# A Performance Benchmarking Methodology for MQTT Broker Implementations

Ilie-Daniel Gheorghe-Pop
SQC
Fraunhofer FOKUS
Berlin, Germany
ilie-daniel.gheorghe-pop@fokus.fraunhofer.de

Alexander Kaiser
Relayr GmbH
Berlin, Germany
alexander.kaiser@relayr.de

Axel Rennoch
SQC
Fraunhofer FOKUS
Berlin, Germany
axel.rennoch@fokus.fraunhofer.de

Sascha Hackel
SQC
Fraunhofer FOKUS
Berlin, Germany
sascha.hackel@fokus.fraunhofer.de

***Abstract*— The rapid growth of IoT across the globe has been significant over the past decade. As the number of connected devices increases by the order of billions year over year, the capacity and operating costs of IoT networks and associated communications software becomes crucial. The manufacturers, software developers, integrators, telco operators as well as business-end users face an increasing need of a benchmarking reference that covers performance aspects of IoT transport protocols. This paper introduces a performance benchmarking methodology as well as examples for the definition of performance tests for the MQTT protocol. The implementation work was done within the open source project IoT Testware project which is part of the Eclipse Foundation. The test suites were specified in TDL-TO and realized in TTCN-3 using the open source IDE Eclipse Titan. The test specifications are covered by the standardization activities of the ETSI working group MTS TST.**

***Keywords— Performance testing, Benchmarking methodology, MQTT, IoT, Open Source, TTCN-3***

## I. INTRODUCTION

IoT has become a major field in computing. The IoT ecosystem has quickly evolved, driven by rapid developments in standards, technologies and platforms. The continuous adoption of IoT across major economic sectors such as Agriculture, e-Health, Manufacturing, Automotive, Retail, Transport and Mobility as well as building and home automation[1] has provided both challenges as well as business opportunities for the IT&C domain.

Cloud adoption has also been rising across all major industries. As the business evolves, tailored services are being developed addressing the vertical sectors. Among these, PaaS Cloud services for IoT applications have seen a significant increase over the last years[2].

As a wide range of IoT devices are constantly entering the market. At the same time, numerous new cloud solutions from small vendors are being developed and racing to create integrated commercial systems. For IoT communication in particular, cloud adoption has created the demand for increasingly reliable, secure and scalable IT solutions. For developers, this has become a constant challenge in the rapidly evolving industry.

Due to the wide range of use-cases for IoT applications, solutions are addressing very specific requirements and are very heterogeneous. Many studies [3]-[11] have addressed these characteristics of IoT communication protocols. Additionally, in [12], the authors survey the most common application layer communication protocols based on their main characteristics and try to address the specific issues of communication.

Among the many transport protocols being developed, some have gained more traction due to their applicability. In one survey [13], the authors have provided details about the characteristics and specific IoT applications for such protocols. One specific challenge for the provider of an IoT communications component that implements such a protocol, is to be able to calibrate it in order to adapt to the network conditions and service demands. In order for such a system component to be commercially selected, it must first of all fulfil all the specific system requirements. The next important aspect lies in the operating costs and features. This is where scalability and performance become a main differentiating factor.

The authors in[12] have concluded that the message queue telemetry transport protocol (MQTT) is one of the most mature protocols suitable for IoT communication cloud architectures. MQTT has been introduced in 1999 by IBM and published as a standard in 2013 by OASIS [14] and further developed throughout the years to address new industry requirements, currently reaching version 5.0 [15].

Although MQTT broker performance requirements differ significantly depending on the targeted application, this paper aims to provide a generic approach towards its evaluation. As such, it proposes a benchmarking methodology for the MQTT broker implementations. This methodology addresses the specific requirements in terms of timing constraints, capacity and robustness. At the same time, it also presents the work performed towards standards development in the performance evaluation of the MQTT protocol [33].

This paper is organized as follows: Section II provides the background and related work. Section III introduces the performance metrics and measurement considerations. Section IV presents the benchmarking methodology and the general approach for performance evaluations. Section V presents different benchmarking scopes as well as a practical example.

Finally, section VI presents the conclusions and outlook on further work.

