International Journal of Engineering Trends and Technology
ISSN: 2231–5381 / https://doi.org/10.14445/22315381/IJETT-V71I10P209

Volume 71 Issue 10, 94-104, October 2023

Original Article

Impact and Analysis of Environment Temperature on MQTT Performance using Raspberry PI 3

Bouchra Allaoui[1], Ahmed Mouhsen[2], Mohamed Lamhamdi[1], Rachid Dakir[3]

[1]*Laboratory of Radiation-Matter and Instrumentation, Hassan First University of Settat, Faculty of Sciences and Technology, Settat, Morocco.*
[2]*Laboratory of Engineering, Industrial Management and Innovation, Hassan First University of Settat, Faculty of Sciences and Technology, Settat, Morocco.*
[3]*LabSIV Laboratory, Ibn Zohr University of Ouarzazate, Polydisciplinary Faculty, Ouarzazate, Morocco.*

[1]*Corresponding Author : b.allaoui@uhp.ac.ma*

Received: 17 June 2023 Revised: 02 August 2023 Accepted: 13 September 2023 Published: 03 October 2023

Abstract *- This study aims to present and discuss the impact of increasing the environment temperature on CPU usage, memory usage, processor temperature, and response time when the number of clients increases. In this work, the MQTT protocol with different QoS levels is used. A testbed system comprises a laptop, a Raspberry Pi 3 Model B, an air conditioner, a DS18B20 thermometer, and a WiFi access point. The two first hardware functions as multiple MQTT clients and an MQTT broker, respectively. Note that the Raspberry Pi is chosen because the community widely accepts it, whereas the purpose of collecting the MQTT clients in a single machine is to increase their number up to 201 automatically. For each number of clients, metrics such as the environment temperature, CPU usage, memory usage, and processor temperature continue to be measured until the number of clients reaches the 66400th PUBLISH packet, while the response time metric is determined by calculating the duration between the CONNECT and the 66400th PUBLISH packets. In this experiment, the environment temperature is varied using the air conditioner for each QoS level. The results indicate that the high environment temperature can either increase the CPU usage and decrease the response time or keep constant the CPU usage and increase the response time.*

Keywords *- CPU Usage, Environment temperature, Internet of Things, MQTT protocol, Response time.*

1. Introduction

The Internet of Things (IoT) describes a network of physical devices that contain several embedded sensing, processing and communication technologies for the purpose of collecting and communicating sensory data over the internet [1]. This concept was used for the first time by Kevin Ashton in 1999 [2]. For communication at a lower layer, several radio technologies such as WMAN, RFID, WPAN, and WLAN are used by IoT Networks. Irrespective of the radio technology used, to create an M2M network, IoT devices must make their data accessible through the internet [3,4]. They are divided into rich resources ones, such as smartphones and servers and limited resources, such as sensor nodes and actuators [5]. Due to the low cost of resource-constrained IoT devices, they have been incorporated into all kinds of technology solutions [4,6]. They function with small memory, limited battery power and low computing power [4,5]. As a result, the performance of M2M communication largely depends on messaging protocols specially designed for M2M communication in Internet of Things applications [3,4]. Currently, several messaging protocols exist, such as CoAP, XMPP, AMQP, and MQTT [1]. Choosing a standardized and efficacious messaging protocol, which is a constant quandary for the IoT industry, is a really demanding challenge as it all depends on the IoT system's features and its messaging requirements [3]; therefore, there is a large number of papers in this field that analyze their performance with different metrics and in different situations [3,6-12].

Nevertheless, they do not consider the impact of the increased number of clients and the environment temperature on metrics, which is the purpose of this article. The main goal is to highlight the environment temperature's significance and determine its value that keeps the IoT system stable in terms of memory usage, CPU usage, processor temperature, and response time. In the present work, emphasis is placed on the MQTT protocol since it is widely used in varying applications [13-15].

The rest of this article is divided into five sections: section II summarizes several related works, followed by a description of the testbed architecture and its realization in section III. Section IV introduces the method used to measure the metrics automatically. Section V presents, examines and analyses the results. Finally, the conclusion of this paper is in section VI.

2. Literature Review

Many studies have been published in the literature that analyze the performance of various network protocols for IoT in different situations and with various metrics. In this section, some related works published in recent years are summarized.

N. Naik aims to guide ordinary users to choose among HTTP, AMQP, COAP, and MQTT, the appropriate one for their IoT-specific system according to various balances between features (e.g. resource requirement, bandwidth, latency, power consumption, etc.) based on static components and some empirical evidence from the literature. Note that the comparison results are determined by considering the network is lossless [3]. S. Imane et al. clear up the importance of IoT in healthcare to provide high-quality services. They began with a summary of some smart healthcare systems by identifying their weaknesses and strengths before giving some criteria for selecting the most appropriate IoT protocol between COAP and MQTT for a smart healthcare IoT application. They concluded that COAP is useful in local networks with less bandwidth and higher-speed transmission, while MQTT is useful in limited bandwidth. They also gave in detail some threats and attacks on smart healthcare systems [7]. M. Zorkany et al. presented a practical comparison of MQTT and COAP on an e-health system according to the number of both bytes utilized in messages and messages lost and the delay. To do so, they evaluated three metrics, namely, the IoT Delay message, the IoT Average Byte, and the IoT Success Message according to the packet loss percentages. Metrics were determined by using the WANEM and WIRESHARK programs. The results showed that MQTT outperforms COAP [8]. Y. Guamán et al. conducted a comparative study between MQTT and COAP regarding jitter and delay to determine which is suitable for controlling objects at home. The experiment system consisted of a Raspberry Pi 3, an Optocoupler circuit and a Lamp. The results concluded that in a domestic environment, the MQTT protocol performed better than the COAP protocol, not only in terms of jitter and delay but also in terms of its implementation [9].

Other performance evaluation work focuses on MQTT. For example, D. Borsatti et al. focused on analyzing the MQTT protocol at different QoS levels regarding latency. Their goals were to evaluate the impact of the end-to-end network delay and the publisher, subscriber, and broker placement on the publisher-to-subscriber data delivery latency. To achieve that, a testbed was set up. The testbed was created using two Ubuntu Virtual Machines, VM1 and VM2; VM1 functions as both a subscriber and a publisher machine, while VM2 functions as a broker machine. The results showed that the latency is up to 7 times the average network delay at the QoS2, and the placements of publisher, broker, and subscriber did not affect the latency [10]. E. Baranauskas et al. determined the energy consumption of all MQTT QoS levels with TLS. The experimental system consists of an ESP32, a Raspberry Pi 2, a client power supply, a digital multimeter and a WIFI. The ESP32 and Raspberry Pi 2 function as two MQTT clients and a broker, respectively. The result showed that QoS0 consumed 6.7% more energy than QoS1. Note that QoS 2 consumed 1.7% less energy than QoS0, while it consumed 5 % more energy than QoS1 [11]. T. Prantl et al. conducted an analysis of the impact of securing MQTT using TLS on energy efficiency, broker connection establishment times, and throughput in different network situations. They used an ESP8266 microcontroller, a windows 10 laptop, a client power supply, a power meter, and a WiFi for their experiment. The microcontroller and laptop function as an MQTT client and broker, respectively. The results showed that TLS negatively affects the performance only in deteriorated network situations [12]. T. N. Ford et al. focused on evaluating the performance of the Raspberry Pi 3 Model B, Zero W, and Zero 2 W in terms of messages' throughput and transmission time according to the number of packet sizes. They determined not only the impact of the WiFi bandwidth, payload size, both client and broker Hardware, QoS level, and a DoS attack on the transmission time but also the impact of the security on the throughput. These two metrics were determined using one subscriber and one publisher [6].

