Accredited SINTA 2 Ranking
Decree of the Director General of Higher Education, Research, and Technology, No. 158/E/KPT/2021
Validity period from Volume 5 Number 2 of 2021 to Volume 10 Number 1 of 2026

Published online at: **http://jurnal.iaii.or.id**

JURNAL RESTI
(Rekayasa Sistem dan Teknologi Informasi)
Vol. 8 No. 1 (2024) 119 - 126 e-ISSN: 2580-0760

A Comparative Study of HTTP and MQTT for IoT Applications in Hydroponics

Irvan Rizki Nugraha[1], Widhy Hayuhardhika Nugraha Putra[2], Eko Setiawan[3]
[1,2]Information System Department, Fakultas Ilmu Komputer, Universitas Brawijaya, Malang, Indonesia
[3]Informatics Engineering Department, Fakultas Ilmu Komputer, Universitas Brawijaya, Malang, Indonesia
[1]irvanriskinugraha90@gmail.com, [2]widhy@ub.ac.id, [3]ekosetiawan@ub.ac.id

Abstract

Hydroponics is based on nutrients in water. It must be regularly monitored to prevent plant defects. The Internet of Things has become a solution for remote hydroponic monitoring and is currently being tested on the Yuan Hidroponik Kelompok Wanita Tani (KWT). This system will send data every minute, and each data has a possibility of loss in transmission. There is a chance that this system will be implemented in other hydroponic organizations. As more devices are involved, it will affect server resources. This research will compare Message Queue Telemetry Transport (MQTT) and Hypertext Transfer Protocol (HTTP) as popular protocols used in IoT. A test with increasing clients shows that at 50 clients HTTP needs 87% CPU, while MQTT needs 22.63% CPU. A test with increasing payload shows that at 10,000 payload HTTP needs 94% CPU while MQTT needs 28.35% CPU. A test with fixed clients and payloads shows that HTTP has a CPU limit based on the clients involved. A transfer time test shows that HTTP needs 177.344 seconds while MQTT needs 3.24 seconds. An acceptance rate is calculated by incrementing the count for every incoming payload. It shows that HTTP can receive 30,000 payloads, unlike MQTT which can only receive 1680 payloads before losses.

Keywords: *hydroponics; IoT; web service; MQTT; HTTP*

How to Cite: I. R. Nugraha, W. H. N. Putra, and E. Setiawan, "A Comparative Study of HTTP and MQTT for IoT Applications in Hydroponics", J. RESTI (Rekayasa Sist. Teknol. Inf.), vol. 8, no. 1, pp. 119 - 126, Feb. 2024.
DOI: https://doi.org/10.29207/resti.v8i1.5561

1. Introduction

Hydroponics is a method of growing plants using a solution of mineral nutrients in water [1]. Since it relies on nutrients in the water, lack or excess nutrients can lead to discoloured or defective leaves [2]. This makes hydroponics necessary to be monitored regularly.

The Internet of Things (IoT) comes as a solution to help farmers remotely monitor hydroponics. The solution was tested on *Kelompok Wanita Tani* (KWT) Yuan Hidroponik. The owner of KWT Yuan Hidroponik mentioned the flow of nutrition addition in a manual way. The nutrient value in the container must be checked before and after adding nutrients. On the basis of that habit, the system will send data every minute. Each minute of data has the possibility of loss in transmission.

Currently, this system is focused on KWT Yuan Hidroponik as a single user. There is a chance this system will be implemented in other hydroponic organizations as a multi-user. This may lead to more devices involved that impact server resources. The popular protocols used in IoT are MQTT (Message Queue Telemetry Transport) and HTTP (Hypertext Transfer Protocol) based on the Eclipse 2023 IoT & Edge Developer Survey. This survey also shows that MQTT has a higher use rate than HTTP.

MQTT is a lightweight publish-subscribe protocol used for IoT communication [3]. This protocol is designed to focus on minimizing network bandwidth and device resources to ensure reliable delivery [4]. Meanwhile, HTTP is a protocol that becomes the basis of the client-server model used in websites [5]. This protocol can transfer many data in tiny packets, which can cause a large overhead. This overhead can cause serious bandwidth problems in IoT [6].

The HTTP and MQTT protocols exhibit packet loss, according to a study on the use of IoT in vehicle air quality. Despite HTTP's slower speed than MQTT, this study demonstrates that it has less packet loss [7]. A

Received: 20-11-2023 | Accepted: 14-01-2024 | Published Online: 18-02-2024

study on the implementation of IoT on the electrocardiogram (ECG) shows that packet loss in MQTT follows the growing number of patients [8]. Network congestion is termed packet loss at the network level in this study. In almost every experiment, a slightly higher packet loss on MQTT than HTTP is observed in another study on the integration of IoT in smart wheelchairs [9]. This study describes packet loss as the loss of a few data packets in a network. Thus, both studies highlight similar issues about packet loss at the network level in their IoT implementation, respectively.