## II. Background

Because the IoT hardware is usually not acquired directly by the IoT communication software provider, and the IoT communication software solution is usually deployed on a commercial cloud, realistic end-to-end test cycles are needed to assess the performance of the solution and establish commercial SLAs. For MQTT, a reliable and generic evaluation methodology is currently not standardized. Because of this, various dimensioning and capacity principles are used for the overall solution architecture. These principles need to adapt to the specifics of the environments ranging from local datacentres to commercial public or private cloud. Additionally, even within the same cloud provider there is a product range to choose from.

An important element is the connectivity between the IoT platform solution components. Another critical element is the connectivity with the external network components (MQTT publishers and subscribers). Moreover, the compute and storage resource consumption across system components are rarely linearly dependent of the number of connected devices so the scaling parameters of the solution need to be specifically calculated.

Over the past few years, there have been several studies and evaluation methods [18]-[18] proposed to characterize and evaluate MQTT brokers. The most popular brokers chosen for the evaluation were open source due to their accessibility. Our work proposes a standardized requirements-driven performance evaluation methodology and the associated example implementation within the open-source IoT Testware project [21][20][21] which is part of the Eclipse Foundation. Following the standards-oriented testing model presented in [22], the test suites developed were specified in TDL-TO[23][24][25] and realized in TTCN-3[26] using the open source IDE Eclipse Titan[27]. Furthermore, the standardization work in this direction is currently ongoing within ETSI[28] complementing the conformance[29] and security testing[30].

This approach aims to provide a uniform means of evaluating the performance and benchmark testing of MQTT broker implementations in accordance with the industry practices. In the next section the MQTT broker performance metrics are introduced.

## III. Performance Metrics

The performance metrics specified herein pertain to the specifics of a MQTT broker implementation. As such, the objective is to use these metrics in order to determine how well the MQTT broker is performing its' functions. As MQTT is a transport protocol, the metrics will be focused on how fast, reliable and efficient the transport of data is handled. The metrics are designed to fit this purpose while covering multiple use-case scenarios.

In order for the collected measurement data to be useful, special consideration needs to be given to the test system (TS) setup. Given that the performance evaluation is targeting one or several brokers, identical TS setup characteristics are required in order for the evaluation results to allow valid comparisons between them. Some of the characteristics may refer to infrastructure, hardware, physical or virtual resources as well as network connectivity resources.

### A. Measurement Methodology

From the performance perspective, all measurable metrics related to the protocol should be considered. Although not exhaustive, these metrics can be categorized as initially proposed in [31] as follows:

Powerfulness metrics include 3 sub categories: Responsiveness, Capacity and Scalability. From the Responsiveness category the response time, roundtrip time and latency time metrics are used. From the Capacity category the arrival capacity, peak capacity, in progress capacity, streaming capacity and Throughput capacity metrics are used. From the Scalability category the scaling capacity metric is used.

Reliability metrics include 6 sub categories: Quality-of-Service, Stability, Availability, Robustness, Recovery, and Correctness. The Quality of Service sub category refers to well defined requirements which may include acceptable values or ranges for metrics from other categories. Stability refers to the capacity of the System to deliver acceptable performance over time. From the Availability sub category, the logical availability metric is used. From the Robustness sub category, the service capacity reduction and service responsiveness deterioration metrics are used. From the Recovery sub category, the service restart characteristics metric is used. Correctness metrics cover the ability of delivering correctly processed requests under high or odd load conditions.

Efficiency metrics cover resource utilization. The metrics cover the characteristics of resource usage, linearity, scalability and bottleneck. The efficiency metrics in this context are referring to the service level and not covering the platform level.

### B. Test Parameters

As a general practice, the benchmarking test environment should reflect as close as possible the production environment. For this reason, deployment platform and network parameters should be the same as in production. For the benchmark test script controlling the test system (TS) three types of parameter categories have been identified: Test input specific parameters, test output specific results and MQTT protocol standard v3.1.1. [14] specific elements further described in the following tables:

Table I contains a non-exhaustive list of test parameters defined for the benchmark standard. The list is expected to grow over time, as additional subsystems and system configurations are developed. These cover the test duration and measurement intervals, type of protocol specific messages, transport network specifics as well as performance metric validation thresholds.