As can be seen from this literature survey, many of them evaluate the performance of MQTT using one or two MQTT clients at most, while the environment temperature is not considered. Note that few published works evaluate the performance of MQTT on Raspberry Pi devices, although they are a suitable solution for the IoT system [6]. Recently, the authors have evaluated Raspberry Pi's performance as an MQTT broker regarding CPU usage, memory usage, processor temperature, and response time when the number of clients and QoS levels increase. This work will determine how increasing the environment temperature impacts the previous results. Consequently, this article is the continuation of the previous work.

3. Testbed Architecture and Realisation

In the previous work, the authors developed an appropriate evaluation environment to carry out measurements and analyze the impact of the increase of both the clients and the QoS levels on the performance of Raspberry Pi as an MQTT broker in an IoT context. Briefly, the testbed components consist of an MQTT broker, MQTT clients, a WiFi Access point, a digital thermometer, and an air conditioner. MQTT clients are divided into 100 publishers (Pubs) and 101 subscribers (Subs). Each publisher sends values of Power (P), Temperature (T) and Humidity (H) while each subscriber receives them, as illustrated in Figure 1. Note that the Last Will and Testament, the Persistent Session, the Retained Message and the Keep Alive are used. This latter was fixed at 50 seconds for all clients, whereas the Retained messages are the initial value of P, T and H for each publisher and each MQTT client's status.

At the hardware level, the MQTT broker runs on the Raspberry Pi 3 Module B [16], while MQTT clients run on the ThinkPad T480 I7-8550U Lenovo laptop. The authors used the MF253V ZTE 4G Wireless Router to create a local network and the DS18B20 digital thermometer [17] to determine the environment temperature during the tests. At the software level, the used operating system is Raspberry Pi OS Lite since it has the necessary software without a desktop environment [18, 19], while the used MQTT broker is the Mosquitto broker because it is suitable for constrained-resources devices, as in this case [4].

The Python programming language and the library of PAHO-MQTT [20] and thread [21] are used to create MQTT clients' programs and let them run concurrently in the same script. Note that the current CPU usage and memory usage for the Mosquitto broker were provided by the command "top -p <pid>" [22], where <pid> is given by the command "systemctl status mosquitto.service", whereas the processor temperature and the environment temperature were provided by commands "cat / sys / class / thermal / thermal_zone0 / temp " [23, 24] and "cat /sys/bus/w1/devices/28-800000264462/w1_slave" [25], respectively. Moreover, the command "renice -12 <pid>" was used to increase the priority of the mosquitto service execution [26]. In this paper, the same testbed will be used.

4. Methodology for Measurement Analysis

In the previous work, they developed a suitable technique for measuring the five metrics: both CPU usage and memory usage for Mosquitto Broker, processor temperature, environment temperature, and response time when the number of clients is (0Pub and 0Sub), (100Pubs and 0Sub), (100Pubs and 1Sub), (100Pubs and 11Subs), (100Pubs and 21Subs) and so on up to (100Pubs and 101Subs). Briefly, the technique goes as follows: after end measuring the first four metrics for zero clients, the sequential launch of MQTT clients begins automatically.

This latter starts first with 100 publishers, then a subscriber afterward, the first ten subscribers, and so on until the tenth ten subscribers. This serial launch is done after one minute of the end of measuring the first four metrics corresponding to them. Note that the programs of 100 publishers, 1 subscriber, and each of the 10 subscribers were written in separate scripts. Consequently, 100 publishers simultaneously send PUBLISH packets, and every 10 subscribers simultaneously receive PUBLISH packets.

The first five metrics are determined during 30 min without clients, whereas they start to be determined once clients are online and end after publishers or each subscriber reaches the 66400th PUBLISH packet. At the end of the test, 13 files are obtained that refer to the total active both subscribers and publishers and the QoS level. Each of them consists of CPU usage, memory usage, processor temperatures, and environment temperatures, as illustrated in Figure 3. Note that they were taken in each update of the Mosquito service's vital information. The cause of the absence sometimes of the environment temperature is that the DS18B20 updates every 1 or 2 seconds, and the command "top" updates every 3 seconds. On the other hand, the response time is the duration between the first sent CONNECT packet and the 66400th PUBLISH packet for each subscriber and between the 664th PUBLISH packet for each publisher. Note that the same initial conditions were taken for each repeated test.

The average and standard deviation of CPU usage, memory usage, environment temperature, and processor temperature for each obtained file and the average and standard deviation of the response time for each number of concurrently launched clients are determined. In this paper, this technique will be used.

5. Results and Discussion

The measurement results are presented for each QoS level in Figures 4 to 8 and more detail in Tables 1 to 3 after increasing the environment temperature using the air conditioner. Note that all the results were obtained without any clients disconnected during the test; therefore, these will allow us to make a correct comparison.

Based on the results of the memory usage for all QoS levels, as shown in Figure 4, it can be remarked that the required memory for Mosquitto broker increases from 0.5% to 0.6% once publishers are launched. It remains constant until the number of subscribers reaches 101 for QoS levels 0 and 1, while it increases to 0.69838% when the eighth ten subscribers are launched and remains constant at 0.7% for QoS level 2. Depending on the results of CPU usage for all QoS levels, as illustrated in Figure 5, it can be observed that CPU usage increases with the number of MQTT clients.

From 0 clients to 100 publishers, the CPU usage increases by 0.4%, 0.5% and 0.9% for QoS levels 0, 1 and 2, respectively, and it remains almost constant until the first ten subscribers are reached, the first ten subscribers, where it begins to increase obviously. For QoS levels 0 and 2, the CPU usage continues to increase until the subscribers' number reach as 101, whereas, for QoS level 1, it continues to increase until the subscribers' number reaches as 91 and then decreases slightly by 0.2565%.

