2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)

CoAP and MQTT based Models to deliver Software and Security Updates to IoT Devices Over the Air

Anurag Thantharate
School of Computing and Engineering
University of Missouri
Kansas City, MO, USA
adtmv7@mail.umkc.edu

Cory Beard
School of Computing and Engineering
University of Missouri
Kansas City, MO, USA
beardc@umkc.edu

Poonam Kankariya
School of Computing and Engineering
University of Missouri
Kansas City, MO, USA
pkkdg@mail.umkc.edu

***Abstract* – The Internet of Things (IoT) is poised to revolutionize how people, industries, and enterprises connect to customers and individuals. Network Protocols, Technology and Standards such as Narrowband IoT (NB-IoT), LTE-M, 5G, LoRaWAN, Message Queue Telemetry Transport (MQTT), Constrained Application Protocol (CoAP), Device management such as Open Mobile Alliance (OMA) for Machine to Machine (M2M) are being developed to support a variety of IoT applications and services. IoT ecosystem is creating tremendous business opportunities and opening the doors for innovation. With the explosive growth of connected devices, approximately five quintillion bytes of data is estimated to be generated by the Internet every day. Not all these connections and ecosystems are secure; the security vulnerabilities are steadily increasing in parallel due to the lack of secure updating mechanisms especially for IoT ecosystem. This paper proposes three different models using the CoAP and MQTT application protocol, which aims at providing efficient mechanisms and methods for Over the Air delivery of Software Updates and Security Patches to IoT devices and evaluates which protocol is better suited for proposed models and applications.**

***Keywords* – Internet of Things, MQTT, CoAP, TCP, UDP, QoS, Software Updates, Security Patches, Application Protocols**

I. Introduction

The Internet of Things (IoT) is transforming how we interact today. Billions of devices are becoming ubiquitous in our daily lives; whether it's home, workplace, car or in our hands, they are providing instant information and keeping us connected with the world. IoT devices are the intelligent system of system, and when this intelligent system of systems share data over the cloud with each other, processes and analyzes, they transform businesses & improved society in countless ways, whether by improving medical care, developing better products quicker with lower development costs, making shopping more seamless and enjoyable or optimizing energy consumption. Many analytical and industry leaders have predicted that there will be around 25 billion connected devices by 2025 with more than one trillion dollars of IoT revenue opportunity [1] [2] [3].

Any embedded device that transmits and receives information over an IP network is an Internet of Things device. These embedded-based microcontroller devices run software on devices with constrained resources. Devices using Linux, real-time operating system (RTOS) and Android can be described as embedded systems [4]. Internet of Things embedded devices often use 8/16-bit microcontrollers and 32-bit architectures are becoming a new choice as they allow for more flexible and extensible software to run on these systems. A RTOS is one of the preferred options, with a kernel, user interface, file system, USB support, networking and more that can fit in a memory space of less than 1 MB. With RTOS the software architecture of an embedded system can be more flexible; troubleshooting and adding new features becomes dramatically simplified; it is also simpler to perform firmware upgrades. Fig. 1 presents the software stack for a typical IoT device [5]. It's essential that all devices have a robust and secure over-the-air software update mechanism that automatically patches security vulnerabilities keeps them bug-free and up- to-date with the latest features, enhances and improves stability.

