

**BUIDING A HA MQTT BROKERAGE SOLUTION USING MOSQUITTO**

**BY**

**WONG KEI YIN**

**A REPORT**

**SUBMITTED TO**

**Universiti Tunku Abdul Rahman**

**in partial fulfillment of the requirements**

**for the degree of**

**BACHELOR OF COMPUTER SCIENCE (HONOURS)**

**Faculty of Information and Communication Technology  
(Kampar Campus)**

**MAY 2021**

## REPORT STATUS DECLARATION FORM


**Title:** BUIDING A HA MQTT BROKERAGE SOLUTION USING  
MOSQUITTO  
\_\_\_\_\_

**Academic Session:** \_\_\_\_\_MAY 2021\_\_\_\_\_

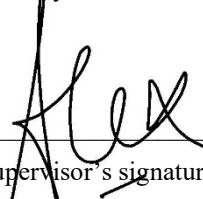
I \_\_\_\_\_WONG KEI YIN\_\_\_\_\_  
**(CAPITAL LETTER)**

declare that I allow this Final Year Project Report to be kept in  
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

  
\_\_\_\_\_  
(Author's signature)

Verified by,

  
\_\_\_\_\_  
(Supervisor's signature)

**Address:**

\_\_\_\_\_132, Jalan Suasa 5, 31900\_\_\_\_\_  
\_\_\_\_\_Perak\_\_\_\_\_  
\_\_\_\_\_

\_\_\_\_\_TS DR OOI BOON YAIK\_\_\_\_\_  
Supervisor's name

**Date:** 1 September 2021

**Date:** 2 September 2021

<b>Universiti Tunku Abdul Rahman</b>			
Form Title : <b>Sample of Submission Sheet for FYP/Dissertation/Thesis</b>			
Form Number: <b>FM-IAD-004</b>	Rev No.: <b>0</b>	Effective Date: <b>21 JUNE 2011</b>	Page No.: <b>1 of 1</b>

## **FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY**

1 September 2021 **UNIVERSITI TUNKU ABDUL RAHMAN**

Date: 1 September 2021

### **SUBMISSION OF FINAL YEAR PROJECT /DISSERTATION/THESIS**

It is hereby certified that WONG KEI YIN (ID No: 19ACB00582) has completed this final year project entitled “BUIDING A HA MQTT BROKERAGE SOLUTION USING MOSQUITTO” under the supervision of TS DR OOI BOON YAIK (Supervisor) from the Department of Computer Science, Faculty of Information and Communication Technology.

I understand that University will upload softcopy of my final year project in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,



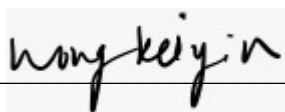

---

(WONG KEI YIN)

## DECLARATION OF ORIGINALITY

I declare that this report entitled “**BUIDING A HA MQTT BROKERAGE SOLUTION USING MOSQUITTO**” is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature :



Name :

Wong Kei Yin

Date :

1 September 2021

## **ACKNOWLEDGEMENTS**

I would like to express my sincere thanks and appreciation to my supervisors, Ts Dr Ooi Boon Yaik who have given me a golden opportunity to involve in the Internet of Things field study. It is my first step to establish a career in the software development field. A million thanks to you

Finally, a million thanks to my family and parents who have given me all the support I need throughout this project

## **ABSTRACT**

Message Queuing Telemetry Transport (MQTT) is a protocol that has been widely used by IoTs because of it has less bandwidth requirement, lightweight and suitable for unreliable connection. It is a publish/subscribed pattern-based protocol. Mosquitto is one of the famous message brokers that implement MQTT protocol. Due to MQTT brokered architecture, typically broker is a single point of the failure. Single broker may spend more time to respond, and performance is affected when high traffic. In this context, high availability refers to ability of system to be continuously operational and also quick response time to user request. Other message brokers such as HiveMQ and Bevywise brokers are support clustering, but not the Mosquitto. Without clustering, there is no high availability solution for Mosquitto. In this project, two high availability solutions have been proposed for Mosquitto. By using the solution, Mosquitto brokers work together and can be view as a single broker. If one of the brokers fail, the remaining broker can cover the same task. In order to develop this solution successfully, the PPDIIO Methodology is being adopted, which is a commonly used methodology for network design.

## **Table of Contents**

<b>TITLE PAGE</b>	<b>i</b>
<b>REPORT STATUS DECLARATION FORM</b>	<b>ii</b>
<b>FYP THESIS SUBMISSION FORM</b>	<b>iii</b>
<b>DECLARATION OF ORIGINALITY</b>	<b>iv</b>
<b>ACKNOWLEDGEMENTS</b>	<b>v</b>
<b>ABSTRACT</b>	<b>vi</b>
<b>TABLE OF CONTENTS</b>	<b>vii</b>
<b>LIST OF FIGURES</b>	<b>ix</b>
<b>LIST OF TABLES</b>	<b>xi</b>
<b>LIST OF ABBREVIATIONS</b>	<b>xii</b>
<b>CHAPTER 1 INTRODUCTION</b>	<b>1</b>
1.1 Problem Statement and Motivation	2
1.2 Project Scope	4
1.3 Project Objectives	4
1.4 Contribution	4
1.5 Background Information	5
<b>CHAPTER 2 LITERATURE REVIEW</b>	<b>6</b>
2.1 MQTT	6
2.1.1 Publish/Subscribe Pattern	6
2.1.2 Topic and Quality of Service	7
2.2 High availability	8
2.3 High availability MQTT clusters using HiveMQ	9
<b>CHAPTER 3 SYSTEM DESIGN AND CONFIGURATION</b>	<b>11</b>
3.1 System Overview	11

3.2 HA Mosquitto broker using Mosquitto bridge	11
3.2.1 Implementation of HA Mosquitto broker using Mosquitto bridge	12
3.3 HA Mosquitto broker using python bridge	18
3.3.1 Overview of python bridge	18
3.3.2 Detailed Callback Function Flowcharts	21
3.5.3 Configuration of HA Mosquitto broker using python bridge	27
<b>CHAPTER 4 METHODOLOGY AND TOOLS</b>	<b>31</b>
4.1 Methodology	31
4.2 Technologies and Tools Involved	32
<b>CHAPTER 5 TESTING</b>	<b>34</b>
5.1 Performance testing	34
5.1.1 Performance testing configuration	35
5.2 Network bandwidth consumption testing	36
5.2.1 Network bandwidth consumption testing configuration	36
5.3 Comparison Results	38
5.3.1 Performance comparison results	38
5.3.2 Network bandwidth consumption comparison results	39
5.4 Result Discussion	39
<b>CHAPTER 6 CONCLUSION AND FUTURE WORK</b>	<b>41</b>
6.1 Conclusion	41
6.2 Future Work	41
<b>REFERENCES</b>	<b>43</b>
<b>WEEKLY LOG</b>	<b>45</b>
<b>POSTER</b>	<b>49</b>
<b>PLAGIARISM CHECK RESULT</b>	<b>50</b>
<b>FYP2 CHECKLIST</b>	<b>53</b>



## LIST OF FIGURES

<b>Figure Number</b>	<b>Title</b>	<b>Page</b>
Figure 2.1	MQTT Publish/Subscribe Architecture	6
Figure 2.2	MQTT topic structure	7
Figure 2.3	High availability structure	9
Figure 2.4	HiveMQ cluster with load balancer	10
Figure 3.1	Network design of using Mosquitto bridge	11
Figure 3.2	Running Mosquitto on broker 1	13
Figure 3.3	Running Mosquitto on broker 2	15
Figure 3.4	Network design of using python bridge	18
Figure 3.5	on_message_forward() Function Flowchart	21
Figure 3.6	on_message_forward_from_otherbroker() Function Flowchart	22
Figure 3.7	on_message_sub_top() Function Flowchart	23
Figure 3.8	on_message_unsub_top() Function Flowchart	24
Figure 3.9	on_message_disconn_topic() Function Flowchart	25
Figure 3.10	on_message_get_topic() Function Flowchart	26
Figure 3.11	Running Mosquitto in broker 1	28
Figure 3.12	Running python bridge on broker 1	28
Figure 3.13	Running Mosquitto in broker 2	29
Figure 3.14	Running python bridge on broker 2	30
Figure 4.1	Phases of PPIDOO	31
Figure 5.1	Figure 5.1 Network design for HA brokers performance testing	34
Figure 5.2	Network design for single broker performance testing	35
Figure 5.3	Network design for HA brokers network bandwidth consumption testing	36

Figure 5.4	Interface of iftop	37
Figure 5.5	Msg time mean mean (msg/sec) comparison result	38
Figure 5.6	Average bandwidth (msg/sec) comparison result	39
Figure 5.7	Network bandwidth consumption (MB) between brokers	39

## LIST OF TABLES

<b>Table Number</b>	<b>Title</b>	<b>Page</b>
Table 3.1	Callback functions for python bridge	20
Table 1.1	Brief description of each phase in PPDIOO	32
Table 4.2	Specification of Desktop	32
Table 4.3	Specification of virtual machine	33

## LIST OF ABBREVIATIONS

<i>IoT</i>	Internet of Things
<i>MQTT</i>	Message Queuing Telemetry Transport
<i>HA</i>	High Availability
<i>MEP</i>	Message Exchange Patterns
<i>QoS</i>	Quality of Services
AMQP	Advanced Message Queuing Protocol
HTTP	Hyper Text Transfer Protocol
VRRP	Virtual Router Redundancy Protocol

## CHAPTER 1 INTRODUCTION

The term Internet of Things (IoT) is a collective term for the number of electronic devices that are able to connect to the Internet, capable of sending data, receiving instructions, or both, was first proposed by British technical experts (Fruhlinger, 2020). The use of IoT has increased significantly because of the evolution of multiple technologies such as sensor network, wireless connection, and low power electronics over the last few years. IoTs connect the physical and digital worlds by bringing the internet's power, data processing, and analytics to the real world of objects. According to Fruhlinger (2020), there are more than 50 billion IoT devices as of 2020, and those devices will generate 4.4 zettabytes of data. IoTs can communicate to each other through the internet, it means that they can be monitored and controlled remotely. Therefore, IoTs infiltrate various fields such as industrial automation, smart cities, and etc. Apart from it, a myriad of IoT protocols have been proposed for communication between IoT devices. MQTT (Message Queuing Telemetry Transport) has been widely used because of it has less bandwidth requirement, lightweight and suitable for unreliable connection.

Compared to other protocols such as HTTP protocols and Advanced Message Queuing Protocol (AMQP), MQTT is a lightweight publish/subscribe protocol for IoT and It can be used on restricted devices as well as high-latency networks. HTTP is a synchronous protocol, and it applied the request and response pattern. Client always need to wait for the server to respond and result in poor scalability. In IoT worlds, the high latency and low bandwidth network always make the synchronous communication problematic. Moreover, HTTP also is a heavy weight protocol that include many headers and rules which make expensive to broadcast messages to all IoTs over the network. The synchronous messaging protocol is far more suitable for IoT applications than the synchronous protocol. Besides HTTP protocols, AMQP is a binary TCP-based protocol that can use either a publish/subscribed model or a request/pattern (Solovev & Petrova, 2020). At the aspect of memory and power consumption, AMQP consumes more memory and power than MQTT because of AMQP has more complex messaging system. Unfortunately, it is not suitable for resource constrained IoT applications although it has a lot of features.

The key feature that makes MQTT protocol lightweight and flexible is its publish-subscribe model. Brokers and clients are the two types of network entities that make up the MQTT protocol. (IBM, 2017). Clients connect to the broker and subscribe to or publish data on specific topics. When a topic receives a new message, the broker will send the message to all devices that have been connected to that topic. The responsibility of broker is receiving all messages, filtering the messages, classifying the subscribers based on the topic and publishing the message to them. Depending on implementation, thousands of MQTT clients may connect to a broker concurrently. Due to broker to handle various tasks such as filtering, publishing, therefore high availability is significant for the broker. Any component that without redundancy is considered as a single point of failure. The goals of high availability are to eliminate the single point of failure in the system and handle increased loads and high levels of traffic. (Digital Ocean, 2016).

Mosquitto is a lightweight MQTT broker written in C compared to others and it is famous and open source. Unfortunately, Mosquitto does not support clustering, it makes high availability difficult. This work proposed high-availability solutions for MQTT brokers using the Mosquitto, which aim to improve the performance of Mosquitto brokers and provide redundancy for Mosquitto brokers. This paper organized as follows. The first chapter contains a brief explanation of MQTT, the problem of Mosquitto, and the objectives of the project. Chapter 2 is about the definition of high availability, the detail explanation of MQTT protocol and the existing high availability solution using other broker. The next few chapters are about the proposed solution and the configuration, methodology, and testing result. The last chapter concludes this paper and talk about the future improvement.

## **1.1 Problem Statement and Motivation**

1. No cluster functionality built-in for Mosquitto to achieve high availability

Unlike other brokers such as HiveMQ, Mosquitto does not support clustering (Tomosvari, 2017). High availability can be achieved through architecture of cluster. For example, HiveMQ can form MQTT broker clusters through its built-in cluster functionality to eliminate the single point of failure (HiveMQ, n.d.). With cluster functionality, HiveMQ broker allows MQTT clients

reconnect to any HiveMQ cluster node and can resume their MQTT session. HiveMQ cluster also has sophisticated and very efficient message routing between brokers. However, due to Mosquitto does not support clustering, it does not handle high availability in it (Kumar, 2020).

2. The period of downtime can cause negative impact

High availability is essential for any organization to against the lost cause by service outage. However, no matter of how reliable the system and software are, situation like power outage and equipment failure that can bring down the servers are inevitable. A single MQTT broker could be a single point of failure, causing service disruption if it went down. Without high availability, when the broker goes down, all messages cannot be forward, and it may cause the business processes that depend on the broker stop. For business, short period of downtime can cause negative impacts such as business brand reputation will be damaged, loss of trust among customer and etc.

