Comparative Performance of TCP and MQTT

Bishal Thapa
Ingram School of Engineering
Texas State University
San Marcos, TX USA
email: b_t220@txstate.edu

Bishal Sharma
Ingram School of Engineering
Texas State University
San Marcos, TX USA
email: dxa6@txstate.edu

Stan McClellan
Ingram School of Engineering
Texas State University
San Marcos, TX USA
email: stan.mcclellan@txstate.edu

***Abstract*— This paper compares the performance of conventional Transmission Control Protocol (TCP) with the popular Message Queueing Telemetry Transport (MQTT) protocol in private and public network settings. Higher-layer protocols, such as MQTT may be problematic for important constraints in some Internet of Things (IoT) scenarios, whereas a simpler "bare socket" TCP may be sufficient for certain scenarios. This research examines the network performance of these protocols by analyzing goodput and transmission time for a specific scenario with data security embedded at the application layer. Other features that may be suitable to be used in IoT applications are also discussed, along with their shortcomings.**

Keywords-TCP/IP; MQTT; TCP Sockets.

I. INTRODUCTION

Communication between one or more devices in a network is possible due to a specific set of protocols. The use of appropriate protocols can improve application performance or can impose extra burdens for processing and transmitting overhead data in addition to application information. Different protocols function in various layers of the familiar Open Systems Interconnection (OSI) 7-layer model [1]. The Transmission Control Protocol / Internet Protocol (TCP/IP) suite is the collection of protocols used for communication between Internet-connected devices.

The TCP transport protocol (OSI layer 4) is a connection-oriented protocol that guarantees delivery of IP packets. TCP segments the data packets received from the network layer and sends them in an ordered sequence using a specified port number [2]. Many common application layer protocols rely on TCP for guaranteed data transmission. TCP is typically used in well-known client-server scenarios, and is a critical part of most Internet communications.

MQTT is a lightweight messaging protocol (OSI layers 5-7) designed for constrained devices with low bandwidth and high latency [3]. It is widely used in IoT applications, where devices communicate with each other via cloud-based servers. In MQTT, messages are organized into topics, which describe the content of the message. Clients can subscribe to one or more topics to receive messages, or they can publish messages to one or more topics. The protocol is intended to be efficient, with minimal overhead and support for offline messaging [4]. The MQTT communication structure depends on an intervening server or "broker" to distribute messages via a publish/subscribe structure.

Many messaging protocols have been used in IoT applications, including MQTT, Hypertext Transfer Protocol (HTTP), Constrained Application Protocol (CoAP), and Advanced Message Queuing Protocol (AMQP) [2-5]. Comparing such protocols is not straightforward due to application requirements and constraints. For instance, traffic reduction, protocol efficiency improvement, communication delay reduction, and better quality of service are some obstacles to IoT implementations [5]. These considerations are important when choosing the correct protocol for specific application [5]. However, the use of additional protocol layers induces additional costs for the overall system which may be critical in certain IoT scenarios. This paper presents findings of a comparative evaluation of MQTT (layers 5-7) with "naked" TCP (layer 4) using standardized payload data. The comparison is presented in terms of header length and total time to transfer or receive the message by the client. MQTT applications are often built using TCP as the transport layer. Thus, this work attempts to quantify the "penalty" or "cost" imposed by MQTT over TCP in a relative sense. This understanding can be very important in the implementation of efficient transport and application protocols in resource constrained scenarios, such as IoT.

The remainder of the paper is organized as follows: Section II provides the overall background including the literature review, and our approach to the experiment. Section III presents the experimental setup, while Section IV presents the data gathered and summarized results of experiments. Section V presents conclusions drawn from the experiments. Section VI discusses possible future work.

II. BACKGROUND

A. Socket

A socket is an endpoint of a two-way communication in an IP network. It is an abstract data structure provided by the operating system to establish communication to transfer messages between endpoints [7]. Each endpoint is identified by a port number and IP address. The TCP transport layer recognizes the application that data is intended to be transferred to by the port number bound to the socket. Thus, the performance of data transfer via a "naked" socket establishes a baseline for comparison of application performance via additional, higher-layer protocols.