Table II presents a non-exhaustive list of test output metrics. These cover the time-measurement results of the protocol specific application message calls. The listed metrics include success and error rates, number of processed protocol specific operations per time unit (e.g. connection requests per second) as well as minimum, maximum and average durations for protocol specific operations (e.g. min/max/average PUBLISH duration)

TABLE III. TEST PARAMETERS

| Parameter | Description |
|---|---|
| Duration | Amount of time that a system load is presented to a SUT |
| Type of call | Type of messages contained within a workload |
| NoC | number of clients generating or subscribing to data/control traffic |
| NoS | Number of servers handling data/control traffic |
| Transport interface | Underlying transport interface |
| WLF for GTW | Work load factor for gateway expressed in number messages received per second, by type of message |
| Payload | Size of the data in Bytes carried within a message. |
| Monitoring Window(s) | The time interval window for which the monitored metrics are recorded. This reflects the measuring accuracy (e.g. per second, minute, hour etc.) |
| Validation threshold(s) | The specific metric thresholds used for validating whether a system performs at specifications. |

TABLE II. TEST OUTPUT

| Metric | Description |
|---|---|
| Minimum call duration | The minimum duration of a successful message request/response interaction within a Monitoring Window |
| Maximum call duration | The maximum duration of a successful message request/response interaction within a Monitoring Window |
| Average call duration | The average duration of a successful message request/response interaction within a Monitoring Window |
| Total number of calls | The total number of workload specified request/response type operations executed during the test |
| Success rate | Percentage number of successful workload operations relative to the total workload operations |
| Error rate | Percentage number of failed workload operations relative to the total workload operations |
| Requests processed per time unit | This metric reflects the average number of successfully processed requests per preferred time unit (second/minute/etc.) |

TABLE I. MQTT MESSAGE TYPES

| Control Packet Name | Description | Client-> Server | Server-> Client | Payload |
|---|---|---|---|---|
| CONNECT | client requests a connection to the server | ✓ | | Required |
| CONNACK | acknowledge connection request | | ✓ | None |
| PUBLISH | Publish message | ✓ | ✓ | Optional |
| PUBACK | Publish acknowledgement | ✓ | ✓ | None |
| PUBREC | Publish received (QoS 2 publish received) | ✓ | ✓ | None |
| PUBREL | Publish release | ✓ | ✓ | None |
| PUBCOMP | Publish complete | ✓ | ✓ | None |
| SUBSCRIBE | Subscribe to topics | ✓ | | Required |
| SUBACK | Subscribe acknowledgement | | ✓ | Required |
| UNSUBSCRIBE | Unsubscribe from Topics | ✓ | | Required |
| UNSUBACK | Unsubscribe acknowledgement | | ✓ | Required |
| PINGREQ | Ping request | ✓ | | None |
| PINGRESP | Ping response | | ✓ | None |
| DISCONNECT | Disconnect notification | ✓ | ✓ | None |
| AUTH | Authentication Exchange | ✓ | ✓ | None |

Table III includes the MQTT set of control packet messages as well as their specific source and destination as well as whether they have an associated payload.

In the following section, the benchmarking methodology is presented.

## IV. BENCHMARKING METHODOLOGY

This sections aims to describe a viable methodology for benchmarking the performance of an MQTT broker. The approach is inspired from the examples in [32] from the point of view of measurement preconditions, approach and statistically consistent measurement sampling.

As a general precondition for performance benchmarking, a functionally correct implementation is a prerequisite. For this the general assumption is that the broker has passed the conformance testing described in [29].

### *A. Benchmarking Steps*

First, the System under Test is described, including hardware, resource manager (bare-metal/virtualization technology) and network connectivity (type: wired/air, latency, throughput capacity). This includes both the resources dedicated to the broker as well the ones for the test system.

Second, the type of performance benchmarking is established: whether the tests aim to determine the system KPI values or the tests aim to check whether the system meets established performance requirements. Depending on the objective, the approach will differ. In this step the KPIs and metrics w/o thresholds are selected. Two examples are presented in section V reflecting the specific approach.

Third, the appropriate tests are selected and the test input parameters are specified. These include the test types, duration, metric threshold requirements, sample size and validation

checks. Then, the monitoring system is configured, and the appropriate metrics are selected for observation.

Fourth, the tests are executed and the metrics are collected. For this stage it is highly relevant that the TS and broker are not affected by external factors in terms of compute and network resources. As an example, the monitoring system load on both the network and compute resources is commonly not taken into consideration and this leads to skewed results.

Finally, the test results are checked and validated leading to a verdict whether the performance tests are passed or failed.

