Design and Implementation of MQTT Load Test System

In Hwan Jung*, Jae Moon Lee and Kitae Hwang

School of Computer Engineering, Hansung University, 116 Samseongyoro-16 Gil, Seongbuk-gu Seoul, 136-792, Korea
*Corresponding author. Tel.: +82-10-2261-6307; Email address: ihjung@hansung.ac.kr

Article History:Received:11 november 2020; Accepted: 27 December 2020; Published online: 05 April 2021

Abstract: In this paper, we design and implement an MQTT load test tools for evaluating the performance of an MQTT appliance system which is capable of handling large amounts of MQTT traffic. The implemented system consists of a master that can centrally control the performance evaluation process and a set of clients that creates virtual MQTT devices and generates MQTT traffic. The MQTT Load Test system was developed using the Java language and the Jpcap library. We developed a GUI version load test and also developed a console mode system that can be executed in an environment where GUI is not available. Using the MQTT Load Test system implemented in this paper, we were able to effectively evaluate the performance of a high-performance MQTT Appliance, another important research topic. Since MQTT Load Test Master and Slave have developed not only the GUI version but also the Console version, it is possible to test the performance even in a remote connection situation where GUI is not supported. The experiment could be conducted in various scenarios, and the number of virtual clients that had to be created per a computer could be confirmed so that the computers participating in the experiment could generate or receive MQTT message traffic of a predetermined level.
Applications: The MQTT Load Test system implemented in this paper can be used as a tool for performance evaluation of high-performance MQTT Appliances.

Keywords: MQTT, MQTT Broker, MQTT Appliance, Load Test, Performance Evaluation.

1. Introduction

The MQTT(Message Queueing Telemetry Transport) protocol is a communication protocol for sending and receiving messages in a specific topic by using the Subscribe and Publish method[1][2]. It provides an efficient communication environment for collecting data of a large number of IoT devices due to a short transmission / reception data and simple communication procedure[3][4][5]. MQTT Appliance means a dedicated hardware equipped with MQTT Broker. In another research project, we are developing a high-performance MQTT Appliance. In this paper, we design and implement an MQTT load test system that can generate MQTT traffic in order to analyze the performance of an MQTT appliance which can handle massive MQTT traffic load. The implemented performance analysis system consists of a master for central control and slaves each of which acts as either a number of virtual publishers which generate MQTT traffic or virtual subscribers which receive MQTT data. The slave creates a number of virtual MQTT clients as thread processes based on the criteria set by the load test master.

The MQTT Load Test System implemented in this paper can simulate an environment in which a large number of MQTT clients exist using a small number of computers with minimum overhead. Especially, since both Master and Slave are implemented as Java-based, both GUI version and Console version, they can be used regardless of any platform such as Windows or Linux.

The composition of this paper is as follows. In Section 2, related review is described. In Section 3, we describe the design and implementation of the MQTT load test system. In section 4, the experimental results are described. Finally, Section 5 presents conclusions and future research.

2. Related Works

2.1 MQTT Protocol Analyzer

In our former study[6], we have developed an MQTT protocol analyzer, MQTTAnalyzer, which can monitor and debug the MQTT application system using packet capture method. The MQTTAnalyzer can check the detailed contents of MQTT messages and the traffic applied to the MQTT Appliance at the network level. On the other hand, the MQTT Load Test system implemented in this paper can generate real-world traffic and measure the performance at peer point, both publisher and subscriber. In other words, the MQTTAnalyzer measures the details of the message and traffic from the perspective of the appliance, but the MQTT Load Test system can measure the performance from the perspective of the clients. Thus, the two tools can be used cooperatively.

2.2 MQTT Simulator

A number of MQTT simulator exist such as MQTTLens[7], MQTTfx[8] and Gambit[9]. Although MQTTLens and MQTTfx can be used to simulate MQTT client by using GUI screen and is easy to use, they are very basic MQTT client which can not be used to generate large MQTT traffic, which is the main research goal

***Corresponding author:** In Hwan Jung
School of Computer Engineering, Hansung University, 116 Samseongyoro-16 Gil, Seongbuk-gu Seoul, 136-792, Korea . Email address: ihjung@hansung.ac.kr

of this paper. On the other hand, Gambit is an MQTT simulator which is an effective MQTT performance evaluation tool by using GUI screen for user convenience and generating virtual IoT devices as virtual MQTT clients. It is mainly designed to demonstrate an environment where large number IoT devices exist. In this paper, we implemented a GUI-based MQTT load test system based on Java framework and Winpcap[10] based Jpcap[11] which is portable for any type of platform such as Windows or Linux. Furthermore, a text-based Console version master and client was also implemented for an environment in which a GUI program cannot be used.