3. Performance of single broker may be affected when high traffic

Depending on implementation, a broker can support up to a thousand concurrent connected MQTT clients. It is critical for broker able to handle increased loads and high traffic. A single broker may spend more time to respond or even inaccessible when high traffic. Instead of a single broker, some implementation such as multiple brokers with load balancing may increase reliability, maximize throughput, minimize respond time and avoid overload. Without high availability, MQTT broker cannot provide optimal performance during the period of high traffic.

Although failover can be implemented on broker, but without cluster functionality, redundant broker cannot work together with the original broker to optimize performance. The key motivation of this project is to provide high availability solution for Mosquitto brokers because of Mosquito does not support clustering to achieve high availability.

## 1.2 Project Scope

This project is about developing solutions to achieve high availability MQTT brokerage using Mosquitto. In this project, high availability refers to the capability to recover from unexpected events in the shortest time possible and quick response time to users' requests. The reliability (of hardware and software components) and performance (response-time) are parts of system availability. The previous work done by HiveMQ team was using the HiveMQ brokers to build high availability environment because HiveMQ broker support cluster. However, Mosquitto does not support cluster. Bridge is a technique which allows two MQTT brokers connect together, generally used for sharing messages between system. This study proposed two methods for Mosquitto broker to achieve high availability. The first method is using Mosquitto bridge, the second method is using python script provided by this study.

## 1.3 Project Objectives

The objectives of this project be summarized as below:

1. Create redundant broker that able to cover same task and eliminate single point of failure.
2. Use bridge to connect original and redundant broker together for messages sharing.
3. Efficiently distribute incoming traffic across MQTT brokers to minimize response time.

This project mainly focuses on providing high availability for the Mosquitto. The security for Mosquitto broker did not cover in this project. For example, this project did not configure the authentication and authorization for Mosquitto broker. it allows anonymous client to connect to the broker.

## 1.4 Contribution

With the rapid development of science and technology, IoT technology is becoming more accessible, allowing a wider range of businesses to benefit from IoT applications. The lightweight features make MQTT has been widely used in IoT world. There are many types of public and private brokers available by different vendors, one of the



famous brokers is Mosquitto. Due to the MQTT architecture, MQTT system typically have a single point of failure which is broker. Broker that does not support clustering such as Mosquitto will not be able to provide high availability. Through the solutions provided by this project, user able to build Mosquitto brokers with high availability. This project not only create redundant brokers but also allows Mosquitto brokers connect and work together to optimize the performance without clustering. By using the solution, Mosquitto brokers able to continuously work without any interruption and able to handle the load when high traffic, minimize the request response time and provide better experience to the user.

### **1.5 Background Information**

Before proceeding, there are a few terms need to be explained here. MQTT clusters is distributed system that represent a logical MQTT brokers. MQTT broker nodes are installed on different physical machine and connected over the internet. From client's perspective, a cluster of brokers behaves like a single broker. MQTT cluster able to eliminate the single point of failure since multiple brokers acts as a single broker. The message can be distributed across brokers by using clustering.

Bridging is one of the Mosquitto feature which basically let us connect two or more brokers together. It is supporting multiple connections to share about the publish/subscribe each topic. For example, broker 1 and broker 2 have configured bridge, message that publish on broker 1 will be distributed to broker 2 via bridge. Any client that subscribed to the broker 2 will get the message as well. Bridging is a main technology used by this project to replace the clustering.

Load balancing is the technique of distributing network or application traffic among multiple servers in a server farm in an efficient manner. By distributing traffic to all devices, it can reduce the load on a single device and improves application responsiveness. HA proxy is one of the software that offering load balancing.

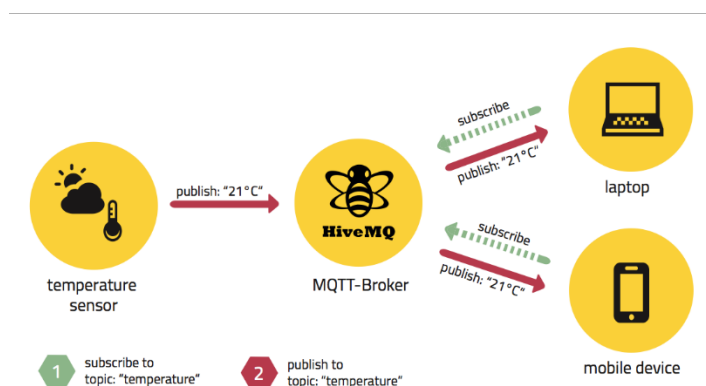
## CHAPTER 2 LITERATURE REVIEW

### 2.1 MQTT

MQTT protocol was originally created by IBM in 1999 (IBM, 2017). It was developed for minimal bandwidth to connect with oil pipelines via satellite. In late 2014, MQTT became an officially approved OASIS open standard (IBM, 2017). Despite the fact that it was designed for remote site communication, it has been widely adopted for IoT because it is suitable for wireless network that experience various level of latency due to bandwidth constraints or unreliable network.

#### 2.1.1 Publish/Subscribe Pattern

Message exchange patterns (MEP) are a set of templates that define how messages should be exchanged. (Flylib, n.d.). The most popular MEP has been widely used is request/response pattern that uses a synchronous communication model for communication. Another common pattern is publish/subscribe pattern. The publish/subscribe pattern provides an alternative to traditional request/response model. The publish/subscribe pattern decouples the data senders (publishers) and receivers (subscribers) to facilitate distribution of messages to subscribers. The connection between them through a third component which is broker. The publishers and subscribers never contact to each other directly. The MQTT protocol utilises broker to exchange messages between clients based on the publish/subscribe pattern. MQTT protocol surround three subject which are publisher, broker, and subscriber as shown in Figure 1.



*Figure 2.1 MQTT Publish/Subscribe Architecture (Eclipse foundation, 2014)*

The publishers generate and broadcast message on the particular topic to all of the topic's subscribers through the brokers. The central communication point is the MQTT broker, in charge of filtering all incoming message, decide who is interested in them and distributing them correctly to all subscribed clients (Eclipse foundation 2014). When a device send data to broker, it is called publish. When the operation is reversed, it is called subscribe. The advantage of this pattern is the decoupling of the publisher message from the subscriber. The decoupling can be broker down into the following three dimensions shown below (Eugster, Felber, Guerraoui, & Kermarrec, 2003):

- **Space decoupling:** The interacting parties which are data sender (publisher) and data receiver (subscriber) do not know each other because there is a broker between them.
- **Time decoupling:** The interacting parties which are publisher and subscriber do not to be active at the same time.
- **Synchronization decoupling:** publishers are not blocked while producing events, and subscribers can get asynchronously notified (through a callback) of the occurrence of an event while performing some concurrent activity.

### 2.1.2 Topic and Quality of Service

MQTT broker plays a vital role because it filters all the message so that every subscriber only receives the message of interest. To route the messages to relevant subscribers, MQTT used subject-based filtering. Every message contains a topic. Subject-based filtering is based on subject or topic, the brokers ensures that the subscriber gets all the message published to the subscribed topics. A topic is a simple string that can consists of one or more hierarchy level. Each topic level is separated by a slash. The Figures 2 shows the MQTT topic structure.



*Figure 2.2 MQTT topic structure (STS MQTT n.d.)*

In MQTT, it also supports wildcards to subscribe to multiple topics simultaneously. For example, multilevel wildcard (#) allows to subscribe entire subtree.

MQTT provides quality level of service, which is called QoS. MQTT protocol supports three levels of QoS (Eclipse foundation 2014). Level 0 message does not have guarantee at all, is sent only once, no acknowledgement from receiver. Level 1 ensures the messages are delivered at least once at the arrive. Receiver can receive the message multiple times. Level 2 is the highest QoS provided in MQTT. It ensures that the exactly once message arrives the destination, avoided from duplication of message sent.

## **2.2 High availability**

Availability is divided into two parts: how long a service is available and how long it takes the system to respond to user request. When high traffic, the server needs to take a long time responds to the user request or become inaccessible, then it does not consider as high availability. High availability refers to those systems that can operate continuously without fail and provide a high level of operational performance, are commonly equipped with redundant components. s (AVI Networks, n.d). The impact of unexpected incidents that can bring the servers down such as hardware and software failures, environmental anomaly can be mitigated via high availability. We must identify and eliminate single points of failure in the system to achieve high availability. (Digital Ocean 2016). To eliminate single points of failure, there are two things needed

1. Redundant components can cover the same task.
2. The mechanism that is able to detect failures in the components such as load balancer.

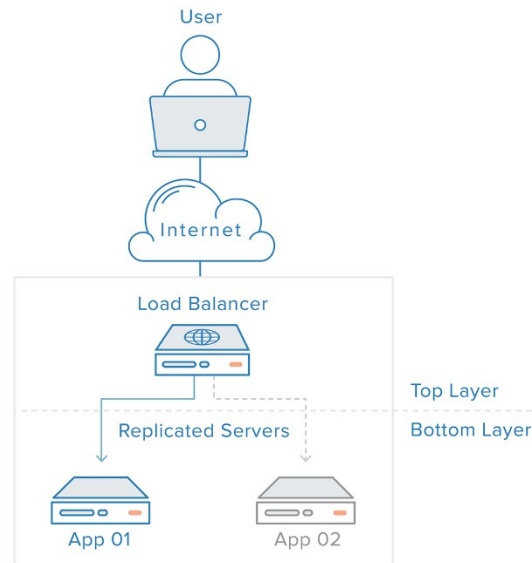
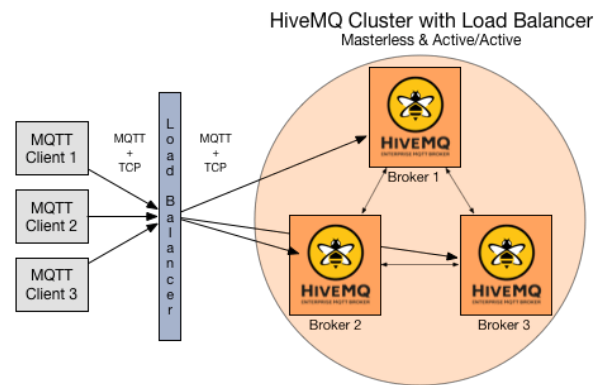


Figure 2.3 High availability structure (Digital Ocean 2016)

Imagine that there are two redundant servers behind the load balancer. The traffic coming from clients can be spread evenly among the servers. It can provide optimal performance if servers can work together. When one of them goes down, traffic will be redirected by load balancer to remaining online server, eliminate single of points failure. The Figure 3 shows the high availability architecture.

### 2.3 High availability MQTT clusters using HiveMQ

One of the unique features of HiveMQ is its cluster ability to form an MQTT broker cluster (HiveMQ, n.d.). A MQTT broker clusters is a distributed system that one behalf of one logical MQTT broker. From client's perspective, it can be viewed as a single MQTT broker. Due to HiveMQ come with cluster functionality built-in, HiveMQ team (2016) suggested to build a high availability cluster environment by using a TCP load balancer and the HiveMQ broker cluster. Figure 4 shows the HiveMQ cluster with load balancer.



*Figure 2.4 HiveMQ cluster with load balancer. (The HiveMQ Team, 2016)*

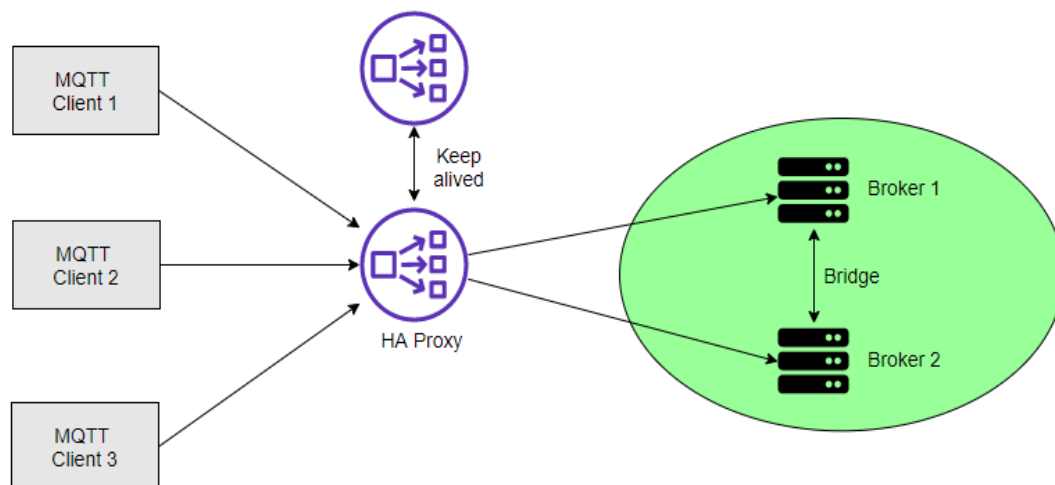
MQTT broker cluster consists of different MQTT broker nodes that connected over a network. Load balancer are also used together in this architecture. HiveMQ cluster is clever for distributing messages across the clusters. The message only gets distributed among the cluster nodes, if necessary, it can prevent nodes from forwarding unnecessary messages. Compare to bridging, cluster subscriptions work dynamically. By using cluster, it can eliminate the single point of failure since multiple brokers act as single broker. For example, if one of the brokers goes down, the load balancer will automatically transfer all incoming traffic to the active broker.

## CHAPTER 3 SYSTEM DESIGN AND CONFIGURATION

### 3.1 System Overview

In this project, two high availability methods have been proposed for the Mosquitto brokers. The first method is using Mosquitto bridge to forward messages within brokers. By using Mosquitto bridge, any message received by broker will be sent to the other broker. Even though nobody subscribes to the topic, the message also will be forwarded to the other broker. Therefore, the second method is using the python bridge that is provided by this study to replace the Mosquitto bridge. The python bridge able to determine whether forward the message to the other broker. It can significantly reduce the network bandwidth consumption between brokers.