By focusing on application-level performance, this study filled the research void left by the two network-level studies. Rather than watching the network, it will use the count variable to evaluate the message it received. Resource overload can cause the application or the MQTT broker to drop, which can impact application-level performance. Thus, although there is a small packet loss for both protocols at the network level, there may be a greater packet loss at the application level. This research is essential since packet loss can result in erroneous nutrient values, as was the case with IoT-based ECG [8]. More packet loss will occur if the application is down due to resource overload, which will stop the hydroponic system.

2. Research Methods

2.1 Testing Architecture

Figure 1 illustrates the architecture of the system used to test the tools associated with each protocol. A virtual machine (VM) is used for each protocol. VM for MQTT is equipped with the MQTT Broker EMQX. This is done to demonstrate the impact of the local MQTT broker on CPU usage. Htop was used to monitor CPU usage in several tests [10].

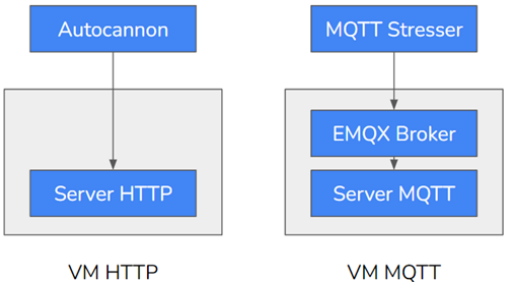

Figure 1. System Architecture for Testing

Both VMs were hosted on Google Cloud Platform (GCP) using the n1-standard-1 machine type. The more detailed specification of n1-standard-1 is explained in Table 1 [11]. There are also differences in how the HTTP and MQTT protocols operate. HTTP uses a request-and-response approach [12], while MQTT uses a publish-and-subscribe approach [13]. This leads to the use of different testing tools.

The MQTT protocol in this research operates on Quality of Service (QoS) 0. This QoS level will make public that the reader does not receive any information about the delivery of messages [14].

Table 1. VM Specification

Property	Value
vCPU	1
RAM	3.75 GB
Disk	10 GB

The HTTP protocol will be tested using Autocannon to send many requests to the HTTP server [15], while the MQTT protocol will be tested using MQTT Stresser to send many messages to the local MQTT broker [16].

2.2 Payload Count Definition

There is a difference in defining the payload count between Autocannon and MQTT Stresser. In Autocannon, the payload count can be defined directly. Meanwhile, the payload count in MQTT Stresser must be defined for each client. For a more detailed explanation, see Table 2.

Table 2. Payload Rounding for MQTT

Client	Payload	Payload per Client	Rounding
60	1,000	17	1,020
60	5,000	84	5,040
60	10,000	167	10,020
60	15,000	250	15,000
60	20,000	334	20,040

To calculate the payload per client in MQTT Stresser, the payload count is divided by the client count as shown in Formula 1.

$$payload\ per\ client = \left\lceil \frac{Payload}{Client} \right\rceil \quad (1)$$

This division results in a decimal value. The value will be rounded up to the nearest whole number. This rounding process ensures that each client receives an equal share of the payload. It is done to prevent fractional portions that could lead to imbalances in test conditions.

To calculate the rounding value, the payload per client is multiplied by the client count as shown in Formula 2.

$$rounding = payload\ per\ client \times payload \quad (2)$$

The difference between the initial payload and the rounded payload will be calculated from the rounding result. For example, a test with 5,000 payloads distributed among 60 clients would result in a rounded payload value of 5040. The difference of 40 is a direct consequence of the rounding operation.

2.3. *Client Count Testing*

This test aims to calculate the effectiveness of increasing the client count of both HTTP and MQTT. The parameters used in this test are detailed in Table 3. The number of clients is incrementally increased until CPU utilization reaches 100%. Each test is repeated ten times, and the results are averaged.

Table 3. Client Count Parameters

Property	Value
Fixed Payload	10,000
HTTP Client Increment	5
MQTT Client Increment	100

A fixed payload of 10,000 is used for both protocols. For HTTP, the testing starts with 5 clients and increases incrementally by 5. Meanwhile, for MQTT, the testing starts with 10, 50, and 100 clients with increments in multiples of 100. The huge increase in MQTT clients is designed to accelerate MQTT testing.

This study follows the fixed scenario described in the previous literature [17]. However, our research uses more scenarios and will stop if CPU usage reaches 100%. The results of this test are presented in a table and graph. The first protocol that reaches almost 100% CPU usage will be compared with other protocols on the same clients. The MQTT data points will be very few compared to HTTP. Trendline is used to draw the line in the combined graph for a better view.

2.4 Payload Count Testing

This test aims to calculate the effectiveness of increasing the payload count on both HTTP and MQTT. The parameters used in this test are detailed in Table 4.