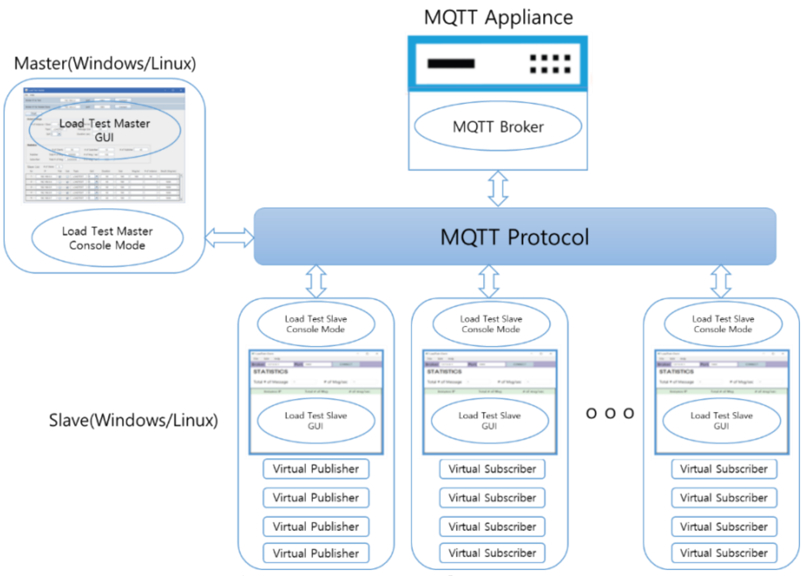


Figure 1 System Architecture

3. System Design and Implementation

3.1 System Architecture and Flow

Figure 1 shows the overall system structure. The entire system is composed of one Load Test Master, multiple Load Test Slaves, and MQTT Appliance equipped with MQTT Broker, which is the target of performance evaluation. Master and Slave communicate using MQTT Topic. There may be a separate broker for communication between them, but basically it is not a problem even if you use the broker of MQTT Appliance. Since Master and Slave programs are developed in Java, they can be executed in both Windows and Linux environments. In particular, text-based Console version was also developed in preparation for an environment where GUI is not possible.

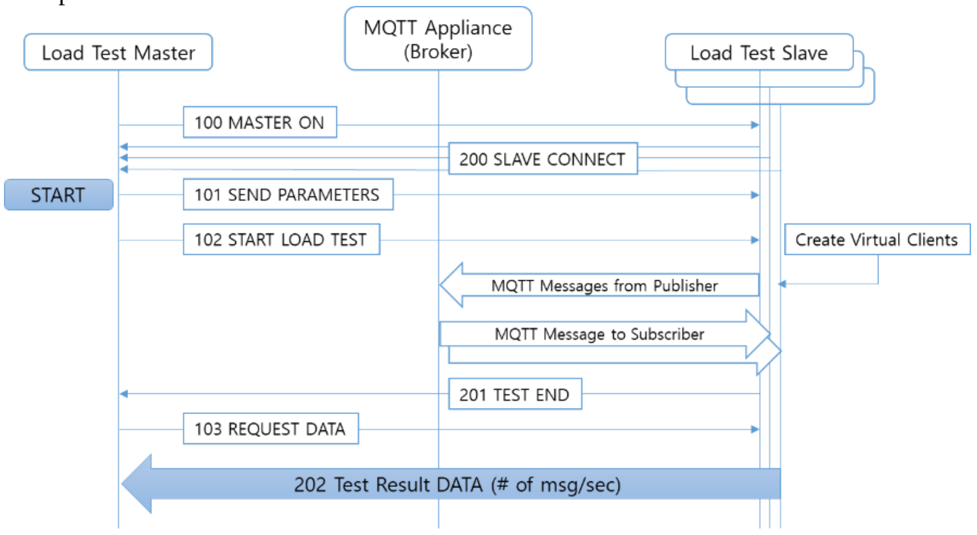


Figure 2 Load Test Execution Flow

Figure 2 is MQTT Load Test execution flow. Both Master and Slave communicate using MQTT Topic. They use the broker of the MQTT appliance under the test as a default but it does not affect the performance of the broker because it communicates before the actual load test starts. The order of execution is as follows.

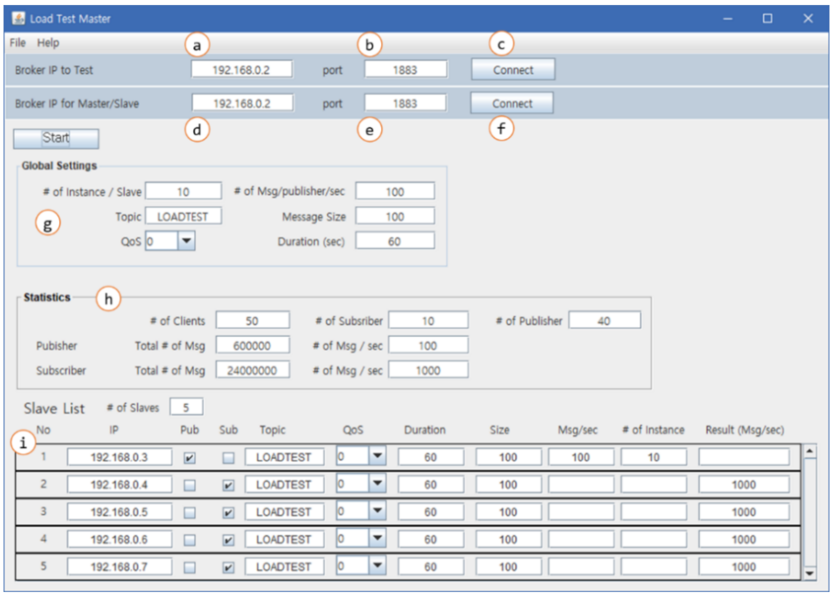
a. When the Master program is executed, it subscribes to MQTT Topic and waits for Slave connection.