According to Figure 6, it can be noticed that from 0 clients to 100 publishers, the processor temperature increases by 3.2°C, 2.2°C and 2.6°C for QoS levels 0, 1 and 2, respectively. After that, it remains approximately constant until the subscribers' number reaches 31 for QoS 0 and 1 for QoS 1 and 2, from which it begins to increase until the maximum number of clients is reached. In terms of test time, it lasted 09h13min33.0164300s, 09h15min17.4760520s and 09h33min35.7679680s for QoS levels 0, 1 and 2, respectively.

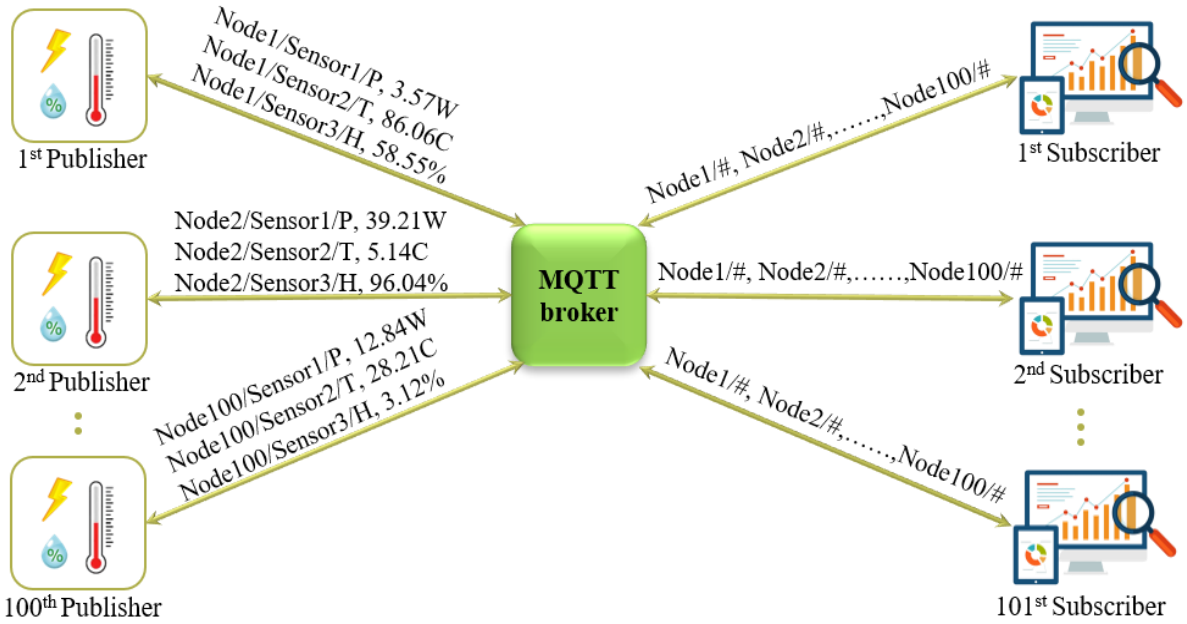

Fig. 1 Testbed architecture

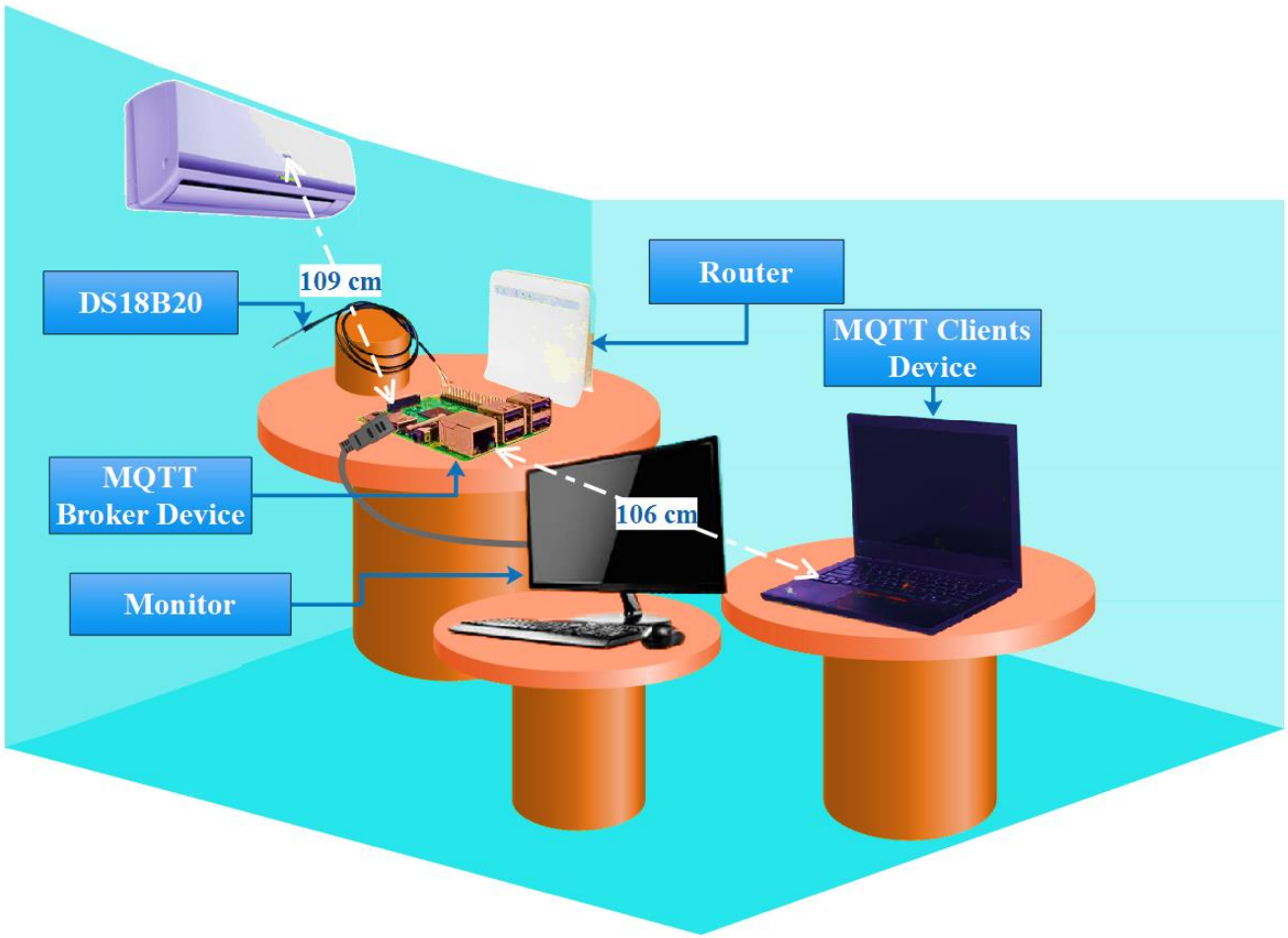

Fig. 2 Actual implementation

100PUB_QoS0_51SUB.txt - Notepad

File Edit Format View Help

Time	CPU %	MEM %	Temp	TempR
12:33:07	2.3	0.6	62.3	29.312
12:33:10	3.7	0.6	61.224	29.312
12:33:13	3.3	0.6	59.61	29.312
12:33:16	3.0	0.6	59.61	
12:33:19	3.3	0.6	59.072	
12:33:22	3.7	0.6	58.534	29.312
12:33:25	3.3	0.6	58.534	29.312
12:33:28	3.3	0.6	57.996	29.312
12:33:31	3.3	0.6	57.996	
12:33:34	4.0	0.6	57.458	29.312
12:33:37	3.0	0.6	56.92	29.312
12:33:40	4.0	0.6	56.92	29.312