### B. Benchmarking Metric Examples

Finally, the test results are checked and validated leading to a verdict whether the performance tests are passed or failed.

- connection-release delay: the time delay between DISCONNECT message and TCP connection closing. Value expressed in milliseconds (ms).
- setup-delay: the time interval starts when a CONNECT message is sent and ends when the corresponding CONNACK message has been received back. Value expressed in milliseconds (ms).
- publish delay: the time interval starts when a PUBLISH message is sent and ends when the corresponding PUBACK/PUBCOMP message has been received. Value expressed in milliseconds (ms).
- subscription delay: the time between SUBSCRIBE and SUBACK message. Value expressed in milliseconds (ms).
- unsubscription delay: the time between UNSUBSCRIBE and UNSUBACK message. Value expressed in milliseconds (ms).
- ping delay: the time between PINGREQ and PINGRESP message. Value expressed in milliseconds (ms).

For each of the measurements enumerated above, the minimum, maximum and average values are also calculated according to Table 2 and reported to the specified monitoring windows.

### C. Benchmark Types

For evaluating the metrics described in sub-section B, the benchmark tests can be grouped in 3 main categories.

*1) Load Tests:* These tests are used for determining or validating the broker workload range. The workload consists of one or multiple message exchanges between the broker and the Test System. The aim is to observe the Powerfulness and Efficiency metrics as well as the Correctness (Reliability category) metric in order to determine or validate the maximum operating workload handled by the broker.

*2) Endurance Tests:* These tests are used for determining or validating the broker Reliability and Efficiency. These tests generally consist of exposing the broker to a variable or high operational workload for long periods of time. The metrics observed are the Reliability and Efficiency ones. This type of testing covers operational aspects such as degradation over time, memory leaks and resource consumption estimates.

*3) Stress Tests: These tests are used for determining or validating the broker Robustness and Recovery (Reliability category) metrics. This is achieved by injecting workload spikes throughout the test and observing the degradation and recovery patterns of the system as well as the maximum workload operational limits.*

## V. Benchmarking Examples

### A. KPI Determination

A KPI determination benchmark is an exploratory performance evaluation that aims to determine the operational performance of a broker. The KPIs are specified as an input. The scope of this evaluation is to establish a reliable indication of how the System under Test is expected to perform in production. The KPIs are determined according to the intended use-case scenario for the broker.

### B. KPI Validation

The KPI validation benchmark is performance evaluation that aims to validate whether the broker performs according to requirement specifications. The KPIs and their thresholds are specified as an input. The scope of this evaluation is to establish a reliable indication of how the broker is expected to perform in production. The KPIs are determined according to the intended use-case scenario for the broker.

*a) KPI Determination:* As a first example, a device manufacturer has finished a hardware MQTT broker prototype for the industrial IoT market. The target objective is to provide communication in small industrial buildings serving a potential capacity of 50 to 5000 MQTT clients. The expected use-case requires QoS1 for data transmission and foresees a 1000-10000 published messages per second load. Additionally, the manufacturer wants to determine the system reliability, specifically Stability, Availability and Correctness.

As presented in section IV the first step is to specify the SuT hardware and network resources. In our example we consider the underlying hardware to be a bare-metal SoC box running Ubuntu 18.04 OS. It has a dual core 2,4 GHz CPU, 2Gb or RAM, 120GB SSD and a 1Gb Ethernet connection. For simplification, the network is considered wired, with a 1Gbps throughput running TCP/IP over a 1000BASE-T Ethernet LAN connection with an estimated 1ms end-to-end delay for all connections. The test system (TS) consists of a quad-core power pc with 2.4GHz CPU, 8Gb of RAM, 500GB SSD and 1Gb Ethernet connection. The TS is deployed in virtual containers running over a Unix environment with direct access to the network (non-virtualized network connection). The monitoring system resource consumption is considered negligible.

The second step is to select the KPIs and metrics of interest. The KPIs selected by the manufacturer are Capacity (max number of publish requests handled per second), responsiveness (average delay of processing client requests), number of registered subscribers, resource usage and stability. The Broker operates with QoS 1. The selected associated metrics are as follows:

- Publish delay: the time interval starts when a PUBLISH message is sent and ends when the corresponding PUBACK (QoS1) message has been received. Value expressed in milliseconds (ms).
- Subscriber Count: the maximum number of registered subscribers within a measurement window.
- Capacity: number of successfully handled Publish messages per second. Threshold is initially set to 1000
- Correctness: percentage of successfully handled Publish messages per second.
- Resource usage – amount of CPU, Memory used by the IuT during the Capacity evaluation. with a min, max and average values.