b. The Slave connects to the Master using the configured Broker IP and Topic and waits for the Master's command.
c. The master shows the slave list every time the slave is connected, and the user sets the load test parameters, such as # of instances/slave, # of msg/sec, message size, test duration and etc. In particular, it is important to set whether each slave operates as a publisher or a subscriber.
d. When the user presses start, the master transmits performance evaluation variables to the slave, the slave creates virtual clients, the publisher generates an MQTT message corresponding to the specified amount of traffic, and the subscriber starts receiving messages from the broker.

3.2 GUI Version Load Test System

In this study, we developed two types of MQTT Load Test. At first, we implemented GUI(Graphical User Interface) version like conventional application. The other type is Console version which does not have GUI screen, instead it accepts command line input and prints out load test results in text format.

Figure 3 shows the GUI version Load Test Master. It controls the entire performance test. Using its GUI, load test parameters, such as # of virtual instance/slave, # of messages/sec, test duration and etc.., can be easily set.



Broker IP to Test	192.168.0.2	port	1883	Connect
Broker IP for Master/Slave	192.168.0.2	port	1883	Connect

Global Settings

# of Instance / Slave	10	# of Msg/publisher/sec	100
Topic	LOADTEST	Message Size	100
QoS	0	Duration (sec)	60

Statistics

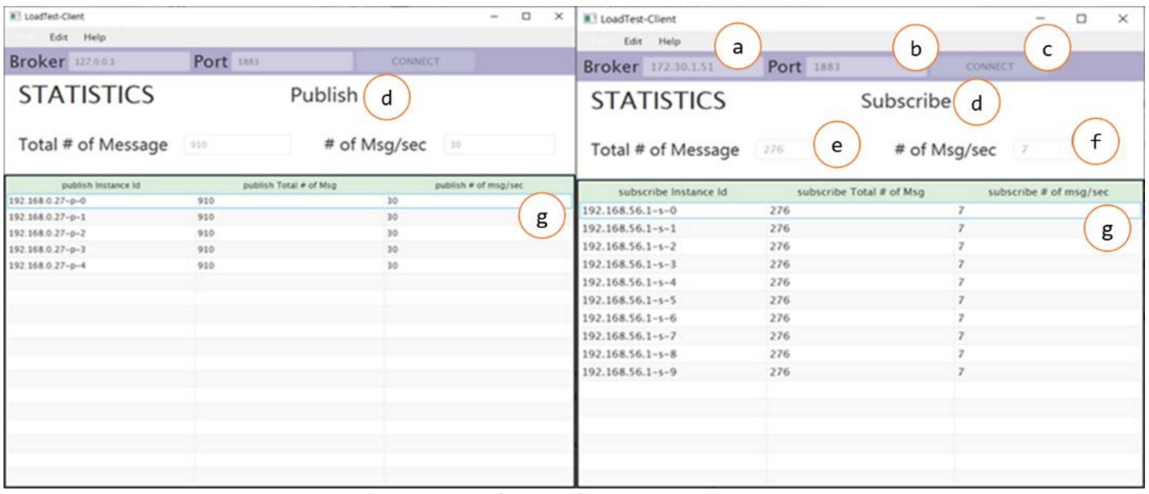
	# of Clients	50	# of Subsriber	10	# of Publisher	40
Pubisher	Total # of Msg	600000	# of Msg / sec	100		
Subscriber	Total # of Msg	24000000	# of Msg / sec	1000		

Slave List # of Slaves 5

No	IP	Pub	Sub	Topic	QoS	Duration	Size	Msg/sec	# of Instance	Result (Msg/sec)
1	192.168.0.3	✓		LOADTEST	0	60	100	100	10	
2	192.168.0.4		✓	LOADTEST	0	60	100			1000
3	192.168.0.5		✓	LOADTEST	0	60	100			1000
4	192.168.0.6		✓	LOADTEST	0	60	100			1000
5	192.168.0.7		✓	LOADTEST	0	60	100			1000