B. MQTT

MQTT is a many-to-many messaging protocol capable of transferring messages between multiple producers and consumers [8]. MQTT is described as an efficient, bi-directional, scalable, reliable, and security enabled IoT messaging protocol which can be "scaled down" effectively and implemented on microcontrollers. MQTT may use TCP to establish communication between an end-device and the intervening server, or "broker" [4].

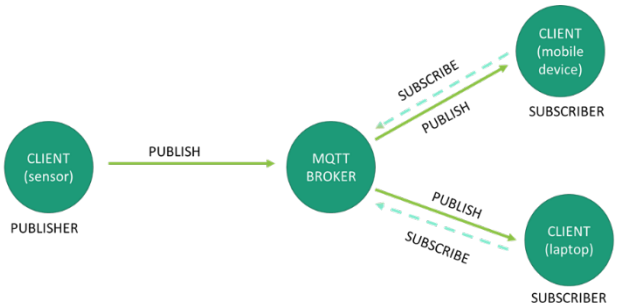

Figure 1. MQTT Publish/Subscribe Model

MQTT uses a Publish/Subscribe construct to achieve one-to-many message distribution [4]. In this structure, end-devices (clients) talk to a central authority (broker). Figure 1

 ISBN: 978-1-68558-034-6

presents the Publish/Subscribe architecture used by MQTT. Essential components of this architecture include:

Publisher (sender): The publisher is a client which sends data (messages) to the broker. Many publishers can connect to the same broker. The publisher may provide data as a part of a 'topic', to which other clients (subscribers) may subscribe in order to retrieve related messages.

Broker: The broker is the interface for the publisher and subscriber to exchange message data. The broker forwards messages to clients (subscribers) based on the topic for which the message was published, and the client is subscribed [3].

Subscriber (receiver): A client must subscribe to certain 'topic' to receive messages for that topic from the broker. If the topic specified in the SUBSCRIBE message from the subscriber matches the topic in the PUBLISH message from the publisher, then the broker forwards the message to the subscriber [5]. Subscribers can subscribe to several topics, and each topic's published messages are forwarded [8].

MQTT systems can deliver messages using three different Quality of Service (QoS) classifications [5]:

QoS 0 (At most once): The message is delivered at most once. There is no guaranteed delivery of packets and no extra methods for quality checks [5]. As a result, messages can be lost and connection reliability is dependent on the transport layer (e.g., TCP) [3].

QoS 1 (At least once): The message is delivered at least once [5]. Each message is sent multiple times, and may overlap, until acknowledged by the recipient [3].

QoS 2 (Exactly once): The message is delivered exactly once, which avoids overlapping of identical messages that may occur in QoS 1 [3].

Several studies have examined the viability of MQTT protocol in IoT and how it compares with other IoT communication protocols. A brief exploration of MQTT and CoAP outlining their architecture and message transmission mechanism is presented in [9]. An extensive comparison of MQTT and CoAP concluded that the MQTT protocol is less bandwidth efficient [10].

The comparison of the different IoT messaging protocols is not straightforward. As a result, various authors have examined the effectiveness of certain protocols under different network circumstances. For example, common messaging protocols for IoT have been compared with message overhead classified as higher for MQTT than CoAP but lesser than AMQP and HTTP. Similar conclusions are presented for bandwidth, latency, power consumption and resource requirements [11].

The performance of MQTT and CoAP have been studied under different network conditions, concluding that MQTT messages suffered lower delays for lower packet loss and higher delays for higher packet loss [12]. Additionally, the overhead for MQTT was higher for different message sizes when compared to CoAP [11]. MQTT has been compared with CoAP and Open Platform Communications Unified Architecture (OPC UA) over different cellular networks, reaching similar conclusions [3]. Additionally, the transmission time of CoAP increases every 1024 bytes leading to higher transmission time than MQTT for high payload sizes [3]. A higher payload for MQTT was expected as CoAP used UDP for its transport layer, whereas MQTT used TCP. However, for larger messages, the overhead can become higher for CoAP due to inefficiencies of User Datagram Protocol (UDP) vs. TCP for connection management [12]. Sockets have been used to send text-based data (e.g., JavaScript Object Notation) in an attempt to establish communication between Android mobile applications and IoT embedded systems [13], and some work has compared connection-oriented and connectionless transports [6].