### 3.2 HA Mosquitto broker using Mosquitto bridge



*Figure 3.1 Network design of using Mosquitto bridge*

Figure 3.1 describes the network design of using Mosquitto bridge. Broker 2 is a redundant broker in order to eliminate single point of failure. Both of the brokers are connected using the Mosquitto bridge to forward message within brokers. The bridge connects two brokers together, to share about the publish/subscribe message of each topic.

HA proxy is used in the project to act as a load balancer. HA proxy dispatch request to the different brokers according to the simple algorithm round robin and make services of broker to appear as a virtual service on single IP address. If one of the brokers down,

HA proxy able to detect the failed broker and redirect to remaining online broker. In case HA proxy become a single point of the failure, redundant HA proxy are also used to cover the task when the HA proxy failed. Keepalived are configured to enable failover between two HA proxy. If the primary HA proxy goes down, the floating IP will be moved to the second HA proxy automatically.

### 3.2.1 Implementation of HA Mosquitto broker using Mosquitto bridge

Brokers and HA proxies are implemented on different virtual machine with Kali Linux OS respectively. Four virtual machines are used in this project for two brokers and two HA proxies.

#### **Broker 1 Configuration**

##### 1. Installation

Install Mosquitto on the virtual machine. This is a package for Mosquitto broker.

```
$ sudo apt-get install mosquitto
```

##### 2. Mosquitto Configuration

Change to directory `/etc/mosquitto/` and create a new mosquitto configuration file called `mosquitto_bridge.conf`. To create a mosquitto configuration file, administrative privilege is needed.

```
$ cd /etc/mosquitto/  
$ sudo nano /etc/mosquitto/mosquitto_bridge.conf
```

Follow the configuration file below,

```
1 #Place your local configuration in /etc/mosquitto/conf.d/  
2 #  
3 # A full description of the configuration file is at  
4 # /usr/share/doc/mosquitto/examples/mosquitto.conf.example  
5  
6 #pid_file /run/mosquitto/mosquitto.pid  
7  
8 #persistence true  
9 #persistence_location /var/lib/mosquitto/  
10  
11 #log_dest file /var/log/mosquitto/mosquitto.log  
12
```



```
13 #include_dir /etc/mosquitto/conf.d
14
15 listener 1883 0.0.0.0
16 allow_anonymous true
17
18 log_type all
19 log_dest topic
20 log_dest stdout
```

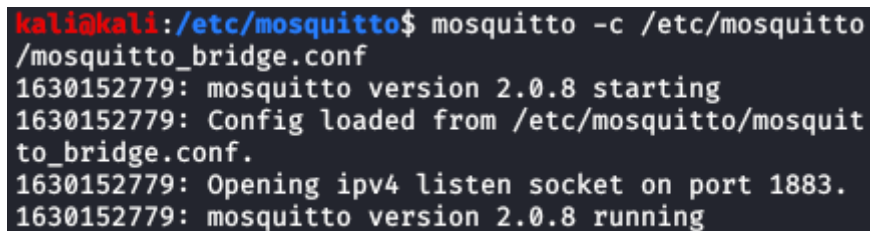
*mosquitto\_bridge.conf*

The network ports that mosquitto listens on can be controlled using *listener*. In the configuration file, broker 1 listen all incoming network connection (0.0.0.0) at port 1883. It sends all log messages to two destinations which are console and topic using *log\_dest*. Broker 1 does not have the bridge configuration because only one of the brokers need to be configured to act as bridge. Bridge is configured at broker 2.

### 3. Run the Mosquitto

Run the command line below to start the Mosquitto broker. `-c` is used to specify the configuration file for Mosquitto broker.

```
$ mosquitto -c /etc/mosquitto/mosquitto_bridge.conf
```



```
kali@kali:/etc/mosquitto$ mosquitto -c /etc/mosquitto
/mosquitto_bridge.conf
1630152779: mosquitto version 2.0.8 starting
1630152779: Config loaded from /etc/mosquitto/mosquit
to_bridge.conf.
1630152779: Opening ipv4 listen socket on port 1883.
1630152779: mosquitto version 2.0.8 running
```

*Figure 3.2 Running Mosquitto on broker 1*

## **Broker 2 Configuration**

### 1. Installation

Install Mosquitto on the virtual machine.

```
$ sudo apt-get install mosquitto
```

## 2. Mosquitto Configuration

Change to directory `/etc/mosquitto/` and create a new mosquitto configuration file called `mosquitto_bridge.conf`. To create a mosquitto configuration file, administrative privilege is needed.

```
$ cd /etc/mosquitto/
$ sudo nano /etc/mosquitto/mosquitto_bridge.conf
```

Follow the configuration file below,

```
1 # Place your local configuration in /etc/mosquitto/conf.d/
2 #
3 # A full description of the configuration file is at
4 # /usr/share/doc/mosquitto/examples/mosquitto.conf.example
5
6 #pid_file /run/mosquitto/mosquitto.pid
7
8 #persistence true
9 #persistence_location /var/lib/mosquitto/
10
11 #log_dest file /var/log/mosquitto/mosquitto.log
12
13 listener 1883 0.0.0.0
14 allow_anonymous true
15
16 connection bridge-01
17 address 192.168.0.139
18 topic # out 2
19 topic # in 2
20
21 log_type all
22 log_dest topic
23 log_dest stdout
```

*mosquitto\_bridge.conf*

The red rectangle box is about the bridge setting. The **connection** is the variable marks the start of a new bridge connection. It is also used to give the bridge a name which is used as the client id on the remote broker. In the configuration, the bridge name is bridge-01. The **address** is the address that bridge to connect to, which is 192.168.0.139 in this project. The last two lines are about the topic and its pattern.

## 3. Start the Mosquitto

Run the command line below to start the Mosquitto broker. `-c` is used to specify the configuration file for Mosquitto broker.

```
$ mosquitto -c /etc/mosquitto/mosquitto_bridge.conf
```

```

kali@kali:/etc/mosquitto$ mosquitto -c mosquitto_bridge.conf
1630155542: mosquitto version 2.0.8 starting
1630155542: Config loaded from mosquitto_bridge.conf.
1630155542: Opening ipv4 listen socket on port 1883.
1630155542: Bridge local.kali.bridge-01 doing local SUBSCRIBE on topic #
1630155542: Connecting bridge (step 1) bridge-01 (192.168.0.139:1883)
1630155542: mosquitto version 2.0.8 running
1630155542: Connecting bridge (step 2) bridge-01 (192.168.0.139:1883)
1630155542: Bridge kali.bridge-01 sending CONNECT
1630155542: Received CONNACK on connection local.kali.bridge-01.
1630155542: Bridge local.kali.bridge-01 sending UNSUBSCRIBE (Mid: 2, Topic: #)
1630155542: Bridge local.kali.bridge-01 sending SUBSCRIBE (Mid: 3, Topic: #, QoS: 2, Options: 0x00)
1630155542: Received PUBACK from local.kali.bridge-01 (Mid: 1, RC:0)
1630155542: Received UNSUBACK from local.kali.bridge-01
1630155542: Received SUBACK from local.kali.bridge-01

```

Figure 3.3 Running Mosquitto on broker 2

### HA Proxy 1 Configuration

#### 1. Installation

Install keepalived and HA proxy in the virtual machine.

```

$ sudo apt-get install haproxy
$ sudo apt-get install keepalived

```

#### 2. HA proxy configuration

Change to directory */etc/haproxy/* and modify *haproxy.cfg*. To modify the *haproxy.cfg*, administrative privileges are needed.

```

$ cd /etc/haproxy/
$ sudo nano /etc/haproxy/haproxy.cfg

```

Add the following code to the *haproxy.cfg*

```

1 listen mqtt
2     bind *:1883
3     mode tcp
4     option tcplog
5     balance roundrobin
6     server broker1 192.168.0.139:1883 check
7     server broker2 192.168.0.107:1883 check

```

*haproxy.cfg*

HA proxy listens for all coming requests to port 1883, forwarding them to two Mosquitto brokers using the round robin algorithm. The *server* setting first argument is a name, followed by the IP address of Mosquitto broker. Each broker performs health checks by adding a check argument.

### 3. Keepalived configuration

Configure the *keepalived.conf* which is located at */etc/keepalived/keepalived.conf*. To modify the *keepalived.conf*, administrative privileges are needed. Create a *keepalived.conf* if no configuration file at */etc/keepalived/*.

```
$ sudo nano /etc/haproxy/keepalived.conf
```

Add the following code to the *keepalived.conf*

```
1 vrrp_instance p1 {
2     state MASTER
3     interface eth0
4     virtual_router_id 101
5     priority 100
6     virtual_ipaddress{
7         192.168.0.181
8     }
9 }
```

*keepalived.conf*

*Vrrp\_instance* describe the instance of the VRRP protocol. The *state* MASTER indicates active server. Interface defines the interface the VRRP runs on. The priority specifies the order in which the assigned interface takes over in a failover, the higher the number, the higher the priority. Finally, the *virtual\_ipaddress* specifies the floating IP address shared by two HA proxy which is 192.168.0.181.

### 4. Start the HA proxy and Keepalived

Run the command line below to start the HA proxy and Keepalived.

```
$ sudo service haproxy start
$ sudo service keepalived start
```

## **HA Proxy 2 Configuration**

### 1. Installation

Install keepalived and HA proxy in the virtual machine.

```
$ sudo apt-get install haproxy
$ sudo apt-get install keepalived
```

## 2. HA proxy configuration

Configure the `haproxy.conf` which is located at `/etc/haproxy/haproxy.cfg`. To modify the `haproxy.cfg`, administrative privileges are needed. The HA proxy 2 configuration is same as the HA proxy 1.

```
$ sudo nano /etc/haproxy/haproxy.cfg
```

## 3. Keepalived configuration

Configure the `keepalived.conf` which is located at `/etc/keepalived/keepalived.conf`. To modify the `keepalived.conf`, administrative privileges are needed. Create a `keepalived.conf` if no configuration file at `/etc/keepalived/`.

```
1 vrrp_instance p2 {
2     state BACKUP
3     interface eth0
4     virtual_router_id 102
5     priority 200
6     virtual_ipaddress{
7         192.168.0.181
8     }
9 }
```

The state BACKUP designates the backup HA proxy. HA proxy 2 use the same virtual IP address as HA proxy 1 which is 192.168.0.181. HA proxy 2 priority is lower than broker 1.

## 4. Start the HA proxy and Keepalived

Run the command line below to start the HA proxy and Keepalived.

```
$ sudo service haproxy start
$ sudo service keepalived start
```

### 3.3 HA Mosquitto broker using python bridge

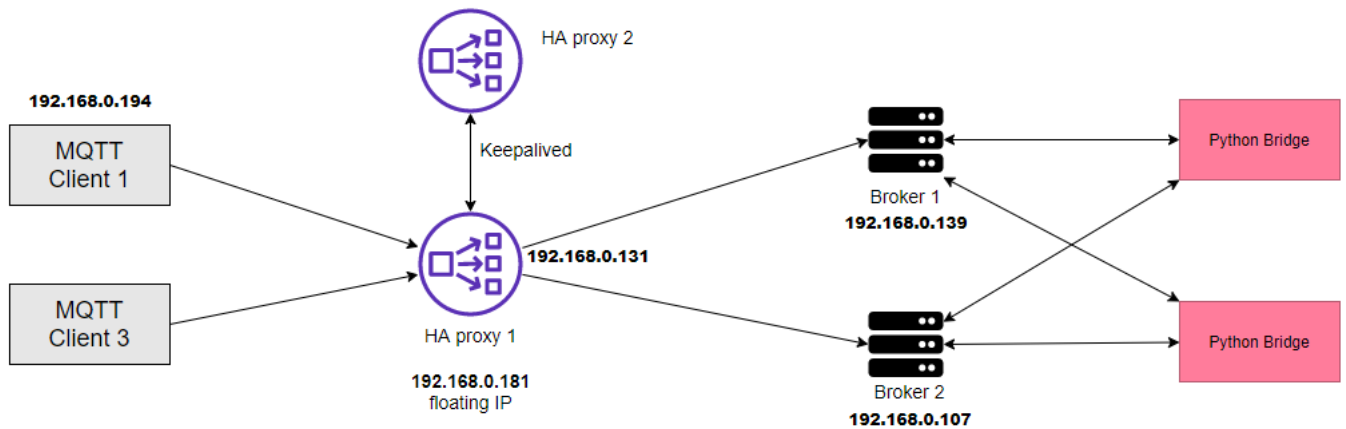


Figure 3.4 Network design of using python script

Figure 3.4 describes the network design of using python script. Instead of using Mosquitto bridge, this method use python bridge to forward message within brokers. Python bridge is a python script provided by this study. It use Paho Python Client provide by Eclipse to perform various functions. Both python bridges are running on their respective brokers. Compare to Mosquitto bridge, the python bridge able to determine whether forward the message to the other broker.

#### 3.3.1 Overview of python bridge

The functions of python bridge can be summarized as below:

1. Subscribes to all topic at local broker. Any message received by local broker will be sent to python bridge. Python bridge determines whether forward the message to the other broker using a list of available topics that is obtained from the other broker.
2. Subscribes to system log topic at local broker. Use log information to manage available topic and publish a list of available topics to topic “topic” at local broker with retained message flag.
3. Get a list of available topic from the other broker by subscribing topic “topic”.