Unix (LF) Ln 25, Col 69 100%

Fig. 3 One of the obtained files

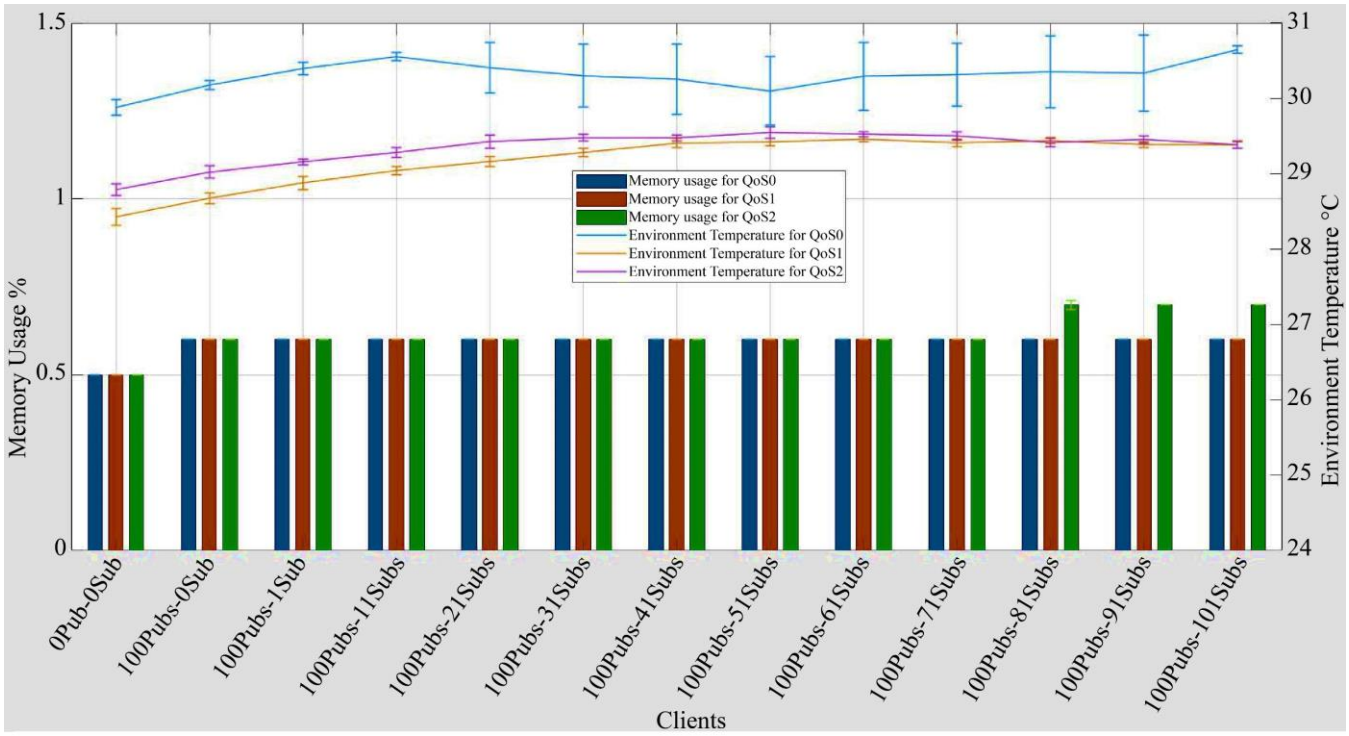

Fig. 4 Memory usage for various QoS levels

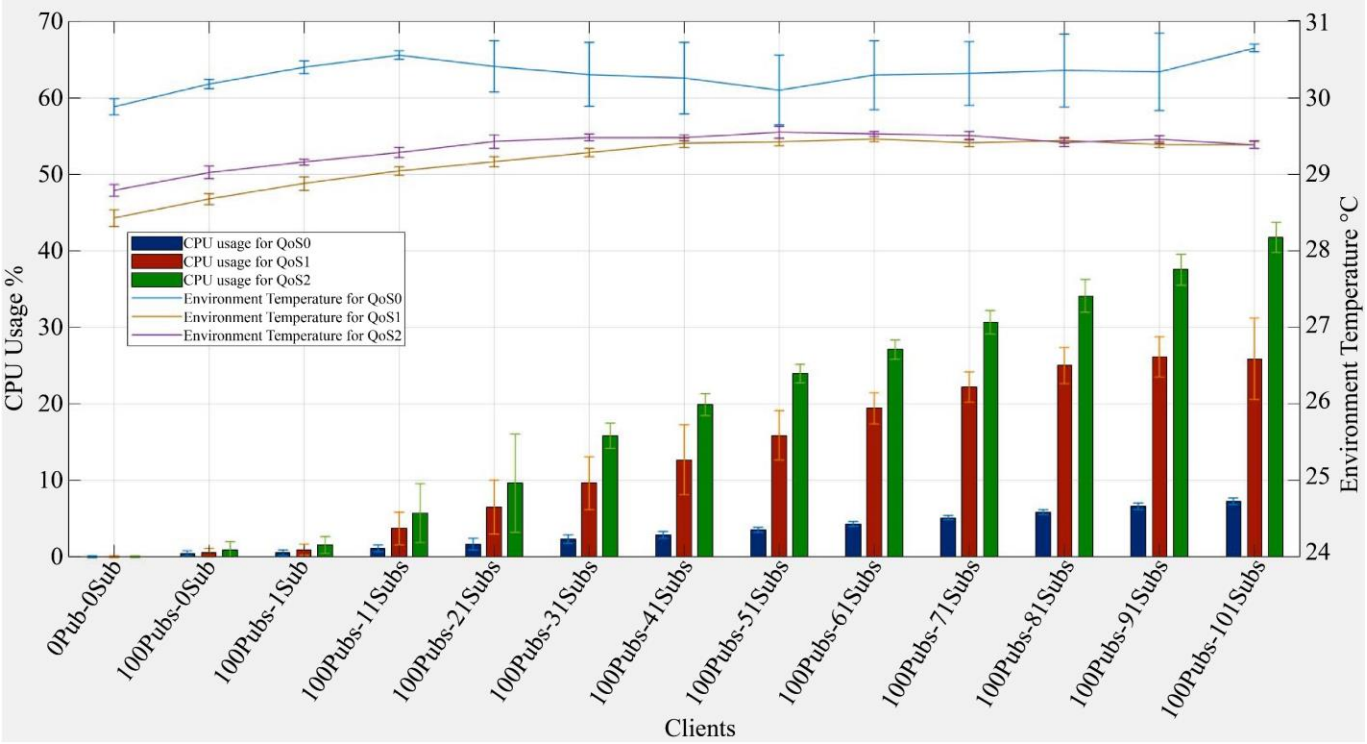

Fig. 5 CPU usage for various QoS levels

Table 1. Metrics results for QoS level 0

Metrics / Clients	Mem_{usage}(%)		CPU_{usage}(%)		$T_{processor}$(°C)		$T_{environment}$(°C)		$t_{response}$(mm:ss.sssssss)	
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
0Pub-0Sub	0.5	0	0.0418	0.1083	50.4775	1.1983	29.8794	0.1021	-	-
100Pubs-0Sub	0.6	0	0.4659	0.3362	53.6422	0.3923	30.1765	0.0579	44:58.1554704	14.0232814
100Pubs-1Sub	0.6	0	0.5560	0.3590	53.7292	0.6171	30.3961	0.0793	45:34.7375210	0
100Pubs-11Subs	0.6	0	1.1118	0.4340	53.9097	0.5426	30.5546	0.0507	45:36.2048793	0.1291951
100Pubs-21Subs	0.6	0	1.6814	0.7417	54.1913	0.7086	30.4069	0.3384	45:20.0541590	0.4384503
100Pubs-31Subs	0.6	0	2.3439	0.5653	54.5747	0.6668	30.3001	0.4159	45:12.7283447	0.1231222
100Pubs-41Subs	0.6	0	2.9441	0.4226	55.9746	0.4881	30.2555	0.4629	45:04.6052301	0.0277305
100Pubs-51Subs	0.6	0	3.6011	0.3474	55.8776	0.5679	30.0968	0.4560	45:03.3460545	0.0475846
100Pubs-61Subs	0.6	0	4.3156	0.3024	57.1380	0.6652	30.2960	0.4482	44:59.2386617	0.0085228
100Pubs-71Subs	0.6	0	5.1244	0.2789	57.8059	0.6184	30.3155	0.4140	45:00.2166870	0.0413740
100Pubs-81Subs	0.6	0	5.8889	0.3604	58.3778	0.6654	30.3543	0.4806	45:00.3861349	0.0715964
100Pubs-91Subs	0.6	0	6.6321	0.4272	58.3727	0.6659	30.3360	0.5094	45:01.0727710	0.0645858
100Pubs-100Subs	0.6	0	7.3080	0.4456	58.9101	0.5694	30.6477	0.0480	44:58.9132257	0.0755639

Table 2. Metrics results for QoS level 1

Metrics / Clients	Mem_{usage}(%)		CPU_{usage}(%)		$T_{processor}$(°C)		$T_{environment}$(°C)		$t_{response}$(mm:ss.sssssss)	
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
0Pub-0Sub	0.5	0	0.0417	0.1060	49.9547	0.6883	28.4281	0.1067	-	-
100Pubs-0Sub	0.6	0	0.5871	0.5169	52.2020	0.5402	28.6758	0.0683	44:47.8622464	8.1224992
100Pubs-1Sub	0.6	0	0.9496	0.7012	52.1118	0.7749	28.8803	0.0898	45:34.1839030	0
100Pubs-11Subs	0.6	0	3.7369	2.0945	53.6025	0.5156	29.0432	0.0554	45:32.7291222	0.0487289