Table 4. Payload Count Parameters

Property	Value
Fixed Client	60
HTTP Client Increment	500
MQTT Client Increment	5,000

A fixed client of 60 is used for both protocols. For HTTP, the test starts with 1,000 payloads and increases incrementally by 500. Meanwhile, for MQTT, testing starts with 1,000, 5,000 and increases incrementally by 5,000. The huge increase in MQTT payloads is designed to accelerate MQTT testing.

The results of this test are presented in a table and graph. The first protocol that reaches almost 100% CPU usage will be compared with other protocols on the same payloads. The MQTT data points will be very few compared to HTTP. Trendline is used to draw the line in the combined graph for a better view.

2.5 Key Determining Factors

The purpose of this test is to determine whether increasing the client or payload affects CPU usage more on HTTP and MQTT. Table 5 provides specific information on the test parameters. Twenty, forty, and sixty clients are used in this test, with payload sizes ranging from 1000 to 5000. Growth will persist in multiples of 5,000 to 30,000. Even if one of the protocols does not reach 100% CPU utilization, the test will still be terminated.

The results of this test are presented in a table and graph. If data points increase with an increasing number of clients, it indicates that the protocol is more affected by increasing number of clients. Otherwise, if the data points increase with increasing payload, it indicates that the protocol is more impacted by increasing payload.

Table 5. Key Determining Factors Parameters

Property	Value
Client Increment	20
Maximum Client	60
Payload Increment	5,000
Maximum Payload	30,000

2.6 Total Transfer Time

This test aims to calculate the time needed to transfer all payloads on both HTTP and MQTT. The parameters used in this test are detailed in Table 6.

Table 6. Total Transfer Time Parameters

Property	Value
Fixed Client	60
Payload Increment	10,000
Maximum Payload	100,000

This test starts with 1,000 and 10,000 payloads increasing in multiples of 10,000 to 100,000. The results of this test are presented in a table and graph. The line with a lower gradient indicates that the protocol is faster than the others.

2.7 Acceptance Rate

This test aims to calculate the number of payloads received on both HTTP and MQTT. The parameters used in this test are detailed in Table 7.

Table 7. Acceptance Rate Parameters

Property	Value
Fixed Client	60
Payload Increment	5,000
Maximum Payload	30,000

The payloads received are calculated by the count variable. The count will increase by 1 for every incoming payload. A study using a similar approach involves inserting data into the database for each incoming payload [18].

This test starts with 1,000, 5,000 and increases incrementally in multiples of 5,000 to 30,000. The results of this test are presented in a table and graph. The protocol with packet loss greater than 50% indicated that the protocol is not reliable. It will be tested to identify its saturation point.

There is a study that uses the opposite calculation method that focuses on the percentage of packet loss [19]. The packet loss can still be tolerated if it occurs for about 2%.

2.8 Saturation Point

This test aims to calculate the more accurate saturation point of the unreliable protocol based on the previous test. There is a chance that packet loss occurs below 5000 payloads. The parameters used in this test are detailed in Table 8.

Table 8. Saturation Point Parameters

Property	Value
Fixed Client	60
Payload Increment	125

This test starts with 125 payloads and increases incrementally by 125. The test will stop to determine whether payload loss has occurred. The payload count will be recorded for the reverse test. The reverse test uses a payload count from the previous test. It begins with 5 clients and increases incrementally by 5. The test will stop to determine whether payload loss has occurred. The parameters used in this reverse test are detailed in Table 9.

Table 9. Reverse Test Parameters

Property	Value
Fixed Payload	<from the last test>
Client Increment	5

The results of this test are presented in two tables and graphs. Data points that dropped below 50% indicate the saturation point of that protocol.

3. Results and Discussions

3.1 Client Count Testing

The results of comparing the CPU usage of the HTTP protocol with the increasing client count can be seen in Table 10.

Table 10. HTTP Client Count Testing

Payload	Client	Average CPU Usage
10,000	5	17.75%
10,000	10	28.00%
10,000	15	40%
10,000	20	45.28%
10,000	25	56.11%
10,000	30	60.73%
10,000	35	68.63%
10,000	40	75.28%
10,000	45	80.67%
10,000	50	87%
10,000	55	91.31%
10,000	60	97%

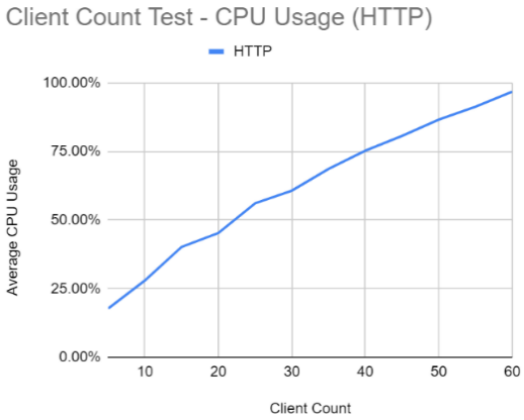

Figure 2. HTTP Client Count Testing

Table 10 shows that HTTP can handle up to 60 clients before reaching almost maximum CPU usage. The test results are also shown graphically in Figure 2.