Python bridge consists of two clients which are “client\_sys” and “client\_sys\_forward”. “client\_sys” connects to local broker and “client\_sys\_forward” connects to the other broker. Message exchanges between local broker and python bridge do not consume any network bandwidth. The main concept of the python bridge is using mqtt clients

subscribe to different topics. Different callback functions will be assigned to different topics. It will trigger different callback functions according to the topic when received messages. For example, “client\_sys” subscribes to all topic at local broker by using topic “#”. Any messages received by local broker will be sent to python bridge and trigger the callback function which is *on\_message\_forward()*. The *on\_message\_forward()* determines whether forwards the message to the other broker. Python bridge uses callback functions to perform various actions. The callback functions are listed in Table 3.1 .

<b>Client object variable name</b>	<b>Callback function</b>	<b>Topic</b>	<b>Description</b>
client_sys	on_message_forward()	#	It subscribes all topic on local broker. The callback function can determine whether publish the message to the other broker.
client_sys	on_message_forward_from_otherbroker()	forward/#	In order to prevent looping, all messages come from the other broker, prefix “forward” will be added to the topic. The callback function removes the topic prefix and republishes the message to local broker.
client_sys	on_message_sub_top()	\$SYS/broker/log/M/subscribe	The callback function add the topic and client id to the dictionary and publishes a list of available topic to topic “topic” on local broker when client subscribes topic.

client_sys	on_message_unsub_top()	\$SYS/broker/log /M/unsubscribe	The callback function removes topic and client id from dictionary and publishes a list of available topics to topic “topic” on local broker when client unsubscribes topic.
client_sys	on_message_disconnect_topic()	\$SYS/broker/log /N	The callback function removes topic and client id from dictionary and publishes a list of available topics to topic “topic” on local broker when client disconnects from local broker.
client_sys _forward	on_message_get_topic()	topic	Get a list of available topics from the other broker.

*Table 3.1 Callback functions for python bridge*



### 3.3.2 Detailed Callback Function Flowcharts

#### 3.3.2.1 on\_message\_forward() Function Flowchart

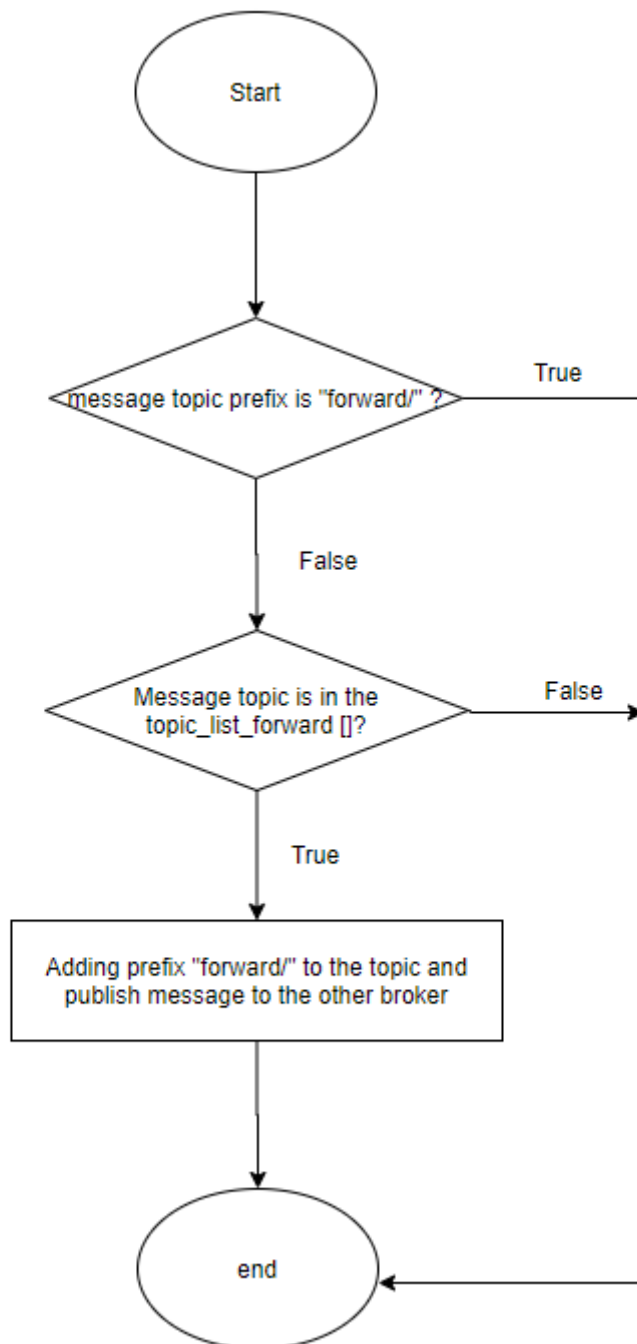
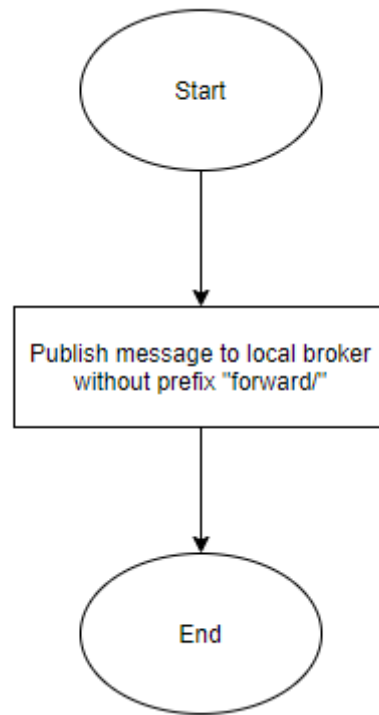