100Pubs-21Subs	0.6	0	6.5557	3.4927	55.1336	0.8040	29.1633	0.0657	45:30.7725962	1.2347048
100Pubs-31Subs	0.6	0	9.6696	3.4441	56.0419	0.6683	29.2820	0.0578	45:25.8577509	0.3351441
100Pubs-41Subs	0.6	0	12.7247	4.5039	57.3925	1.2229	29.4064	0.0615	45:26.2970319	0.4554824
100Pubs-51Subs	0.6	0	15.9089	3.1906	59.1757	0.9813	29.4233	0.0490	45:15.8914025	0.0638623
100Pubs-61Subs	0.6	0	19.4295	2.0271	59.5795	0.7554	29.4599	0.0367	45:12.2849025	0.1404345
100Pubs-71Subs	0.6	0	22.2271	1.9944	60.2573	0.8651	29.4108	0.0474	45:10.5244150	0.1094722
100Pubs-81Subs	0.6	0	25.0337	2.3812	61.8616	0.5974	29.4395	0.0370	45:11.4668670	0.0960955
100Pubs-91Subs	0.6	0	26.1496	2.6274	60.7618	1.0320	29.3879	0.0417	45:17.8326053	0.1053612
100Pubs-100Subs	0.6	0	25.8931	5.3138	60.1713	1.0594	29.3855	0.0440	45:21.4915965	0.3210148

Table 3. Metrics results for QoS level 2

Metrics / Clients	Mem_{usage}(%)		CPU_{usage}(%)		$T_{processor}$(°C)		$T_{environment}$(°C)		$t_{response}$(mm:ss.sssssss)	
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
0P [a] **-0S** [b]	0.5	0	0.0410	0.1030	49.5404	1.0968	28.7890	0.0801	-	-
100P-0S	0.6	0	0.9225	1.1391	52.1868	0.5654	29.0208	0.0831	44:40.3623696	1.6210172
100P-1S	0.6	0	1.5634	1.0809	52.6350	0.6662	29.1599	0.0376	45:34.8774110	0
100P-11S	0.6	0	5.7685	3.8327	54.0888	0.8282	29.2826	0.0687	46:35.4629899	11.4967979
100P-21S	0.6	0	9.6668	6.4024	55.3071	0.9952	29.4283	0.0895	48:07.5951529	22.6834192
100P-31S	0.6	0	15.9054	1.6473	58.0239	0.4192	29.4772	0.0425	45:33.7561482	0.0874794
100P-41S	0.6	0	19.9036	1.4143	59.2242	0.7673	29.4777	0.0380	45:38.1831383	0.0283345
100P-51S	0.6	0	23.9557	1.2171	60.3773	0.7025	29.5480	0.0755	45:31.2897997	0.1795318
100P-61S	0.6	0	27.1226	1.2515	60.4371	0.5856	29.5268	0.0343	45:35.8941220	0.1296544
100P-71S	0.6	0	30.6574	1.5488	60.7268	0.6786	29.5038	0.0588	45:45.8201724	0.1517876
100P-81S	0.6984	0.0126	34.1196	2.1214	61.7510	0.6128	29.4154	0.0506	46:14.3851415	9.8990647
100P-91S	0.7	0	37.5317	2.0035	63.2206	0.4897	29.4558	0.0467	48:30.5348309	1:42.9210329
100P-100S	0.7	0	41.7216	1.9940	64.4157	0.5690	29.3851	0.0511	48:02.8420009	2:15.9251051

