Implementation of Message Queuing Telemetry Transport Protocol in Model Rocket

Muchiri I. Ngethe, Yator C. Kiplimo and Shohei Aoki

***Abstract*——** Microcontroller devices are incorporated in the flight computer that convey data to the ground station in the design and implementation of model rocketry. In the field of rocketry, data transfer speed and precision are critical. Due to its small code footprints and low network bandwidth requirements, the Message Queuing Telemetry Transport (MQTT) messaging protocol tackles both of these issues. MQTT uses Transmission Control Protocol (TCP) which ensures packet delivery, as opposed to User Datagram Protocol (UDP), which is faster but does not guarantee packet delivery or even the sequence in which packets are delivered. MQTT also uses a single connection to send messages, making it faster than Hypertext Transfer Protocol (HTTP), which requires a three-way TCP handshake every time a message is transmitted. This paper describes how to incorporate MQTT messaging protocol to the flight software to facilitate communication between the onboard computer and the ground station. Publish and subscribe architecture in MQTT was utilized where the onboard computer published sensor values to particular topics and the ground station subscribed to these topics so as to receive the data. This communication was facilitated by MQTT broker that acted as an intermediary between the two clients. From field tests carried out, data transmission was found to be fast and reliable as all the data packets transmitted were received from a distance of 300 m, the onboard computer and the ground station were able to interact through Wi-Fi, and the sensor data from the onboard computer was plotted and displayed on the ground station dashboard.

***Keywords*—** HTTP, Microcontrollers, model rocketry, MQTT, MQTT broker, TCP, UDP.

I. INTRODUCTION

Developing model rockets is a difficult endeavor, owing to the large number of crucial parameters that occur during high-altitude flights [1]. During rocket flight, there is need to accurately collect and transmit flight data for active monitoring of the rocket flight variables. The embedded avionics play a key role in this regard: collecting, transmitting, and storing data [2]. The real-time monitoring of rocket flight parameters is made possible through data transmission in the field of rocketry. The transfer of commands from the ground station to the rocket, such as remote ignition and remote ejection, is made easier by communication between the rocket and the ground station. Communication needs to be efficient and reliable in order to facilitate all these operations. This will ensure that the system responds in real-time and that the various metrics, including altitude, velocity, acceleration, and Global Positioning System (GPS) coordinates for recovery, are monitored in real-time.

Wireless communication is used to establish communication between the rocket and the base station. Long-Range (LoRa), Wi-Fi, and RF communication are just a few examples of the wireless communication technologies that could be used. Different protocols may be used in Wi-Fi communication to simplify communication between two devices. HyperText Transfer Protocol (HTTP), and MQTT are a few of the different protocols that are used that utilize TCP for communication. This research focuses on the usage of the MQTT protocol for data transfer between the onboard flight computer and the ground station.

A. HTTP PROTOCOL

HTTP is an application layer protocol that uses a request/response model. A client sends a request to the server in the form of a request method containing a request header and a message body. The server responds with a status code success followed by a message containing server information [3]. By default, HTTP uses TCP as its transport protocol, while TLS/SSL is used for security [4]. This makes the connection between the client and the server to be connection-oriented. Analogous to publish/subscribe models, HTTP employs Uniform Resource Identifiers (URI) instead of topics. The URI is used by the client and server to receive and send data respectively. Further, HTTP requires additional support for Quality of Service (QoS) assurance [5].

B. MQTT PROTOCOL

Client Servers broadcast or subscribe to the messaging transport protocol known as Message Queue Telemetry Transport (MQTT). It is transparent, compact, and made to be easy to use. TCP/IP or any network protocol that permits bi-directional, lossless connections in a systematic way is used to build the protocol. Among the features of MQTT are the publish/subscribe message pattern, which permits one-to-many message dissemination, and a messaging transport that is unrestricted by the payload. Three levels of message delivery quality are included in this protocol, including "At most once,"

Muchiri I. Ngethe, Department of Telecommunication and Information Engineering, JKUAT (+254706230020; e-mail: ianmuchiri8@gmail.com).
Yator C. Kiplimo, Department of Telecommunication and Information Engineering, JKUAT (email: christianyator7@gmail.com)
Shohei Aoki, Department of Mechatronics Engineering, JKUAT (email: aoki@jkuat.ac.ke)

where messages are sent using the best efforts of the operating environment. This level may be employed if there is message loss. Second, "At least once," when message delivery is assured but message duplication is likely. The final selection, "Precisely once," ensures that messages will be delivered precisely once. Although it would significantly reduce network traffic, this level might be used [6]. In addition to eliminating transport overhead and protocol exchange to save network traffic, the MQTT protocol has a noteworthy feature that notifies interested parties when an unexpected disconnection occurs [7].

Many microcontrollers available today support the TCP/IP stack. As a result, integrating MQTT into devices or projects that employ the aforementioned microcontrollers is made simple. The two main elements required for MQTT implementation are MQTT client installed on the microcontroller or device and MQTT broker server to manage publish and subscribe data. The main advantage of the publish/subscribe system is that a broker stands between the publisher and the subscriber, preventing direct communication between the two devices [8]. The MQTT architecture is as shown in Fig.