Previous research may be helpful in understanding protocol differences. However, these outcomes didn't provide an explicit understanding of the overhead incurred by different network configurations for a range of payload sizes. Although seemingly pedestrian, this level of understanding is critical in the efficient implementation of an IoT application, which may be severely resource-constrained.

Thus, the present paper provides a comparative evaluation of MQTT and socket communications. To accomplish this comparison, two distinct experimental environments are used: LAN (controlled) and Internet (uncontrolled). Overhead for a range of message payloads is compared statistically in each environment, using TCP sockets to provide a baseline for comparison with MQTT. Further, a unique application payload is used to compartmentalize data communications, provide intrinsic security, and regularize payload structure.

C. Intelligent Cipher Transfer Object (ICTO)

ICTO is a security technology that includes mechanisms for participant authentication and authorization for access of data, which is protected by cloaking patterns. A portable dynamic rule set, which includes executable code for managing access to the protected set of participants and the protected data, is included within the ICTO. For a given user, the ICTO may provide access to some participants while preventing access to other participants based on this set of access constraints. The ICTO concept extends the conventional Authentication, Authorization, and Accounting (AAA) and Role-Based Access Control (RBAC) concepts by cloaking data at the point of generation with specific user-defined rule sets for access. The owner of data is in command to configure how or when protected data can be accessed by another party [14].

The ICTO technology is independent of what security measures are followed at the channel. Once user data has been encapsulated by a secure ICTO object, the data remains secure even in the absence of any security measures on the communication channel. As a result, the use of ICTO in IoT communications may be of particular interest. In such cases, a clear understanding of the efficiency of messaging protocols is critical. In an IoT environment, for instance, assuming a secure ICTO object has been generated, we then need an efficient data exchange protocol to reliably transmit the object from one device to another. Thus, the results of the present research may be useful in establishing an efficient framework for the exchange of ICTO objects.

III. Experimental Setup

A. TCP socket

A wireless TCP socket connection was established between two PCs similar in hardware and software where one was configured as the client that sends user data of given size and the other was configured as server that receives data sent from the client. Sending and receiving machines used Intel i5 2.4 GHz processors with Debian Linux kernels with 8 Gigabytes of memory. Python scripts were used to set up the client, server, and the packet logger programs. To trace/analyze packets being exchanged between client and server, the Python library "pcapy" was used. Pcapy is based on tcpdump [13]. The data recorded with the Python program was validated with the well-known Wireshark application-layer packet analyzer.

The machine running the client program was configured to send user data to the server and the machine running the server was configured to receive data sent by the client. Python scripts to record every packet that is seen on the network interface were run on both client and server machines during the experiments.

 ISBN: 978-1-68558-034-6

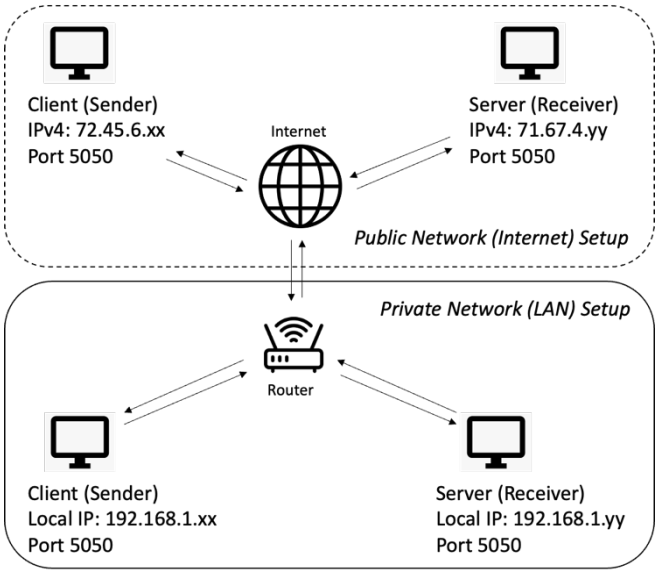

Figure 2. Experimental Setup for TCP socket communication

Experiments were performed for two different configurations of user data: file and byte sequence. In the "file" experiments, user data to be sent by the client is in a pre-existing file. In the "byte" experiments, the user data to be sent by the client is sequence of dynamically generated bytes. This procedure was repeated for two different network environments: Internet (uncontrolled) and Local Area Network (LAN) – a controlled network environment. Figure 2 illustrates the configuration in both environments.