Figure 3 Load Test Master GUI version

The detailed screen configuration is as follows.
a. MQTT Appliance IP
b. MQTT Appliance Port # (default 1883)
c. Connect to Appliance
d. MQTT Broker IP for communication between Master and Slave (same as Appliance)
e. MQTT Broker Port # for communication between Master and Slave (same as Appliance)
f. Connect to Broker
g. Global Load Test Parameters. This is default values for all slaves. The # of instances/slave(a) determines how many virtual clients should be created. The # of Msg/Publisher/sec(b) is the amount of traffic that should be generated by each virtual publisher. Therefore, the traffic generated by each slave computer becomes (a)x(b). In this respect, the total traffic applied to the MQTT appliance can be calculated by (a)x(b) multiplied by # of slave computers enrolled as publishers.
h. Load Test Result Statistics. It shows # of clients which is the sum of # of publishers and # of subscribers. For the publisher, the value Total # of Msg means the total number of published message incoming into the MQTT appliance. On the other hand, for the subscriber, the Total # of Msg accounts for how many outbound messages are delivered to the subscribers. In this respect, the # of Msg/sec represents the inbound or outbound throughput accomplished by the MQTT appliance.
i. List and Status of Slaves. When a slave computer connects to a slave, new slave information is added to the list. The user can set each slave as either a publisher or a slave. In addition, ignoring global settings in (g), each slave can be configured differently. It means, for each slave computer, the QoS level, test duration, message size, traffic (msg/sec) or # of instance(virtual client) values can be set differently according to required application environment.



Broker 127.0.0.1 Port 1883

STATISTICS Publish

Total # of Message 910 # of Msg/sec 30

publish Instance Id	publish Total # of Msg	publish # of msg/sec
192.168.0.27-p-0	910	30
192.168.0.27-p-1	910	30
192.168.0.27-p-2	910	30
192.168.0.27-p-3	910	30
192.168.0.27-p-4	910	30

Broker 172.30.1.51 Port 1883

STATISTICS Subscribe

Total # of Message 276 # of Msg/sec 7

subscribe Instance Id	subscribe Total # of Msg	subscribe # of msg/sec
192.168.56.1-s-0	276	7
192.168.56.1-s-1	276	7
192.168.56.1-s-2	276	7
192.168.56.1-s-3	276	7
192.168.56.1-s-4	276	7
192.168.56.1-s-5	276	7
192.168.56.1-s-6	276	7
192.168.56.1-s-7	276	7
192.168.56.1-s-8	276	7
192.168.56.1-s-9	276	7

Figure 4 Load Test Slave GUI version

Figure 4 shows Load Test Slave GUI version. Once a slave is executed, it connects to the Master using MQTT topic and waits for Master's command for Load Test. When the load test scenario begins, the slave creates virtual MQTT clients as threads and starts to generate MQTT traffic in case of publisher or wait for message in case of subscriber. The detailed screen configuration of Figure 4 is as follows.

a. Broker IP (for communicating with Master, same as the Appliance)
b. Broker Port # (default 1883)
c. Connect to Broker (Master)
d. Operating mode : Publisher or Subscriber
e. Total Number of messages sent/received
f. Average Number of messages sent/received
g. Status of Virtual Clients. The number of virtual clients is determined by the master. For each virtual client, the Total # of Msg and the # of Msg/sec are displayed after the load test finished

3.3 Console Version Load Test System

In this study, considering the case where the MQTT Appliance is located at a remote location such as an IDC(Internet Data Center) where GUI is not applicable, a text-based console version without GUI was also developed. In order to perform a load test in such an environment, a remote access program such as Telnet[12] or SSH[13] must be used, and a load test program that can be executed from the command line is required. In this respect, the developed Console version can be run by omitting the Java GUI and setting it as an option in the Command Line like an ordinary Linux commands. It is executed by inputting the test scenario file in XML format generated by the GUI tool. Figure 5 shows the flow of Master GUI and Console version program. Once the user set parameters and starts the load test, it generates a scenario file and invokes the Console Version Master which eventually communicates with load test slaves and starts the load test. Figure 6 shows an example of scenario file which is generated by Load Test Master GUI. In Figure 6, the scenario.xml contains all the parameters set by GUI screen. The first client is set as a publisher and the number of instance is set by 10. The second client is configured as subscriber and also the number of instances is 10.

