IDs rather than long topic names. At the moment MQTT-SN is only supported by a few platforms. There is a free broker implementation called Really Small Message Broker (Xu, Mahendran, Guo, & Radhakrishnan, 2017) and also EMQTT MQTT-Broker which supports MQTT-SN through a plugin (EMQTT, 2013). Since it was designed to be lightweight, MQTT does not provide encryption, and instead, data is exchanged as plain-text, which is clearly an issue from the security standpoint. Therefore, encryption needs to be implemented as a separate feature, for instance via TLS, which on the other hand increases overhead. Authentication is implemented by many MQTT brokers, through one of the MQTTs control type message packets, called CONNECT. Brokers require from clients, that when sending the CONNECT message, they should define username/password combination before validating the connection, or refusing it in case the authentication was unsuccessful. Overall, security is an ongoing research effort for MQTT (Lesjak et al., 2015).

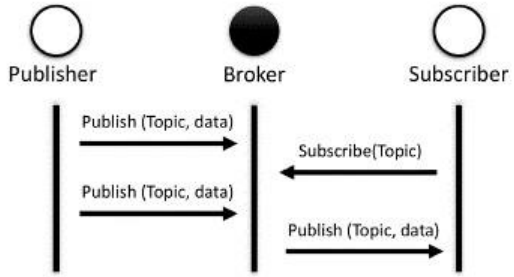

Figure 2: Publish subscribe architecture of. MQTT

3 CONSTRAINED APPLICATION PROTOCOL (CoAP)

This protocol was designed for use in constrained devices with limited processing capability by the Constrained RESTful Environments (CoRE) working group of IETF (Shelby, 2013). Similar to HTTP it is based on the request/response architecture and one of its most defining characteristics is its use of tested and well accepted REST architecture. CoAP is considered a lightweight protocol, so the headers, methods and status codes are all binary encoded, thus reducing the protocol overhead in comparison with many protocols. It also runs over less complex UDP transport protocol instead of TCP, further reducing the overhead. When a CoAP client sends one or multiple CoAP requests to the server and gets the response, this response is not sent over a previously established connection, but exchanged asynchronously over CoAP messages. The Figure 3 below shows the request response architecture of CoAP. The price paid for this reduction is reliability. Since UDP features reduced reliability, IETF has recently created an additional standard document, opening up the possibility of CoAP running over TCP (Bormann, Lemay, Tschofenig, Hartke, & Silverajan, 2018). CoAP relies on a structure that is logically divided into two layers, the request/response layer and the message layer. The request/response layer, implements RESTful architecture and allows for CoAP clients to use methods like GET, PUT, POST or DELETE when sending requests to specific URI. (Nguyen & Iacono, 2015). The request and responses are matched through a token; a token in the response has to be the same as the one defined in the request. It is also possible for a client to push data, for example updated sensor data, to a device by using method POST to its URL. As we can see, in this layer CoAP uses the same methods as REST HTTP. What makes CoAP different from HTTP is the second layer. CoAP uses its second layer known as message layer for reliability by retransmitting lost packets since UDP does not ensure reliable connection. The message layer defines four types of messages: CON (Confirmable), NON (non-confirmable), ACK (Acknowledgement), and RST (reset). The CON messages demand an ACK message as reply from the receiver while the NON messages don't request any reply. CoAP has an optional feature that allows clients to continue receiving changes on a requested resource from the server (Correia, Sacramento, & Schutz, 2016). This is achieved by adding an observe option to a GET request. The server then adds the client to the list of observers for the specific resource which allows the client to receive the notifications when resource state changes. This feature which is considered a variant of the publish-subscribe architecture, eliminates the need to poll the server repeatedly for a specific resource trying to get the changed resource state. In an attempt to get even closer to publish/subscribe paradigm, IETF has recently released the draft of Publish-Subscribe Broker that extends the capabilities of CoAP for supporting nodes with long interruptions in connectivity and/or up-time (Koster, SmartThings, Keranen, Jimenez, & Ericsson, 2019). As a security mechanism CoAP uses DTLS (Rescorla & Modadugu, 2012) on top of its UDP transport protocol based on TLS protocol with necessary changes to run over an unreliable connection giving a secure CoAPS protocol version. Most of the modifications in comparison to TLS include features that stop connection termination in case of lost or out of order packets. As an example, there is a possibility to retransmit handshake messages. Handshaking process is very similar to the one in TLS, with the exchange of client and server 'hello' messages, but with the additional possibility for a server to send a verification query to making sure that the client was sending its 'hello' message from the authentic source address. This mechanism helps prevent Denial-of-Service attacks. Through these messages, client and server also exchange supported cipher suits and keys, and agree on the ones both sides support, which will further be used for data exchange protection during the communication. Since DTLS was not originally designed for IoT and constrained devices, new