Figure 2.5 on\_message\_forward() Function Flowchart

### 3.3.2.2 on\_message\_forward\_from\_otherbroker() Function Flowchart



*Figure 3.6 on\_message\_forward\_from\_otherbroker() Function Flowchart*

### 3.3.2.3 on\_message\_sub\_top() Function Flowchart

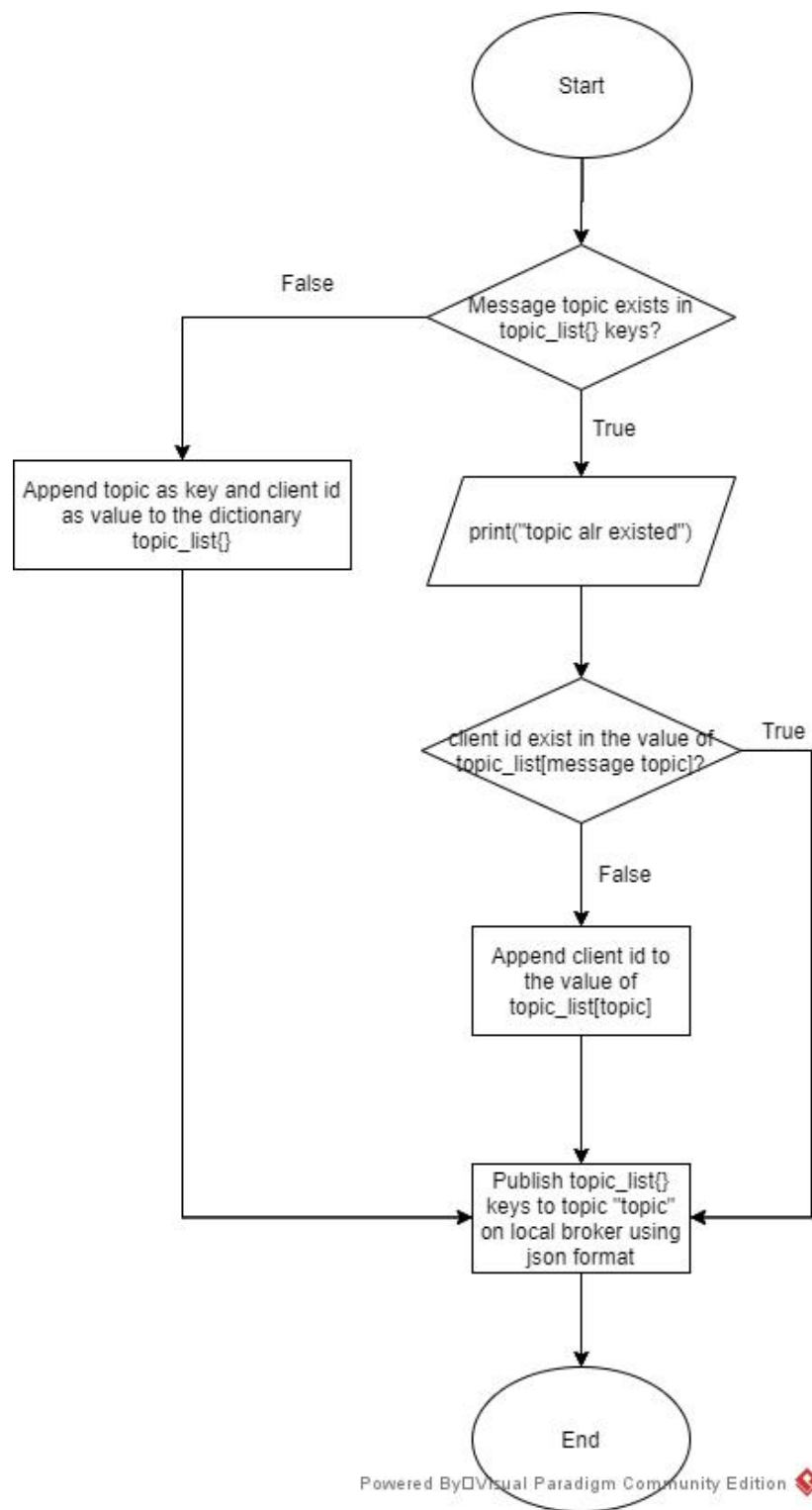


Figure 3.7 on\_message\_sub\_top() Function Flowchart

### 3.3.2.4 on\_message\_unsub\_top() Function Flowchart

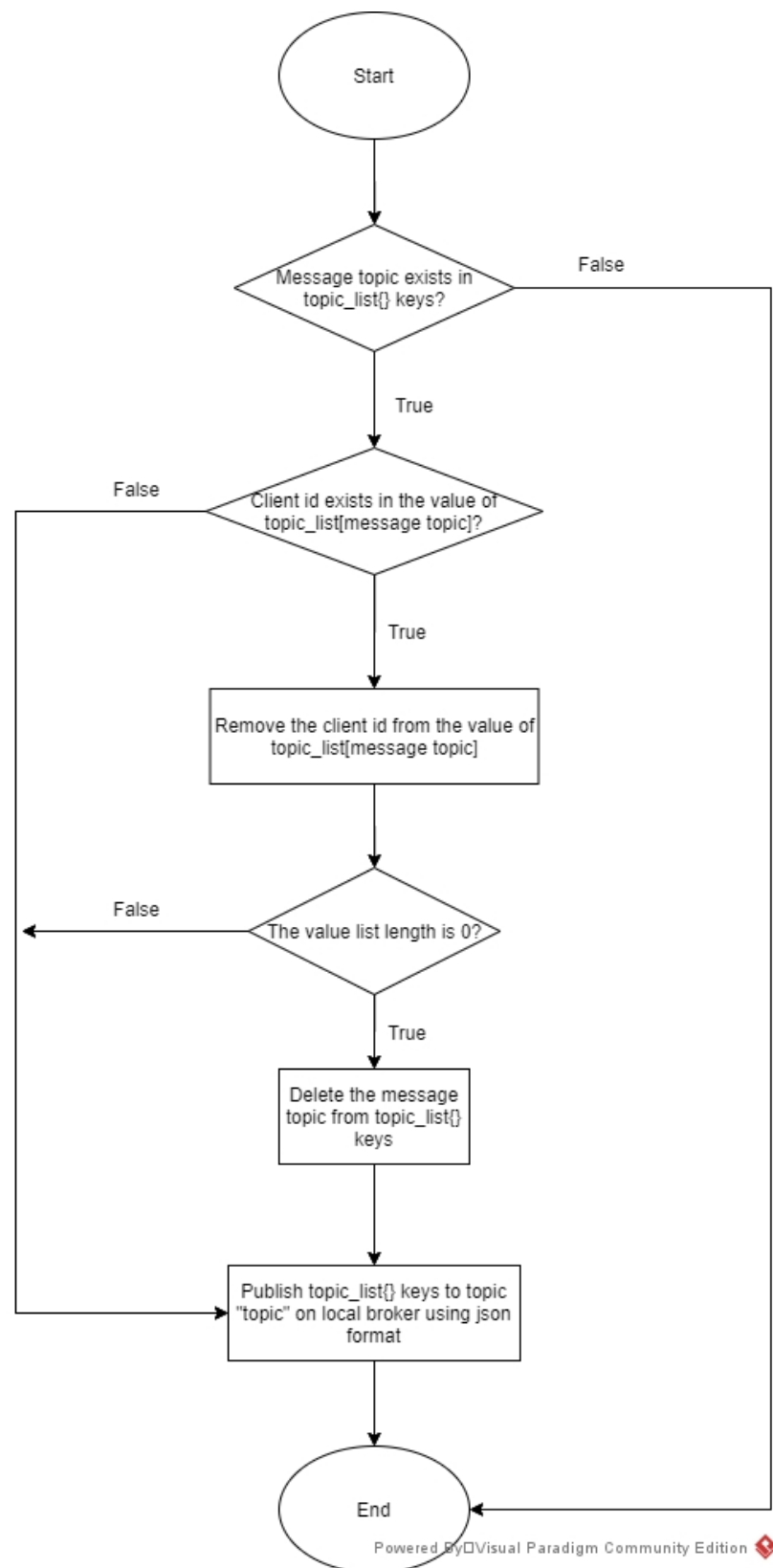


Figure 3.8 on\_message\_unsub\_top() Function Flowchart

### 3.3.2.5 on\_message\_disconn\_topic() Function Flowchart

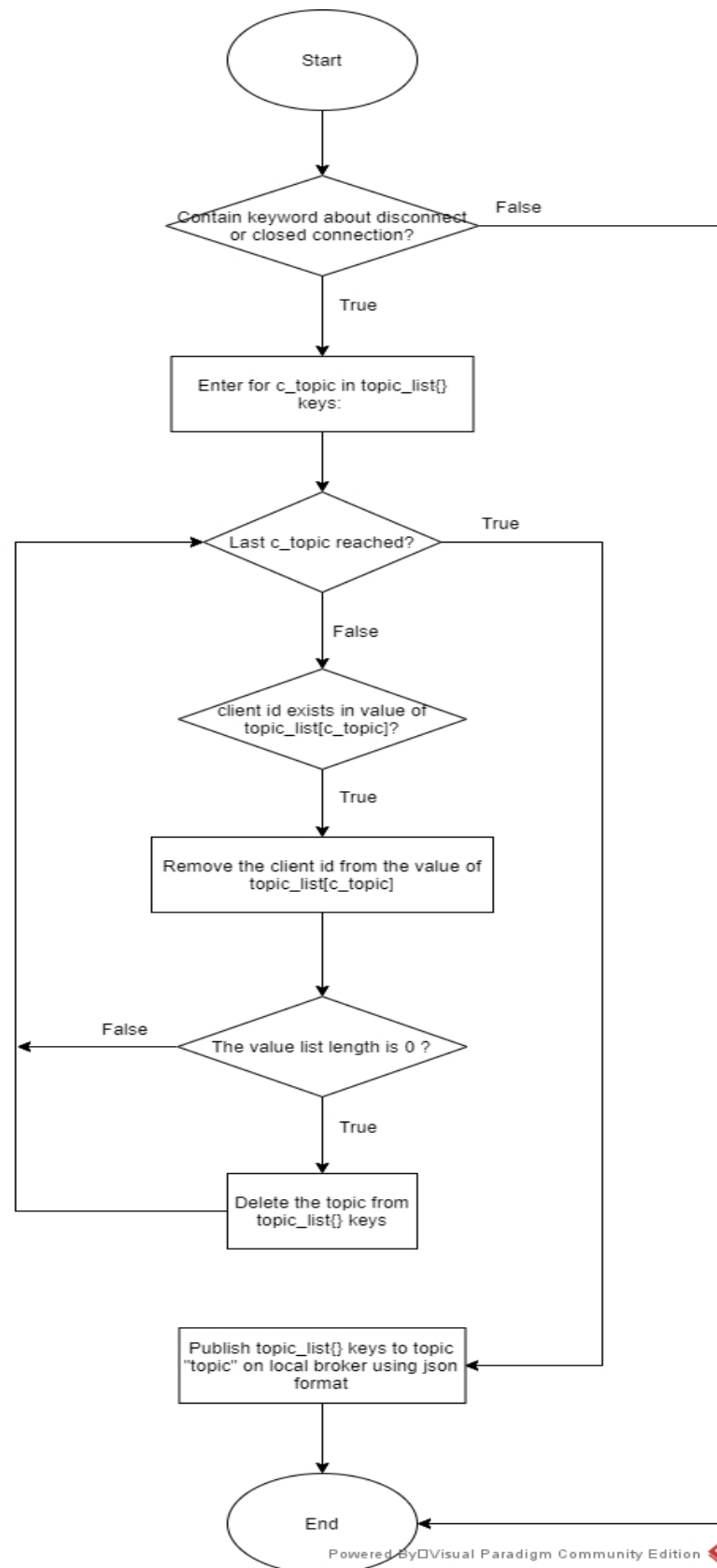


Figure 3.9 on\_message\_disconn\_topic() Function Flowchart

### 3.3.2.6 on\_message\_get\_topic() Function Flowchart

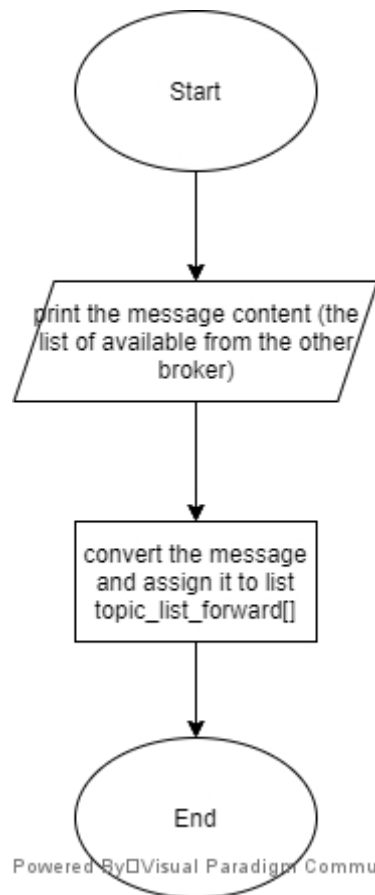


Figure 3.10 on\_message\_get\_topic() Function Flowchart

### 3.5.3 Configuration of HA Mosquitto broker using python bridge

Brokers and HA proxies are implemented on different virtual machine with Kali Linux OS respectively. Four virtual machines are used in this project for two brokers and two HA proxies. The configuration of Haproxy 1 and Haproxy 2 are same as above (configuration using Mosquitto bridge).

#### **Broker 1 Configuration**

##### 1. Install Mosquitto

Install Mosquitto on the virtual machine. This is a package for Mosquitto broker.

```
$ sudo apt-get install mosquitto
```

##### 2. Mosquitto Configuration

Change to directory */etc/mosquitto/* and create a new mosquitto configuration file called *mosquitto\_python\_bridge.conf*. To create a mosquitto configuration file, administrative privilege is needed.

```
$ cd /etc/mosquitto/  
$ sudo nano /etc/mosquitto/mosquitto_python_bridge.conf
```

Follow the configuration file below,

```
1 listener 1883 0.0.0.0  
2 allow_anonymous true  
3  
4 log_type all  
5 log_dest topic  
6 log_dest stdout
```

*mosquitto\_python\_bridge.conf*

The network ports that mosquitto listens on can be controlled using *listener*. In the configuration file, broker 1 listen all incoming network connection (0.0.0.0) at port 1883. It sends all log messages to two destinations which are console and topic using *log\_dest*.

##### 3. Run the Mosquitto

Run the command line below to start the Mosquitto broker. *-c* is used to specify the configuration file for Mosquitto broker.

```
$ mosquitto -c /etc/mosquitto/mosquitto_python_bridge.conf
```

```
kali@kali:/etc/mosquitto$ mosquitto -c /etc/mosquitto
/mosquitto_bridge.conf
1630152779: mosquitto version 2.0.8 starting
1630152779: Config loaded from /etc/mosquitto/mosquit
to_bridge.conf.
1630152779: Opening ipv4 listen socket on port 1883.
1630152779: mosquitto version 2.0.8 running
```

*Figure 3.11 Running Mosquitto in broker 1*

4. Install paho-mqtt

```
$ pip install paho-mqtt
```

5. Run the python bridge

```
$ python3 python_script.py -b 192.168.0.107
```

The argument “-b” is used to specify the other broker’s IP address. In this project, the other broker’s IP address for broker 1 is 192.168.0.107 which is broker 2 IP address.

```
kali@kali:~/Downloads$ python3 python_script.py -b 19
2.168.0.107
Not connected, another broker
Connected to local broker
The local broker topic status: {'$SYS/broker/log/#':
['broker_ext']}
The local broker topic status: {'$SYS/broker/log/#':
['broker_ext'], '#': ['broker_ext']}
```

*Figure 3.12 Running python bridge on broker 1*

### **Broker 2 Configuration**

- Install Mosquitto

Install Mosquitto on the virtual machine. This is a package for Mosquitto broker.

```
$ sudo apt-get install mosquitto
```

- Mosquitto Configuration



Change to directory */etc/mosquitto/* and create a new mosquitto configuration file called *mosquitto\_python\_bridge.conf*. To create a mosquitto configuration file, administrative privilege is needed.

```
$ cd /etc/mosquitto/  
$ sudo nano /etc/mosquitto/mosquitto_python_bridge.conf
```

Follow the configuration file below,

```
1 listener 1883 0.0.0.0  
2 allow_anonymous true  
3  
4 log_type all  
5 log_dest topic  
6 log_dest stdout
```

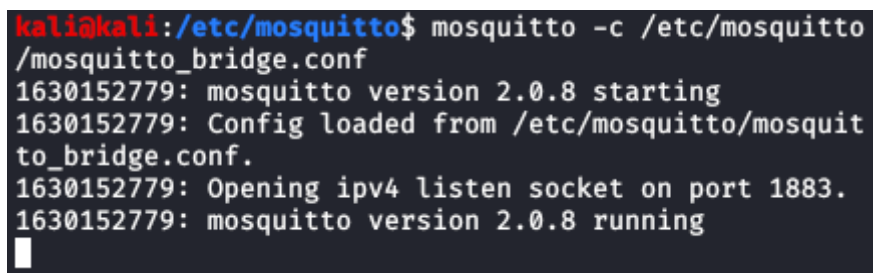
*mosquitto\_python\_bridge.conf*

The network ports that mosquitto listens on can be controlled using *listener*. In the configuration file, broker 1 listen all incoming network connection (0.0.0.0) at port 1883. It sends all log messages to two destinations which are console and topic using *log\_dest*.

- Run the Mosquitto

Run the command line below to start the Mosquitto broker. *-c* is used to specify the configuration file for Mosquitto broker.

```
$ mosquitto -c /etc/mosquitto/mosquitto_python_bridge.conf
```



```
kali@kali:/etc/mosquitto$ mosquitto -c /etc/mosquitto  
/mosquitto_python_bridge.conf  
1630152779: mosquitto version 2.0.8 starting  
1630152779: Config loaded from /etc/mosquitto/mosquit  
to_python_bridge.conf.  
1630152779: Opening ipv4 listen socket on port 1883.  
1630152779: mosquitto version 2.0.8 running
```

*Figure 3.13 Running Mosquitto in broker 2*

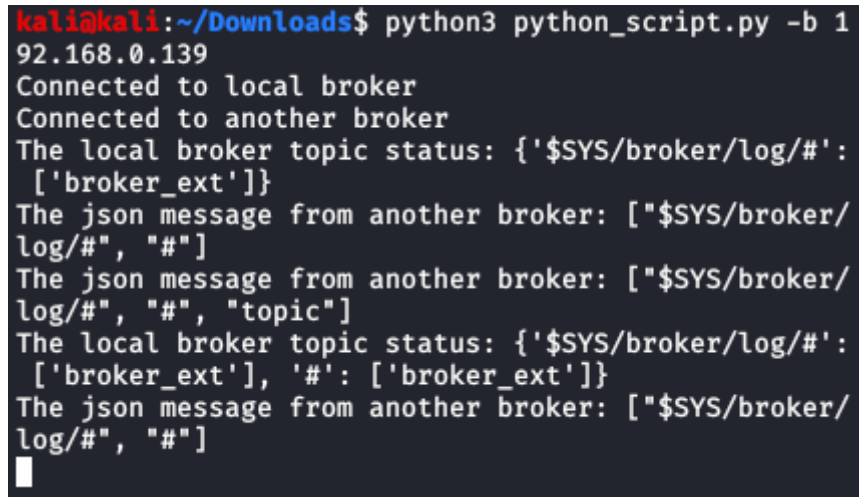
- Install paho-mqtt

```
$ pip install paho-mqtt
```

- Run the python bridge

```
$ python3 python_script.py -b 192.168.0.139
```

The argument “-b” is used to specify the other broker’s IP address. In this project, the other broker’s IP address for broker 2 is 192.168.0.139 which is broker 1 IP address.

A terminal window with a dark background and light-colored text. The prompt is 'kali@kali:~/Downloads\$'. The command entered is 'python3 python\_script.py -b 192.168.0.139'. The output shows the script connecting to a local broker and another broker, displaying topic status and JSON messages. The messages include paths like '\$SYS/broker/log/#' and topics like 'broker\_ext'.

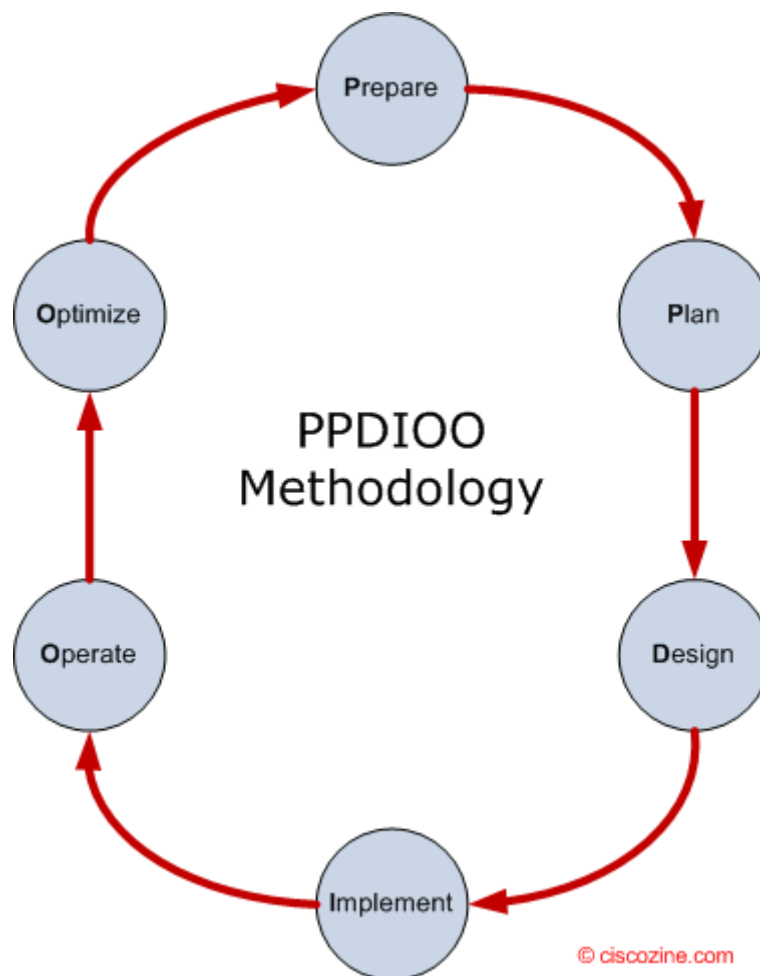
```
kali@kali:~/Downloads$ python3 python_script.py -b 192.168.0.139
Connected to local broker
Connected to another broker
The local broker topic status: {'$SYS/broker/log/#': ['broker_ext']}
The json message from another broker: ["$SYS/broker/log/#", "#"]
The json message from another broker: ["$SYS/broker/log/#", "#", "topic"]
The local broker topic status: {'$SYS/broker/log/#': ['broker_ext'], '#': ['broker_ext']}
The json message from another broker: ["$SYS/broker/log/#", "#"]
█
```

*Figure 3.14 Running python bridge on broker 2*

## CHAPTER 4 METHODOLOGY AND TOOLS

### 4.1 Methodology

Since this is a project mainly on network deployment and deployment, the PPDIOO network design methodology is used. The methodology is composed by six phases which are prepare, plan, design, implement, operate, optimize. The Figure 4.1 shows the PPDIOO lifecycle phase.