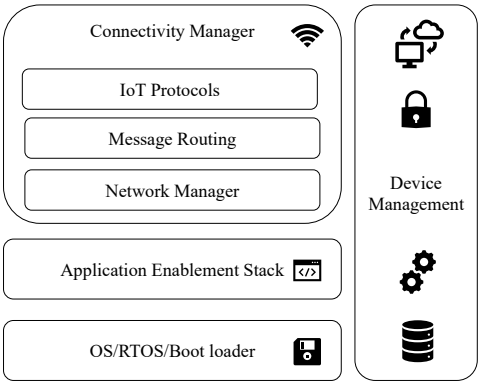

Fig. 1. IoT Device Software Stack

In the real world, the scale of typical IoT deployment is vast, and it is not practical to quickly update the software components or security patches on devices with constrained power and low memory, and it is not efficient in the cases where the devices are remote. Over-the-air requires a lot of attention to security and robustness, and it's a complex process with many moving pieces. Any interruption to update such as power loss could break the device and make it unusable. Use of insecure or outdated components, unsafe data transfer and storage, and lack of secure device management are the primary sources of vulnerabilities. This mandates device manufacturers and service providers to frequently release Over the Air (OTA) software updates and security patches to avoid unethical access and data lost/stolen by a foreign entity as we interact over IP network [6] [7]. Because IoT in general are constrained heterogeneous devices with different configurations, operating systems, and applications, it is important to choose which communication protocols are best for battery life, security, and data transfer. The system must make sure whether the firmware is legitimate and what encryption and decryption method while unloading the packaging is being used. Who is releasing the update, if it's identity and certification is legitimate or not? What type of update we are sending to devices is critically important, whether it's a software update to bootloader, system application or file systems or if it's a security update to deliver some critical vulnerability patches [6]. In the proposed model, we categorized the OTA update approach based on severity - urgent (can't wait), important (must have) and trivial (good to have). Both MQTT and CoAP protocols are widely used

978-1-7281-2980-8/19/$31.00 ©2019 IEEE
DOI 10.1109/iThings/GreenCom/CPSCom/SmartData.2019.00183

application protocols in IoT ecosystem, however these must be evaluated based on reliability, security & transmission speed to deliver software & security updates.

The rest of the paper is organized as follows: Section II provides an overview of the IoT Ecosystem and Architecture. Section III discusses the MQTT and CoAP protocol features. Section IV discusses the related work from other researchers. Section V provides a study based on experiments between MQTT and CoAP. Section VI presents our proposed model to deliver Software and Security updates. Section VII concludes the paper and talks about future work.

II. IoT Ecosystem and Architecture

An IoT ecosystem (as shown in Fig. 2) varies from solution to solution but typically consists of many different devices (i.e., things) which can communicate directly with each other, or use gateways to talk or to connect to the world (i.e., Internet). A gateway (or a broker) processes data from different devices and manages traffic between network which use different protocols. Gateway provides protocol translations and proxy-related functions in the IoT ecosystem. As these are heterogeneous devices, they usually cannot talk to the Internet or IoT applications and services directly; a gateway receives data from devices and process it for transmission to the Internet over TCP/IP and acts as a proxy. Fig. 3 represents the IoT architecture stack [8] which is developed based upon TCP/IP and broadly consist of Perception layer, Transport layer and Application layer. The IoT stack mainly runs on TCP and UDP.