The third step consists of determining the tests and configuring the monitoring system. According to the KPI requirements, the type of tests required are load testing for determining the IuT Capacity and Resource usage and Endurance testing for determining the system Responsiveness, Correctness and Availability. The monitoring system is configured to record number of subscribers, PUBLISH delay, rate of success for PUBLISH messages, CPU, Mem and Network in/out usage. The monitoring window is set to 1 second and post-processing averages are configured to cover 1 minute for load tests and 1 hour for endurance tests.

*1) Load Testing*

1000 Clients connect to the IuT and subscribe to topics. Each client starts sending Publish messages at a rate of 1 message per second. The rate increases by 1 up to a maximum of 10 every minute. The test duration is set to 10 minutes and executed 10 times. The test is repeated for an incremented number of connected clients up to 5000 in incremental steps of 1000 clients. The input parameters are no longer incremented if the test fails. Test duration: 10 minutes per test.

Metrics:
- number of connected clients
- number of PUBLISH messages processed per second
- average PUBLISH messages processing delay
- rate of successfully processed PUBLISH messages
- CPU load user time %
- Memory load (Mb)
- Network Packet In (Kb)
- Network Packet Out (Kb)

Test Success Criteria
- Fail criteria: rate of successfully processed PUBLISH messages falls under 90%
- Fail criteria: CPU load goes over 80%
- Fail criteria: Memory load goes over 90%
- Pass criteria: Test ended without fail criteria triggered

*2) Endurance Testing*

Starting from the highest load successfully passed, the test parameters are noted with max X where X is the metric from the Load test. For example: max Clients connect to the broker and subscribe to topics.

Each client starts sending Publish messages at a rate of max message per second. The rate remains constant. The test duration is set to 600 minutes and executed 10 times. The test is repeated for an decremented number of connected clients down to 1000 in decremented steps of 1000 clients in case of failure. The input parameters are no longer decremented if the test succeeds. Test duration: 600 minutes per test.

Metrics:
- number of connected clients
- number of PUBLISH messages processed per second
- average PUBLISH messages processing delay
- rate of successfully processed PUBLISH messages
- CPU load user time %
- Memory load (Mb)
- Network Packet In (Kb)
- Network Packet Out (Kb)

Test Success Criteria:
- Fail criteria: rate of successfully processed PUBLISH messages falls under 90%
- Fail criteria: CPU load goes over 80%
- Fail criteria: Memory load goes over 90%
- Pass criteria: Test ended without fail criteria triggered.

The fourth step consists of executing the tests. As a general precondition: the SuT is operational – MQTT broker is active. TS is operational and connected to the SuT. Finally, the results are collected and the Powerfulness, Reliability and Efficiency selected KPI values are determined.

### C. Examples of Tests

A test should be presentable as a document, with accompanying data files, that provides a full description of an execution of a performance test on a test system. Description of the test case and objective of the test case, e.g., the definition of the targeted metrics should be contained therein. The following sections should be addressed in general:

- Test procedure: Description of the execution of the test
    - Test sequence to run the test case: sequence of actions for running the experiment and collect the measurements needed to compute the metrics
    - Test duration (per iteration).
    - Number of iterations of the experiment. Number of repetitions of the experiments to obtain relevant statistical results.
    - Measurements collected to compute the metrics.
- Procedure for metrics calculation
    - Description of the procedure applied (statistical aggregation, algorithm, etc. to compute the metrics based on the raw measurements collected.
- Expected output of the test case
    - Test report

An example recommended test report is presented in Table IV. Furthermore, a series of examples for load, endurance and stress tests examples with TDL-TO are presented in Tables V-VII.