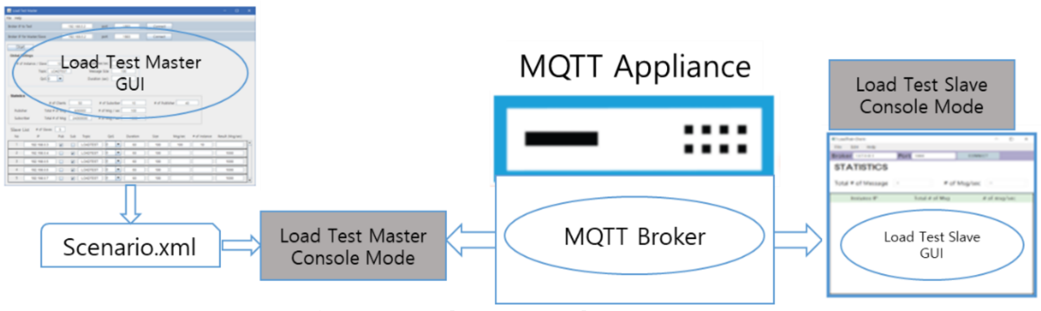
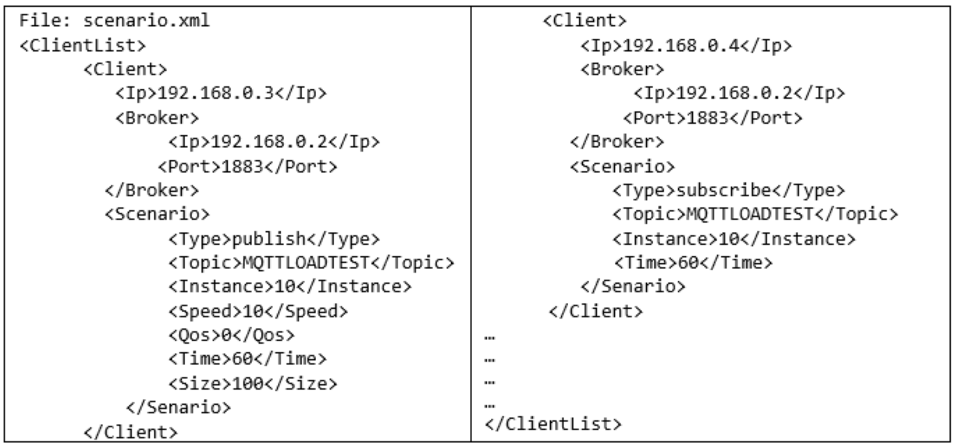
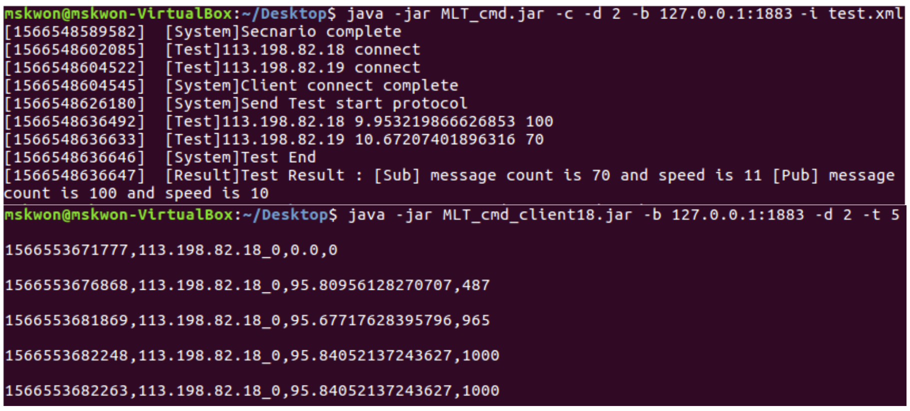


Figure 5 Load Test Console Version



```
File: scenario.xml
<ClientList>
      <Client>
         <Ip>192.168.0.3</Ip>
         <Broker>
              <Ip>192.168.0.2</Ip>
             <Port>1883</Port>
        </Broker>
        <Scenario>
              <Type>publish</Type>
              <Topic>MQTTLOADTEST</Topic>
              <Instance>10</Instance>
              <Speed>10</Speed>
              <Qos>0</Qos>
              <Time>60</Time>
              <Size>100</Size>
          </Senario>
      </Client>
      <Client>
         <Ip>192.168.0.4</Ip>
         <Broker>
              <Ip>192.168.0.2</Ip>
             <Port>1883</Port>
        </Broker>
        <Scenario>
            <Type>subscribe</Type>
            <Topic>MQTTLOADTEST</Topic>
            <Instance>10</Instance>
            <Time>60</Time>
         </Senario>
      </Client>
…
…
…
…
</ClientList>
```

Figure 6 Example of scenario.xml



```
mskwon@mskwon-VirtualBox:~/Desktop$ java -jar MLT_cmd.jar -c -d 2 -b 127.0.0.1:1883 -i test.xml
[1566548589582]  [System]Secnario complete
[1566548602085]  [Test]113.198.82.18 connect
[1566548604522]  [Test]113.198.82.19 connect
[1566548604545]  [System]Client connect complete
[1566548626180]  [System]Send Test start protocol
[1566548636492]  [Test]113.198.82.18 9.953219866626853 100
[1566548636633]  [Test]113.198.82.19 10.67207401896316 70
[1566548636646]  [System]Test End
[1566548636647]  [Result]Test Result : [Sub] message count is 70 and speed is 11 [Pub] message
count is 100 and speed is 10
```

```
mskwon@mskwon-VirtualBox:~/Desktop$ java -jar MLT_cmd_client18.jar -b 127.0.0.1:1883 -d 2 -t 5

1566553671777,113.198.82.18_0,0.0,0

1566553676868,113.198.82.18_0,95.80956128270707,487

1566553681869,113.198.82.18_0,95.67717628395796,965

1566553682248,113.198.82.18_0,95.84052137243627,1000

1566553682263,113.198.82.18_0,95.84052137243627,1000
```

Figure 7 Console Version Master and Slave

Figure 7 shows the Console Version Master and Slave screen. The Console Version Master is executed with the following syntax and the execution options are as follows.

- Syntax : java –jar [Master jar file name] –c –b IP:port –d level –i scenario.xml
- Options

  -c: Console version of Master is executed. If -g is entered in place of -c, GUI version of Master is executed as shown in Figure 3.

