Ratchanont Treevijitpaisan (Fame) u7444134

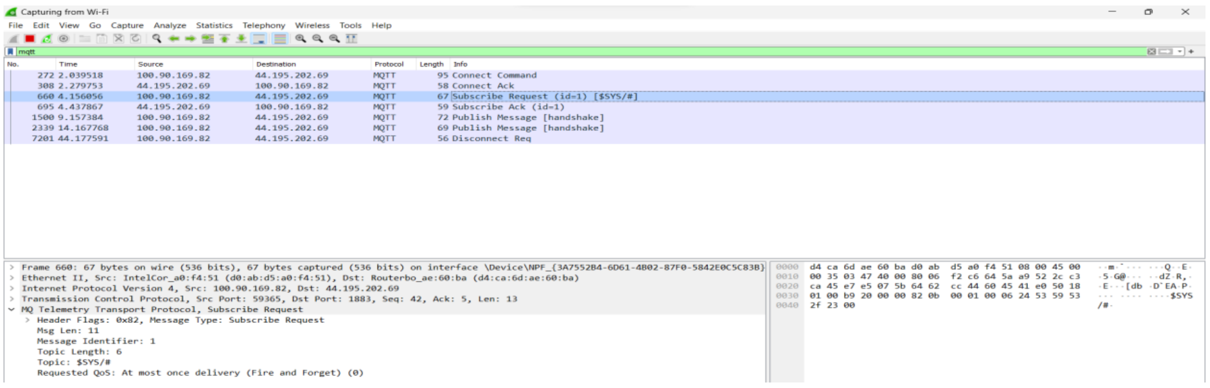
COMP3310/COMP6331 – Computer Networks

Assignment 3 – Testing MQTT

1. Subscribe to some broker, you can use any mqtt client you like for this, to retrieve some $SYS/# value. Wireshark the handshake for one example of each of the differing QoS-levels (0,1,2), include screenshots in your report that show the wireshark capture (filter for mqtt), and briefly explain how each QoS transfer works, and what it implies for message duplication and message order. Discuss briefly in your report in which circumstances would you choose each QoS level, and offer an example of a sensor that would use each of them. *[around 1 page]*

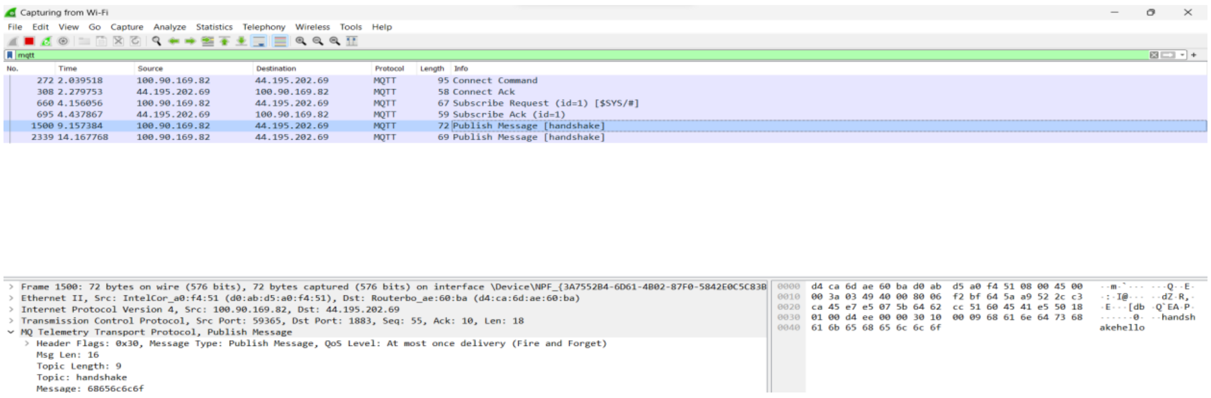
During the communication between mqtt client and mqtt broker, mqtt protocol has a Quality of Service (QoS) specification function while communicating. The Quality of Service (QoS) is an agreement between the mqtt client which in this case can be Publisher (Client who send some messages in some topics) and Subscribe (Client who subscribe to some topics) which defines the level that those messages will be delivered. There are 3 levels of QoS in mqtt.

First, QoS level 0 – at most once (The HiveMQ Team, 2015)


Capturing from Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
mqtt

```
No.   Time        Source          Destination     Protocol Length Info
  272 2.039518    100.90.169.82   44.195.202.69   MQTT     95 Connect Command
  308 2.279753    44.195.202.69   100.90.169.82   MQTT     58 Connect Ack
  660 4.156056    100.90.169.82   44.195.202.69   MQTT     67 Subscribe Request (id=1) [$SYS/#]
  695 4.437867    44.195.202.69   100.90.169.82   MQTT     59 Subscribe Ack (id=1)
 1500 9.157384    100.90.169.82   44.195.202.69   MQTT     72 Publish Message [handshake]
 2339 14.167768   100.90.169.82   44.195.202.69   MQTT     69 Publish Message [handshake]
 7201 44.177591   100.90.169.82   44.195.202.69   MQTT     56 Disconnect Req

> Frame 660: 67 bytes on wire (536 bits), 67 bytes captured (536 bits) on interface \Device\NPF_{3A7552B4-6D61-4B02-87F0-5842E0C5C83B}
> Ethernet II, Src: IntelCor_a0:f4:51 (d0:ab:d5:a0:f4:51), Dst: Routerbo_ae:60:ba (d4:ca:6d:ae:60:ba)
> Internet Protocol Version 4, Src: 100.90.169.82, Dst: 44.195.202.69
> Transmission Control Protocol, Src Port: 59365, Dst Port: 1883, Seq: 42, Ack: 5, Len: 13
v MQ Telemetry Transport Protocol, Subscribe Request
  > Header Flags: 0x82, Message Type: Subscribe Request
    Msg Len: 11
    Message Identifier: 1
    Topic Length: 6
    Topic: $SYS/#
    Requested QoS: At most once delivery (Fire and Forget) (0)

0000  d4 ca 6d ae 60 ba d0 ab  d5 a0 f4 51 08 00 45 00   ··m·`··· ···Q··E·
0010  00 35 03 47 40 00 80 06  f2 c6 64 5a a9 52 2c c3   ·5·G@··· ··dZ·R,·
0020  ca 45 e7 e5 07 5b 64 62  cc 44 60 45 41 e0 50 18   ·E···[db ·D`EA·P·
0030  01 00 b9 20 00 00 82 0b  00 01 00 06 24 53 59 53   ··· ···· ····$SYS
0040  2f 23 00                                           /#·
```

Figure 1: QoS 0 subscribe.


Capturing from Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
mqtt

```
No.   Time        Source          Destination     Protocol Length Info
  272 2.039518    100.90.169.82   44.195.202.69   MQTT     95 Connect Command
  308 2.279753    44.195.202.69   100.90.169.82   MQTT     58 Connect Ack
  660 4.156056    100.90.169.82   44.195.202.69   MQTT     67 Subscribe Request (id=1) [$SYS/#]
  695 4.437867    44.195.202.69   100.90.169.82   MQTT     59 Subscribe Ack (id=1)
 1500 9.157384    100.90.169.82   44.195.202.69   MQTT     72 Publish Message [handshake]
 2339 14.167768   100.90.169.82   44.195.202.69   MQTT     69 Publish Message [handshake]

> Frame 1500: 72 bytes on wire (576 bits), 72 bytes captured (576 bits) on interface \Device\NPF_{3A7552B4-6D61-4B02-87F0-5842E0C5C83B
> Ethernet II, Src: IntelCor_a0:f4:51 (d0:ab:d5:a0:f4:51), Dst: Routerbo_ae:60:ba (d4:ca:6d:ae:60:ba)
> Internet Protocol Version 4, Src: 100.90.169.82, Dst: 44.195.202.69
> Transmission Control Protocol, Src Port: 59365, Dst Port: 1883, Seq: 55, Ack: 10, Len: 18
v MQ Telemetry Transport Protocol, Publish Message
  > Header Flags: 0x30, Message Type: Publish Message, QoS Level: At most once delivery (Fire and Forget)
    Msg Len: 16
    Topic Length: 9
    Topic: handshake
    Message: 68656c6c6f

0000  d4 ca 6d ae 60 ba d0 ab  d5 a0 f4 51 08 00 45 00   ··m·`··· ···Q··E·
0010  00 3a 03 49 40 00 80 06  f2 bf 64 5a a9 52 2c c3   ·:·I@··· ··dZ·R,·
0020  ca 45 e7 e5 07 5b 64 62  cc 51 60 45 41 e5 50 18   ·E···[db ·Q`EA·P·
0030  01 00 d4 ee 00 00 30 10  00 09 68 61 6e 64 73 68   ······0· ··handsh
0040  61 6b 65 68 65 6c 6c 6f                            akehello
```