TABLE V. TEST REPORT EXAMPLE

| **Test Number** | T-01 | **Test Category** | Performance | **Test Type** | Load Testing |
|---|---|---|---|---|---|
| **Test Objective** | "Determine if the IUT(broker) can handle the given incremental load for a determined period of time without exceeding the delay threshold within a given acceptable message loss rate." | | | | |
| **Test Description** | Test Scenario 1<br>Test Case 1<br>Configuration 1: Against Mosca Server | | Test Scenario 1<br>Test Case 1<br>Configuration 2: Against Mosquitto Server | | Reference<br>2ms |
| **Preconditions** | the CLIENT having a MQTT_CONNECTION to the IUT | | | | |
| **Expected Behaviour** | ensure that {<br>when {<br>(.) at time point t1: the tester entity send multiple PUBLISH messages and assure the INCREMENTAL_RATE and<br>(!) during the INTERVAL after t1: the IUT entity receive multiple PUBLISH message containing<br>topic_name corresponding to TOPIC,<br>payload corresponding to RETAINED_MESSAGE;<br>}<br>then {<br>(!) INTERVAL after t1 :<br>the IUT entity assure and send the PUBACK messages and<br>the IUT entity assure the packet_loss_limit and<br>the IUT entity assure the DELAY;<br>} | | | | |
| **Output** | Average PUBLISH/PUBACK delay in ms (KPIx), | | | | |
| **Measurements** | Publish Success Rate | | Sequence Delay | | Reference |
| **Values TC1** | 100% | | 0.998ms | | 2 ms |
| **Values TC2** | 100% | | 0.93ms | | 2 ms |

TABLE IV. PERFORMANCE LOAD TEST PURPOSE EXAMPLE WITH TDL-TO

| **TP Id** | TP_MQTT_Performance_Broker_Endurance_003 |
|---|---|
| **Test Objective** | Determine if the IUT(broker) can handle the given incremental load for a determined period of time without exceeding the delay threshold within a given acceptable message loss rate. |
| **Reference** | [MQTT-3.1.2-9], [MQTT-3.1.4-1], [MQTT-3.2.2-6] |
| **PICS Selection** | PICS_BROKER_BASIC and PICS_BROKER_PERFORMANCE and PICS_CLIENT_BASIC |
| **Initial Conditions** | |
| with {<br>the CLIENT having a MQTT_CONNECTION to the IUT<br>} | |
| **Expected Behaviour** | |
| ensure that {<br>when {<br>(.) at time point t1:<br>the tester send multiple PUBLISH messages and assure the RATE and<br>(!) during the INTERVAL after t1:<br>the IUT receive multiple PUBLISH message containing<br>topic_name corresponding to TOPIC,<br>payload corresponding to RETAINED_MESSAGE;<br>}<br>then {<br>(!) INTERVAL after t1 :<br>the IUT assure and send the PUBACK messages and<br>the IUT assure the packet_loss_limit and<br>the IUT assure the DELAY<br>}<br>} | |
| **Final Conditions** | |
| | |

TABLE VII. PERFORMANCE ENDURANCE TEST PURPOSE EXAMPLE WITH TDL-TO

| **TP Id** | TP_MQTT_Performance_Broker_Endurance_003 |
|---|---|
| **Test Objective** | Determine if the IUT(broker) can handle the given incremental load for a determined period of time without exceeding the delay threshold within a given acceptable message loss rate. |
| **Reference** | [MQTT-3.1.2-9], [MQTT-3.1.4-1], [MQTT-3.2.2-6] |
| **PICS Selection** | PICS_BROKER_BASIC and PICS_BROKER_PERFORMANCE and PICS_CLIENT_BASIC |
| **Initial Conditions** | |
| with {<br>the CLIENT having a MQTT_CONNECTION to the IUT<br>} | |
| **Expected Behaviour** | |
| ensure that {<br>when {<br>(.) at time point t1:<br>the tester send multiple PUBLISH messages and assure the RATE and<br>(!) during the INTERVAL after t1:<br>the IUT receive multiple PUBLISH message containing<br>topic_name corresponding to TOPIC,<br>payload corresponding to RETAINED_MESSAGE;<br>}<br>then {<br>(!) INTERVAL after t1 :<br>the IUT assure and send the PUBACK messages and<br>the IUT assure the packet_loss_limit and<br>the IUT assure the DELAY<br>}<br>} | |
| **Final Conditions** | |