Table 11. MQTT Client Count Testing

Payload	Client	Average CPU Usage
10,000	10	17.51%
10,000	50	22.63%
10,000	100	27.54%
10,000	200	33.99%
10,000	300	39.55%
10,000	400	46.71%
10,000	500	55.92%
10,000	600	61.80%
10,000	700	67.70%
10,000	800	70.44%
10,000	900	73.94%
10,000	1,000	77.08%
10,000	1,100	82.55%
10,000	1,200	86.50%
10,000	1,300	88.38%
10,000	1,400	92%
10,000	1,500	97.14%

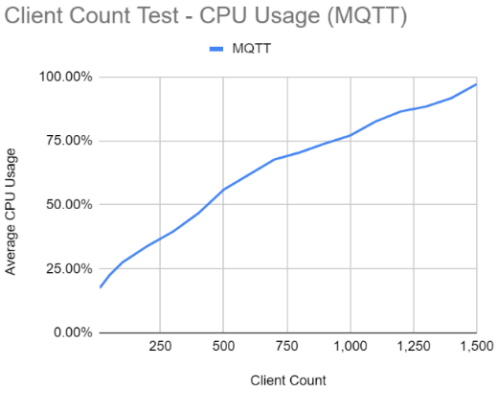

Figure 3. MQTT Client Count Testing

The same test was also carried out for the MQTT protocol. The result of comparing CPU usage of the MQTT protocol with increasing client count can be seen in Table 11. Table 11 shows that MQTT can handle up to 1,500 clients before reaching almost maximum CPU usage. The test results are also shown graphically in Figure 3.

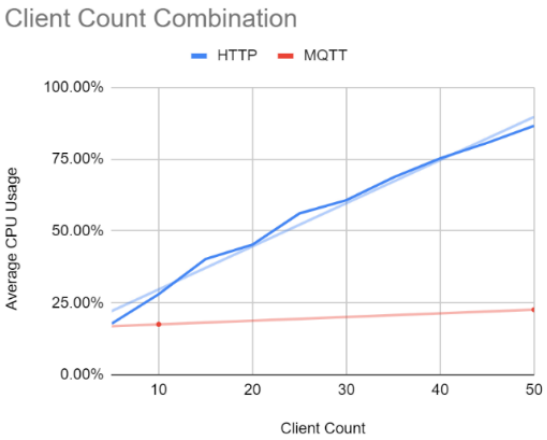

Figure 4. Combination Client-Count Testing

The combination of HTTP and MQTT client count graphs is depicted graphically in Figure 4. Trendline is used due to the imbalanced data point of MQTT. Figure 4 shows that for 50 clients, MQTT only needs 22.63% CPU usage. Meanwhile, HTTP needs 87% CPU usage.

This test shows that MQTT is more resource efficient than HTTP in terms of client count.

3.2 Payload Count Testing

The results of comparing the CPU usage of MQTT with increasing payload count can be seen in Table 12.

Table 12. HTTP Payload Count Testing

Client	Payload	Average CPU Usage
60	1,000	21.68%
60	1,500	30.22%
60	2,000	41.16%
60	2,500	44%
60	3,000	51%
60	3,500	62.73%
60	4,000	65.29%
60	4,500	70.04%
60	5,000	74.19%
60	5,500	76.09%
60	6,000	78.35%
60	6,500	80.02%
60	7,000	87.00%
60	7,500	89%
60	8,000	90.28%
60	8,500	91.97%
60	9,000	93%
60	9,500	93.30%
60	10,000	94%

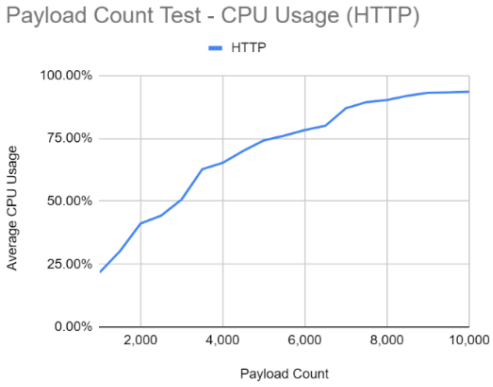

Figure 5. HTTP Payload Count Testing

Table 12 shows that HTTP can handle up to 10,000 payloads before reaching almost maximum CPU usage. The test results are also shown graphically in Figure 5. The same test was also carried out for the MQTT protocol. The result of comparing CPU usage of the MQTT protocol with increasing payload count can be seen in Table 13.