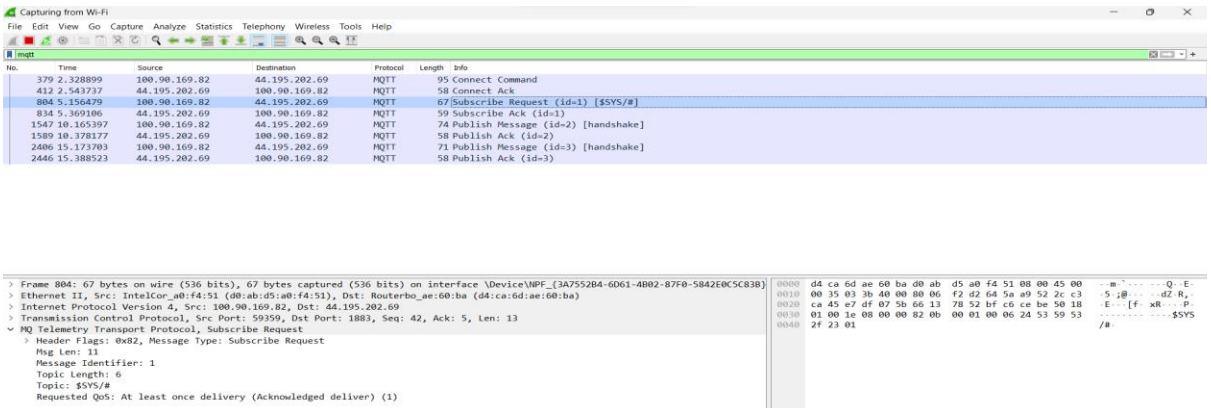
Figure 2: QoS 0 publish.

QoS level 0 is the lowest level services which not guaranteed that the receiver will get the publishing messages. The broker will not store the publishing messages on this QoS levels that labelling flag as Fire and Forget which acknowledges and immediately flush out this message to the subscribers. Subscriber might not retrieve the message but in trade off with the speed.

QoS 0 circumstances criteria (Steve, 2022):

- The communication between publisher, subscriber and broker are held on the private network area such as LAN or personal home server which have a stable connection.
- The sending messages are not at high priority, messages losses are acceptable.
- Messages are not required to send or receive in order or perfect queuing.

Secondly, QoS level 1 – at least once (The HiveMQ Team, 2015)

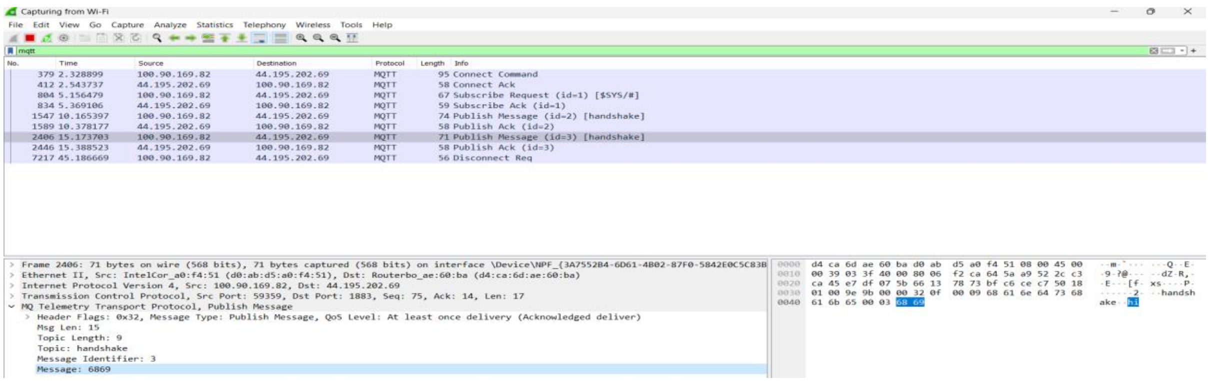

Capturing from Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
mqtt

```
No.    Time       Source          Destination     Protocol Length Info
   379 2.328899   100.90.169.82   44.195.202.69   MQTT       95 Connect Command
   412 2.543737   44.195.202.69   100.90.169.82   MQTT       58 Connect Ack
   804 5.156479   100.90.169.82   44.195.202.69   MQTT       67 Subscribe Request (id=1) [$SYS/#]
   834 5.369106   44.195.202.69   100.90.169.82   MQTT       59 Subscribe Ack (id=1)
  1547 10.165397  100.90.169.82   44.195.202.69   MQTT       74 Publish Message (id=2) [handshake]
  1589 10.378177  44.195.202.69   100.90.169.82   MQTT       58 Publish Ack (id=2)
  2406 15.173703  100.90.169.82   44.195.202.69   MQTT       71 Publish Message (id=3) [handshake]
  2446 15.388523  44.195.202.69   100.90.169.82   MQTT       58 Publish Ack (id=3)
```

```
> Frame 804: 67 bytes on wire (536 bits), 67 bytes captured (536 bits) on interface \Device\NPF_{3A7552B4-6D61-4B02-87F0-5842E0C5C83B}
> Ethernet II, Src: IntelCor_a0:f4:51 (d0:ab:d5:a0:f4:51), Dst: Routerbo_ae:60:ba (d4:ca:6d:ae:60:ba)
> Internet Protocol Version 4, Src: 100.90.169.82, Dst: 44.195.202.69
> Transmission Control Protocol, Src Port: 59359, Dst Port: 1883, Seq: 42, Ack: 5, Len: 13
v MQ Telemetry Transport Protocol, Subscribe Request
  > Header Flags: 0x82, Message Type: Subscribe Request
    Msg Len: 11
    Message Identifier: 1
    Topic Length: 6
    Topic: $SYS/#
    Requested QoS: At least once delivery (Acknowledged deliver) (1)
```

```
0000  d4 ca 6d ae 60 ba d0 ab  d5 a0 f4 51 08 00 45 00   ··m·`···  ···Q··E·
0010  00 35 03 3b 40 00 80 06  f2 d2 64 5a a9 52 2c c3   ·5·;@···  ··dZ·R,·
0020  ca 45 e7 df 07 5b 66 13  78 52 bf c6 ce be 50 18   ·E···[f·  xR····P·
0030  01 00 1e 08 00 00 82 0b  00 01 00 06 24 53 59 53   ········  ····$SYS
0040  2f 23 01                                           /#·
```

Figure 3: QoS 1 subscribe.


Capturing from Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
mqtt

```
No.    Time       Source          Destination     Protocol Length Info
   379 2.328899   100.90.169.82   44.195.202.69   MQTT       95 Connect Command
   412 2.543737   44.195.202.69   100.90.169.82   MQTT       58 Connect Ack
   804 5.156479   100.90.169.82   44.195.202.69   MQTT       67 Subscribe Request (id=1) [$SYS/#]
   834 5.369106   44.195.202.69   100.90.169.82   MQTT       59 Subscribe Ack (id=1)
  1547 10.165397  100.90.169.82   44.195.202.69   MQTT       74 Publish Message (id=2) [handshake]
  1589 10.378177  44.195.202.69   100.90.169.82   MQTT       58 Publish Ack (id=2)
  2406 15.173703  100.90.169.82   44.195.202.69   MQTT       71 Publish Message (id=3) [handshake]
  2446 15.388523  44.195.202.69   100.90.169.82   MQTT       58 Publish Ack (id=3)
  7217 45.186669  100.90.169.82   44.195.202.69   MQTT       56 Disconnect Req
```

```
> Frame 2406: 71 bytes on wire (568 bits), 71 bytes captured (568 bits) on interface \Device\NPF_{3A7552B4-6D61-4B02-87F0-5842E0C5C83B
> Ethernet II, Src: IntelCor_a0:f4:51 (d0:ab:d5:a0:f4:51), Dst: Routerbo_ae:60:ba (d4:ca:6d:ae:60:ba)
> Internet Protocol Version 4, Src: 100.90.169.82, Dst: 44.195.202.69
> Transmission Control Protocol, Src Port: 59359, Dst Port: 1883, Seq: 75, Ack: 14, Len: 17
v MQ Telemetry Transport Protocol, Publish Message
  > Header Flags: 0x32, Message Type: Publish Message, QoS Level: At least once delivery (Acknowledged deliver)
    Msg Len: 15
    Topic Length: 9
    Topic: handshake
    Message Identifier: 3
    Message: 6869
```

```
0000  d4 ca 6d ae 60 ba d0 ab  d5 a0 f4 51 08 00 45 00   ··m·`···  ···Q··E·
0010  00 39 03 3f 40 00 80 06  f2 ca 64 5a a9 52 2c c3   ·9·?@···  ··dZ·R,·
0020  ca 45 e7 df 07 5b 66 13  78 73 bf c6 ce c7 50 18   ·E···[f·  xs····P·
0030  01 00 9e 9b 00 00 32 0f  00 09 68 61 6e 64 73 68   ······2·  ··handsh
0040  61 6b 65 00 03 68 69                               ake··hi
```