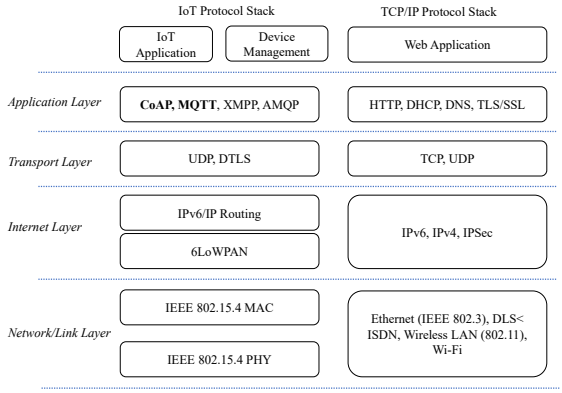

Fig. 3. Typical IoT vs. TCP/IP Protocol Stack

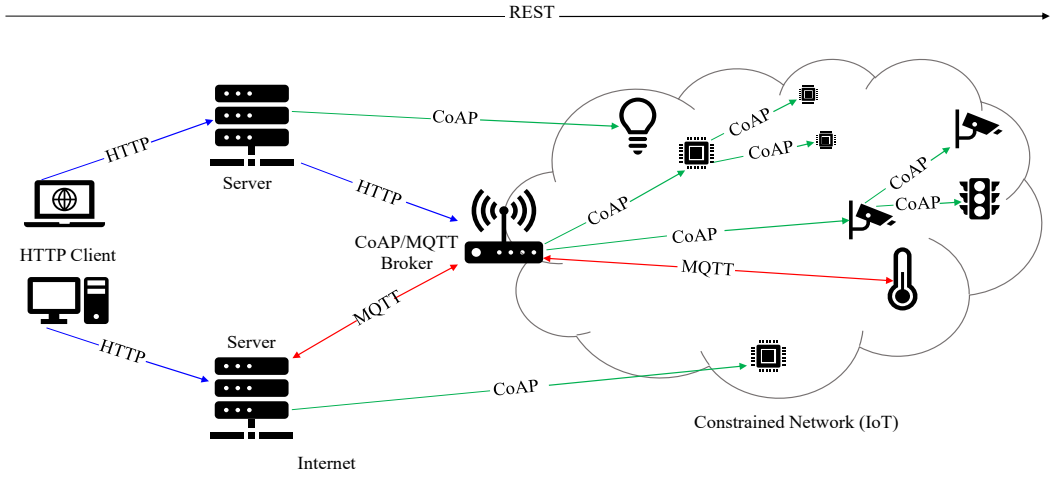

Fig. 2. IoT Protocol Ecosystem

TCP is a more sophisticated and reliable protocol but has a more complex frame structure and is not an ideal choice for lossy networks and constrained devices. IoT primarily employs UDP with application protocols like CoAP, and it incorporates TCP for the MQTT protocol. The IoT devices (actuators, sensors for example) sense and gather information about the environment, The Internet and Transport layers store, analyse, and process this data and transfer between different protocol layers through networks such as 5G, LTE, LAN, or Bluetooth. The application layer is responsible for delivering application-specific services to the users, it defines various applications in which the IoT can be deployed.

III. Messaging Protocols For IoT

This section discusses the most commonly used messaging protocols for IoT - MQTT and CoAP [8] [9].

A. CoAP (Constrained Application Protocol)

The Constrained Application Protocol (CoAP) is designed by the IETF's CoRE (Constrained RESTful Environments) [10] [12] working group to handle data communication, application and services between low power constrained nodes on lossy, constrained RF network.

A CoAP clients run on as little as an 8, 16 or 32-bit microcontroller with a limited amount of RAM and ROM and capable of throughput up to 100 kbps with high probability of packet loss. Fig. 4 show the CoAP protocol stack, where it employs Request/Response and a variant of Publish/Subscribe (Resource/Observe) architectures. The Request/Response model is like the Client/Server model in HTTP. CoAP is intended to play a similar role as HTTP does for Web Internet and is being considered a replacement of HTTP for IoT networks and is becoming a standard protocol for many IoT solutions. CoAP operates over UDP (and or CoAP over DTLS over UDP) which includes support for reliable delivery, simple congestion and flow control to communicate between endpoints. The use of CoAP over TCP is possible, but it leads to a larger overhead, more roundtrips, increased RAM and resource requirements for a device leads to larger packet sizes.

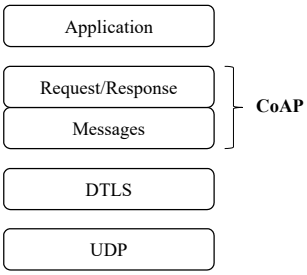

Fig. 4. Abstract Layering of CoAP

UDP handshaking is lighter, and it's easier to integrate on devices with low power microcontrollers; it is a simple protocol with low overhead. CoAP resources are specified by the Uniform Resource Identifier (URI) and these resources are accessed based on a REST method such as GET, POST, PUT or DELETE. In CoAP, the publisher publishes data to the URI and subscribers subscribed to those resources indicated by URI get notified for the new data. CoAP also uses UDP's broadcast and multicast, an essential feature for large scale IoT network. CoAP supports asynchronous message exchanges, which helps especially when the device runs on a low power cycle when the device is not always on. The message reliability is handled at the application layer, to overcome UDP unreliability in the system. CoAP employs a Request/Response model with retransmission mechanism feature. The Message layer supports four different message types CON (confirmable), NON (non-confirmable), ACK (Acknowledgement), RST (Reset). To improve security, DTLS (TLS on UDP datagram) is being used over UDP to solve reordering and packet loss problems in the constrained network.