versions optimized for the lightweight devices have emerged recently (Panwar & Kumar, 2015; Raza, Shafagh, Hewage, Hummen, & Voigt, 2013). Some of the DTLS optimization mechanisms with a goal of making it more lightweight include IPv6 over Low-power Wireless Personal Area Network (6LoWPAN) header compression mechanisms to compress DTLS header (Raza, Trabalza, & Voigt, 2012). Due to the limitations of DTLS, its optimization is an open and ongoing research issue (Granjal, Monteiro, & Sa Silva, 2015; Lakkundi & Singh, 2014).

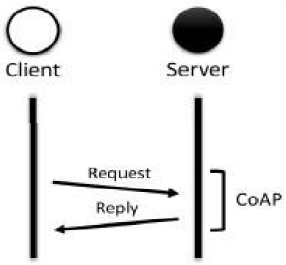

Figure 3: Request-response architecture of CoAP

4 RELATIVE ANALYSIS AND COMPARISON OF MQTT AND CoAP PROTOCOLS

This section presents a comparative analysis of the two protocols in terms of message size and overhead, power consumption and resource requirement, bandwidth and latency, reliability/QoS and Interoperability, security and provisioning, IoT usage and standardization.

4.1 Message size and overhead

In terms of both message size and message overhead, studies have shown that CoAP has the lowest compared to MQTT (Bandyopadhyay & Bhattacharyya, 2013; De Caro, Colitti, Steenhaut, Mangino, & Reali, 2013; Ngo Manh Khoi, Saguna, Mitra, & Ahlund, 2015; Thangavel et al., 2014). This is due to the fact that MQTT runs on TCP unlike CoAP which runs on UDP, thereby incurring a higher message size and message overhead for MQTT with a header size of 2-bytes per message (Naik, 2017).

4.2 Power consumption and Resource requirement

When comparing MQTT and CoAP in terms of normal power consumption and resource requirement, studies have shown that CoAP requires the lowest power and resource compared to MQTT. Various experimental studies found that CoAP consumes slightly less power and resources in similar circumstances: unreliable scenario (MQTT QoS 0 vs. CoAP NON), and reliable scenario (MQTT QoS 1 or 2 vs. CoAP CON), while assuming that no packet losses happened (Bandyopadhyay & Bhattacharyya, 2013; Colitti, Steenhaut, & Caro, 2011; De Caro et al., 2013; Ngo Manh Khoi et al., 2015; Thangavel et al., 2014).

4.3 Bandwidth and Latency

Experimental studies have shown that CoAP has the lowest bandwidth consumption and latency compared to MQTT when transferring the same payload under the same network condition (MQTT QoS 1 or 2 vs. CoAP CON) (Bandyopadhyay & Bhattacharyya, 2013; Colitti et al., 2011; De Caro et al., 2013; Ngo Manh Khoi et al., 2015; Thangavel et al., 2014). TCP is a major factor as to why MQTT consumes more bandwidth and has more latency because the technique employed by TCP during congestion is for the TCP sender to open the congestion window and double the number of packets in each round-trip time (RTT). In CoAP, a UDP transaction requires only two UDP datagrams, one in each direction; this reduces the network load response times (Naik, 2017). Experimental studies have also shown that MQTT messages experienced lower delays than CoAP for lower packet loss and higher delays than CoAP for higher packet loss. Moreover, when the message size is small and the loss rate is equal to or less than 25%, CoAP generates less extra traffic than MQTT to ensure reliable transmission (Chen & Kunz, 2016; Thangavel et al., 2014).

4.4 Reliability/QoS and Interoperability

CoAP protocol does not explicitly define a QoS whereas MQTT defines three QoS levels: 0- at most once (only TCP guarantee), 1- at least once (MQTT guarantee with confirmation), 2- exactly once (MQTT guarantee with handshake) (Bandyopadhyay & Bhattacharyya, 2013). The use of TCP in MQTT ensures higher reliability compared to CoAP. In terms of interoperability, CoAP is a HTTP-based RESTful protocol and thus offers higher interoperability compared to MQTT because all that is needed to support message exchange is in the HTTP stack (Naik, Jenkins, Davies, & Newell, 2016).