*Figure 4.1 Phases of PPDIOO*

Prepare	Study all the necessary knowledge for problem. For example, the MQTT protocol, Mosquitto and etc.
Plan	Perform analysis on the existing problem and determine infrastructure or technology that could be used in the proposed system.
Design	Design the details for the proposed solution. For example, the configuration file details for implementing the bridge.
Implement	The building process of the proposed solution.
Operate	Testing for the proposed solution.
Optimize	Revise the proposed solution and solve the problem that found in operate phase. Improve the proposed solution if possible

*Table 3.2 Brief description of each phase in PPDIOO*

## 4.2 Technologies and Tools Involved

### Hardware

Desktop:

- Used to run *multiple\_client\_sub.py* and Krylovsk mqtt benchmark tool in testing.
- Used to run 4 virtual machines.

Processor	Intel(R) Core (TM) i5-3340 CPU @ 3.10GHz, 3101 Mhz, 4 Core(s), 4 Logical Processor(s)
Graphics	GTX 1050
RAM	16GB
OS	Microsoft Windows 10 Pro

*Table 4.3 Specification of Desktop*

Virtual Machine:

- Used to run HA proxy 1, HA proxy 2, broker 1 and broker 2 in this project.

Processor	Intel(R) Core (TM) i5-3340 CPU @ 3.10GHz, 3101 Mhz, 1 Core
RAM	1048MB
OS	Kali Linux

*Table 4.4 Specification of virtual machine*

### **Software**

1. Oracle virtual box:

In order to run Mosquitto and HA proxy, virtual machine is needed in this project. It is an ideal utility for running virtual machine on a Window or Linux PC.

2. Kali Linux OS

Kali Linux OS will be installed on each virtual machine to run Mosquitto.

3. Mosquitto

An open-source message broker that implements MQTT protocol.

4. Paho – MQTT

This is Eclipse Paho MQTT Python client library.

5. HA proxy

It is an open-source software that can provide load balancing and proxying for TCP.

6. Keepalived

Enable a shared IP address between two servers.

7. krylovsk mqtt benchmark tool

This is a simple MQTT (broker) benchmarking tool provided by krylovsk. (Krylovskiy A, n.d.)

8. iftop

A Real Time Linux Network Bandwidth Monitoring Tool.

## CHAPTER 5 TESTING

### 5.1 Performance testing

To investigate performance comparison of single broker, HA Mosquitto brokers using Mosquitto bridge and HA Mosquitto brokers using python bridge, [krylovsk mqtt benchmark tool](#) has been used to perform load tests in a publish scenario. This is a simple MQTT (broker) benchmarking tool written in GO. Besides that, a python script *multiple\_client\_sub.py* is created to simulate subscribers. The script creates 500 subscribers. Each subscriber subscribes one topic, topic from “test/0, test/1, ..., test/498, test/499”, total 500 topics. For HA brokers, HA proxy forward messages according to the round robin algorithm, therefore subscribers connect broker alternately. It results in

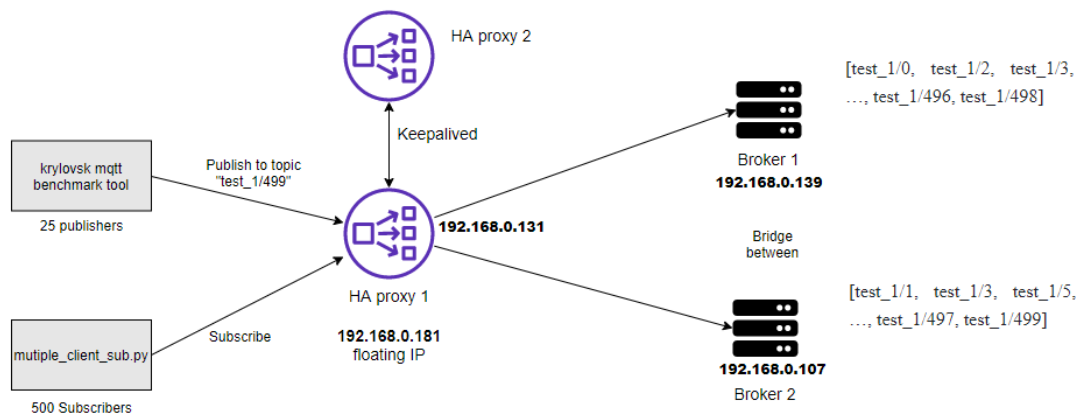


Figure 5.1 Network design for HA brokers performance testing

Broker 1: [test\_1/0, test\_1/2, test\_1/3, ..., test\_1/496, test\_1/498]

Broker 2: [test\_1/1, test\_1/3, test\_1/5, ..., test\_1/497, test\_1/499].

Krylovsk mqtt benchmark tool is used to create 25 publishers. Each publisher connects to the broker alternately through the HA proxy. Each publisher publishes 3000 messages to topic test\_1/499, the size of the message payload is 100 bytes. The quality of service is 1. Figure 5.1 shows the network design for HA brokers. For single broker, subscribers and publishers directly connect to the broker. The mosquitto broker start with the default configuration. Figure 5.2 shows the testing network design for single broker.

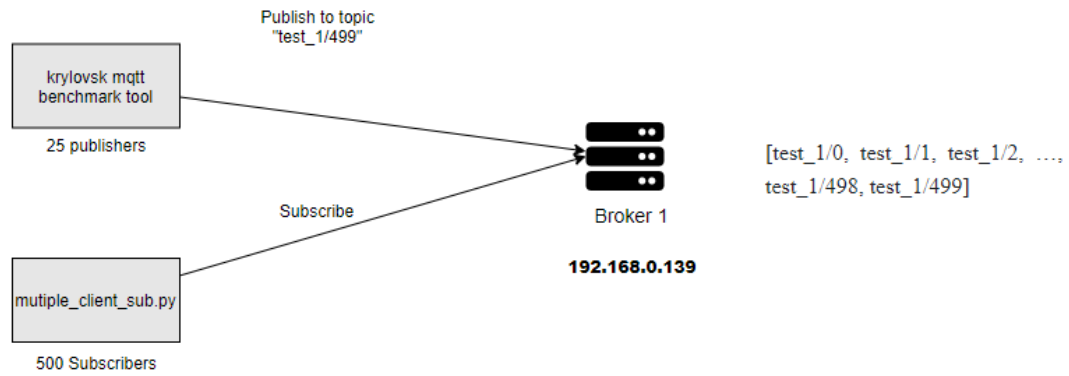


Figure 5.2 Network design for single broker performance testing

### 5.1.1 Performance testing configuration

#### For Single Broker

1. Run the *multiple\_client\_sub.py* to create subscribers subscribe to different topic. All subscribers directly connect to the broker 1.

```
$ python multiple_client_sub.py -b 192.168.0.139
```

The argument “-b” is used to specify the broker IP address.

2. Run the krylovsk mqtt benchmark tool to perform load tests in a publish scenario. After finish publishing, krylovsk mqtt benchmark tool will return result.

```
$ mqtt-benchmark.exe --broker tcp://192.168.0.139:1883 -count 3000 --clients 20 --qos 1 --topic test_1/1 --size 100
```

#### For HA brokers (Mosquitto bridge and python bridge)

1. Run the *multiple\_client\_sub.py* to create subscribers subscribe to different topic. Each subscriber connects to brokers alternately through the HA proxy.

```
$ python multiple_client_sub.py -b 192.168.0.181
```

The argument “-b” is used to specify the HA proxy floating IP address. In this project, the HA proxy floating IP address is 192.168.0.181.

- Run the krylovsk mqtt benchmark tool to Run the krylovsk mqtt benchmark tool to perform load tests in a publish scenario. After finish publishing, krylovsk mqtt benchmark tool will return result.

```
$ mqtt-benchmark.exe --broker tcp://192.168.0.181:1883 -count 3000 --clients 20 --qos 1 --topic test_1/1 --size 100
```

## 5.2 Network bandwidth consumption testing

This testing aims to test the network bandwidth consumption between brokers when message forward from one broker to the other broker. The test targets are Mosquitto bridge and python bridge. Figure 5.3 illustrates the testing of network bandwidth consumption between brokers.

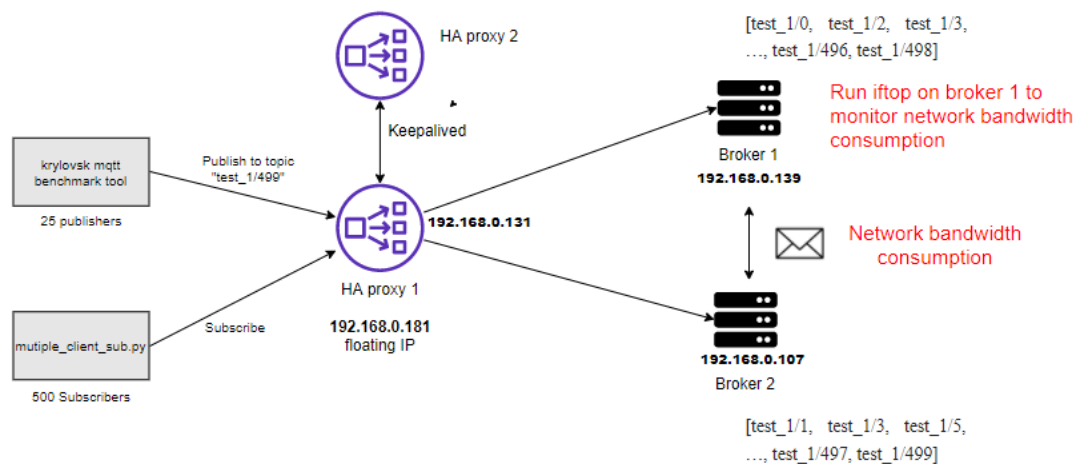


Figure 4.3 Network design for HA brokers network bandwidth consumption testing

To monitor the network bandwidth consumption between brokers, iftop program is used on broker 1. iftop is a real time Linux console-based network bandwidth monitoring tool. It will show a quick overview of network activities based on port. Same as performance testing, *multiple\_client\_sub.py* is used to create subscribers subscribe to topic on broker 1. Krylovsk mqtt benchmark is used to create publishers and publish message with qos 0 to both broker through HA proxy.

### 5.2.1 Network bandwidth consumption testing configuration

#### For HA brokers (Mosquitto bridge and Python bridge)

- Install iftop on broker 1



```
$ sudo apt install iftop
```

2. Run iftop on broker 1

```
$ sudo iftop
```

Press T to display cumulative line total.