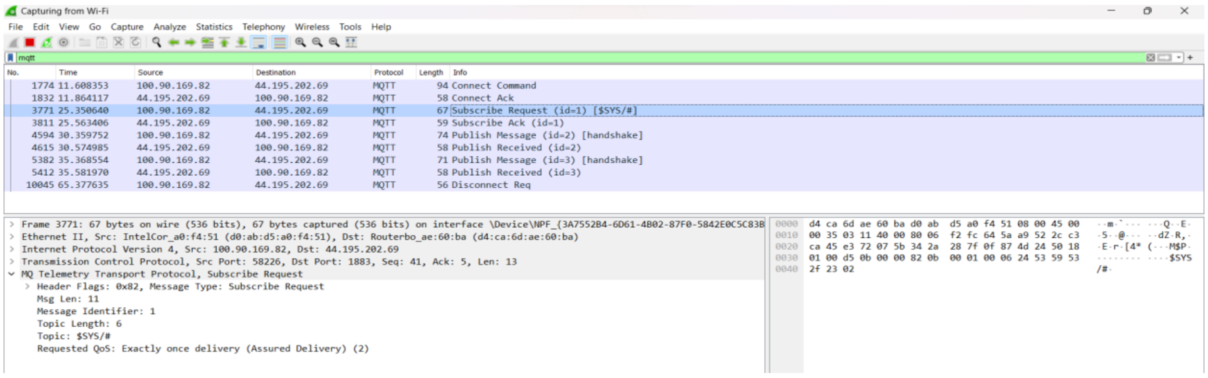
Figure 4: QoS 1 publish.

QoS level 1 guarantees that the messages will receive by subscribers “at least once” which in some circumstances it might causes the duplication of the delivered messages due to the method behind this QoS level which is broker will provide the PUBACK packages which in the figure 4 is id 2 or id 3 when the messages is published to remind the publisher that the receiver got the messages. In addition, the duplication messages event might occur because the delay of PUBACK packages to acknowledges the publisher.

QoS 1 circumstances criteria (The HiveMQ Team, 2015):

- The duplication of messages is acceptable which trade off in speed compared to QoS 2
- The guaranteed messages acknowledgement on subscriber, and it can handle the duplicate data

Consequently, QoS level 2 – exactly once (The HiveMQ Team, 2015)

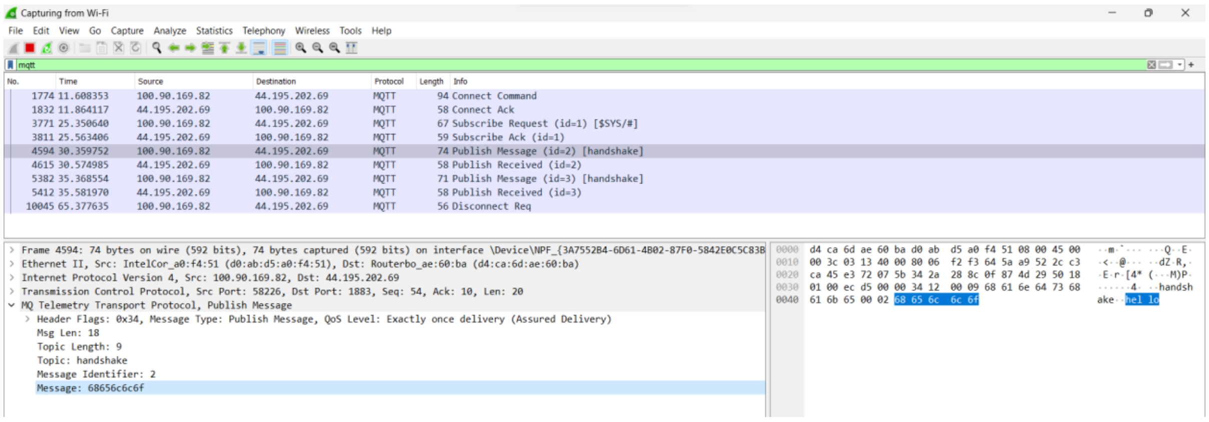

Capturing from Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
mqtt

```
No.    Time       Source          Destination     Protocol Length Info
 1774 11.608353  100.90.169.82   44.195.202.69   MQTT     94 Connect Command
 1832 11.864117  44.195.202.69   100.90.169.82   MQTT     58 Connect Ack
 3771 25.350640  100.90.169.82   44.195.202.69   MQTT     67 Subscribe Request (id=1) [$SYS/#]
 3811 25.563406  44.195.202.69   100.90.169.82   MQTT     59 Subscribe Ack (id=1)
 4594 30.359752  100.90.169.82   44.195.202.69   MQTT     74 Publish Message (id=2) [handshake]
 4615 30.574985  44.195.202.69   100.90.169.82   MQTT     58 Publish Received (id=2)
 5382 35.368554  100.90.169.82   44.195.202.69   MQTT     71 Publish Message (id=3) [handshake]
 5412 35.581970  44.195.202.69   100.90.169.82   MQTT     58 Publish Received (id=3)
10045 65.377635  100.90.169.82   44.195.202.69   MQTT     56 Disconnect Req
```

```
> Frame 3771: 67 bytes on wire (536 bits), 67 bytes captured (536 bits) on interface \Device\NPF_{3A7552B4-6D61-4B02-87F0-5842E0C5C83B
> Ethernet II, Src: IntelCor_a0:f4:51 (d0:ab:d5:a0:f4:51), Dst: Routerbo_ae:60:ba (d4:ca:6d:ae:60:ba)
> Internet Protocol Version 4, Src: 100.90.169.82, Dst: 44.195.202.69
> Transmission Control Protocol, Src Port: 58226, Dst Port: 1883, Seq: 41, Ack: 5, Len: 13
v MQ Telemetry Transport Protocol, Subscribe Request
  > Header Flags: 0x82, Message Type: Subscribe Request
    Msg Len: 11
    Message Identifier: 1
    Topic Length: 6
    Topic: $SYS/#
    Requested QoS: Exactly once delivery (Assured Delivery) (2)
```

```
0000  d4 ca 6d ae 60 ba d0 ab  d5 a0 f4 51 08 00 45 00   ··m·`··· ···Q··E·
0010  00 35 03 11 40 00 80 06  f2 fc 64 5a a9 52 2c c3   ·5··@··· ··dZ·R,·
0020  ca 45 e3 72 07 5b 34 2a  28 7f 0f 87 4d 24 50 18   ·E·r·[4* (···M$P·
0030  01 00 d5 0b 00 00 82 0b  00 01 00 06 24 53 59 53   ········ ····$SYS
0040  2f 23 02                                           /#·
```

Figure 5: QoS 2 subscribe.


Capturing from Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
mqtt

```
No.    Time       Source          Destination     Protocol Length Info
 1774 11.608353  100.90.169.82   44.195.202.69   MQTT     94 Connect Command
 1832 11.864117  44.195.202.69   100.90.169.82   MQTT     58 Connect Ack
 3771 25.350640  100.90.169.82   44.195.202.69   MQTT     67 Subscribe Request (id=1) [$SYS/#]
 3811 25.563406  44.195.202.69   100.90.169.82   MQTT     59 Subscribe Ack (id=1)
 4594 30.359752  100.90.169.82   44.195.202.69   MQTT     74 Publish Message (id=2) [handshake]
 4615 30.574985  44.195.202.69   100.90.169.82   MQTT     58 Publish Received (id=2)
 5382 35.368554  100.90.169.82   44.195.202.69   MQTT     71 Publish Message (id=3) [handshake]
 5412 35.581970  44.195.202.69   100.90.169.82   MQTT     58 Publish Received (id=3)
10045 65.377635  100.90.169.82   44.195.202.69   MQTT     56 Disconnect Req
```

```
> Frame 4594: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\NPF_{3A7552B4-6D61-4B02-87F0-5842E0C5C83B
> Ethernet II, Src: IntelCor_a0:f4:51 (d0:ab:d5:a0:f4:51), Dst: Routerbo_ae:60:ba (d4:ca:6d:ae:60:ba)
> Internet Protocol Version 4, Src: 100.90.169.82, Dst: 44.195.202.69
> Transmission Control Protocol, Src Port: 58226, Dst Port: 1883, Seq: 54, Ack: 10, Len: 20
v MQ Telemetry Transport Protocol, Publish Message
  > Header Flags: 0x34, Message Type: Publish Message, QoS Level: Exactly once delivery (Assured Delivery)
    Msg Len: 18
    Topic Length: 9
    Topic: handshake
    Message Identifier: 2
    Message: 68656c6c6f
```

```
0000  d4 ca 6d ae 60 ba d0 ab  d5 a0 f4 51 08 00 45 00   ··m·`··· ···Q··E·
0010  00 3c 03 13 40 00 80 06  f2 f3 64 5a a9 52 2c c3   ·<··@··· ··dZ·R,·
0020  ca 45 e3 72 07 5b 34 2a  28 8c 0f 87 4d 29 50 18   ·E·r·[4* (···M)P·
0030  01 00 ec d5 00 00 34 12  00 09 68 61 6e 64 73 68   ······4· ··handsh
0040  61 6b 65 00 02 68 65 6c  6c 6f                     ake··hel lo
```

Figure 6: QoS 2 publish.