  -b: This is an option to enter the address of the broker for communication to be connected. It should be written in IP:port format, and enter it as "java -jar Master.jar -c -b 192.168.0.2:1883". The IP address must be separated from the broker port number with a colon (:).

  -d: There are two debug options, 0 and 1. 0 simply prints the System and Result logs, and 1 prints the System, Result, and Client logs. Enter "java -jar Master.jar -c -b 192.168.0.2:1883 -d 0". System, Result, and Client logs are described in Log Type below.

  -i: Specifies the XML file saved in the GUI as an information option. This option is an improved option for the scenario.xml file previously inserted as redirection.
- Output (Log Types)

  [*Current time*] [*tag*] [*Message*]

  a. *current time*: Displays the current time in milliseconds units.

  b. *Tag*

    -System: This tag means the test situation. The log shows that the program is running normally, such as the start of the test, the execution of the test, and the end.

    -Client: It is a tag related to Slave pc. Notifies that the slave pc is connected to the communication broker or outputs the message sent by the slave pc to the master pc as a log. This message includes the test result of the Slave PC. If 0 level is entered in the -d option, the tag is not output.

    -Result: This is a tag that means the result of the test. It also means the end of the program. The test result value transmitted from each slave pc to the master pc is calculated in the master pc, the average value is output, and the program is terminated.

  c. *Message*: Test result message

The Console version Slave is executed with the following syntax and the execution options are as follows.

- Syntax : java –jar [Slave jar file name] –c –b IP:port –d 0/1
- Options

  -c: Console version of Slave is executed. If -g is entered in place of -c, GUI version of Slave is executed as shown in Figure 4.

  -b: This is an option to enter the address of the broker for communication to be connected.

  -d: There are two debug options, 0 and 1. 0 simply prints the System and Result logs, and 1 prints the System, Result, and Client logs
- Output (Log Types)

  [*Current time*] [*tag*] [*Message*]

  a. *Ccurrent time*: Displays the current time in milliseconds units.

  b. *Tag*

  -System, Client or Result. Same as Master.

  c. *Message*: Test result message after load test finishes. It shows number of messages/sec and total number of message for each virtual clients.

4. Experiment

The purpose of the experiment is to verify whether the MQTT Load Test system can simulate various application environments for the MQTT Appliance subject to performance testing. The performance of an MQTT Appliance should be tested differently according to two extremely different application environments. First case is that there are many outbound messages. In this case, like broadcast message transmission, there are few publishers but many subscribers for the same topic. The second is the case where there are many inbound messages because there are a small number of subscribers and a large number of publishers, such as when many IoT devices send data to a single data collecting server.

Table1 Experiment 1 Configurations

Slave Computer Specification	CPU : Intel 4 core i7-7700 Memory : 4 Giga bytes OS : Ubuntu Linux/Windows
Publishing Speed (# of Msg/sec)	10, 20, 50, 100
# of Instances	50, 100, 200, 300, 400, 500
Test duration (sec)	60
Size of Message (bytes)	100
QoS Level	0

4.1 Experiment 1 – Effect of # of Instances

The purpose of this experiment is to check the maximum number of instances of one Load Test Slave computer and to see if the maximum instances are applicable in a test environment where multiple slaves participate. This Load Test system creates a number of virtual clients, publishers to generate MQTT messages, corresponding to the # of instances value as threads on the slave computer. Since the # of instances value is greatly affected by the performance of the computer on which the slave is running, an experiment to know the performance limit of the slave is required in advance to full integration test. Table 1 shows the slave computer hardware specifications and experimental conditions to be tested. Figure 8 shows the experimental results. Since the computer used in the experiment was not sufficiently capable, it was not executed if the instance value was 400 or higher. This process is essential before proceeding with full-scale integration testing. In this way, b measuring the performance of the slave computer used in the load test in advance, it will be possible to predict an appropriate value for the full integration test where may slave computers are involved.

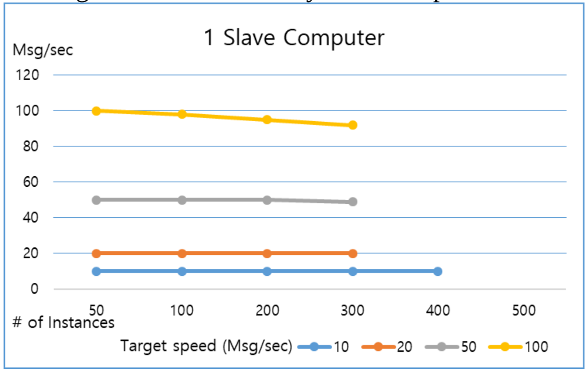


Figure 8 Experiment 1 Result – Effect of # of Instances

As shown in Fig. 8, as a result of testing instances up to 500 for one slave computer, the experiment was successful for instances up to about 300, but in the case of more than 400, the slave computer could not generate the target traffic, rather the slave computer stopped because too many instances of threads were executed. As a result, it can be recognized that the slave computer used in this experiment can create up to around 300 instances. In this way, the goal of this experiment is to determine the maximum instance the slave computer can handle [13,14].