B. MQTT

For setting up MQTT experiments, three PCs with similar hardware and software configurations were used. Since MQTT utilizes a broker to transmit information between multiple clients, the first PC was used as a sending client, the second as the receiving client, and the third as the MQTT broker, all on the wireless interface.

Python scripts were used to run the sending and receiving clients as well as the packet logger. The packet logger program executed on both the sending and receiving machines during each experiment. Other procedures were similar to the setup discussed in Section III.A regarding user data configurations (file and byte) and network environments (Internet and LAN). Figure 3 illustrates the configuration in both environments.

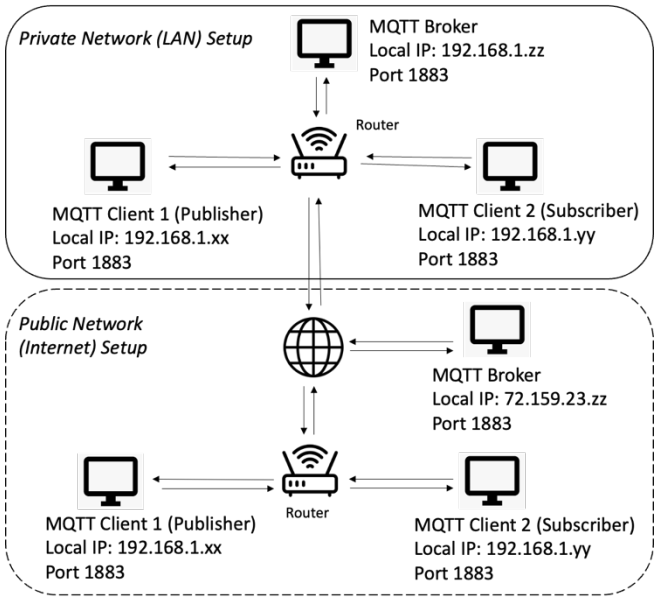

Figure 3. Experimental Setup for MQTT communication

With the experimental configurations detailed in Sections III.A and III.B, at least 150 iterations were completed for each configuration to obtain a statistically valid collection of network performance metrics. Metrics of interest included: total transmit time, total header size, header to payload ratio, and similar valuable measurements. For each experiment, total header size is calculated by summing the header lengths of TCP segment header of each IP packet exchanged between sender and receiver. User payload sizes of 1 byte, 10 bytes, 100 Kbytes, 500 Kbytes, and 1000 Kbytes were used. Figure 3 illustrates experimental setups on different network environments.

IV. Experimental Results

The results of the experiments are divided into two sections: private network (LAN) and public network (Internet). Subsequent plots consist of dependent variables describing network performance metrics (e.g., total transfer time and cumulative header size) versus the independent variable (payload size). Each point in the plot represents the mean of at least 150 iterations of each experiment. The shaded regions surrounding the mean represent the boundaries of a 95% confidence interval with unknown population standard deviation (e.g., via the t-distribution).

A. Private Network (LAN)

Plots generated from experimental data in a controlled environment (LAN) are presented comparing the relative performance of TCP with MQTT for payloads of byte sequences as well as files/objects.

Figure 4 presents the cumulative header size required for varying sizes of user data (transferred as a file or as a sequence of bytes) for both sending and receiving agents of TCP and MQTT in a private network (LAN) setting.

As expected, MQTT requires greater header size for transmitting a user payload of given size as compared to TCP. Surprisingly, with increasing payload size, header overhead for MQTT increases dramatically. In contrast, for the TCP baseline, header overhead remains constant and at least one order of magnitude smaller than MQTT.