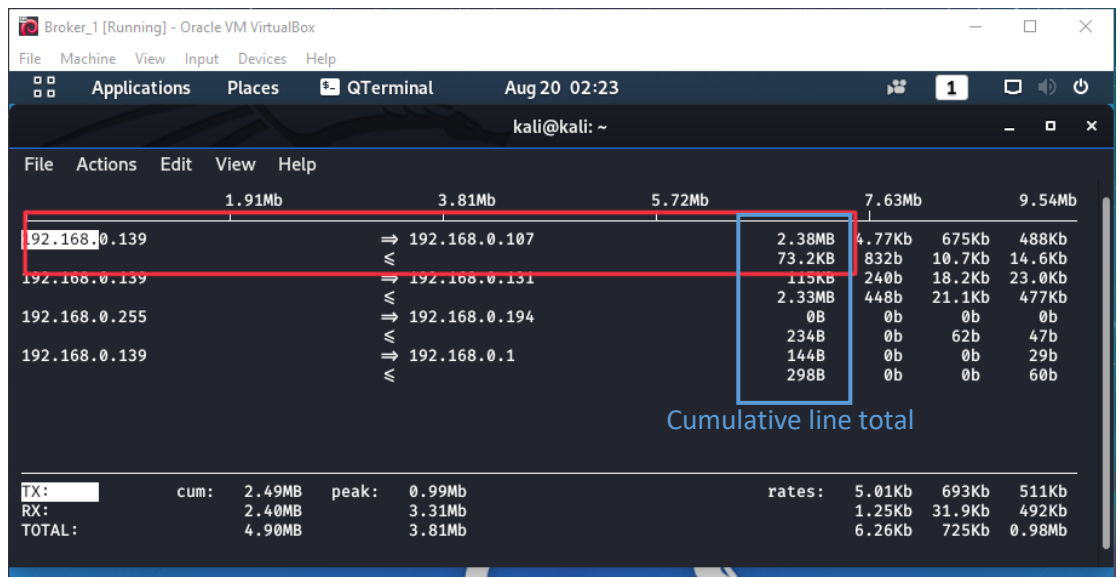


Figure 5.4 Interface of iftop

3. Run the `multiple_client_sub.py` to create subscribers subscribe to different topic. Each subscriber connects to brokers alternately through the HA proxy.

```
$ python multiple_client_sub.py -b 192.168.0.181
```

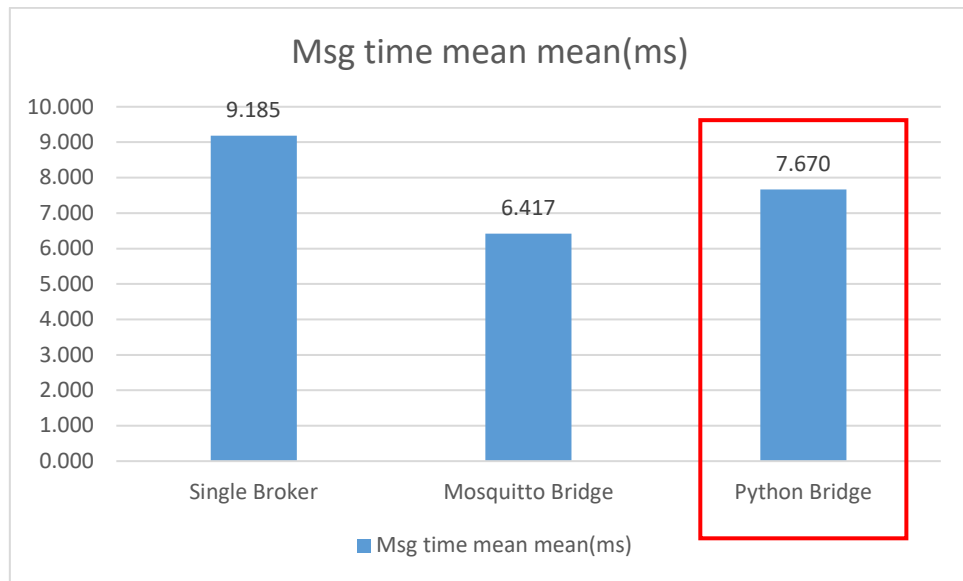
The argument “-b” is used to specify the HA proxy floating IP address. In this project, the HA proxy floating IP address is 192.168.0.181.

4. Run the krylovsk mqtt benchmark tool to act as publishers publish messages to both brokers.

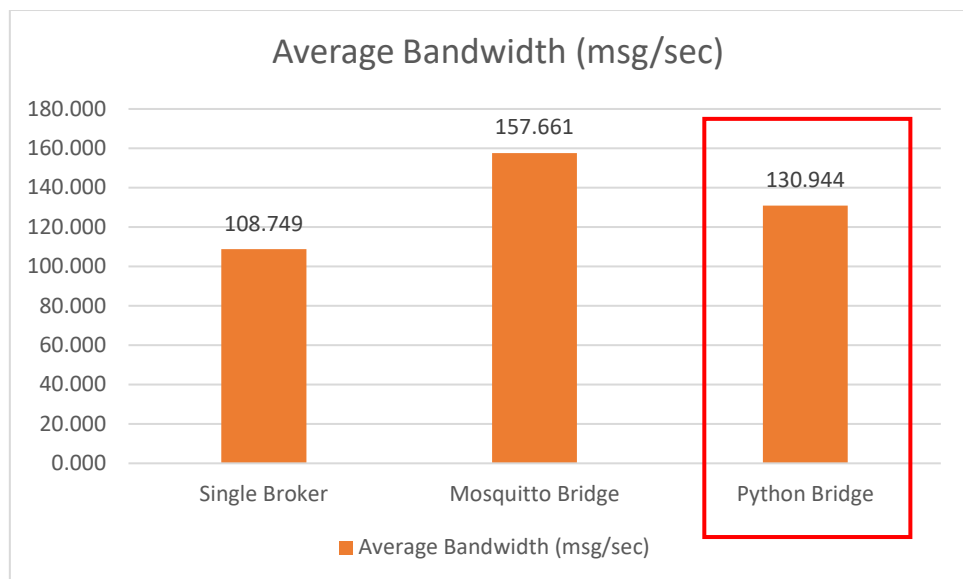
```
$ mqtt-benchmark.exe --broker tcp://192.168.0.181:1883 -count 3000 --clients 20 --qos 0 --topic test_1/1 --size 100
```

## 5.3 Comparison Results

### 5.3.1 Performance comparison results

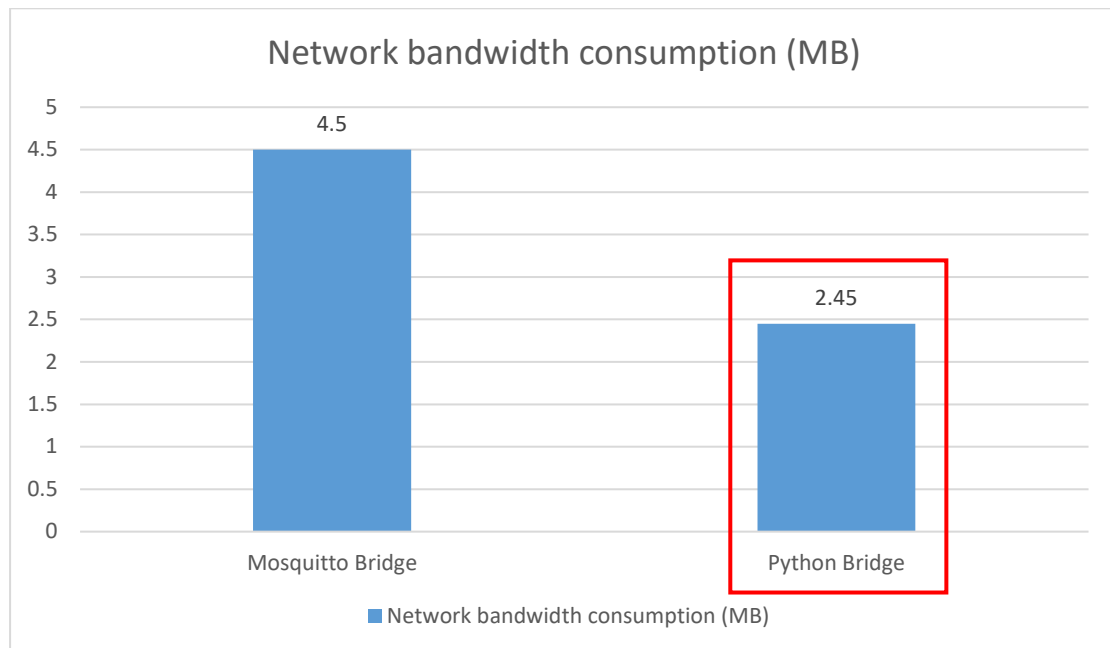


*Figure 5.5 Msg time mean mean (msg/sec) comparison result*



*Figure 5.6 Average bandwidth (msg/sec) comparison result*

### 5.3.2 Network bandwidth consumption comparison results



*Figure 5.7 Network bandwidth consumption (MB) between brokers*

### 5.4 Result Discussion

According to Figure 5.6, Mosquitto bridge is the highest average bandwidth because of it consists of two brokers to handle the messages. The average bandwidth of python bridge is lower than Mosquitto bridge because the script needs to do some stuff to determine whether forward the message to another broker. The single broker is lowest because it only one broker to handle messages. By using the high availability broker solution can achieve higher throughput than single broker under the publishing conditions.

Although the Mosquitto bridge's performance is better than python bridge, but Mosquitto bridge consumes more network bandwidth than python bridge. Figure 5.7 shows the network bandwidth consumption. The bandwidth usage between two brokers using Mosquitto bridge is 4.5 MB. By using the python bridge, the bandwidth usage is 2.45 MB. This is because there is no client subscribes to topic "test\_1/1" on broker 1, the messages that is sent to topic "test\_1/1" on broker 2 are not forwarded to broker 1, therefore the bandwidth usage of python bridge is lower than Mosquitto bridge.

By using the python bridge provided by this study, it can reduce the network usage between brokers. However, the performance is lower than Mosquitto bridge. This is a trade-off between performance and network bandwidth consumption.

## CHAPTER 6 CONCLUSION AND FUTURE WORK

### 6.1 Conclusion

Compared to other protocol such as HTTP, MQTT is a lightweight publish/subscribed messaging protocol design for low-bandwidth and high latency network. These kinds of features make MQTT has been widely used in IoT world. In MQTT based network, broker plays the prime role because it mainly responsible for receiving and processing all the messages from clients. The broker is the heart of the MQTT protocol. Therefore, MQTT broker should be high available. There are many types of public and private brokers available by different vendors, one of them is Mosquitto. However, Mosquitto does not support clustering and it makes high availability become difficult. Although high availability for Mosquitto brokers can be achieved by implementing the failover, but Mosquitto brokers are not able work together to maximizes capacity utilization without clustering. Without high availability, single broker could be a single point of failure and performance will be affected when high traffic.

This project proposed two solutions for Mosquitto brokers to achieve high availability. The first method was using Mosquitto bridge to connect original broker and redundant broker. However, Mosquitto bridge will send all messages to the other broker even though nobody subscribes to the topic on the other broker. Hence, the python bridge was developed to reduce bandwidth usage between brokers. The python bridge is a python script provided by this study and it can determine whether forward the message to the other broker. HA proxy also has been used to act as load balancer in both solutions.

In this study, both solutions are able to eliminate single point of failure by using redundant broker. Redundant broker can work with original broker using bridge. HA proxy can efficiently distribute incoming traffic to both original and redundant broker. The high availability brokers can give better performance than single broker.

### 6.2 Future Work

There is one issue with the python bridge. The python bridge does not support topic wildcard. For example, client A subscribes to a topic “sensor/#” at broker 2. The topic “sensor/#” means any message topic with the prefix “sensor” will be sent to client A. If client B publishes a message to topic “sensor/1” on broker 1, broker 1 will not forward the message to broker 2 because the python bridge does not recognize topic

wildcard. The python bridge can be improved by adding functions to recognize topic wildcard. Moreover, tree data structure can be used instead of python dictionary in python bridge. Due to the mqtt topic is a simple string that can consists of one or more hierarchy level, tree data structure is more suitable for managing topic and client id in python bridge.

## REFERENCES

- AVI Networks, *What is High Availability? Definition & FAQs*. [online] Available from: <https://avinetworks.com/glossary/high-availability/#main>. [27 November 2020]
- Digital Ocean, 2016. *What is High Availability?* [online] Available from: <https://www.digitalocean.com/community/tutorials/what-is-high-availability>. [27 November 2020]
- Eclipse foundation, 2014. *How to Get Started with the lightweight IoT Protocol?* [online] Available from: [https://www.eclipse.org/community/eclipse\\_newsletter/2014/october/article2.php](https://www.eclipse.org/community/eclipse_newsletter/2014/october/article2.php). [29 November 2020]
- Eugster P.T., Felber, P.A., Guerraoui, P., & Kermarrec, A.-M., 2003. The Many Faces of Publish/Subscribe. *Research Gate*. [online] Available from: [https://www.researchgate.net/publication/220566321\\_The\\_Many\\_Faces\\_of\\_PublishSubscribe](https://www.researchgate.net/publication/220566321_The_Many_Faces_of_PublishSubscribe). [29 November 2020].
- Flylib, *Message exchange patterns*. [online] Available from: [https://flylib.com/books/en/2.365.1/message\\_exchange\\_patterns.html](https://flylib.com/books/en/2.365.1/message_exchange_patterns.html). [29 November 2020]
- Fruhlinger, J, 2020. *What is IoT? The internet of things explained*. Available from: [27 November 2020]
- HiveMQ, *Clustering Hive MQ*. [online] Available from: [https://www.hivemq.com/downloads/clustering\\_hivemq.pdf](https://www.hivemq.com/downloads/clustering_hivemq.pdf) [1 April 2020]
- HiveMQ, *HiveMQ Cluster :: HiveMQ Documentation*. [online] Available from: <https://www.hivemq.com/docs/hivemq/4.4/user-guide/cluster.html>. [30 November 2020]
- IBM Developer, 2017. *What is MQTT? Why use MQTT?* [online] Available from: <https://developer.ibm.com/articles/iot-mqtt-why-good-for-iot/>. [29 November 2020]

- Kumar R., 2020. MQTT Broker Comparison – MQTTRoute vs Mosquitto. [online] Available from: <https://www.bevywise.com/blog/mqtt-broker-comparison-mqttroute-vs-mosquitto/>
- Lavoie C., 2017. *Proxy-protocol-figure-2.png (620×338)*. [online] Available from: <https://www.haproxy.com/wp-content/uploads/2017/03/Proxy-protocol-figure-1.png> [7 April 2020]
- Solovev A.& Petrova A., 2020. *Peculiarities of Choosing MQTT Protocol for Your IoT Devices*. [online] Available from: <https://www.integrasources.com/blog/mqtt-protocol-iot-devices/> [10 September 2020]
- STS MQTT, *MQTT Docs: Topics and access control*. Available from: <https://docs.senasetecnic.com/mqtt/topics-and-access-control/>. [29 November 2020]
- The HiveMQ Team, 2016. Creating highly available and ultra-scalable MQTT clusters, *HiveMQ*. [online] Available from: <https://www.hivemq.com/blog/clustering-mqtt-introduction-benefits/>. [30 November 2020]
- Tomosvari I., 2017. Choosing an MQTT broker for your IoT project, *AutSoft*. [online] Available from: <https://blog.autsoft.hu/choosing-an-mqtt-broker-for-your-iot-project/> [1 April 2020]
- Krylovskiy A., krylovsk/mqtt-benchmark: MQTT broker benchmarking tool, *GitHub*. [online] Available from: <https://github.com/krylovsk/mqtt-benchmark> [1 September 2021]



# FINAL YEAR PROJECT WEEKLY REPORT

(Project I / Project II)

<b>Trimester, Year:</b> Semester 2, Year 3	<b>Study week no.:</b> 1, 2
<b>Student Name &amp; ID:</b> Wong Kei Yin (19ACB00582)	
<b>Supervisor:</b> Ts Dr Ooi Boon Yaik	
<b>Project Title:</b> Building A Ha Mqtt Brokerage Solution Using Mosquitto	

## 1. WORK DONE

None

## 2. WORK TO BE DONE

1. Find out how to develop a bridge.


## 3. PROBLEMS ENCOUNTERED

None

## 4. SELF EVALUATION OF THE PROGRESS

None

  
\_\_\_\_\_  
Supervisor's signature

  
\_\_\_\_\_  
Student's signature

# FINAL YEAR PROJECT WEEKLY REPORT

(Project I / Project II)

<b>Trimester, Year:</b> Semester 2, Year 3	<b>Study week no.:</b> 3, 4
<b>Student Name &amp; ID:</b> Wong Kei Yin (19ACB00582)	
<b>Supervisor:</b> Ts Dr Ooi Boon Yaik	
<b>Project Title:</b> Building A Ha Mqtt Brokerage Solution Using Mosquitto	

## 1. WORK DONE

None

## 2. WORK TO BE DONE

2. Write a python script to act as a bridge between brokers. The python bridge able to determine whether forward the messages.