B. MQTT (Message Queuing Telemetry Transport)

MQTT [8] [11] is a TCP-based Publish and Subscribe model messaging protocol, designed for lightweight machine-to-machine IoT communications with reliable bi-directional message delivery. MQTT was originally developed by IBM but is now part of OASIS open standards and is better suited compared to HTTP for IoT devices on constrained networks. MQTT clients publish (Publisher) messages to a Topic (e.g. 'DATA') through the MQTT server (Broker) over TCP. Topics are subscribed by other clients (Subscribers) as illustrated in the Fig. 5. Then all Clients (subscribers) subscribed to that Topic will receive the message. In the Client/Server model (e.g., HTTP), the Client directly talks to an endpoint. MQTT removes the direct communication between publisher and subscriber and talks to the Broker only, Publishers and Subscribers do not know each other's addresses and every message contains a Topic (subject) where Clients can subscribe to more than one topic and receive published messages. The MQTT broker plays a critical role as a receiver or as a server. The MQTT broker receives messages from the publisher and forwards these messages to the subscribers. The MQTT broker uses the list of topics to filter the MQTT clients that will receive the message. In essence, the Topic creates a virtual channel between the Publisher to its Subscriber. MQTT is an asynchronous protocol; it does not block the client while it waits for the message. It uses TCP as a transport layer and TLS/SSL for data security.

MQTT guarantees in-order delivery per-publisher and supports three levels of Quality of Service messages as follows.

- 0 - The broker/client will deliver the message at most once, with no confirmation and guarantee. (fire and forget model)
- 1 - The broker/client will deliver the message at least once, with confirmation required (guaranteed with acknowledgment)
- 2 - The broker/client will deliver the message exactly once with 4 step handshakes, it's most reliable but slow.

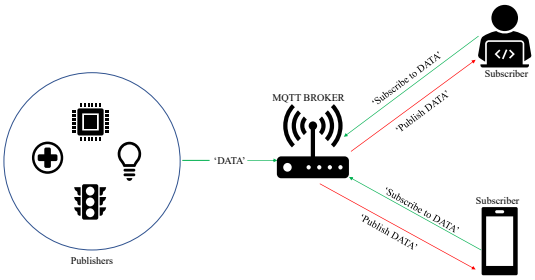

Fig. 5. MQTT Protocol and Architecture

MQTT does not support device-to-device communication nor multicast, but it supports a persistent session which is ideal for intermittent connectivity (for devices where sessions last for weeks or months), It also allows automatic keep alive message during those sporadic periods. QoS1 and QoS2 messages are queued for clients who may be offline but have not timed out. MQTT supports Last Will and Testament messages where the MQTT broker will automatically publish Last Will and Testament messages on behalf of clients with an unintentional terminated session (e.g., to send the last software version).

IV. Related Work

Authors in [13] has evaluated latency performance between CoAP & MQTT and experiments resulted in shorter response time for message delivery using MQTT and had experienced lower delays than CoAP for lower packet loss. How big or small is the payload also impacts the performance of the protocols when you consider the packet delay, packet error rate while transferring different size of data over the channel, authors in [14] conducted experiment with Raspberry Pi device which serves as a Gateway and evaluated CoAP & MQTT performance over two high bandwidth networks, results show that the CoAP performs better and is efficient when client is sending smaller payload but MQTT works better when the payload is bigger. Burak H. Çorak et al. in paper [15] compared the analysis of different IoT application protocol and measure the packet creation time from each protocol and how much time each protocol takes to transmit messages between endpoints. The work in [16] compares the different IoT application protocol efficiency, average Round Trip Time (RTT) with varying payload where CoAP with no ack achieved the highest efficiency compared to MQTT QoS0 & 1 and very similar RTT. Authors in paper [8] evaluated packet loss transmitting delays and the data transferred per messages.