QoS level 2 is the most robustness communication level in Mqtt which the messages are guaranteed that the subscriber will receive exactly once messages due to four handshake behind the mqtt level 2 protocol which receiver will reply on the messages as PUBREC packages which in QoS 1 if it caught delay publisher will re send the message again which cause some duplication, but in case if the receiver collect the right data publisher will acknowledge PUBREL package to drop off all duplicated messages and acknowledge back to publisher with PUBCOMP packages for finish discarding unnecessary message and perfectly receive the right information as the picture below.

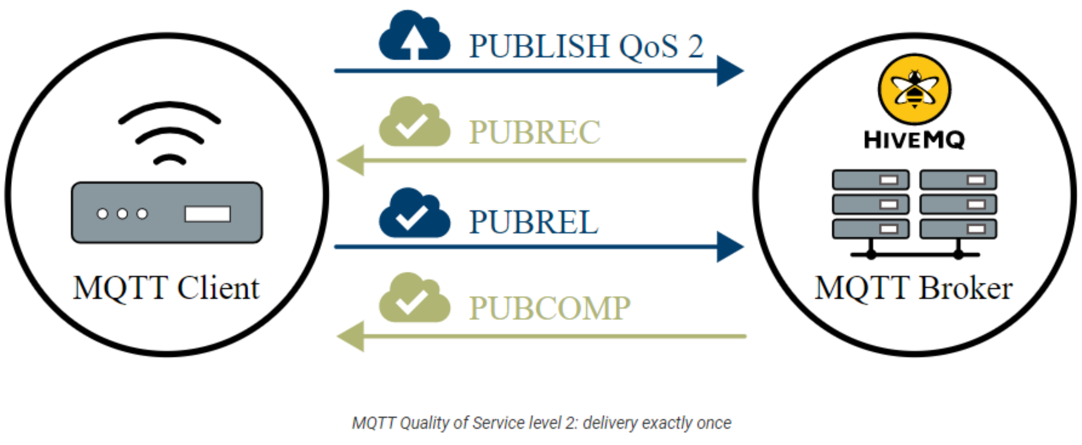


Figure 7: QoS 2 4 handshake

QoS 2 circumstances criteria (The HiveMQ Team, 2015):

- Required a precise delivering messages because message duplication might cause negative consequences.
- Perfect queuing messages due to slow and precise communication

Real world use cases for each QoS level (Susie Cheong, 2022)

QoS level 0 is suitable for some sensors in smart home due to the requirement on the stable connection and the chances on messages loss such as humidity, pressure, temperature sensors which normally used in smart home or sensor that use for counting people in the grocery store.

QoS level 1 is a high speed which guaranteed the message will be received with some issues on messages duplication mostly used in driving such as fasten seat belt sign alert sensor due to repeatedly warn driver to response.

QoS level 2 is the most robust and precise communication level which usually appear on the industry that need to automate with the perfect queue and correct data. For instance, robotic industry for precise move action.

2. **Run your Analyser against your partner's broker**, have it tell the broker (and so **their** Controller) what you want **their** Publisher to send, and record information for 1 minute for each set of the qos/delay/instance-count parameters.
   (*Tips: (i) only print to screen for debugging, otherwise it will slow your code down a lot. (ii) Check with the broker owner what delay there might be between your request and their publisher responding. (iii) Use the counter values to tell you what messages are arriving, or are not arriving as expected, and what values they start/stop at when you subscribe to calculate the rates below. (iv) try publishing at a very low rate (i.e. high-delay) to start with, to check your code)*

   a. Collect statistics, for each instance-count, delay and QoS level, and measure over the 1-minute period:

In the question 2, this analyzer was operating with another colleague mqtt broker local host (mosquitto) on personal hotspot. In addition, this analyzer also implements and test with the public broker (emqx) but will discuss on question 3. The analyzer was run with 60 second.

Friend 1 instance count

Instance 1 / QoS 0

QoS 0	Avg msg (msg/s)	loss msg (%)	out of order msg (%)	mean gap (ms)	median gap (ms)
0.0 s	2367.483333	0	0	4.19E-04	0
0.01 s	63.76666667	0.03	0	0.015543268	0.016
0.02 s	32.01666667	0.05	0	0.031038542	0.031
0.04 s	21.3	0.08	0	0.046642913	0.047
0.08 s	10.65	0.16	0	0.093163009	0.094
0.16 s	5.816666667	0.29	0	0.171247126	0.172
0.32 s	3.05	0.55	0	0.326582418	0.328

Instance 1 / QoS 1

QoS 1	Avg msg (msg/s)	loss msg (%)	out of order msg (%)	mean gap (ms)	median gap (ms)
0.0 s	2778.816667	0	0	3.57E-04	0
0.01 s	63.86666667	0.03	0	0.015543461	0.016
0.02 s	31.86666667	0.05	0	0.031094715	0.031
0.04 s	21.23333333	0.08	0	0.046666143	0.047
0.08 s	10.63333333	0.16	0	0.093210361	0.094
0.16 s	5.816666667	0.29	0	0.170752874	0.172
0.32 s	3.05	0.55	0	0.326489011	0.328

Instance 1 / QoS 2

QoS 2	Avg msg (msg/s)	loss msg (%)	out of order msg (%)	mean gap (ms)	median gap (ms)
0.0 s	1736.183333	0	0	5.70E-04	0
0.01 s	63.73333333	0.03	0	0.015539105	0.016
0.02 s	31.85	0.05	0	0.03107801	0.031
0.04 s	21.25	0.08	0	0.046616954	0.047
0.08 s	10.6	0.16	0	0.093233071	0.094
0.16 s	5.8	0.29	0	0.170974063	0.172
0.32 s	3.033333333	0.55	0	0.326657459	0.328

Friend 2 instance count

Instance 1 / QoS 0

QoS 0/1	Avg msg (msg/s)	loss msg (%)	out of order msg (%)	mean gap (ms)	median gap (ms)
0.0 s	1899.883333	0	0	5.21E-04	0
0.01 s	63.98333333	0.03	0	0.015555758	0.016
0.02 s	31.91666667	0.05	0	0.031078892	0.031
0.04 s	21.3	0.08	0	0.046617854	0.047
0.08 s	10.63333333	0.16	0	0.093185243	0.094
0.16 s	5.8	0.29	0	0.171020173	0.172
0.32 s	3.05	0.55	0	0.326582418	0.328

Instance 2 / QoS 0

QoS 0/2	Avg msg (msg/s)	loss msg (%)	out of order msg (%)	mean gap (ms)	median gap (ms)
0.0 s	1928.866667	0	0	5.13E-04	0
0.01 s	63.98333333	0.03	0	0.015555758	0.016
0.02 s	31.91666667	0.05	0	0.031078892	0.031
0.04 s	21.3	0.08	0	0.046617854	0.047
0.08 s	10.63333333	0.16	0	0.093185243	0.094
0.16 s	5.816666667	0.29	0	0.170977011	0.172
0.32 s	3.05	0.55	0	0.326576923	0.328

Instance 1 / QoS 1

QoS 1/1	Avg msg (msg/s)	loss msg (%)	out of order msg (%)	mean gap (ms)	median gap (ms)
0.0 s	1347.633333	0	0	7.36E-04	0
0.01 s	63.11666667	0.03	0	0.015558901	0.016
0.02 s	31.65	0.05	0	0.031101686	0.031
0.04 s	21.15	0.08	0	0.046628549	0.04
0.08 s	10.63333333	0.16	0	0.093210361	0.094
0.16 s	5.816666667	0.29	0	0.170931034	0.172
0.32 s	3.033333333	0.55	0	0.32639779	0.328