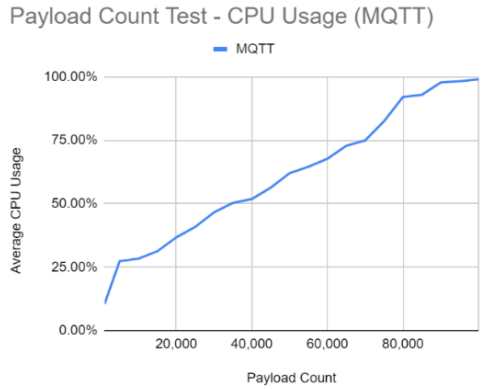

Figure 6. MQTT Payload Count Testing

Table 13 shows that the MQTT protocol can handle up to 100,000 payloads before reaching almost maximum CPU usage. The test results are also shown graphically in Figure 6.

Table 13. MQTT Payload Count Testing

Client	Payload	Average CPU Usage
60	1,000	10.46%
60	5,000	27%
60	10,000	28.35%
60	15,000	31.27%
60	20,000	36.74%
60	25,000	40.85%
60	30,000	46.60%
60	35,000	50%
60	40,000	51.83%
60	45,000	56.33%
60	50,000	62.04%
60	55,000	64.59%
60	60,000	68%
60	65,000	72.83%
60	70,000	75%
60	75,000	82%
60	80,000	92%
60	85,000	92.91%
60	90,000	98%
60	95,000	98%
60	100,000	99%

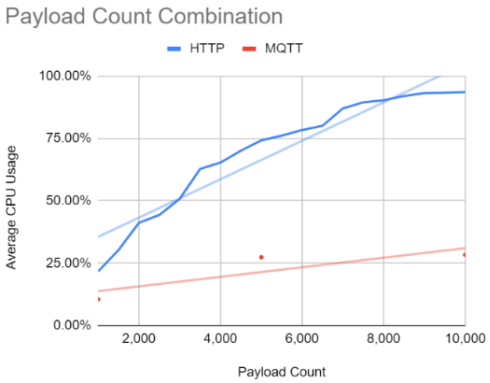

Figure 7. Combination Payload Count Testing

The combination of HTTP and MQTT payload count graphs is shown graphically in Figure 7. Trendline is used due to the imbalanced data point of MQTT. Figure 7 shows that for 10,000 payloads, MQTT only needs 28.35% CPU usage. Meanwhile, HTTP requires 94% CPU usage. This test shows that MQTT is more resource efficient than HTTP in terms of payload count.

3.3 Key Determining Factors

The results to determine which increase in client or payload has the greatest impact on CPU usage can be seen in Table 14.

Table 14. HTTP Key Determining Factors

Payload	Client		
	20	40	60
1,000	48.40%	79.90%	83.40%
5,000	64.90%	71.70%	92.70%
10,000	53.10%	72.70%	95.30%
15,000	50.60%	76.00%	94.70%
20,000	50.30%	77.80%	94.70%
25,000	46.40%	76.30%	95.30%
30,000	48.40%	77.80%	100%

Table 14 shows that each client count maintains a CPU usage range. For example, 40 clients maintain a range of 75% CPU usage.

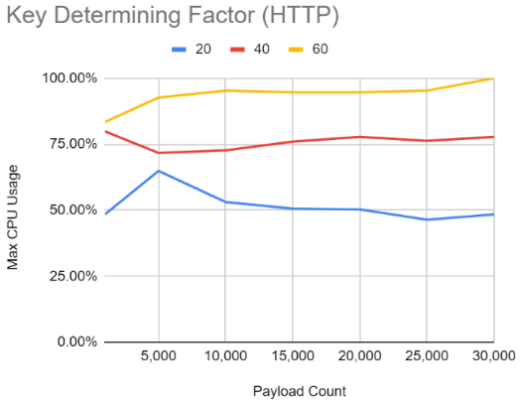

Figure 8. HTTP Key Determining Factors

This test shows that HTTP has a CPU usage limit based on the client involved. The test results are also shown graphically in Figure 8. The same test was also carried out for the MQTT protocol, which can be seen in Table 15.

Table 15. Key Determining Factors for MQTT

Payload	Client		
	20	40	60
1,000	4.60%	5.40%	7.90%
5,000	11.30%	12.70%	14%
10,000	19.50%	20.80%	22.70%
15,000	27.30%	30.70%	31.80%
20,000	37.10%	39.70%	41.30%
25,000	44.70%	46.70%	48%
30,000	57.30%	57.60%	58%

Table 15 shows that each client has almost the same CPU usage as the payload increases. For example, a 30,000 payload maintains a range of 58% CPU usage for every client count. This test shows that the increase in payload in MQTT is more impactful on CPU usage. The test results are also shown graphically in Figure 9.