V. Comparative Analysis and Performance Study between MQTT and CoAP

Both MQTT and CoAP operate on different architectures and serve different use cases and applications. Both protocols have been designed to be used in lightweight IoT environments and works well with low power and constrained RF networks, measuring them against each other is not an ideal comparative study, but we can analyze how both the protocols are performing in different RF environments with different QoS requirements individually. The testbed we have used in this setup is a cloud-based IoT performance testing platform [17], with MQTT & CoAP protocols fully implemented and where we have simulated these protocols with varying parameters of the network.

In this simple setup, we created ten clients each for MQTT and CoAP; both protocol clients are sending messages five times over two conditions provided by the simulator: "Good RF Network" (1000 Mbps) and "Constrained RF Network" (100 Kbps) every 600 seconds in two different setup environments, see the simulation setup parameters in Table 1. We choose QoS1 and QoS2 for MQTT and CON/ACK scheme for CoAP for evaluation as MQTT QoS0 and CoAP NON messages do not provide guaranteed delivery of messages. MQTT QoS1 and CoAP CON provides ACK based reliability while MQTT QoS2 guarantees that duplicate messages are not delivered to the clients. Aim for this experiment is to observe how CoAP and MQTT protocols behave in different but similar RF environment individually and how much time they take to send packets/messages between endpoints with some acknowledgement schemes. The server we have implemented to measure CoAP messages are coap.me and broker.hivemq.com:1883 broker for MQTT.

Normal RF	Constrained RF (Lossy)
{"bandwidth_unit": "Mbps", "bandwidth": 1000, "reorder": 0, "duplicate": 0, "corrupt": 0, "drop": 0, "latency_variance": 0, "latency": 0 }	{"bandwidth_unit": "Kbps", "bandwidth": 100, "reorder": 0, "duplicate": 0, "corrupt": 0, "drop": 0, "latency_variance": 0, "latency": 0 }

Table 1. Simulation Setup Parameters

The simulated results are shown in Table 2; both protocols achieved 100% success for delivery messages in both RF conditions. This means no packet loss which reflects good retransmission and handling schemes for both MQTT and CoAP. In the setup, we choose to exclude the complexity of re-ordering, duplication, corruption, dropping and latency factors. Other settings may very well change results drastically, but our focus was to simulate an ideal RF condition. Results show lower time message for MQTT for both RF conditions with QoS1 but the average time to receive packets increases when change to QoS2 due to the 4-way handshake mechanism. MQTT with QoS1 in Constrained RF conditions outperforms results in both conditions for MQTT with QoS2 and CoAP with CON/ACK schemes. This suggests MQTT QoS1 is a better choice for applications which require reliable and faster delivery.

Protocols	QoS	Network Bandwidth	Number of Clients	Number of Messages per 600 seconds	Protocol Host	Avg. Time/msg
CoAP (Good RF)	CON/ACK	1000 Mbps	10	5	coap.me	691.789 ms
CoAP (Constrained RF)	CON/ACK	100 Kbps	10	5	coap.me	1155.731 ms
MQTT (Good RF)	QoS = 1	1000 Mbps	10	5	broker.hivemq.com:1883	91.720 ms
MQTT (Constrained RF)	QoS = 1	100 Kbps	10	5	broker.hivemq.com:1883	176.460 ms
MQTT (Good RF)	QoS = 2	1000 Mbps	10	5	broker.hivemq.com:1883	214.911ms
MQTT (Constrained RF)	QoS = 2	100 Kbps	10	5	broker.hivemq.com:1883	219.107ms

Table 2. Simulation Results

Figs. 6–10 represent the average time it took for a message across ten iterations for all clients (each with five messages, for total 50 iterations of sending packets) for MQTT and CoAP in different RF conditions. The spike in the graphs represents the longer time taken by the client to send a message due to possible packet loss which leads to retransmission of messages. For a fair comparison, we plotted graphs individually for each protocol in both RF conditions.