Instance 2 / QoS 1

QoS 1/2	Avg msg (msg/s)	loss msg (%)	out of order msg (%)	mean gap (ms)	median gap (ms)
0.0 s	1361.383333	0	0	7.29E-04	0
0.01 s	63.13333333	0.03	0	0.015558754	0.016
0.02 s	31.65	0.05	0	0.031101686	0.031
0.04 s	21.16666667	0.08	0	0.046628842	0.04
0.08 s	10.63333333	0.16	0	0.093210361	0.094
0.16 s	5.816666667	0.29	0	0.170931034	0.172
0.32 s	3.033333333	0.55	0	0.32639779	0.328

Instance 1 / QoS 2

QoS 2/1	Avg msg (msg/s)	loss msg (%)	out of order msg (%)	mean gap (ms)	median gap (ms)
0.0 s	765.2333333	0	0	1.28E-03	0
0.01 s	63.65	0.03	0	0.015563646	0.016
0.02 s	31.51666667	0.05	0	0.031133862	0.031
0.04 s	20.85	0.08	0	0.046912	0.047
0.08 s	10.51666667	0.16	0	0.093253968	0.094
0.16 s	5.766666667	0.29	0	0.17106087	0.172
0.32 s	3	0.56	0	0.326731844	0.328

Instance 2/ QoS 2

QoS 2/2	Avg msg (msg/s)	loss msg (%)	out of order msg (%)	mean gap (ms)	median gap (ms)
0.0 s	746.3833333	0	0	1.31E-03	0
0.01 s	63.66666667	0.03	0	0.01556376	0.016
0.02 s	31.51666667	0.05	0	0.031133862	0.031
0.04 s	20.83333333	0.08	0	0.04696237	0.047
0.08 s	10.51666667	0.16	0	0.093253968	0.094
0.16 s	5.766666667	0.29	0	0.17106087	0.172
0.32 s	3	0.56	0	0.326642458	0.328

Ratchanont Treevijitpaisan (Fame) u7444134

Friend 3 instance count

Instance 1 / QoS 0

QoS 0/1	Avg msg (msg/s)	loss msg (%)	out of order msg (%)	mean gap (ms)	median gap (ms)
0.0 s	1353.183333	0	0	7.30E-04	0
0.01 s	63.31666667	0.03	0	0.015604265	0.016
0.02 s	31.9	0.05	0	0.031143753	0.031
0.04 s	21.21666667	0.08	0	0.046764151	0.047
0.08 s	10.63333333	0.16	0	0.093653061	0.094
0.16 s	5.75	0.29	0	0.17105814	0.172
0.32 s	3.033333333	0.55	0	0.326657459	0.328

Instance 2 / QoS 0

QoS 0/2	Avg msg (msg/s)	loss msg (%)	out of order msg (%)	mean gap (ms)	median gap (ms)
0.0 s	1350.916667	0	0	7.31E-04	0
0.01 s	63.33333333	0.03	0	0.015600158	0.016
0.02 s	31.88333333	0.05	0	0.031151674	0.031
0.04 s	21.21666667	0.08	0	0.04677673	0.047
0.08 s	10.63333333	0.16	0	0.093653061	0.094
0.16 s	5.75	0.29	0	0.17105814	0.172
0.32 s	3.033333333	0.55	0	0.326569061	0.328

Instance 3 / QoS 0

QoS 0/3	Avg msg (msg/s)	loss msg (%)	out of order msg (%)	mean gap (ms)	median gap (ms)
0.0 s	1344.5	0	0	7.35E-04	0
0.01 s	63.35	0.03	0	0.015604474	0.016
0.02 s	31.9	0.05	0	0.031143753	0.031
0.04 s	21.21666667	0.08	0	0.046752358	0.047
0.08 s	10.63333333	0.16	0	0.093653061	0.094
0.16 s	5.75	0.29	0	0.17105814	0.172
0.32 s	3.033333333	0.55	0	0.326657459	0.328

Instance 1 / QoS 1

QoS 1/1	Avg msg (msg/s)	loss msg (%)	out of order msg (%)	mean gap (ms)	median gap (ms)
0.0 s	655.2833333	1.58	0.01309866	1.51E-03	0
0.01 s	63.75	0.03	0	0.015612709	0.016
0.02 s	31.75	0.05	0	0.031127101	0.031
0.04 s	20.93333333	0.08	0	0.046725896	0.047
0.08 s	10.66666667	0.16	0	0.093333333	0.094
0.16 s	5.816666667	0.29	0	0.171155172	0.172
0.32 s	3.033333333	0.55	0	0.326828729	0.328

Instance 2 / QoS 1

QoS 1/2	Avg msg (msg/s)	loss msg (%)	out of order msg (%)	mean gap (ms)	median gap (ms)
0.0 s	638.1666667	1.67	0.013476103	1.54E-03	0
0.01 s	63.76666667	0.03	0	0.015604444	0.016
0.02 s	31.75	0.05	0	0.031127101	0.031
0.04 s	20.95	0.08	0	0.046725318	0.047
0.08 s	10.66666667	0.16	0	0.093309859	0.094
0.16 s	5.816666667	0.29	0	0.171155172	0.172
0.32 s	3.05	0.55	0	0.326752747	0.328

Instance 3 / QoS 1

QoS 1/3	Avg msg (msg/s)	loss msg (%)	out of order msg (%)	mean gap (ms)	median gap (ms)
0.0 s	643.8333333	1.56	0.013176288	1.53E-03	0
0.01 s	63.65	0.03	0	0.015633054	0.016
0.02 s	31.75	0.05	0	0.031127101	0.031
0.04 s	20.93333333	0.08	0	0.046725896	0.047
0.08 s	10.68333333	0.16	0	0.093310938	0.094
0.16 s	5.816666667	0.29	0	0.171158046	0.172
0.32 s	3.05	0.55	0	0.326752747	0.328

Instance 1 / QoS 2

QoS 2/1	Avg msg (msg/s)	loss msg (%)	out of order msg (%)	mean gap (ms)	median gap (ms)
0.0 s	376.0833333	10.8	0.097318857	2.42E-03	0
0.01 s	61.95	0.03	0	0.015629171	0.016
0.02 s	31.43333333	0.05	0	0.031092308	0.031
0.04 s	20.88333333	0.08	0	0.04675	0.047
0.08 s	10.46666667	0.16	0	0.093475279	0.094
0.16 s	5.7	0.29	0	0.170956012	0.172
0.32 s	2.966666667	0.56	0	0.326627119	0.328

Instance 2 / QoS 2

QoS 2/2	Avg msg (msg/s)	loss msg (%)	out of order msg (%)	mean gap (ms)	median gap (ms)
0.0 s	369.7	10.69	0.096159048	2.44E-03	0
0.01 s	61.95	0.03	0	0.015629171	0.016
0.02 s	31.43333333	0.05	0	0.031092308	0.031
0.04 s	20.88333333	0.08	0	0.04675	0.047
0.08 s	10.46666667	0.16	0	0.093475279	0.094
0.16 s	5.7	0.29	0	0.170956012	0.172
0.32 s	2.966666667	0.56	0	0.32680226	0.328

Instance 3 / QoS 2

QoS 2/3	Avg msg (msg/s)	loss msg (%)	out of order msg (%)	mean gap (ms)	median gap (ms)
0.0 s	371.15	11.03	0.09802865	2.43E-03	0
0.01 s	61.96666667	0.03	0	0.015629271	0.016
0.02 s	31.43333333	0.05	0	0.031092308	0.031
0.04 s	20.9	0.08	0	0.04673743	0.047
0.08 s	10.46666667	0.16	0	0.093475279	0.094
0.16 s	5.7	0.29	0	0.170956012	0.172
0.32 s	2.966666667	0.56	0	0.326887006	0.328

b. While measuring the above also
   i. Subscribe to and record the $SYS topics, and identify what, if anything, on the broker do any loss/misordered-rates correlate with. *(Look at measurements under e.g. 'load', as well as the 'heap' and 'active clients' and 'messages', anything that seems relevant. See https://mosquitto.org/man/mosquitto-8.html though it may not quite match your broker)*

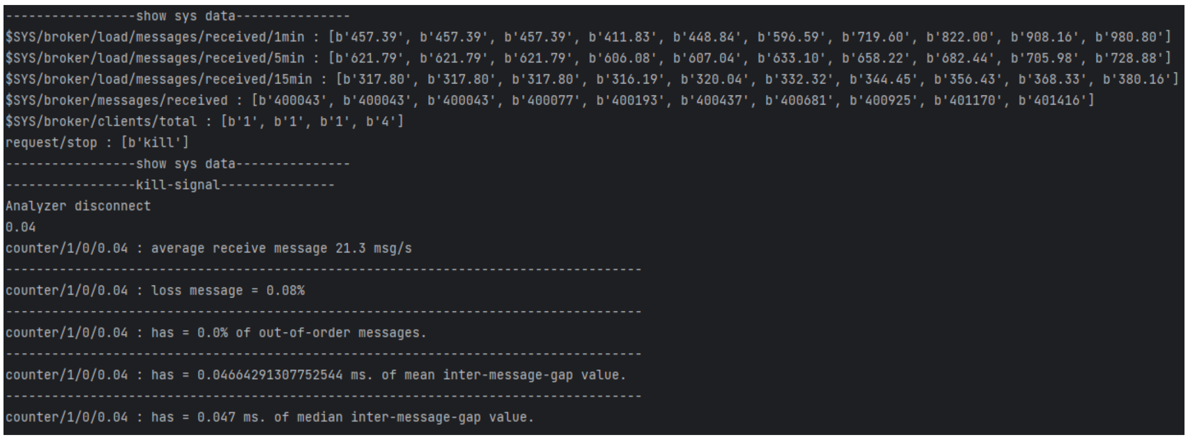
Based on the mosquito $SYS topics (Eclipse Mosquitto, 2019)

On topic $SYS/broker/load/messages/received, broker track the moving average of the number of bytes corresponding to the time different time interval based on broker setting which in this case is 1, 5, 15 minutes.