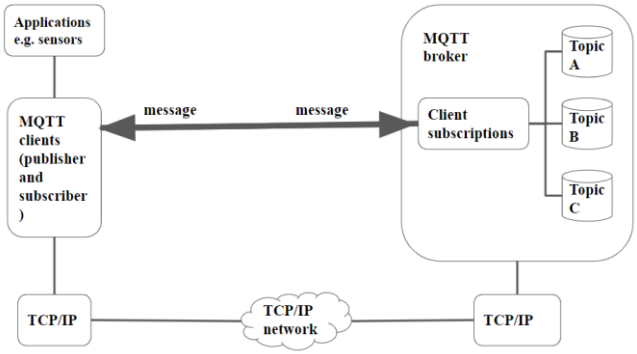

Fig. 1 MQTT Architecture

II. SYSTEM IMPLEMENTATION

The following parts were needed to implement the MQTT protocol in the telemetry system: a Raspberry Pi 4, a NodeMCU-ESP32S, a BMP180, an MPU6050, a Neo 6M GPS module, a PowerBeam M2-400 antenna, and a Router.

The system communicated over Wi-Fi thanks to the NodeMCU-ESP32s, a low-cost, low-power, dual core system on chip (SoC) microcontroller. During flight, the rocket's altitude was measured using the BMP180 barometric pressure sensor to calculate how high the rocket would soar. The MPU6050 is a 6-axis motion tracking sensor consisting of a three-axis accelerometer and a three-axis gyroscope [9]. It was used to calculate displacement, acceleration, direction, and speed. The rocket's location was determined via the low-cost, high-efficiency NEO-6M global positioning system (GPS) module [10]. At the ground station, a 2.4 GHz directional parabolic antenna called the PowerBeam served as a signal receiver. Our ability to broadcast over Wi-Fi across great distances was made possible by its high gain of 18 dBi and receiver sensitivity of -94 dBm. A router was employed to establish an access point at the ground station by utilizing the PowerBeam antenna's signal as the Wide Area network (WAN) input. The ground station server was a low-cost quad core 32-bit Wi-Fi and Bluetooth single-board computer called the Raspberry Pi 4.

The onboard microcontroller served as the publisher, publishing flight data to the various topics for transmission. The microcontroller also acted as a subscriber to ejection topics which enabled a user at the ground station to send commands to the onboard computer such as remote ejection. The Raspberry Pi 4 was an MQTT broker and was used to facilitate the exchange of data between the publisher and the subscriber. The subscriber to the topics published by the onboard flight computer was the node.js server which subscribed to the topics and used the data to visualize graphs on a react web application.

The sensors were connected to the nodeMCU-ESP32s and transmitted data to it via serial communication. The data was processed in the microcontroller to filter out noise before being transmitted. The microcontroller created an access point to be used for Wi-Fi communication with the ground station. The data that had been processed was published to the various topics before being transmitted wirelessly to the ground station where the broker was located. The signal was received at the ground station by the PowerBeam antenna which connected to the onboard access point. The PowerBeam antenna sent the received signal to the router which then created an access point at the ground station. The Raspberry Pi 4 was connected to the ground station access point hence it was linked to the onboard computer. Thus, the published messages were received by the broker installed in the Raspberry Pi. In our case we used an MQTT broker known as Mosquitto broker. The Node.js server was also installed in the Raspberry Pi and it subscribed to the topics published to the Mosquitto broker by the onboard computer. The Node.js then sent the data to the react web application which visualized the data using graphs and texts. The complete system architecture is as depicted in Fig.

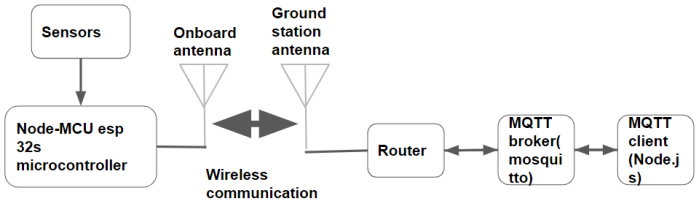

Fig. 2 Telemetry system Architecture

III. RESULTS AND DISCUSSION

A ground test was carried out, and data was successfully transmitted over a distance of about 300 meters as shown in Fig. The onboard station was carried across a field while still remaining within line of sight of the ground station antenna.

Fig. 3 Transmission Distance

The onboard station collected the data from the various sensors and published the data to the topic esp32. The data transmitted from the onboard station was successfully received by the ground station Mosquitto MQTT broker. The Node.js which was the web application's server, subscribed to the topic and received the data from the broker. The system interaction was as shown in fig. 4.

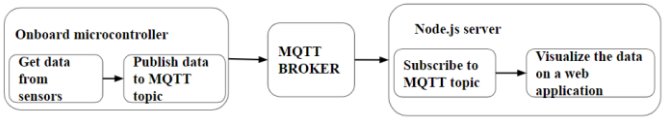

Fig. 4 MQTT Architecture

The server then transferred the data to the client side of the web application which was a react based web application for visualization.