TABLE VI. PERFORMANCE STRESS TEST PURPOSE EXAMPLE WITH TDL-TO

| **TP Id** | TP_MQTT_Performance_Broker_Stress_003 |
|---|---|
| **Test Objective** | Determine if the IUT(broker) can handle the given spiking load for a determined period of time without exceeding the delay threshold within a given acceptable message loss rate. |
| **Reference** | [MQTT-3.1.2-9], [MQTT-3.1.4-1], [MQTT-3.2.2-6] |
| **PICS Selection** | PICS_BROKER_BASIC and PICS_BROKER_PERFORMANCE and PICS_CLIENT_BASIC |
| **Initial Conditions** | |
| with {<br>the CLIENT having a MQTT_CONNECTION to the IUT<br>} | |
| **Expected Behaviour** | |
| ensure that {<br>when {<br>(.) at time point t1:<br>the tester send multiple PUBLISH messages and assure the RATE and<br>(!) during the INTERVAL after t1:<br>the IUT receive multiple PUBLISH message containing<br>topic_name corresponding to TOPIC,<br>payload corresponding to RETAINED_MESSAGE; and<br>(.) at time point t2:<br>the tester send multiple PUBLISH messages and assure the SPIKE_RATE and<br>(!) during the INTERVAL after t1: the IUT receive multiple PUBLISH message containing<br>topic_name corresponding to TOPIC,<br>payload corresponding to RETAINED_MESSAGE;<br>}<br>then {<br>(!) INTERVAL after t2 :<br>the IUT assure and send the PUBACK messages and<br>the IUT assure the packet_loss_limit and<br>the IUT assure the DELAY<br>}<br>} | |
| **Final Conditions** | |

## VI. Conclusions and Future Work

In this paper we described a performance evaluation benchmarking methodology for MQTT broker implementations. The proof of concept implementation was performed using the Eclipse Titan framework within the IoT Testware open source project. In addition, the work presented herein is also part of standardization efforts [33].

A benchmark example was provided using the described methodology. Additionally, test examples specified in TDL-TO and realized in TTCN-3 Eclipse Titan were presented. Future work will attempt to use this methodology on a real-life production system and disseminate the results.

## Acknowledgment

The work is supported by the experts from the ETSI TC MTS working group TDL and TST. The work on IoT test purposes has been partly funded by the German Federal Ministry of Economics and Technology with its project IoT-T. The authors appreciate the Eclipse foundation, in particular the members of the IoT-Testware and Titan projects for their contributions.

## References

[1] Rayes A., Salam S. (2019) “IoT Vertical Markets and Connected Ecosystems”. In: Internet of Things From Hype to Reality. Springer, https://doi.org/10.1007/978-3-319-99516-8_9

[2] [Online] Statista Research Department, (May , 2020) “Number of publicly known Internet of Things (IoT) platforms worldwide from 2015 to 2019” Available: https://www.statista.com/statistics/1101483/global-number-iot-platform/

[3] A. Al-Fuqaha, A. Khreishah, M. Guizani, A. Rayes and M. Mohammadi, "Toward better horizontal integration among IoT services," in IEEE Communications Magazine, vol. 53, no. 9, pp. 72-79, September 2015, doi: 10.1109/MCOM.2015.7263375.

[4] [Online] Andrew Foster (PrismTech Whitepaper) “Messaging Technologies for the Industrial Internet and the Internet of Things”, 2014. Available: https://www.smartindustry.com/assets/Uploads/SI-WP-Prismtech-Messaging-Tech.pdf

[5] N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," 2017 IEEE International Systems Engineering Symposium (ISSE), Vienna, 2017, pp. 1-7, doi: 10.1109/SysEng.2017.8088251.

[6] J. Ramirez and C. Pedraza, "Performance analysis of communication protocols for Internet of things platforms," 2017 IEEE Colombian Conference on Communications and Computing (COLCOM), Cartagena, 2017, pp. 1-7, doi: 10.1109/ColComCon.2017.8088198.

[7] S. N. Swamy, D. Jadhav and N. Kulkarni, "Security threats in the application layer in IOT applications," 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Palladam, 2017, pp. 477-480, doi: 10.1109/I-SMAC.2017.8058395.

[8] M. B. Yassein, M. Q. Shatnawi and D. Al-zoubi, "Application layer protocols for the Internet of Things: A survey," 2016 International Conference on Engineering & MIS (ICEMIS), Agadir, 2016, pp. 1-4, doi: 10.1109/ICEMIS.2016.7745303.