This result indicates that TCP is substantially more efficient than MQTT for resource-constrained environments. The convenience and apparent simplicity of using MQTT for a publish/subscribe scenario comes with a "penalty" of vastly more non-payload data transfer.

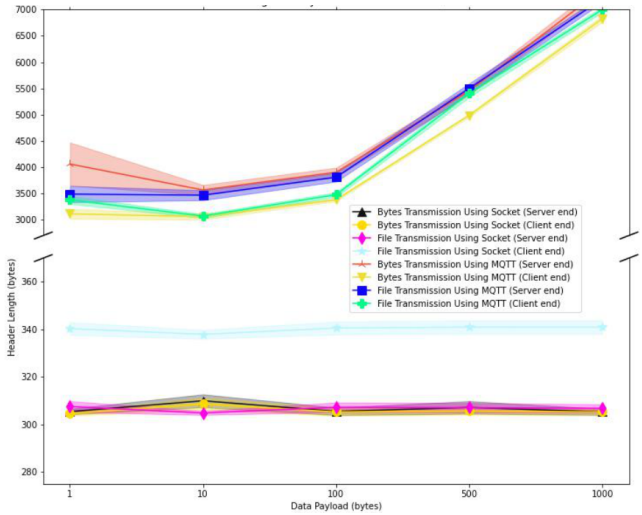

Figure 4. Cumulative Header Size vs Payload Size in LAN

This result indicates that TCP is substantially more efficient than MQTT for resource-constrained environments. The convenience and apparent simplicity of using MQTT for a publish/subscribe scenario comes with a "penalty" of vastly more non-payload data transfer.

Figure 5 presents the total transmission time for varying sizes of user data (transferred as a file or as a sequence of bytes). This data is presented for both sending and receiving agents of TCP and MQTT in a private network (LAN) environment. Although the difference between transfer times for a given payload size is relatively small, it is present and observable.

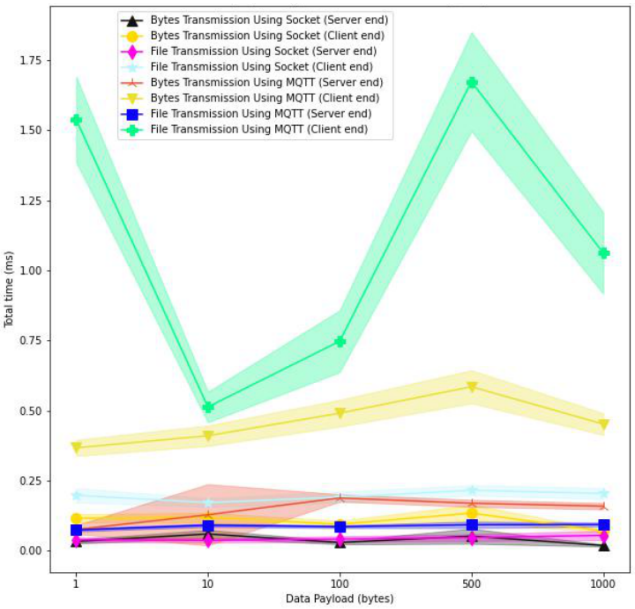

Figure 5. Total Time (ms) vs Payload Size in LAN

From Figure 5, it is clear that the time required for data exchange for most MQTT configurations is substantially higher (by a factor of 2 or more) than those for TCP. A significant difference in the cumulative header size for MQTT and TCP (observed in Figure 4) may be an intuitive reason for the observed time difference.

B. *Public Network (Internet)*

Plots generated from experimental data in a public network (uncontrolled) are presented comparing the relative performance of TCP with MQTT for payloads of byte sequences as well as files/objects.

Figure 6 presents the cumulative header size required for varying sizes of user data (transferred as a file or as a sequence of bytes). This data is presented for both sending and receiving agents of TCP and MQTT in a public network (Internet) environment.