[a)]PUB; [b)]SUB

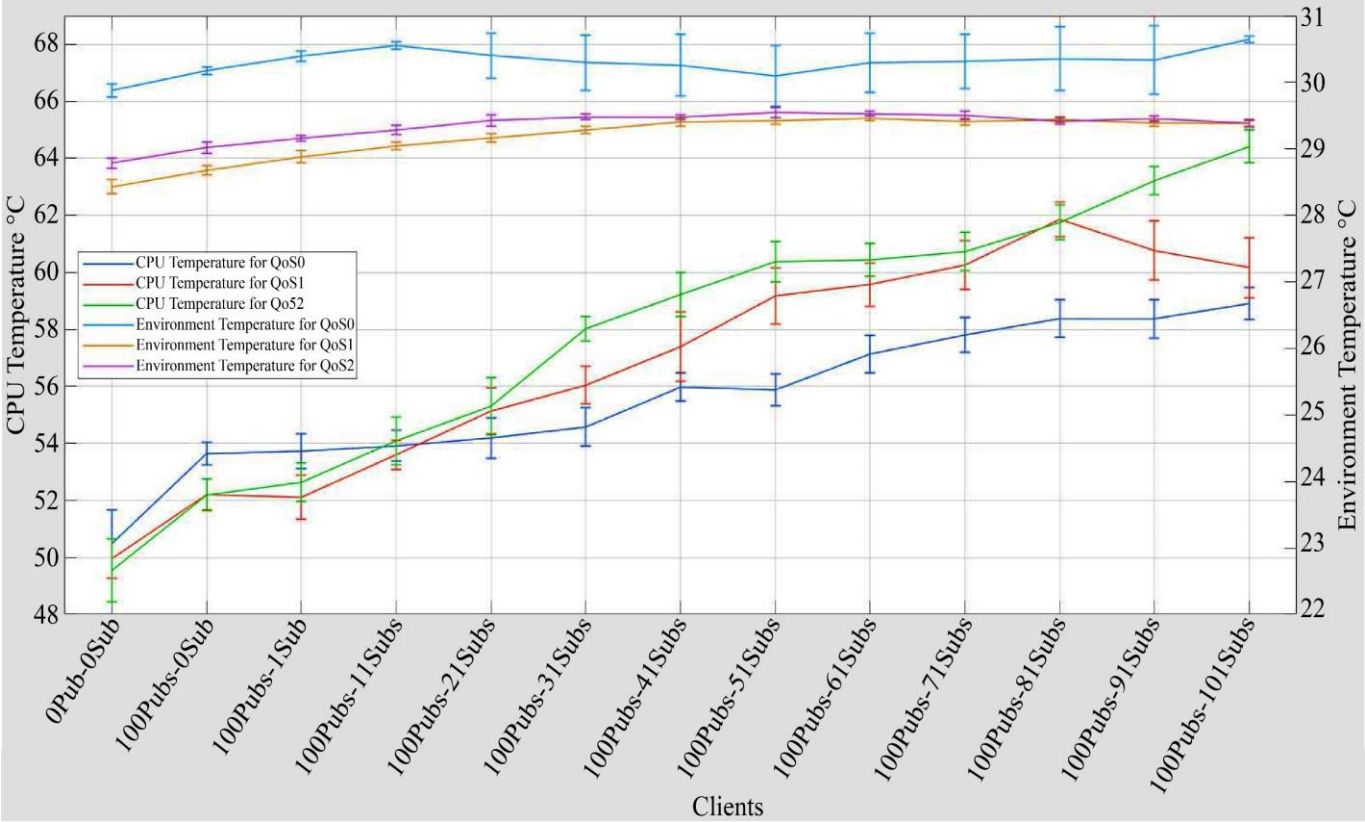

Fig. 6 Processor temperature for various QoS levels

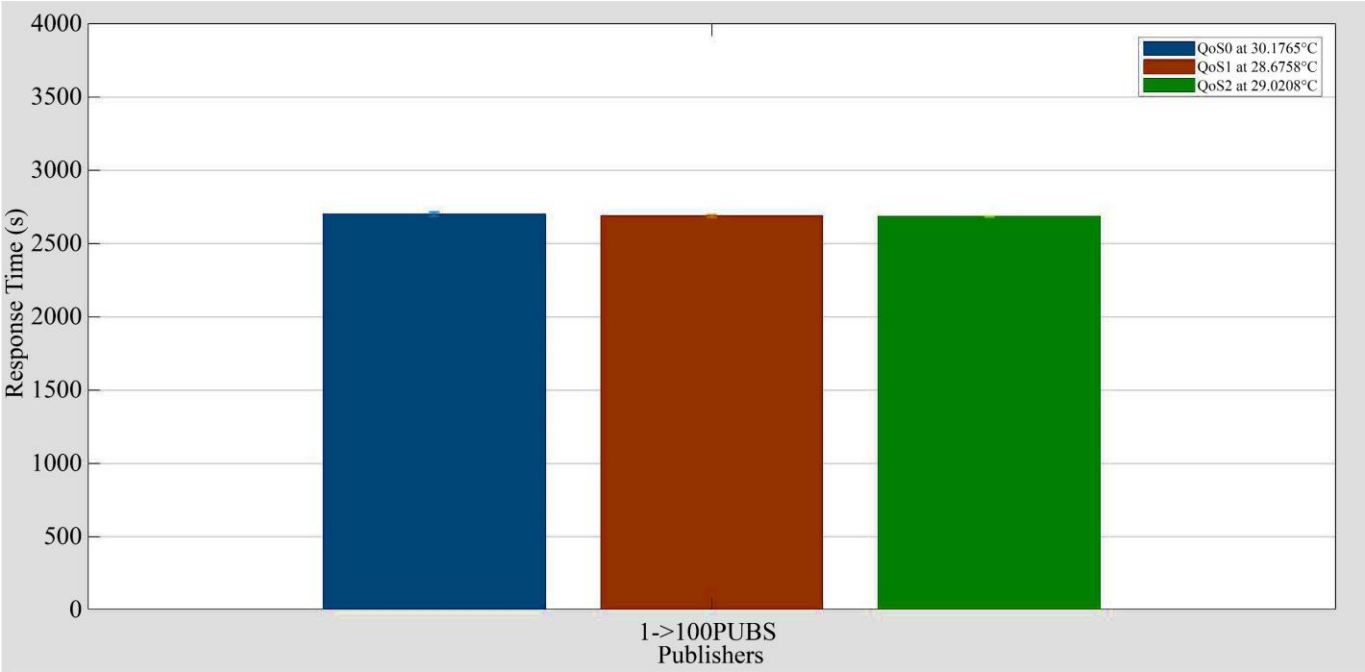

Fig. 7 Publishers' response time for various QoS levels

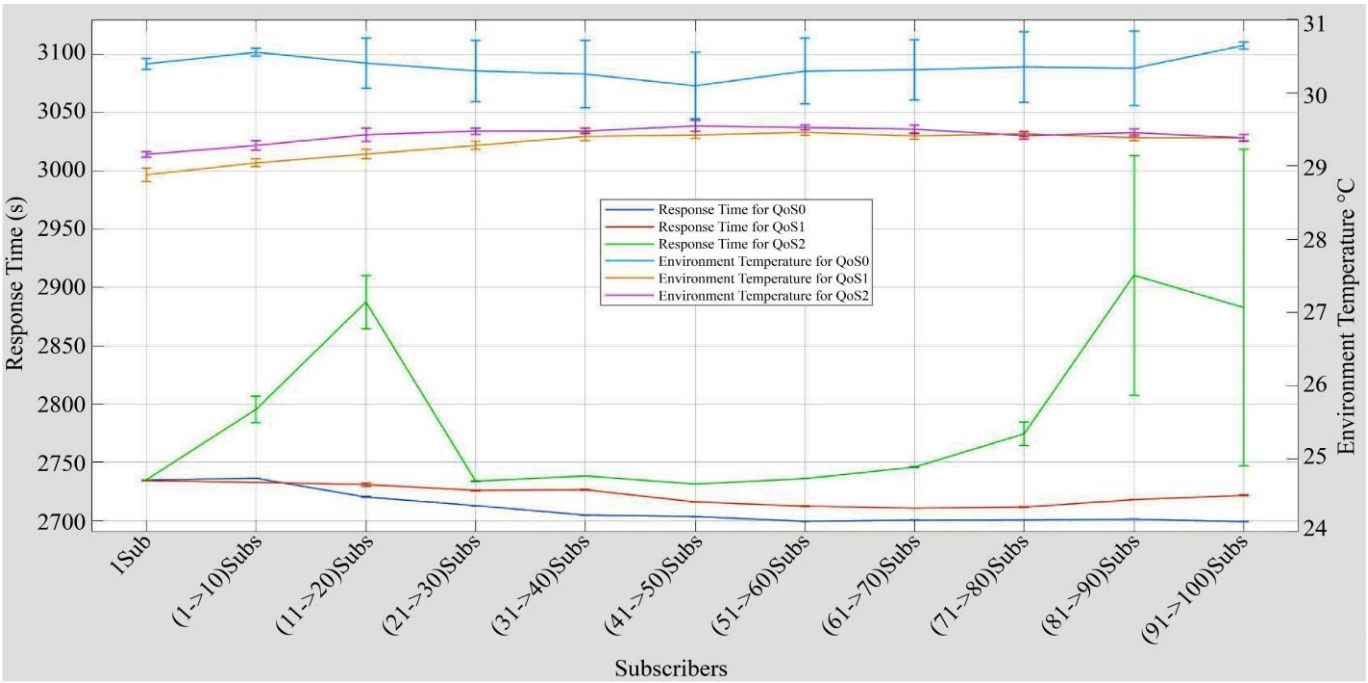

Fig. 8 Subscribers' response time for various QoS levels

Comparing these results with previous ones taken at the low environment temperature, it can be noticed that both the CPU usage and the processor temperature increase with the number of clients regardless of the used environment temperature. The graph shape not only the processor temperature and the CPU usage for all QoS levels but also the subscribers' response time for QoS0 measured at the high and low environment temperature are the same. To compare the actual results with the previous ones, the average and standard deviation of metrics were determined for each QoS level, as illustrated in Table 4. For QoS0, the memory usage and CPU usage remain constant, but the processor temperature increases by 4.8°C and the publishers' response time increases by 15.6s, while the subscribers' response time average increases by 4.2s. At the level of QoS1, the difference between the memory usage is from the seventh ten subscribers. It decreases slightly by 0.03%, whereas the average of the processor temperature and CPU usage increase by 4.5°C and 1.2%, respectively. Publishers' response time increases by 18.4s, while subscribers' response time average decreases by 22.6s. At this level, it can be noticed that the high environment temperature can either increase the CPU usage and decrease the response time or keep constant the CPU usage and increase the response time. The test at 27°C for QoS2 was repeated to verify these results. Between 24°C and 27°C, the average processor temperature and the CPU usage increase by 4.2°C and 0.93%, respectively, while publishers' response time and subscribers' response time average decrease by 21.4s and 12.8s, respectively. Between 27°C and 29°C, the processor temperature average increases by 0.9°C. The publishers' response time increases by 4.2s, whereas the subscribers' response time average increases by 1.7s. The CPU usage at 29°C remains inferior to its counterpart at 27°C, but the reverse occurs when the eighth ten subscribers are surpassed. For QoS2, when repeated test at 30°C, it was noticed that the MQTT clients disconnected.