On topic $SYS/broker/messages/received, broker track down the total number of messages since the broker established.

On topic $SYS/heap/#, contains two subtopics which is current size & maximum size. In addition, it demonstrated the heap memory that used by mosquito. Unfortunately, heap topics is unavailable due to compile time options.

On topic $SYS/broker/clients/total, is a useful when running on public broker to demonstrate all the client that connected to the broker which can use to interpret the huge different between public and private broker, In addition, client subtopics $SYS/broker/clients/connected will show the number of currently active clients which will be a decent choice to demonstrated the publisher instance count while conduct the analysis.

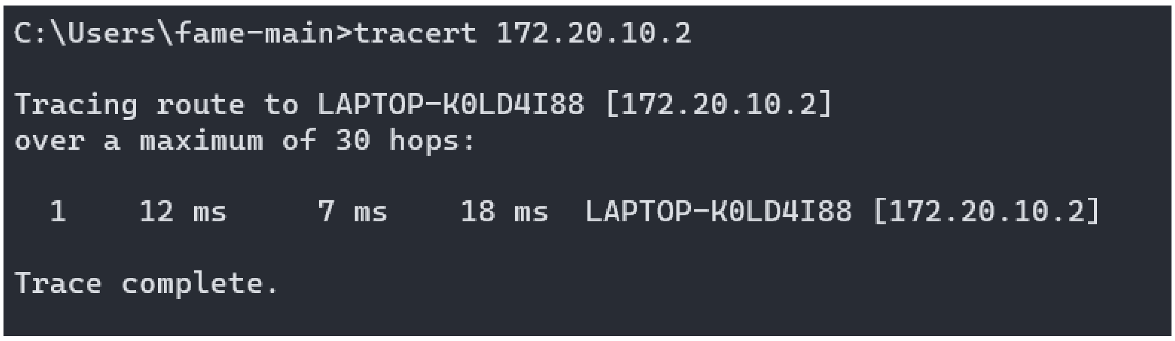


```
-----------------show sys data---------------
$SYS/broker/load/messages/received/1min : [b'457.39', b'457.39', b'457.39', b'411.83', b'448.84', b'596.59', b'719.60', b'822.00', b'908.16', b'980.80']
$SYS/broker/load/messages/received/5min : [b'621.79', b'621.79', b'621.79', b'606.08', b'607.04', b'633.10', b'658.22', b'682.44', b'705.98', b'728.88']
$SYS/broker/load/messages/received/15min : [b'317.80', b'317.80', b'317.80', b'316.19', b'320.04', b'332.32', b'344.45', b'356.43', b'368.33', b'380.16']
$SYS/broker/messages/received : [b'400043', b'400043', b'400043', b'400077', b'400193', b'400437', b'400681', b'400925', b'401170', b'401416']
$SYS/broker/clients/total : [b'1', b'1', b'1', b'4']
request/stop : [b'kill']
-----------------show sys data---------------
-----------------kill-signal---------------
Analyzer disconnect
0.04
counter/1/0/0.04 : average receive message 21.3 msg/s
--------------------------------------------------------------------------------
counter/1/0/0.04 : loss message = 0.08%
--------------------------------------------------------------------------------
counter/1/0/0.04 : has = 0.0% of out-of-order messages.
--------------------------------------------------------------------------------
counter/1/0/0.04 : has = 0.04664291307752544 ms. of mean inter-message-gap value.
--------------------------------------------------------------------------------
counter/1/0/0.04 : has = 0.047 ms. of median inter-message-gap value.
```

Figure 8: $SYS analyzer demonstrated bytes data from $SYS topics.

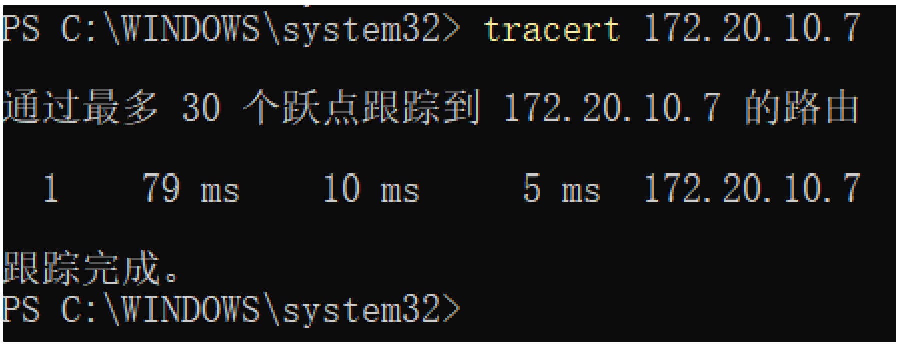
ii. Record a traceroute between the Analyser and the broker, and between the Publisher and the broker *(you'll need the broker owner to provide the second one)*

The analyzed result was conduct on personal hotspot on peers mosquito broker, controller and publisher which the trace router hop will definitely not over 3 – 5 hops because testing were conduct in person. There are also some tracings while testing with public broker and cloud broker as well. The picture below shows the traceroute hop on author side and peers' side while running the test.



```
C:\Users\fame-main>tracert 172.20.10.2

Tracing route to LAPTOP-K0LD4I88 [172.20.10.2]
over a maximum of 30 hops:

  1    12 ms     7 ms    18 ms  LAPTOP-K0LD4I88 [172.20.10.2]

Trace complete.
```

Figure 9: Analyzer side traceroute.



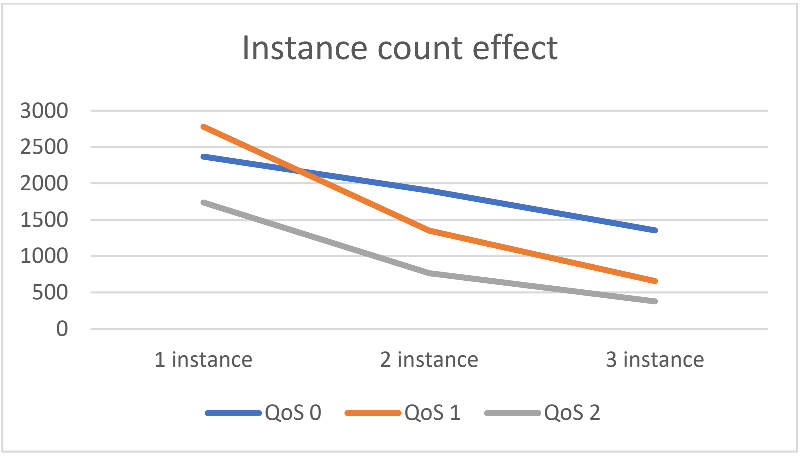
```
PS C:\WINDOWS\system32> tracert 172.20.10.7

通过最多 30 个跃点跟踪到 172.20.10.7 的路由

  1    79 ms    10 ms     5 ms  172.20.10.7

跟踪完成。
PS C:\WINDOWS\system32>
```

Figure 10: Publisher and Controller traceroute on colleague.

3. In your report: *[around 2-3 pages of text, plus any charts]*
    a. Summarise your measurements. Explain what you expected to see, especially in relation to the different QoS levels, and whether your expectations were matched.