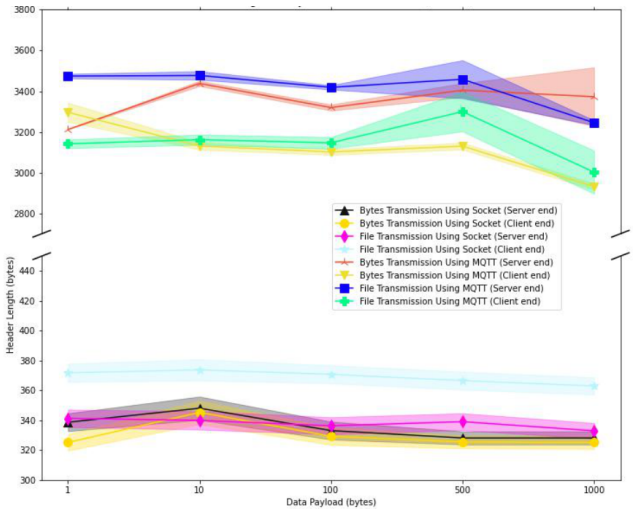

Figure 6. Cumulative Header Size vs Payload Size in Internet

As expected, and as clearly displayed in Figure 6, the header overhead for MQTT is significantly greater than TCP. However, unlike the rising trend of total header size with increasing payload size observed in Figure 4, header overhead for MQTT seems to be steady with increasing payload size in the Internet environment, but is still an order of magnitude greater than TCP.

Figure 7 presents the total transmission time for varying sizes of user data (transferred as a file or as a sequence of bytes). This data is presented for both sending and receiving agents of TCP and MQTT in a public network (Internet) environment. Wide confidence intervals may be due to the dynamic/unpredictable nature of routing, packet loss, and other factors present in Internet traffic.

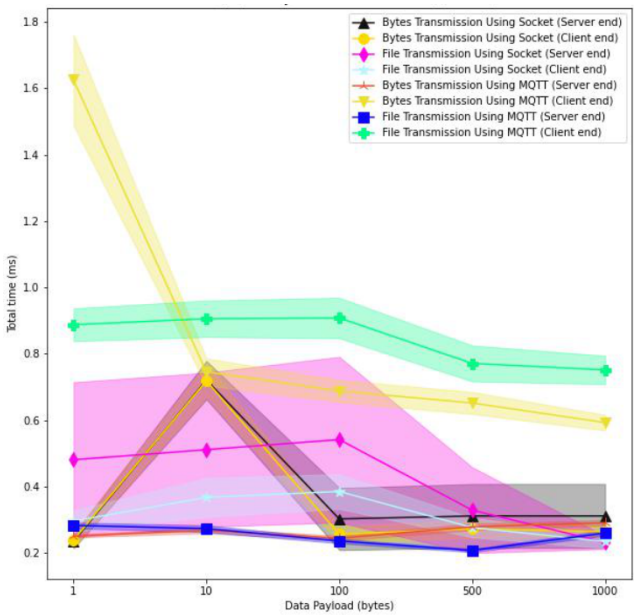

Figure 7. Total Time (ms) vs Payload Size in Internet

Regardless, a clear comparison between MQTT and TCP is evident from Figure 7 in that total transmission time for TCP is typically faster by a factor of 2 or more. This is consistent with the observations and conclusions derived from Figure 5, which performed the same experiment in the LAN (controlled) environment.

V. CONCLUSION

From the results presented in Section IV, we find that for normal data transfer, either as a file or a series of bytes, TCP to performs better in total transmit time and payload to header ratio (goodput). This is unsurprising, because MQTT leverages TCP as the transport layer. However, the overall inefficiency of MQTT is surprising, providing transmission delay of at least a factor of 2 (and typically much greater), and an overhead inefficiency of an order of magnitude, regardless of network environment. Thus, for the purpose of transmitting information, TCP sockets are substantially more efficient. The presence of a broker to moderate communication between publishers and subscribers in MQTT may provide application flexibility, but the resulting operational inefficiencies are concerning.

As in any application, additional aspects of each alternative must be considered. For instance, MQTT can operate in various QoS modes and therefore certain performance parameters like reliability and transfer time may be bounded. Additionally, the message queueing feature of MQTT enables relatively passive IoT devices to transmit/fetch data from the broker regardless of operating concurrency between the publisher and subscriber. However, MQTT typically provides minimal security through basic authentication (e.g., username, password).