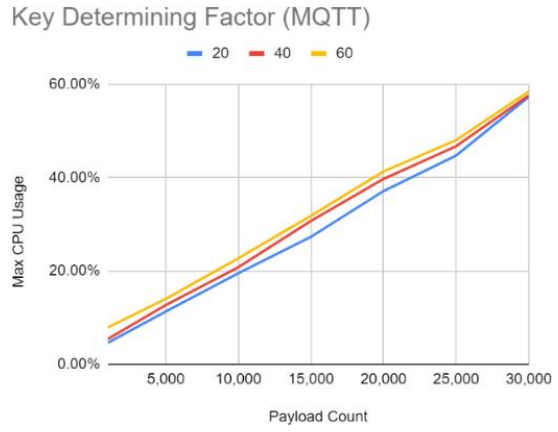

Figure 9. Key Determining Factors for MQTT

The results of both tests are consistent with previous studies. There is a study that shows that HTTP has a higher CPU usage compared to MQTT [20].

3.4 Total Transfer Time

The results of comparing the total transfer time of both protocols with increasing payload count can be seen in Table 16.

Table 16. Total Transfer Time

Payload	Total Transfer Time	
	MQTT	HTTP
1000	0.565	3.315
10000	0.839	20.335
20000	1.092	37.338
30000	1.371	56.343
40000	1.596	74.336
50000	1.918	93.369
60000	2.157	109.352
70000	2.443	127.347
80000	2.77	143.35
90000	3.038	161.349
100000	3.24	177.344

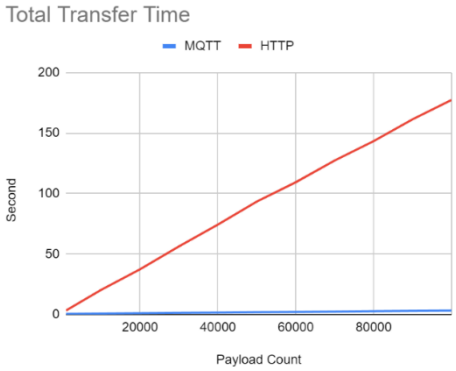

Figure 10 Total Transfer Time

Table 16 provides an overview of the fact that the MQTT protocol is faster to receive the entire payload compared to HTTP. The entire payload is received using the MQTT protocol in only 3.24 seconds. Meanwhile, HTTP takes up to 177.344 seconds. The test results are also shown graphically in Figure 10. Figure 10 shows a line of HTTP that has a higher gradient than MQTT. It shows that HTTP has a longer transfer time than MQTT.

3.5 Acceptance Rate

The results of comparing the acceptance rate of both protocols with increasing payload count can be seen in Table 17.

Table 17 shows that HTTP can receive all 30,000 payloads without loss. On the contrary, MQTT experiences payload losses of more than 50% when it receives 5,000 payloads. The test results are also shown graphically in Figure 11. This test shows that MQTT has poorer reliability than HTTP, so MQTT will be tested to identify its saturation point.

Table 17. Acceptance Rate

Payload	Acceptance Rate	
	HTTP	MQTT
1,000	100.00%	100.00%
5,000	100.00%	8.51%
10,000	100.00%	14.19%
15,000	100.00%	3.63%
20,000	100.00%	6.80%

25,000	100.00%	2.29%
30,000	100.00%	3.46%

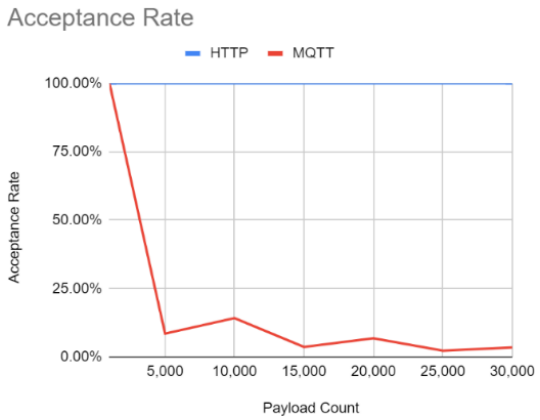

Figure 11. Acceptance Rate

3.5 MQTT Saturation Point

As MQTT is unreliable on the basis of the previous test, there is a chance that packet loss occurs below 5,000 payloads. This test is done to find the exact point that causes packet loss for both client and payload parameters. The results of the comparison of the acceptance rate of the MQTT protocol with increasing payload count can be seen in Table 18.