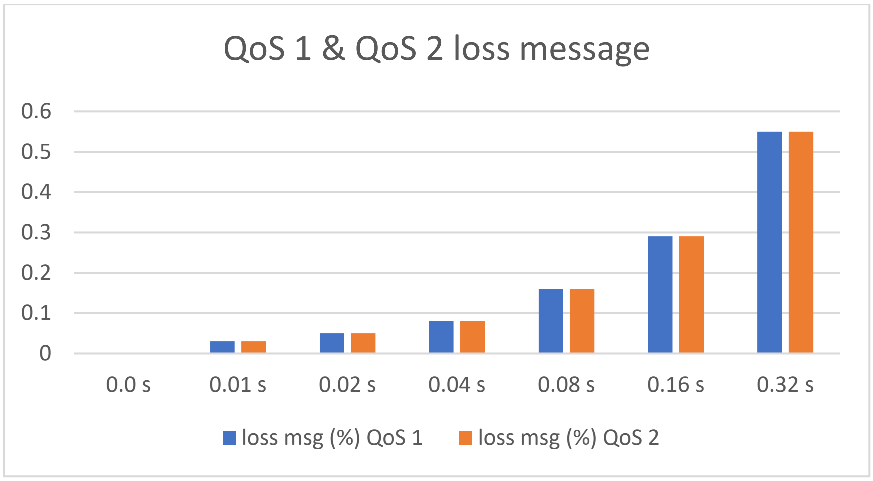
Firstly, there are a statement on the multiple publisher that will stress the broker and network which referred to mqtt client v3 which build on TCP on the network side, multiple connections of publisher will obviously lead to multiple handshake connections establishment which broker need to handle multiple connections which probably cause bottlenecks, performance drop, message lost, error, etc. On the broker side, as we subscribe and also analyze the data from the $SYS topics, broker also need to spend more resources to handle multiple publishing messages and handle on the specific packet on QoS 1 and QoS 2 which needed to be received by the subscriber.

Delay 0.0	Avg msg (msg/s) 1 instance	Avg msg (msg/s) 2 instance	Avg msg (msg/s) 3 instance
QoS 0	2367.483333	1899.88333	1353.183333
QoS 1	2778.816667	1347.63333	655.2833333
QoS 2	1736.183333	765.233333	376.0833333



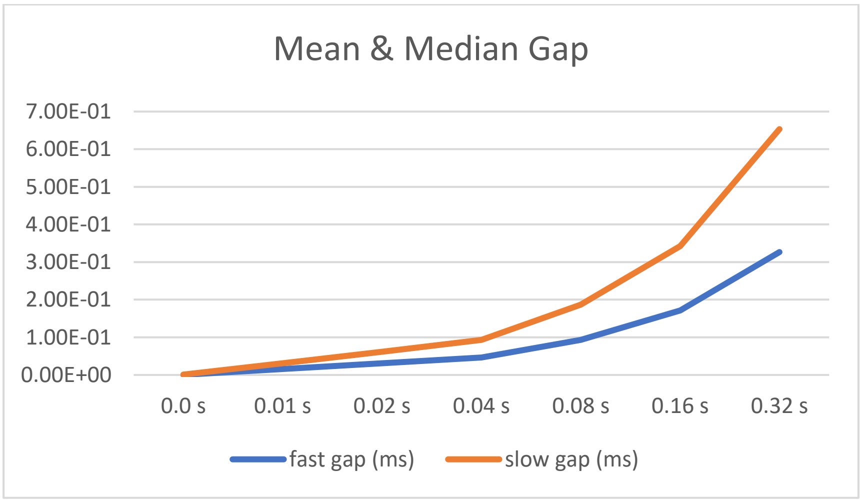
Based on the table above, it obvious that the average message continuously decreasing corresponds to the increased of the publisher. On QoS level 0 rate is stably decreasing due to the fire and forget delivery method which didn't required to set an establishment for handshake and guaranteed message delivered. The broker only handles the performance and memory resources to track down the client connection, and detail. Unlike, QoS 1 and 2 which provide message delivered guaranteed option the average message significantly drops down compared to the QoS level 0 because of the handshake establishment which caused amount of time for assured the packet delivered as level 2 is the most robustness but slowest.

Secondly, on QoS level 0 which theoretically might occur message loss issues due to the characteristic that broker will not memorize the message only acknowledged and flush out immediately. However, surprisingly, there are also have a message loss issues appeared on QoS level 1 and QoS level 2 which will guaranteed that the receiver will get the messages as the detailed below.



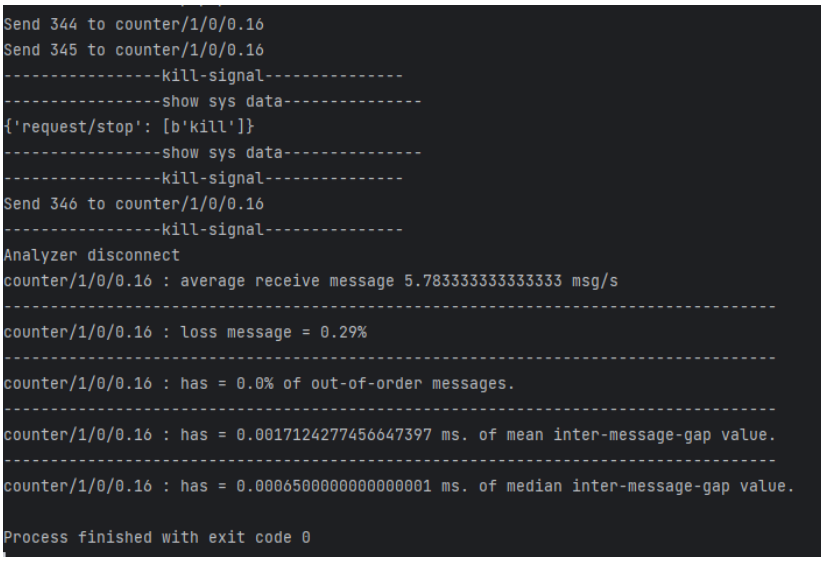
These issues might occur due to the sudden disconnect of the analyzer which might lost a message packet. It apparent that on the slow message which is high delay (0.32s) the loss rate is the highest because the duration of message delivery while analyzer disconnect from the broker due to time limit. Also, on the QoS level 1, there are the messages duplication issues which rarely appeared due to the private and local broker server which have the stable network. Unfortunately, there are also out of order issues appeared on the QoS level 2 which out of order message is only occurring during conduct a fast speed publishing which is 0.0 delay.

Finally, on the gap between message the result is reasonable due to the QoS level and delay speed which the fast publishing should contain the tightest gap compared to the slow publishing as the graph below.



b. Describe what correlations of measured rates with $SYS topics you expect to see and why, and whether you do, or do not.

Based on the testing environment there are less spot on the $SYS topic. For instance, the broker/load/message/received topic was complicated to analyzed due to the broker property that will retain some message in a meantime which is acknowledges us that broker still got the message. Unfortunately, it still complexes to distinct the retain message which QoS level 1 or 2 because both retain the message for guaranteed delivery. On the broker/message/received is useful to track down all messages especially on a ton of flooding message on QoS 0 delay 0.0 which will appear a ton of message received detail. On the client topics, its also remind tester to know that there is a specific amount of client that connect to the broker. In addition, it also a good assumption to check the amount of user who are on the public broker which should be a million of client. Unfortunately, based on the corresponding test on public broker, we also know that the broker might not always have the same structure such as $SYS not in the emqx broker.

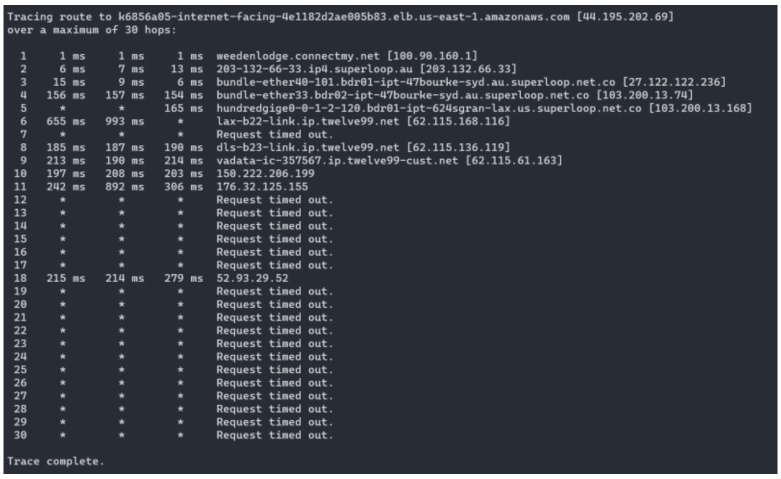


```
Send 344 to counter/1/0/0.16
Send 345 to counter/1/0/0.16
----------------kill-signal--------------
----------------show sys data--------------
{'request/stop': [b'kill']}
----------------show sys data--------------
----------------kill-signal--------------
Send 346 to counter/1/0/0.16
----------------kill-signal--------------
Analyzer disconnect
counter/1/0/0.16 : average receive message 5.783333333333333 msg/s
-----------------------------------------------------------------------------------
counter/1/0/0.16 : loss message = 0.29%
-----------------------------------------------------------------------------------
counter/1/0/0.16 : has = 0.0% of out-of-order messages.
-----------------------------------------------------------------------------------
counter/1/0/0.16 : has = 0.0017124277456647397 ms. of mean inter-message-gap value.
-----------------------------------------------------------------------------------
counter/1/0/0.16 : has = 0.0006500000000000001 ms. of median inter-message-gap value.

Process finished with exit code 0
```

Figure 12: $SYS topic not available on public mqtt

c. Report the traceroutes captured above.