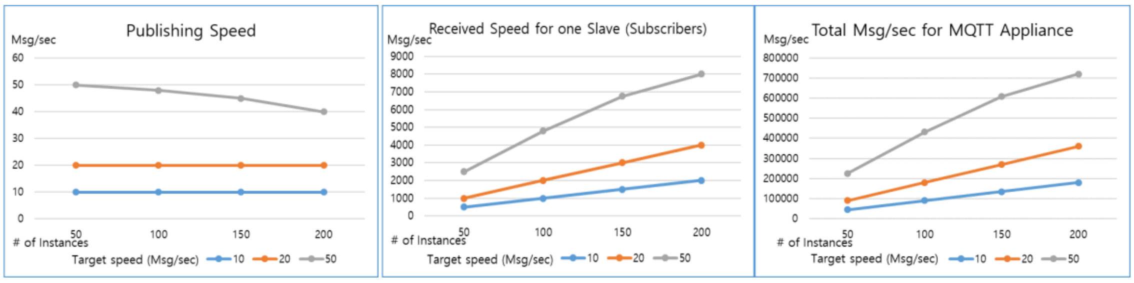
4.2 Experiment 2 – Outbound load test

This experiment is to test an environment with a small number of publishers and a large number of subscribers. That is, assuming an application system with high outbound traffic that simultaneously delivers the same message to a large number of subscribers. Even if a small number of brokers generate MQTT messages, if there are many subscribers to the same topic, the broker internal processing overhead increases and network traffic increases as the broker must copy the same message and deliver it to the subscribers. Table 2 shows experimental configurations.

Table2 Experiment 2 Configurations

Slave Computer Specification	CPU : Intel 4 core i7-7700 Memory : 4 Giga bytes OS : Windows
MQTT Appliance Specification	CPU : Intel 12 Core, Dual Xeon Memory : 32 GB NIC : 40GB x 4 port
Number of Slave Computers	10
Number of Slave for Publisher, Number of Insances	1 1
Number of Slave for Subscriber, Number of Instances	9 50 100 150 200
Publishing Speed (# of Msg/sec	10, 20, 50, 100
Test duration (sec)	60
Size of Message (bytes)	100
QoS Level	0

# of Subscriber Instances/Slave	Number of total		Target speed Msg/Innstance/sec			
	publisher	subscribers	10	20	50	
50	1	450	4500	9000	22500	Target Appliance Total # of outbound Msg/sec
100	1	900	9000	18000	45000	
150	1	1350	13500	27000	67500	
200	1	1800	18000	36000	90000	



(a) Speed of Publishing (b) Received msg/sec for on Slave (c) Outbound Throughput

Figure 9 Result of Experiment 2 - Outbound Load Test

Figure 9 shows the result of the outbound load test. One publisher generates 10, 20, and 50 messages per second on one slave computer. Figure 9 (a) is the result of measuring message generation speed of the publisher. When the number of messages generated per second is less than 20 msg/sec, the same value as the target value is measured, but when it is as large as 50 msg/sec, it is observed to decrease slightly as the number of subscribers increased. This is because MQTT Broker copies a large number of messages when the number of subscribers is large, so the throughput of MQTT Appliance decreases due to increased network traffic and overhead. Figure 9 (b) is the measurement result of how many messages are received in terms of the slave computer running the virtual subscribers. As the instance per slave increases, no perfect proportional value is observed. That is, if the publishing speed is over 50 msg/sec and the number of instances is over 100, it shows that the increase rate decreases due to the increase in overhead applied to the MQTT broker and network. Finally, Figure 9 (c) shows Total Msg/sec generated by the MQTT Broker, the Appliance equipment under the test. This value is a measure of the maximum throughput that the MQTT Appliance can handle. As shown in Figure 9 (c), the MQTT Appliance used in this experiment showed a maximum speed of 720,000 msg/sec.

4.3 Experiment 3 – Inbound load test

This experiment assumes an environment with a large number of publishers and a small number of suscribers. For example, if there are a large number of IoT sensors, the IoT sensor will be the publisher and the server collecting data will be the subscriber. In this case, since a large number of subscribers send a push message to the MQTT borker, the MQTT broker has many incoming messages. In this experiment, contrary to Experiment 2, it is assumed that there is one subscriber and a total of 90 publishers. In order to measure the inbound message throughput of the MQTT Broker Appliance, the subscriber's message reception rate is measured. Table 2 shows experimental configurations.