As a result, in the context of IoT, a choice between the use of TCP or MQTT for ICTO object transport becomes clearer. Although MQTT's message queueing capability may be useful, the lack of security for data in-flight and at-

 ISBN: 978-1-68558-034-6

rest (e.g., on the broker) may be a critical consideration. As a result, the direct use of TCP could be preferable because of the substantial improvement in efficiency. However, ICTO technology offers security regardless of the security imposed by network protocols.

VI. Future Work

In the future, we intend to further assess MQTT in terms of network performance by comparing it to other popular protocols like CoAP. Further, we aim to experiment more thoroughly with MQTT by altering the QoS parameters between experiments, introducing multiple subscribers and publishers, and stress testing the message queuing feature. From the findings, we intend to further explore favorable communication protocols to transport ICTO objects.

References

[1] P. Gralla, “How the Internet Works,” Que Publishing, 1998.

[2] B. A. Forouzan, “TCP/IP Protocol Suite,” Guide books. https://dl.acm.org/doi/abs/10.5555/572565 (accessed Mar. 12, 2023).

[3] L. Durkop, B. Czybik, and J. Jasperneite, “Performance evaluation of M2M protocols over cellular networks in a lab environment,” in 2015 18th International Conference on Intelligence in Next Generation Networks, 2015, pp. 70–75, doi: 10.1109/ICIN.2015.7073809.

[4] “MQTT - The Standard for IoT Messaging.” https://mqtt.org/ (accessed Mar. 16, 2023).

[5] Y. Sueda, M. Sato, and K. Hasuike, “Evaluation of Message Protocols for IoT,” in 2019 IEEE International Conference on Big Data, Cloud Computing, Data Science & Engineering (BCD), May 2019, pp. 172–175. doi: 10.1109/BCD.2019.8884975.

[6] M. Xue and C. Zhu, “The Socket Programming and Software Design for Communication Based on Client/Server,” in 2009 Pacific-Asia Conference on Circuits, Communications and Systems, May 2009, pp. 775–777. doi: 10.1109/PACCS.2009.89.

[7] M. B. Yassein, M. Q. Shatnawi, S. Aljwarneh, and R. Al-Hatmi, “Internet of Things: Survey and open issues of MQTT protocol,” in 2017 International Conference on Engineering & MIS (ICEMIS), May 2017, pp. 1–6. doi: 10.1109/ICEMIS.2017.8273112.

[8] “MQTT Version 5.0.” OASIS. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html (accessed Mar. 10, 2023).

[9] D. B. Ansari, A. Rehman, and R. Ali. "Internet of things (iot) protocols: a brief exploration of MQTT and COAP." International Journal of Computer Applications 179.27 (2018): 9-14.

[10] V. Seoane, C. Garcia-Rubio, F. Almenares, and C. Campo, “Performance evaluation of CoAP and MQTT with security support for IoT environments,” Computer Networks, vol. 197, p. 108338, 2021.

[11] N. Naik, “Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP,” in 2017 IEEE International Systems Engineering Symposium (ISSE), Oct. 2017, pp. 1–7. doi: 10.1109/SysEng.2017.8088251.

[12] D. Thangavel, X. Ma, A. Valera, H.-X. Tan, and C. K.-Y. Tan, “Performance evaluation of MQTT and CoAP via a common middleware,” in 2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), Apr. 2014, pp. 1–6. doi: 10.1109/ISSNIP.2014.6827678.

[13] N. Nikolov and O. Nakov, “Research of Communication Between IoT Cloud Structure, Android Application and IoT Device Using TCP Sockets,” in 2019 X National Conference with International Participation (ELECTRONICA), May 2019, pp. 1–4. doi: 10.1109/ELECTRONICA.2019.8825568.

[14] G. S. Smith, M. L. Smith-Weed, D. M. Fischer, and E.M. Ridenour, “System and methods for using cipher objects to protect data,” (US US20220004649A1) USPTO, 2017.

[15] “tcpdump(1) man page | TCPDUMP & LIBPCAP.” https://www.tcpdump.org/manpages/tcpdump.1.html (accessed Mar. 10, 2023).

 ISBN: 978-1-68558-034-6