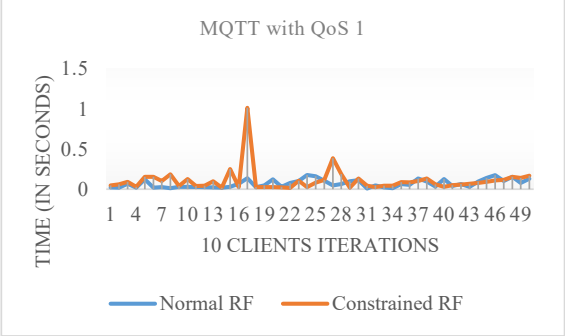

Fig. 6. MQTT Performance with QoS1 (Normal RF vs. Constrained RF)

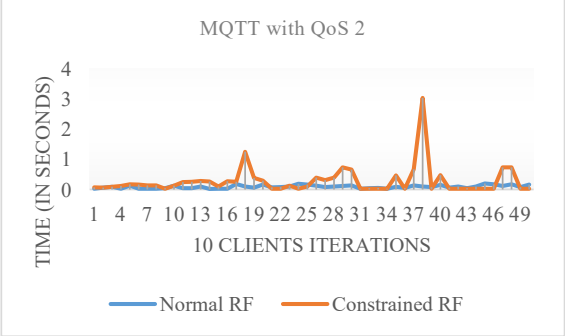

Fig. 7. MQTT Performance with QoS2 (Normal RF vs. Constrained RF)

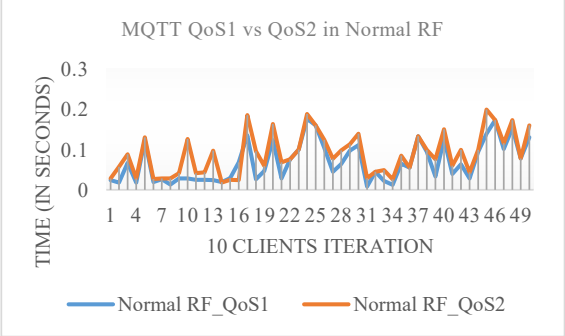

Fig. 8. MQTT Performance QoS1 vs QoS2 (Normal RF)

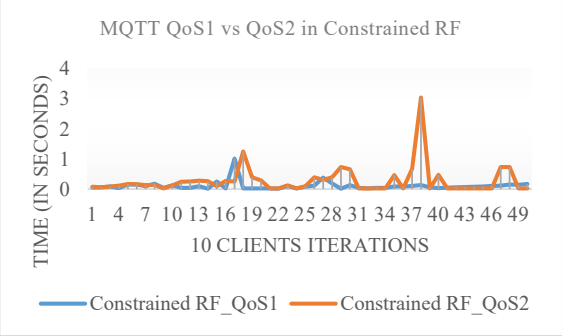

Fig. 9. MQTT Performance QoS1 vs QoS2 (Constrained RF)

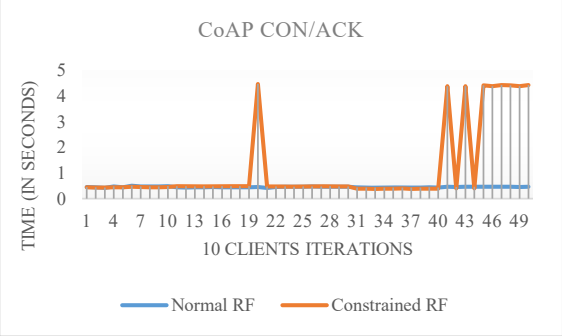

Fig. 10. CoAP Performance (Normal RF vs. Constrained RF)

VI. Proposed Model

The purpose of the evaluation is to determine which application protocol is best suited for our proposed model and use cases to send security patches and software updates OTA to constrain IoT devices. We will refer these updates as 'IOTA' in this paper going forward. Based on our results, specifications, feature support and studies of other research works, both MQTT and CoAP have its advantages and disadvantages in terms of functionality and performance. For one application MQTT works better while for other CoAP. Considering this theory, we propose the use of both protocols individually and together to send software and security updates over the air to satisfy diverse use cases.