## 3. PROBLEMS ENCOUNTERED

None

## 4. SELF EVALUATION OF THE PROGRESS

None

Supervisor's signature

Student's signature

# FINAL YEAR PROJECT WEEKLY REPORT

(Project I / Project II)

<b>Trimester, Year:</b> Semester 2, Year 3	<b>Study week no.:</b> 5, 6
<b>Student Name &amp; ID:</b> Wong Kei Yin (19ACB00582)	
<b>Supervisor:</b> Ts Dr Ooi Boon Yaik	
<b>Project Title:</b> Building A Ha Mqtt Brokerage Solution Using Mosquitto	

## 1. WORK DONE

1. Python bridge was developed

## 2. WORK TO BE DONE

1. Fix the bug in the python bridge
2. Find out how to do the testing for Mosquitto bridge and python bridge.

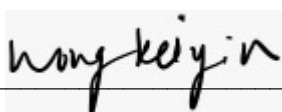
## 3. PROBLEMS ENCOUNTERED

1. A little bug in the python bridge

## 4. SELF EVALUATION OF THE PROGRESS

None

  
\_\_\_\_\_  
Supervisor's signature

  
\_\_\_\_\_  
Student's signature

# FINAL YEAR PROJECT WEEKLY REPORT

(Project I / Project II)

<b>Trimester, Year:</b> Semester 2, Year 3	<b>Study week no.:</b> 9, 10
<b>Student Name &amp; ID:</b> Wong Kei Yin (19ACB00582)	
<b>Supervisor:</b> Ts Dr Ooi Boon Yaik	
<b>Project Title:</b> Building A Ha Mqtt Brokerage Solution Using Mosquitto	

## 1. WORK DONE

1. Testing is done. The direction for testing is correct

## 2. WORK TO BE DONE

1. Make the testing report more readable and more detailed


## 3. PROBLEMS ENCOUNTERED

None

## 4. SELF EVALUATION OF THE PROGRESS

None

  
\_\_\_\_\_  
Supervisor's signature

  
\_\_\_\_\_  
Student's signature



UNIVERSITI TUNKU ABDUL RAHMAN  
FACULTY OF INFORMATION & COMMUNICATION  
TECHNOLOGY (KAMPAR CAMPUS)

# Building A Ha MQTT Brokerage Solution Using Mosquitto

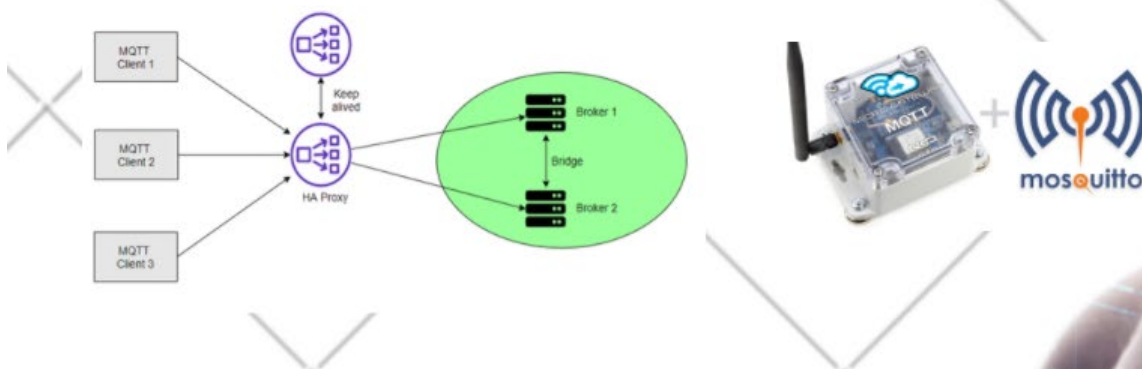
## Problem Statement

- No cluster functionality built-in for Mosquitto to achieve high availability
- The period of downtime can cause negative impact
- Performance of single broker may be affected when high traffic

## Project Objective

- Create redundant broker that able to cover same task and eliminate single point of failure.
- Use bridge to connect original and redundant broker together for messages sharing.
- Efficiently distribute incoming traffic across MQTT brokers to minimize response time.

## System Design



Feedback Studio - Google Chrome

ev.turnitin.com/app/carta/en-us/?s=&lang=en-us&o=1639804048&student\_user=1&u=1116414639

feedback studio Kei Yin Wong Building A Ha MQTT Brokerage Solution Using Mosquitto

### CHAPTER 1 INTRODUCTION

The term Internet of Things (IoT) is a collective term for the number of electronic devices that are able to connect to the Internet, capable of sending data, receiving instructions, or both, was first proposed by British technical experts (Fruhlinger, 2020). The use of IoT has increased significantly because of the evolution of multiple technologies such as sensor network, wireless connection, and low power electronics over the last few years. IoTs connect the physical and digital worlds by bringing the internet's power, data processing, and analytics to the real world of objects. According to Fruhlinger (2020), there are more than 50 billion IoT devices as of 2020, and those devices will generate 4.4 zettabytes of data. IoTs can communicate to each other through the internet, it means that they can be monitored and controlled remotely. Therefore, IoTs infiltrate various fields such as industrial automation, smart cities, and etc. Apart from it, a myriad of IoT protocols have been proposed for communication between IoT devices. MQTT (Message Queuing Telemetry Transport) has been widely used because of it has less bandwidth requirement, lightweight and suitable for unreliable connection.

Compared to other protocols such as HTTP protocols and Advanced Message Queuing Protocol (AMQP), MQTT is a lightweight publish/subscribe protocol for IoT and It can be used on restricted devices as well as high-latency networks. HTTP is a synchronous protocol, and it applied the request and response pattern. Client always need to wait for the server to respond and result in poor scalability. In IoT worlds, the high latency and low bandwidth network always make the synchronous communication problematic. Moreover, HTTP also is a heavy weight protocol that include many headers and rules which make expensive to broadcast messages to all IoTs over the network. The synchronous messaging protocol is far more suitable for IoT applications than the synchronous protocol. Besides HTTP protocols, AMQP is a binary TCP-based protocol

Match Overview

8%

1	Submitted to University... Student Paper	2%
2	mosquito.org Internet Source	1%
3	J. Kotak, A. Shah, A. Sh... Publication	<1%
4	Submitted to University... Student Paper	<1%
5	lipn.univ-paris13.fr Internet Source	<1%
6	www.tecmint.com Internet Source	<1%
7	access.redhat.com Internet Source	<1%
8	Submitted to American... Student Paper	<1%
9	Priyadarshi Tripathy, Ks... Publication	<1%

## Turnitin Originality Report

Processed on: 02-Sep-2021 09:59 +08  
 ID: 1639804048  
 Word Count: 7346  
 Submitted: 1

Building A Ha MQTT Brokerage Solution Using  
 M... By Kei Yin Wong

Similarity Index	Similarity by Source
8%	Internet Sources: 6% Publications: 4% Student Papers: 5%

<a href="#">exclude quoted</a> <a href="#">exclude bibliography</a> <a href="#">exclude small matches</a>	mode: <span>show matches one at a time</span> <span>▼</span> <span>Change mode</span> <a href="#">print</a>
<a href="#">download</a>	
2% match (student papers from 15-Sep-2017) <a href="#">Submitted to University of Portsmouth on 2017-09-15</a>	
1% match (Internet from 15-Jul-2021) <a href="http://mosquitto.org">http://mosquitto.org</a>	
<1% match (publications) <a href="#">J. Kotak, A. Shah, A. Shah, P. Rajdev, "A Comparative Analysis on Security of MQTT Brokers", 2nd Smart Cities Symposium (SCS 2019), 2019</a>	
<1% match (student papers from 25-Nov-2020) <a href="#">Submitted to University of Messina on 2020-11-25</a>	
<1% match (Internet from 23-Mar-2016) <a href="http://lipn.univ-paris13.fr">http://lipn.univ-paris13.fr</a>	
<1% match (Internet from 22-May-2021) <a href="https://www.tecmint.com/iftop-linux-network-bandwidth-monitoring-tool/">https://www.tecmint.com/iftop-linux-network-bandwidth-monitoring-tool/</a>	
<1% match (Internet from 02-May-2020) <a href="https://www.tecmint.com/setup-automatic-updates-for-centos-8/">https://www.tecmint.com/setup-automatic-updates-for-centos-8/</a>	
<1% match (Internet from 11-Mar-2019) <a href="https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/pdf/load_balancer_administration/Red_Hat_Enterprise_Linux-7-Load_Balancer_Administration-en-US.pdf">https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/pdf/load_balancer_administration/Red_Hat_Enterprise_Linux-7-Load_Balancer_Administration-en-US.pdf</a>	
<1% match (student papers from 25-May-2020) <a href="#">Submitted to American Public University System on 2020-05-25</a>	
<1% match (publications) <a href="#">Priyadarshi Tripathy, Kshirasagar Naik, "Impact Analysis", Wiley, 2014</a>	
<1% match (Internet from 13-Dec-2020) <a href="https://www.networkworld.com/article/3207535/what-is-iot-the-internet-of-things-explained.html#:~:text=The%20internet%20of%20things%20(IoT)%20is%20a%20catch-all">https://www.networkworld.com/article/3207535/what-is-iot-the-internet-of-things-explained.html#:~:text=The%20internet%20of%20things%20(IoT)%20is%20a%20catch-all</a>	

<b>Universiti Tunku Abdul Rahman</b>			
<b>Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)</b>			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1 of 1



**FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY**

<b>Full Name(s) of Candidate(s)</b>	WONG KEI YIN
<b>ID Number(s)</b>	19ACB00582
<b>Programme / Course</b>	Bachelor of Computer Science (Honours)
<b>Title of Final Year Project</b>	Building A Ha MQTT Brokerage Solution Using Mosquitto

<b>Similarity</b>	<b>Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)</b>
<b>Overall similarity index: <u>8</u> %</b>  <b>Similarity by source</b> Internet Sources: <u>6</u> % Publications: <u>4</u> % Student Papers: <u>5</u> %	
<b>Number of individual sources listed of more than 3% similarity: <u>0</u></b>	
<b>Parameters of originality required and limits approved by UTAR are as Follows:</b> <b>(i) Overall similarity index is 20% and below, and</b> <b>(ii) Matching of individual sources listed must be less than 3% each, and</b> <b>(iii) Matching texts in continuous block must not exceed 8 words</b> <i>Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.</i>	

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

***Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.***

Signature of Supervisor

Name: Ts Dr Ooi Boon Yaik

Date: 2 September 2021

Signature of Co-Supervisor

Name: \_\_\_\_\_

Date: \_\_\_\_\_





# UNIVERSITI TUNKU ABDUL RAHMAN


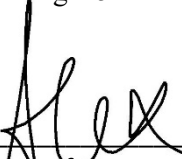
## FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY (KAMPAR CAMPUS)

### CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	19ACB00582
Student Name	WONG KEI YIN
Supervisor Name	Ts Dr OOI BOON YAIK

TICK (✓)	DOCUMENT ITEMS
	Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
N/A	Front Plastic Cover (for hardcopy)
✓	Title Page
✓	Signed Report Status Declaration Form
✓	Signed FYP Thesis Submission Form
✓	Signed form of the Declaration of Originality
✓	Acknowledgement
✓	Abstract
✓	Table of Contents
✓	List of Figures (if applicable)
✓	List of Tables (if applicable)
N/A	List of Symbols (if applicable)
✓	List of Abbreviations (if applicable)
✓	Chapters / Content
✓	Bibliography (or References)
✓	All references in bibliography are cited in the thesis, especially in the chapter of literature review
N/A	Appendices (if applicable)
✓	Weekly Log
✓	Poster
✓	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)

\*Include this form (checklist) in the thesis (Bind together as the last page)

<p>I, the author, have checked and confirmed all the items listed in the table are included in my report.</p> <p></p> <p>(Signature of Student)</p> <p>Date: 1 September 2021</p>	<p>Supervisor verification. Report with incorrect format can get 5 mark (1 grade) reduction.</p> <p></p> <p>(Signature of Supervisor)</p> <p>Date: 2 September 2021</p>
--	---