Table 18. MQTT Saturation Point by Payload

Payload	Payload Acceptance		
	Expected	Received	Percentage
125	180	180	100.00%
250	300	300	100.00%
375	420	420	100.00%
500	540	540	100.00%
625	660	660	100.00%
750	780	780	100.00%
875	900	900	100.00%
1000	1020	1020	100.00%
1125	1140	1140	100.00%
1250	1260	1260	100.00%
1375	1380	1380	100.00%
1500	1500	1500	100.00%
1625	1680	1680	100.00%
1750	1800	120	6.67%

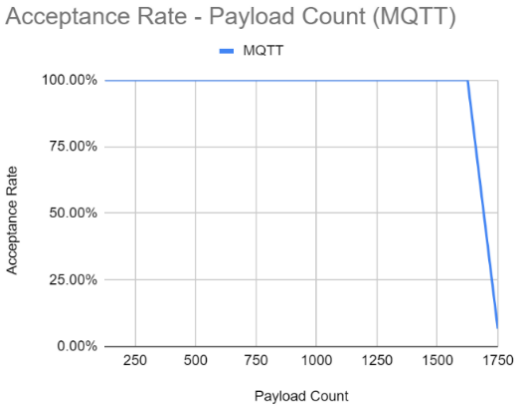

Figure 12. MQTT Saturation Point by Payload

Table 18 shows that the MQTT protocol can receive all payloads up to 1,680. The more payloads are added, the more payload losses. The test result is also shown graphically in Figure 12.

Reverse testing is conducted after the saturation point is established at 1,750 payloads. The results of the comparison of the acceptance rate of the MQTT protocol with the increase in the client count can be seen in Table 19.

Table 19. MQTT Saturation Point by Client

Payload	Payload Acceptance		
	Expected	Received	Percentage
5	1750	1750	100.00%
10	1750	1750	100.00%
15	1755	1755	100.00%
20	1760	1760	100.00%
25	1750	665	38.00%
30	1770	324	18.31%

Table 19 provides an overview of the MQTT protocol that successfully received all payloads when transmitted from 20 clients. As more clients are added, packet loss occurs. The test result is also shown graphically in Figure 13.

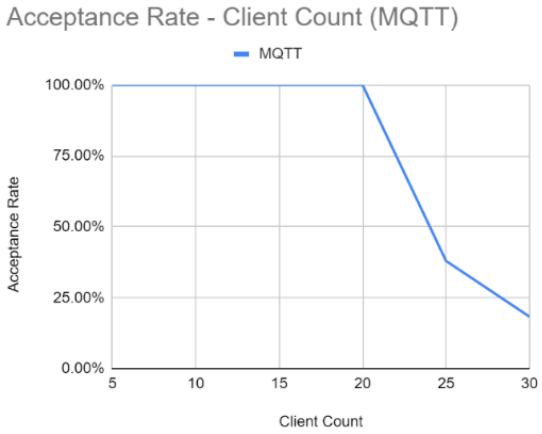

Figure 13. MQTT Saturation Point by Client

4. Conclusions

This study shows that MQTT is more resource-efficient than HTTP. 50 clients on MQTT only need 22.63% CPU usage than HTTP which needs 87% CPU usage. It is also shown that 10,000 payloads on MQTT need 28.35% CPU usage compared to HTTP which needs 94% CPU usage. Another test shows that HTTP has a CPU usage limit based on the client involved. As more payload increases, it does not have a huge impact on CPU usage. This study also shows that MQTT has a faster transfer time than HTTP. It is shown that MQTT only needs 3.24 seconds compared to HTTP which needs 177.344 seconds. However, the faster transfer time on MQTT is inversely proportional to the MQTT acceptance rate, which is worse than HTTP. HTTP can receive all 30,000 payloads without any losses. Meanwhile, MQTT can only receive 1680 payloads before experiencing payload losses. More improvement can be made in this study by comparing the acceptance rate of MQTT and HTTP on larger virtual machines.

References

[1] J. E. Son, H. J. Kim, and T. I. Ahn, "Hydroponic systems," in *Plant Factory*, Elsevier, 2020, pp. 273–283. doi: 10.1016/B978-0-12-816691-8.00020-0.

[2] A. Malik and R. Hartono, "Sistem Otomatis Pembuatan Nutrisi Ideal untuk Tanaman Pakcoy Menggunakan kendali Logika Fuzzy," *Telekontran : Jurnal Ilmiah Telekomunikasi, Kendali dan Elektronika Terapan*, vol. 9, no. 2, pp. 154–164, Oct. 2021, doi: 10.34010/telekontran.v9i2.5624.

[3] M. Bender, E. Kirdan, M.-O. Pahl, and G. Carle, "Open-Source MQTT Evaluation," in *2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*, IEEE, Jan. 2021, pp. 1–4. doi: 10.1109/CCNC49032.2021.9369499.

[4] B. Mishra and A. Kertesz, "The Use of MQTT in M2M and IoT Systems: A Survey," *IEEE Access*, vol. 8, pp. 201071–201086, 2020, doi: 10.1109/ACCESS.2020.3035849.

[5] N. Nikolov, "Research of MQTT, CoAP, HTTP and XMPP IoT Communication protocols for Embedded Systems," in *2020 XXIX International Scientific Conference Electronics (ET)*, IEEE, Sep. 2020, pp. 1–4. doi: 10.1109/ET50336.2020.9238208.