Update Type	Severity	Desired Protocol
Security	High (cannot wait)	MQTT (A)
Software/FUMO	High (cannot wait)	MQTT (A)
Security Only	Medium (can wait)	MQTT + CoAP (B)
Software Update	Medium (can wait)	MQTT + CoAP (B)
Security Only	Low (good to have)	MQTT or CoAP (A or B)
Software Update	Low (good to have)	MQTT or CoAP (A or B)

Table 3. Proposed Model with Priorities and Updates Type

We have categorized IOTA based on update type, severity and delivery timelines to determine which messaging protocol is best suited for sending packets to devices as shown in Table 3. We start with an assumption that devices in Constrained Network require a Gateway, a Broker for communication between IoT and an Internet network. A Broker in turn can support mainly MQTT, CoAP and HTTP stack protocols for a complex solution. Because these are sleeping devices which are meant to last for a number of years with limited power, not every device will be in active state and most of the time devices will be inactive. It is crucial to properly choose protocols as it hugely impacts the life of a device. In the next section we will discuss different methods we are proposing to deliver these updates.

A. MQTT Only

MQTT is a base protocol used in this model, the Publisher who is responsible for providing Security or Software updates will publish the 'IOTA' packets to the Broker (through an HTTP client or as an MQTT publisher to the Broker), where a Broker is using an MQTT protocol stack and is subscribed to by different IoT devices for the 'IOTA' topic. Fig. 11 illustrates the proposed MQTT only model, as the Broker receives the 'IOTA' from the publisher, it stores it, and processes/forwards, i.e., publishes to devices who subscribed to 'IOTA' topic. For sleeping devices, MQTT persistence plays an important role; the Broker stores the information (even if the client is offline). When the client reconnects, the data is available immediately. This model is best suited for critical security patches or Software firmware updates which cannot wait and need reliable, guaranteed and fast delivery. There are a number of MQTT brokers being developed, we validated our model against Mosquito developed by Eclipse and HiveMQ by dc-square.

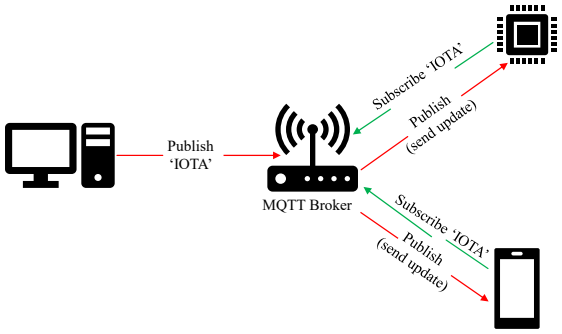

Fig. 11. MQTT Only Model

B. MQTT + CoAP (CoAP encapsulated inside MQTT)

This model will use the MQTT protocol as base and encapsulates CoAP protocol inside MQTT as URL. CoAP supports the publish-subscribe architecture using a GET method where it utilizes URI instead of topics. When a device receives the MQTT messages it reads the data from the CoAP URL and opens a UDP connection to download packets. CoAP URL triggers the predefined function on the node i.e. to download the packets. On a device URL, Model will use a GET method and then expect a packet with that data to return. It's like CoAP clients subscribe to a topic on a resource by

issuing a GET request. You can push a data packet to a device as well as POST to your URL. And a CoAP client, in turn, can send packets to another CoAP client once it receives, thereby creating a multicast scenario.