Table 4. Comparison of metrics in different environment temperature

Metrics / QoS	$T_{environment}$(°C)		Mem_{usage}(%)		CPU_{usage}(%)		$T_{processor}$(°C)		$t_{response}$(mm:ss.sssssss)	
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
0	24.8511	0.0267	0.5923	0.0266	3.2725	2.4016	50.8071	2.1918	45:05.9687279	9.5619550
	30.3089	0.1851	0.5923	0.0266	3.2319	2.3678	55.6139	2.3741	45:10.1366972	13.4397700
1	24.6528	0.1170	0.6227	0.0571	11.8149	8.7054	52.2706	3.2352	45:34.2356689	11.5951825
	29.1835	0.3203	0.5923	0.0266	12.9929	9.7057	56.7882	3.7407	45:21.7574721	8.3403694
2	24.2420	0.0511	0.6274	0.0578	18.3143	13.4919	52.7291	4.0043	46:39.4630146	1:29.7536052
	27.7059	0.1957	0.6202	0.0545	19.2421	13.6720	56.8957	4.8929	46:26.5872859	1:15.1413205
	29.3439	0.2171	0.6153	0.0531	19.1446	14.0803	57.8411	4.4667	46:28.2400825	1:07.5047088

Table 5. Comparison of related works

Articles	Protocols	Metrics	Number of Clients	Number of packets	QoS			Environment temperature
					0	1	2	
[12]	MQTT	Energy efficiency, broker connection establishment times, and throughput	1	720	✓	✓	✓	x
[10]	MQTT	Publisher-to-subscriber data delivery latency	2	10000	✓	✓	✓	x
[27]	MQTT	Rate of MQTT received messages, CPU Usage and Processor Temperature	60	Maximum MTR	✓	✓	✓	x
This work	MQTT	CPU usage, memory usage, processor temperature, and Response time	201	66400	✓	✓	✓	✓

It was also noticed that there is an inverse relationship between CPU usage and response time, meaning that even metrics such as throughput and latency can be affected by changing environment temperature. As a result, the environment temperature is an important factor to consider when measuring metrics. Nevertheless, it was noted that no articles incorporate environment temperature into their measurements, as illustrated in Table 5. The ideal environment temperature, in this case, is 25°C because the corresponding CPU temperature is lower.