[6] B. Wukkadada, K. Wankhede, R. Nambiar, and A. Nair, "Comparison with HTTP and MQTT In the Internet of Things (IoT)," in *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*, IEEE, Jul. 2018, pp. 249–253. doi: 10.1109/ICIRCA.2018.8597401.

[7] C. C. Goh, E. Kanagaraj, L. M. Kamarudin, A. Zakaria, H. Nishizaki, and X. Mao, "IV-AQMS: HTTP and MQTT Protocol from a Realistic Testbed," in *2019 IEEE International Conference on Sensors and Nanotechnology*, IEEE, Jul. 2019, pp. 1–4. doi: 10.1109/SENSORSNANO44414.2019.8940094.

[8] A. A. Fadhel and H. M. Hasan, "Reducing Delay and Packets Loss in IoT-Cloud Based ECG Monitoring by Gaussian Modeling," *International Journal of Online and Biomedical Engineering (iJOE)*, vol. 19, no. 06, pp. 97–113, May 2023, doi: 10.3991/ijoe.v19i06.38581.

[9] Y. V. R. Darujati, L. K. P. Saputra, Y. Lukito, and W. A. Guspara, "Evaluation of HTTP and MQTT Protocols in the Design of Telemetry Data for Embedded Systems on Smart Wheelchairs," in *2023 IEEE International Conference on Data and Software Engineering (ICoDSE)*, IEEE, Sep. 2023, pp. 102–107. doi: 10.1109/ICoDSE59534.2023.10291260.

[10] B. Chen, F. Slyne, and M. Ruffini, "Energy Efficient SDN and SDR Joint Adaptation of CPU Utilization Based on Experimental Data Analytics," Feb. 2023.

[11] R. Bankston and J. Guo, "Performance of Container Network Technologies in Cloud Environments," in *2018 IEEE International Conference on Electro/Information Technology (EIT)*, IEEE, May 2018, pp. 0277–0283. doi: 10.1109/EIT.2018.8500285.

[12] M. W. Asyhari, R. Sigit, and S. Sukaridhoto, "Vending Machine Monitoring System Integrated with Webserver," in *2019 International Electronics Symposium (IES)*, IEEE, Sep. 2019, pp. 556–559. doi: 10.1109/ELECSYM.2019.8901588.

[13] R. Bryce, T. Shaw, and G. Srivastava, "MQTT-G: A Publish/Subscribe Protocol with Geolocation," in *2018 41st International Conference on Telecommunications and Signal Processing (TSP)*, IEEE, Jul. 2018, pp. 1–4. doi: 10.1109/TSP.2018.8441479.

[14] J. Toldinas, B. Lozinskis, E. Baranauskas, and A. Dobrovolskis, "MQTT Quality of Service versus Energy Consumption," in *2019 23rd International Conference Electronics*, IEEE, Jun. 2019, pp. 1–4. doi: 10.1109/ELECTRONICS.2019.8765692.

[15] S. S. N. Challapalli, P. Kaushik, S. Suman, B. D. Shivahare, V. Bibhu, and A. D. Gupta, "Web Development and performance comparison of Web Development Technologies in Node.js and Python," in *2021 International Conference on Technological Advancements and Innovations (ICTAI)*, IEEE, Nov. 2021, pp. 303–307. doi: 10.1109/ICTAI53825.2021.9673464.

[16] B. Mishra, "Performance Evaluation of MQTT Broker Servers," 2018, pp. 599–609. doi: 10.1007/978-3-319-95171-3_47.

[17] Fifin Ayu Mufarroha, Ahmad Farisul Haq, Arifatul Maghfiroh, Devie Rosa Anamisa, Ahmad Afif Supianto, and Achmad Jauhari, "Quality Assurance of Academic Websites using Performance Testing Tools," *Technium: Romanian Journal of Applied Sciences and Technology*, vol. 16, pp. 226–233, Oct. 2023, doi: 10.47577/technium.v16i.9985.

[18] R. A. Atmoko, R. Riantini, and M. K. Hasin, "IoT real time data acquisition using MQTT protocol," *J Phys Conf Ser*, vol. 853, p. 012003, May 2017, doi: 10.1088/1742-6596/853/1/012003.

[19] D. Guha Roy, B. Mahato, D. De, and R. Buyya, "Application-aware end-to-end delay and message loss estimation in Internet of Things (IoT) — MQTT-SN protocols," *Future Generation Computer Systems*, vol. 89, pp. 300–316, Dec. 2018, doi: 10.1016/j.future.2018.06.040.

[20] C. B. Gemirter, C. Senturca, and S. Baydere, "A Comparative Evaluation of AMQP, MQTT and HTTP Protocols Using Real-Time Public Smart City Data," in *2021 6th International Conference on Computer Science and Engineering (UBMK)*, IEEE, Sep. 2021, pp. 542–547. doi: 10.1109/UBMK52708.2021.9559032.