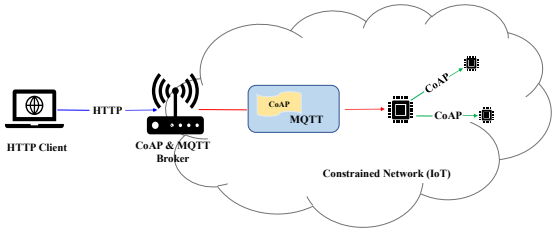

Fig. 12. MQTT + CoAP Model

C. CoAP Only

In this model, communication between IoT devices and a Gateway happens with CoAP. Once the Gateway has received software package from the provider through HTTP, it caches and downloads to forward it to CoAP clients. CoAP uses URI, unlike topics in MQTT. The Publisher publishes data to the URI and the subscriber subscribes to a resource indicated by the URI. In CoAP when a publisher publishes new data to the URI, then all the subscribers are notified about the new value as indicated by the URI. In some instances where we are pushing more substantial firmware updates, CoAP can utilize the block transfer option [18] where a large amount of data can be transferred by dividing it into a small sized data blocks in multiple request/response model. Each request/response can be treated separately which makes the server truly stateless: the server can handle each block transfer independently, with no need for a connection setup or other server-side memory of previous block transfers.

VII. Conclusion and Future Work

In this paper, we studied the working of two lightweight IoT application protocols - MQTT and CoAP in different RF conditions with their respective QoS and reliability scheme to send messages between clients. We proposed the model using both CoAP and MQTT together and individually for sending security patches and software updates to devices depending on the use cases. The quantitative simulation research shows that MQTT performs faster and more reliable compared to CoAP when the use case is to send urgent updates to devices. And CoAP is appropriate for applications which require larger data to transmit between clients in a constrained network with better network utilization. Future work will validate our study on the proposed models using 3GPP-based cellular IoT networks like NB-IoT and LTE-M with real RF conditions (i.e., re-ordering, duplication, corruption, dropping with varied latency). Research will also be conducted on how 5G is going to play a role in IoT and study any new application protocols being developed for IoT constrained networks w.r.t cellular IoT.

References

[1] IoT Cellular Networks - Altice http://www.alticelabs.com/content/WP-IoT-Cellular-Networks.pdf

[2] Internet of Things (IoT) Data Continues to Explode Exponentially https://blogs.cisco.com/datacenter/internet-of-things-iot-data-continues-to-explode-exponentially-who-is-using-that-data-and-how

[3] The internet of things by 2025 - GSMA https://www.gsma.com/iot/wp-content/uploads/2018/08/GSMA-IoT-Infographic-2019.pdf

[4] L. Belli et al., "Design and Deployment of an IoT Application-Oriented Testbed," in Computer, vol. 48, no. 9, pp. 32-40, Sept. 2015.

[5] The Three Software Stacks Required for IoT Architectures - Eclipse IoT https://iot.eclipse.org/resources/white-papers

[6] Ubuntu Security Notices https://usn.ubuntu.com/

[7] Android Security Bulletin https://source.android.com/security/bulletin

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

[9] D. Thangavel, X. Ma, A. Valera, H. Tan and C. K. Tan, "Performance evaluation of MQTT and CoAP via a common middleware," *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, Singapore, 2014, pp. 1-6.

[10] The Constrained Application Protocol (CoAP) https://tools.ietf.org/html/rfc7252

[11] Message Queuing Telemetry Transport http://mqtt.org/

[12] D. Mun, M. L. Dinh and Y. Kwon, "An Assessment of Internet of Things Protocols for Resource-Constrained Applications," *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, Atlanta, GA, 2016, pp. 555-560.

[13] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," in *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347-2376, Fourthquarter 2015 .

[14] U. Tandale, B. Momin, and D. P. Seetharam. An empirical study of application layer protocols for iot. In *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS),* pages 2447–2451, Aug 2017.

[15] B. H. Çorak, F. Y. Okay, M. Güzel, Ş. Murt and S. Ozdemir, "Comparative Analysis of IoT Communication Protocols," *2018 International Symposium on Networks, Computers and Communications (ISNCC)*, Rome, 2018, pp. 1-6.

[16] S. Mijovic, E. Shehu and C. Buratti, "Comparing application layer protocols for the Internet of Things via experimentation," *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, Bologna, 2016, pp. 1-5.

[17] IOTIFY Test bed https://iotify.io/iot-network-simulator/

[18] Block Transfer - https://tools.ietf.org/html/rfc7959