4.5 Security and provisioning

Studies have shown that MQTT is a messaging protocol with the lowest level of security compared to CoAP. MQTT has minimal authentication features aside TLS/SSL and relies only on simple username and password. CoAP on the other hand uses DTLS and IPSec for authentication, integrity and encryption (Naik, 2017).

4.6 IoT Usage and Standardization

MQTT has been employed by a larger number of organizations without yet becoming a global standard while CoAP has been less used by organizations but its technique being much closer to HTTP is considered to be of higher standardization (Naik, 2017). MQTT is an established M2M protocol and has been used and supported by the large number of organizations such as IBM, Facebook, Eurotech, Cisco, Red Hat, M2Mi, Amazon Web

Services (AWS), InduSoft and Fiorano (Bandyopadhyay & Bhattacharyya, 2013; De Caro et al., 2013; Thangavel et al., 2014).

5 RELATED WORK AND JUSTIFICATION

Dizdarevic et al (2019) in their paper on a survey of communication protocols for internet of things and related challenges of fog and cloud computing integration, discussed multiple protocol solutions for IoT to Fog computing and then to Cloud computing. In their study they proposed a HTTP-CoAP solution or an MQTT-AMQP solution but the technique used involved having for example, the CoAP protocol functioning between IoT devices and the Gateway and the HTTP protocol functioning between the gateway and the Cloud. Also, their multiple protocol solution involved protocols of similar architecture.

Thangavel et al (2014) conducted an experimental study on the performance evaluation of MQTT and CoAP using a common middleware. They designed a common middleware on the gateway using a common API capable of interfacing with CoAP, MQTT and any other lightweight protocol that they care to attach. Their multiprotocol implementation was limited to gateway/broker to client/end user and did not offer solutions for working with nodes. This work is an extension of the work done by Thangavel et al with an improvement of implementing the hybrid technique in the nodes as well and not just on the gateway.

A hybrid of MQTT-CoAP protocol is necessary because for WSNs that use only the MQTT protocol, communicating or requesting a resource from the node directly cannot be done without going through the broker which increases the latency and bandwidth consumption. On the other hand, WSNs that use only the CoAP protocol are not as scalable because of the coupling between clients and servers and also since there is no inherent capability in the CoAP protocol to retain messages from nodes.

WSN design is application specific and the application layer protocol is always a factor to consider both for sensor nodes and for gateways. Although the application specific nature of WSN design means that either of the protocols is best suited for any application, there is often need to reprogram the device when network conditions change or device application change for optimal utilization of the system resources. The hybrid MQTT-CoAP protocol system will eliminate the need for this reprogramming and also significantly reduce the time taken to decide which protocol best suits the application and network conditions. Therefore, having a hybrid protocol technique will be invaluable to an engineer looking to optimize data communication in the network.

6 PROPOSED TECHNIQUE FOR HYBRID MQTT-COAP PROTOCOL

CoAP and MQTT differ in architecture and also as discussed in the comparison in section 4 above they have complimentary advantages and disadvantages thus the need for a technique to combine both protocols.

The flow chart in Figure 4 shows the proposed hybrid technique. The technique requires the designer to make resources of their choice available via the CoAP protocol and other or similar resources available via the MQTT protocol so the node publishes topics to a broker using the MQTT protocol and also receives requests and processes these requests using the CoAP protocol. In this protocol technique, the observe feature of the CoAP protocol will be optionally replaced or handled by the MQTT protocol so as to make the hybrid solution scalable.

Since studies have shown that at lower packet loss rates and higher delay, MQTT messages have lower delay than CoAP, a decision algorithm could be developed in the gateway to instruct nodes to communicate with either of the protocols depending on the observed network condition and this is only possible because the hybrid system would make it possible for both protocols to be used simultaneously in the node.

7 CONCLUSION

In this study we briefly overviewed the MQTT and CoAP protocols and made a brief comparison between both protocols in terms of message size and overhead, power consumption and resource requirement, bandwidth and latency, reliability/QoS and Interoperability, security and provisioning, IoT usage and standardization. This is in effort to expose the limitations and strong points of the protocols with a view to hybridizing the complementing strong points of the protocols. It is anticipated that this hybrid will produce a better performing protocol system. We propose a hybrid MQTT-CoAP protocol technique for data communication not only between gateway and server/cloud/end user but also between nodes and between a node and the gateway.