[9] L. Nastase, "Security in the Internet of Things: A Survey on Application Layer Protocols," 2017 21st International Conference on Control Systems and Computer Science (CSCS), Bucharest, 2017, pp. 659-666, doi: 10.1109/CSCS.2017.101.

[10] Karagiannis, Vasileios & Chatzimisios, Periklis & Vázquez-Gallego, Francisco & Alonso-Zarate, J.. (2015). A survey on application layer protocols for the Internet of Things. Trans. IoT Cloud Comput.. 3. 11-17..

[11] Masek, P., Hosek, J., Zeman, K., Stusek, M., Kovac, D., Cika, P., … Kröpfl, F. (2016). Implementation of True IoT Vision: Survey on Enabling Protocols and Hands-On Experience. International Journal of Distributed Sensor Networks. https://doi.org/10.1155/2016/8160282.

[12] Jasenka Dizdarević, Francisco Carpio, Admela Jukan, and Xavi Masip-Bruin. 2019. A Survey of Communication Protocols for Internet of Things and Related Challenges of Fog and Cloud Computing Integration. ACM Comput. Surv. 51, 6, Article 116 (February 2019). https://doi.org/10.1145/3292674

[13] Tara Salman, Raj Jain, ”A Survey of Protocols and Standards for Internet of Things”, Advanced Computing and Communications, Vol. 1, No. 1, March 2017, arXiv:1903.11549

[14] OASIS, “MQTT Version 3.1.1,” 2014, Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html, (visited Nov,2019)

[15] OASIS, “MQTT Version 5.0”, 2019 . Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html, (visited Nov, 2019 )

[16] M. Houimli, L. Kahloul and S. Benaoun, "Formal specification, verification and evaluation of the MQTT protocol in the Internet of Things," 2017 International Conference on Mathematics and Information Technology (ICMIT), Adrar, 2017, pp. 214-221, doi: 10.1109/MATHIT.2017.8259720.

[17] E. Bertrand-Martínez, P. Feio, V. Nascimento, B. Pinheiro, A. Abelém: A Methodology for Classification and Evaluation of IoT Brokers, IFIP 2019, DOI: 978-3-903176-23-2

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

[19] [Online] Eclipse IoT-testware: Available:https://projects.eclipse.org/projects/technology.iottestware

[20] [Online] IoT-T project, Available: https://www.iot-t.de/

[21] I. Schieferdecker, S. Kretzschmann, A. Rennoch, M. Wagner: IoT-Testware – an Eclipse project. IEEE QRS 2017, DOI 10.1109/QRS.2017.59

[22] A. Kaiser, S. Hackel: Standards-Based IoT Testing with Open-Source Test Equipment. IEEE QRS 2019, DOI 10.1109/QRS-C.2019.00085

[23] ETSI Test Description Language (TDL) standards (ES 203 119): Available: https://tdl.etsi.org/index.php/downloads

[24] ETSI TPLan: A notation for expressing Test Purposes. Available: https://portal.etsi.org/Services/Centre-for-Testing-Interoperability/ETSI-Approach/Testing-Languages/TPLAN

[25] ETSI TR 103 119, Methods for Testing and Specification (MTS); The Test Description Language (TDL); Reference Implementation

[26] ETSI TTCN-3 standards (ES 201 873) Available: http://www.ttcn-3.org/index.php/downloads/standards

[27] [Online] Eclipse Titan: https://projects.eclipse.org/projects/tools.titan

[28] ETSI TR 103 119, Methods for Testing and Specification (MTS); The Test Description Language (TDL); Reference Implementation

[29] ETSI Test Specification for MQTT; Part 1: Conformance Tests https://portal.etsi.org/webapp/WorkProgram/Report_WorkItem.asp?WKI_ID=54401

[30] ETSI Test Specification for MQTT; Part 2: Security Tests https://portal.etsi.org/webapp/WorkProgram/Report_WorkItem.asp?WKI_ID=54410

[31] ETSI TR 101 577: Methods for Testing and Specifications (MTS); Performance Testing of Distributed Systems; Concepts and Terminology

[32] RFC2544: Benchmarking Methodology for Network Interconnect Devices

[33] ETSI Test Specification for MQTT; Part 3: Performance Tests Available: https://portal.etsi.org/webapp/WorkProgram/Report_WorkItem.asp?wki_id=54411