Table3 Experiment 3 Configurations

Slave Computer Specification	CPU : Intel 4 core i7-7700 Memory : 4 Giga bytes OS : Windows
MQTT Appliance Specification	CPU : Intel 12 Core, Dual Xeon Memory : 32 GB NIC : 40GB x 4 port
Number of Slave Computers	10
Number of Slave for Subscriber, Number of Insances	1 1
Number of Slave for Publisher, Number of Instances	9 50 100 150 200
Publishing Speed (# of Msg/sec)	10, 20, 30
Test duration (sec)	60
Size of Message (bytes)	100
QoS Level	0

# of Publisher Instances/Slave	# of total		Target speed Msg/Instance/sec			
	subscriber	publishers	10	20	50	
50	1	450	4500	9000	22500	Target Appliance Total # of Inbound Msg/sec
100	1	900	9000	18000	45000	
150	1	1350	13500	27000	67500	
200	1	1800	18000	36000	90000	

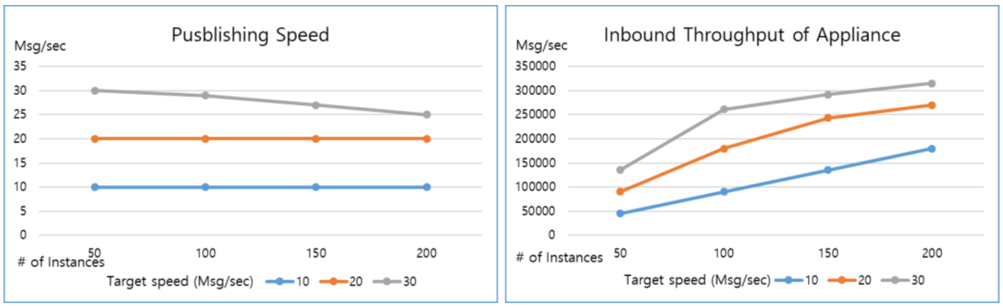


Figure 9 Result of Experiment 3 - Inbound Load Test

Figure 10 shows the results of Experiment 3, an inbound load test. Fig. 10(a) shows the result in which there exists one subscriber and total 90 publishers generate 10, 20 and 30 messages per second. If # of instance increases and target speed is large, publishers show values less than the target value. Figure 10 (b) shows the results of the MQTT Broker Appliance's Inbound throughput. In both graphs, when the number of instance value was large and the publish rate was large, the target throughput was not observed. The reason is that all messages are concentrated in one subscriber.

5. Conclusion

In this paper, we have designed and implemented MQTT load test system, which consists of Master and Slave program, that can be used for evaluating the operation status of MQTT broker in terms of performance aspect. The implemented load test slave system can generate MQTT traffic for simulating large number of MQTT clients like IoT sensors. Since the implemented MQTT Load Test system uses the Java framework, it can be executed on any platform, and it has great usability because not only the GUI version but also the text-based Console version was developed.

Throughout various experiments, it was confirmed that the implemented Load test system can effectively test the performance of the MQTT Broker Appliance. An experimental method was also developed to check how many virtual clients need to be created to generate maximum traffic according to the hardware specifications of the slave computers used for the load test.

Future research on this study is to add detailed parameter setting functions and improve performance so that large capacity MQTT traffic can be generated with minimum overhead, for verifying the performance of MQTT Appliance system being developed in this research. GUI(Graphical.

6. Acknowledgement

This research was financially supported by Hansung University.

7. References

1. https://en.wikipedia.org/wiki/MQTT
2. Oasis, “MQTT v3.1.1” http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html
3. Lampkin V, et al. Building smarter planet solutions with MQTT and IBM WebSphere MQ telemetry IBM. ITSO
4. Hermes Aslava, Luis Alejandro Rojas and Ramon Pereira. Implementation of Machine-to
5. Machine Solutions Using MQTT Protocol in Internet of Things (IoT) Environment to Improve Automation Process for Electrical Distribution Substations in Colombia. Journal of Power and Energy Engineering, pp. 92-96, 2015. DOI:https://doi.org/10.4236/jpee.2015.34014
6. Kitae Hwang, Heyjin Park, Jisu Kim, Taeyun Lee, Inhwan Jung. An Implementation of Smart Gardening using Raspberry_pi and MQTT. The Journal of the Institute of Internet, Broadcasting and Communication, Vol. 18, No. 1, pp.151-159, Feb. 2018. DOI:https://doi.org/10.7236/JIIBC.2018.18.1.151
7. Kitae Hwan, Jae-Moon Lee and In-Hwan Jung, Design and Implementation of MQTT Protocol Analyzer, International Journal of Advanced Science and Technology, Vol. 28, No. 5, pp. 92-99. Sep, 2019.
8. MQTTLens, MQTT Client for Chrome, Chrome Web Store.
9. MQTTfx, The JavaFX based MQTT Client, http://www.mqttfx.org/
10. Gambit MQTT Simulator, https://www.gambitcomm.com/
11. Winpcap, windows packet capture library, https://www.winpcap.org/
12. Jpcap, Capture Network Packages with Java, https://javatutorial.net/tag/jpcap
13. Telnet, https://en.wikipedia.org/wiki/Telnet, RFC 854
14. SSH, Secure Shell, https://en.wikipedia.org/wiki/Secure_Shell, RFC 4251
15. Pradhan, A., Bisoy, S. K., & Mallick, P. K. (2020). Load balancing in cloud computing: Survey. In Innovation in Electrical Power Engineering, Communication, and Computing Technology (pp. 99-111). Springer, Singapore.
16. Bisoy, S. K., Mallick, P. K., & Mishra, A. Fairness Analysis of TCP Variants in Asymmetric Network. International Journal of Engineering & Technology, 7(2.12), 231-233.