6. Conclusion

MQTT publish/subscribe protocol is one of the foundations of IoT communication with IoT devices. This paper evaluated and analyzed Raspberry Pi's performance as an MQTT broker regarding memory usage, CPU usage, processor temperature, and response time in each QoS level when the environment temperature increases. To achieve that, the temperature was increased to 30°C and 29°C for QoS0 and QoS1, respectively, while the temperature was increased to 27°C, 29°C and 30°C for QoS2. The actual measurement obtained is compared with the measurement previously obtained at 24°C for all QoS levels.

The study results showed that the high environment temperature can either increase the CPU usage and decrease the response time or remain constant the CPU usage and increase the response time. Also, it increases the processor temperature. For QoS level 0, the response time average increases by 4.2s, while for QoS level 1, at 29°C, the CPU usage average increases by 1.2%, and the response time average decreases by 22.6s. For QoS2, comparing the results at 27°C to those at 24°C, the CPU usage average increases by 0.93% and the response time average decreases by 12.8s, while, comparing the results at 29°C to those 27°C, the response time average increases by 1.7s and also, at 30°C, the MQTT clients disconnect.

Regardless of the fact that the difference in response time is a few seconds, these seconds in a smart healthcare system, for example, may lead to the loss of life of patients as well as when medical sensors disconnect. This study proves that the environment temperature is an important factor that must be considered for measuring metrics and that 25°C is the ideal environment temperature for maintaining system stability.

References

[1] Shakila Zaman et al., "Thinking Out of the Blocks: Holochain for Distributed Security in IoT Healthcare," *IEEE Access*, vol. 10, pp. 37064-37081, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[2] Kevin Ashton, "That 'Internet of Things' Things," *RFID Journal*, vol. 22, no. 7, pp. 97-114, 2009. [Google Scholar] [Publisher Link]

[3] Nitin Naik, "Choice of Effective Messaging Protocols for IoT Systems: MQTT, CoAP, AMQP and HTTP," *IEEE International Systems Engineering Symposium*, pp. 1-7, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[4] Biswajeeban Mishra, Biswaranjan Mishra, and Attila Kertesz, "Stress-Testing MQTT Brokers: A Comparative Analysis of Performance Measurements," *Energies*, vol. 14, no. 18, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[5] Vishal A. Thakor et al., "Lightweight Cryptography Algorithms for Resource-Constrained IoT Devices: A Review, Comparison and Research Opportunities," *IEEE Access*, vol. 9, pp. 28177-28193, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[6] T.N. Ford, E. Gamess, and C. Ogden, "Performance Evaluation of Different Raspberry Pi Models as MQTT Servers and Clients," *International Journal of Computer Networks & Communications*, vol. 14, no. 2, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[7] Sahmi Imane, Mazri Tomader, and Hmina Nabil, "Comparison between CoAP and MQTT in Smart Healthcare and Some Threats," *International Symposium on Advanced Electrical and Communication Technologies*, pp. 1-4, 2018. [CrossRef] [Google Scholar]

[Publisher Link]

[8] M. Zorkany, K. Fahmy, and Ahmed Yahya, "Performance Evaluation of IoT Messaging Protocol Implementation for E-Health Systems," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 11, pp. 412-419, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[9] Yesenia Guamán et al., "Comparative Performance Analysis between MQTT and CoAP Protocols for IoT with Raspberry Pi 3 in IEEE 802.11 Environments," *2020 15th Iberian Conference on Information Systems and Technologies*, pp. 1-6, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[10] Davide Borsatti et al., "From IoT to Cloud: Applications and Performance of the MQTT Protocol," *22nd International Conference on Transparent Optical Networks*, pp. 1-4, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[11] Edgaras Baranauskas, Jevgenijus Toldinas, and Borisas Lozinskis, "Evaluation of the Impact on Energy Consumption of MQTT Protocol over TLS," *CEUR Workshop Proceedings: IVUS International Conference on Information Technologies: Proceedings of the International Conference on Information Technologies*, pp. 56-60, 2019. [Google Scholar] [Publisher Link]

[12] Thomas Prantl et al., "Performance Impact Analysis of Securing MQTT using TLS," *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, pp. 241-248, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[13] Borislav Toskov et al., "Architecture of Intelligent Guard System in the Virtual Physical Space," *IEEE 10th International Conference on Intelligent Systems*, pp. 265-269, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[14] Aleksandar Velinov et al., "Covert Channels in the MQTT-Based Internet of Things," *IEEE Access*, vol. 7, pp. 161899-161915, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[15] Elias M. Pinheiro, and Sérgio D. Correia, "Software Model for a Low-Cost, IoT Oriented Energy Monitoring Platform," *SSRG International Journal of Computer Science and Engineering*, vol. 5, no. 7, pp. 1-5, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[16] Raspberry Pi 3 Model B. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[17] Maximintegrated, Ds18B20: Programmable Resolution 1-Wire Digital Thermometer, 2015. [Online]. Available: https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf

[18] Operating System Images. [Online]. Available: https://www.raspberrypi.org/software/operating-systems/#raspberry-pi-os-32-bit

[19] Téléchargements. [Online]. Available: https://raspberry-pi.fr/telechargements/

[20] Paho-MQTT 1.6.1. [Online]. Available: https://pypi.org/project/paho-mqtt/

[21] Threading - Thread-Based Parallelism. [Online]. Available: https://docs.python.org/3/library/threading.html#

[22] Top. [Online]. Available: http://manpages.ubuntu.com/manpages/xenial/man1/top.1.html?_ga=2.164635455.1084823020.1640179895-1054292474.1615460534

[23] The Config.txt File. [Online]. Available: https://www.raspberrypi.com/documentation/computers/config_txt.html

[24] Ruth Suehle, and Tom Callaway, *Raspberry Pi Hacks: Tips & Tools for Making Things with the Inexpensive Linux Computer*, 1st ed., O'Reilly Media, pp. 1–62, 2013. [Google Scholar] [Publisher Link]

[25] Warren Gay, *Advanced Raspberry Pi: Raspbian Linux and GPIO Integration*, 2nd ed., Apress Berkely, pp. 245–258, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[26] Renice. [Online]. Available: http://manpages.ubuntu.com/manpages/xenial/en/man1/renice.1.html?_ga=2.179003750.378150101.1624202291-1054292474.1615460534

[27] Diana Bezerra Correia Lima et al., "A Performance Evaluation of Raspberry Pi Zero W Based Gateway Running MQTT Broker for IoT," *IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference*, pp. 76-81, 2019. [CrossRef] [Google Scholar] [Publisher link]