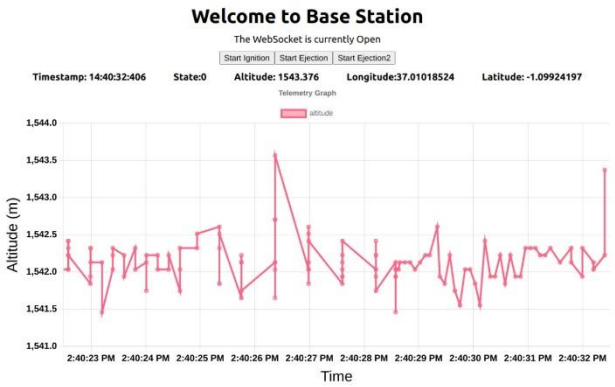

Fig. 5 Data Visualization

Based on the data received at the base station, it was observed that the system's altitude was not increasing significantly since the test movements were performed horizontally on the ground. The minimal fluctuations observed on the altitude were due to sensor noise. Furthermore, from the data collected, the team was able to observe the real-time location of the system as it moved due to the longitude and latitude positions obtained from the GPS sensor.

TABLE I
COMPARISON BETWEEN MQTT AND HTTP

Test number	Quantity of data packets transferred via MQTT	Quantity of data packets transferred via HTTP
1	1865	32
2	1900	30
3	2015	29
4	1810	30
5	2050	29
6	1950	28
7	1825	29
8	1955	30
9	2005	29
10	1995	28
Mean	1937	29.4

Furthermore, this study carried out a comparison test between HTTP and MQTT protocol to determine which protocol had a better performance. The study involved determining the number of data packets transmitted from the onboard station to the ground station over a period of 1 minute. The table above shows the results of the test carried out. From the test results, it was clear that MQTT was faster in data transmission as compared to HTTP. The data in the table depicts that MQTT data transmission in this case was approximately 65 times faster as compared to HTTP

From the results of the test, the data transmission system utilizing MQTT protocol proved to be fast and efficient in data transmission. This is because when utilizing MQTT protocol, we can be able to reuse a single connection to send multiple messages over the channel.

IV. CONCLUSION

In this study, the MQTT protocol was successfully integrated into a model rocket's telemetry system. The test results depict that the MQTT protocol is both fast and reliable in data packet delivery due to the quality of service (QOS) it offers. Thus, MQTT protocol can be used as one of the methods of real-time data transmission in a model rocket telemetry system.

ACKNOWLEDGEMENT

The authors thank Jomo Kenyatta University of Agriculture and Technology for the use of the Integrated Prototyping and Innovation Center. The authors also acknowledge the financial support of AFRICA-ai-JAPAN project (JICA) and the technical support of Mr. Ben Maniafu.

REFERENCES

[1] G. de A. Souza, L. Barbosa, G. Ramalho, and A. Zuquete Guarato, "Low Cost Real Time Rocket Telemetry System Design," no. January, 2019, doi: 10.26678/abcm.cobem2019.cob2019-1585.

[2] "Introduction to Avionics Systems - R.P.G. Collinson - Google Books." https://books.google.co.ke/books?hl=en&lr=&id=NlQFCAAAQBAJ&oi=fnd&pg=PR7&dq=.+Introduction+to+Avionics&ots=lCc51M8akU&sig=yOrCEJbkkKsFPnjEpqMFyC31y_4&redir_esc=y#v=onepage&q=.%20Introduction%20to%20Avionics&f=false (accessed May 29, 2022).

[3] R. Fielding *et al.*, "Hypertext Transfer Protocol – HTTP/1.1," Jul. 1999, doi: 10.17487/RFC2616.

[4] I. Grigorik, "Making the web faster with HTTP 2.0," *Commun ACM*, vol. 56, no. 12, pp. 42–49, Jul. 2013, doi: 10.1145/2534706.2534721.

[5] N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," *2017 IEEE International Symposium on Systems Engineering, ISSE 2017 - Proceedings*, Jul. 2017, doi: 10.1109/SYSENG.2017.8088251.

[6] "MQTT - The Standard for IoT Messaging." https://mqtt.org/ (accessed Jun. 30, 2022).

[7] S. Kraijak and P. Tuwanut, "A survey on internet of things architecture, protocols, possible applications, security, privacy, real-world implementation and future trends," *International Conference on Communication Technology Proceedings, ICCT*, vol. 2016-February, pp. 26–31, Feb. 2016, doi: 10.1109/ICCT.2015.7399787.

[8] G. Sasikala *et al.*, "IoT real time data acquisition using MQTT protocol," *Journal of Physics: Conference Series*, vol. 853, no. 1, p. 012003, May 2017, doi: 10.1088/1742-6596/853/1/012003.

[9] "What Is MPU6050? - Arduino Project Hub." https://create.arduino.cc/projecthub/CiferTech/what-is-mpu6050-b3b178 (accessed Jul. 01, 2022).

[10] "NEO-6M GPS Module — An Introduction." https://www.electroschematics.com/neo-6m-gps-module/ (accessed Jul. 01, 2022).