Mqtt client v3 is still build on TCP protocol which required to establish the connection to perform handshake which based on the peers running result there are a hop until it reach the peers broker which might be a supporting detailed to explain that the when the QoS is higher the delay is higher, it will potentially caused of dropping performance because of the time spending hopping to the peers broker. In addition, some test result on the public broker and traceroute result will be demonstrated below for more clearly visualization.



```
Tracing route to k6856a05-internet-facing-4e1182d2ae005b83.elb.us-east-1.amazonaws.com [44.195.202.69]
over a maximum of 30 hops:

  1     1 ms     1 ms     1 ms  weedenlodge.connectmy.net [100.90.160.1]
  2     6 ms     7 ms    13 ms  203-132-66-33.ip4.superloop.au [203.132.66.33]
  3    15 ms     9 ms     6 ms  bundle-ether40-101.bdr01-ipt-47bourke-syd.au.superloop.net.co [27.122.122.236]
  4   156 ms   157 ms   154 ms  bundle-ether33.bdr02-ipt-47bourke-syd.au.superloop.net.co [103.200.13.74]
  5     *        *      165 ms  hundredgige0-0-1-2-120.bdr01-ipt-624sgran-lax.us.superloop.net.co [103.200.13.168]
  6   655 ms   993 ms     *     lax-b22-link.ip.twelve99.net [62.115.168.116]
  7     *        *        *     Request timed out.
  8   185 ms   187 ms   190 ms  dls-b23-link.ip.twelve99.net [62.115.136.119]
  9   213 ms   190 ms   214 ms  vadata-ic-357567.ip.twelve99-cust.net [62.115.61.163]
 10   197 ms   208 ms   203 ms  150.222.206.199
 11   242 ms   892 ms   306 ms  176.32.125.155
 12     *        *        *     Request timed out.
 13     *        *        *     Request timed out.
 14     *        *        *     Request timed out.
 15     *        *        *     Request timed out.
 16     *        *        *     Request timed out.
 17     *        *        *     Request timed out.
 18   215 ms   214 ms   279 ms  52.93.29.52
 19     *        *        *     Request timed out.
 20     *        *        *     Request timed out.
 21     *        *        *     Request timed out.
 22     *        *        *     Request timed out.
 23     *        *        *     Request timed out.
 24     *        *        *     Request timed out.
 25     *        *        *     Request timed out.
 26     *        *        *     Request timed out.
 27     *        *        *     Request timed out.
 28     *        *        *     Request timed out.
 29     *        *        *     Request timed out.
 30     *        *        *     Request timed out.

Trace complete.
```

Figure 11: Traceroute on public cloud broker.

Instance count 1 public broker.

QoS 0	Avg msg (msg/s)	loss msg (%)	out of order msg (%)	mean gap (ms)	median gap (ms)
0.0 s	3161	0	0.004471159	3.19E-04	0
0.01 s	63.7	0.03	0	0.015561371	0
0.02 s	31.78333333	0.05	0	0.031206716	0
0.04 s	20.78333333	0.08	0	0.047158909	0
0.08 s	10.53333333	0.16	0	0.093581616	0
0.16 s	5.783333333	0.29	0	0.171242775	0.065
0.32 s	3.033333333	0.55	0	0.32640884	0.33

QoS 1	Avg msg (msg/s)	loss msg (%)	out of order msg (%)	mean gap (ms)	median gap (ms)
0.0 s	3.6	0.47	0	2.74E-01	0.22
0.01 s	3.433333333	0.49	0	0.286536585	0.24
0.02 s	2.916666667	0.57	0	0.31862069	0.25
0.04 s	3.366666667	0.5	0	0.294925373	0.27
0.08 s	2.933333333	0.57	0	0.336514286	0.31
0.16 s	2.216666667	0.76	0	0.445227273	0.39
0.32 s	1.4	1.2	0	0.712168675	0.58

QoS 2	Avg msg (msg/s)	loss msg (%)	out of order msg (%)	mean gap (ms)	median gap (ms)
0.0 s	1.833333	1.84999999	0	5.04E-01	0
0.01 s	1.2	4.35	0	0.692608696	0.63
0.02 s	1.833333333	1.85	0	0.51962963	0.48
0.04 s	0.2	20	0	1.632	0
0.08 s	1.266666667	2.7	0	0.723783784	0.61
0.16 s	1.033333333	3.33	0	0.917333333	0.745
0.32 s	1.2	2.86	0	0.806571429	0.78

4. Consider the broader end-to-end (internet-wide maybe) network environment, in a situation with millions of sensors publishing regularly/irregularly to thousands of subscribers. Explain in your report *[around 1-2 pages]*

a. What (cpu/memory/network) performance challenges there might be when using MQTT in general, from the source publishing its messages, all the way through the network and broker to your subscribing client application. If you lose messages, where might they be lost?

Assumed that in the broader end-to-end situation, a million of sensors are interact with a few brokers or in worst cases one broker. Broker can also treat like normal server with specific protocol called mqtt which in this case can brefily point out the performance issues in this scenario (Gheorghe-Pop et al., 2020).

CPU challenges

A million use of sensor also directly mean that CPU required to spend a huge amount of load to deal with the Mqtt packets which either be QoS level 0, 1, or 2. Especially, high level of QoS, high amount of packet like on level 2 as PUBREC, PUBREL, and PUBCOMP packet which required CPU to process with theses kind of packet. In very high level of MQTT broker like Cloud broker also have an authentication method for security which can caused CPU to Encrypt or Decrypt packet which consumed amount of workload.

Memory challenges

Mqtt build on top of TCP/IP which have a connection establishment like publisher and subscriber, so broker need to maintain those session data. In case of a huge amount of end user, it potentially causes a huge amount of memory to maintain a million session. In addition, referred to the QoS level, especially on level 1 and 2 which required to perform handshake, broker also need to store those message packet for those QoS level specification which depends on messages payload size.

Network challenges

Usually, mqtt are used in smart home or private network area which have a stable and reliable network connection. Unfortunately, in the million users even mqtt is a lightweight protocol, it still required a significantly amount of bandwidth due to the size of the payloads corresponding with the QoS level such as re-sending message on the QoS level 1 which consume amount of bandwidth for those messages duplication. Imagine on QoS level 0 which transmitted a data packet with high frequency rate, can lead to packet loss, high latency network issues and bottle neck.

Message loss and where to find it

Message loss issues in mqtt can be derived into many perspectives. For instance, on QoS level 0 which not guaranteed messages delivered, poor internet connection, unpredictable event like client disconnects unintendedly while publishing or subscribing with broker, retained message capacity of broker reach a limit, etc. Unfortunately, the lost messages are not recoverable or discoverable on mqtt. However, we can also config mqtt broker th have a persistent session to retain all subscriber messages, or we still can still acknowledge the lost messages on very high-level approach with the specific configuration on the application that used mqtt broker, for instance as an application log file for track down the lost messages.

b. How the different QoS levels may help, or not, in dealing with the challenges.

As the information discussed in the question 4.a, we can interpret that QoS level also have a potential effect on those challenges either is help or not based on the specification, and purposed usage of those QoS level. We still need to concern and design the best QoS level depends on the various situations. We can derive both consequences from different QoS level below:

QoS 0 consumes least amount of CPU resources, memory usages, and network bandwidth due to broker didn't response to store the messages only fire and flush those data immediately. Unfortunately, the most remarkable issue on level 0 is the unrecoverable on the message lost.

QoS 1 guaranteed that there is no message loss which in exchange with the high amount of CPU consumption, network bandwidth, memory consumption. The duplication messages event potentially lead to network traffic issues and additional resources to handle it.

QoS 2 provide the most robustness message delivery by 4 handshake which can deal with the message duplication issues but it required most CPU, memory resources and network due to the 4-handshake process as well.

Apparently, it also depends on user purposes to choose the most suitable QoS level. In addition, there also a plenty option to optimize the mqtt corresponding with those challenges for instances mqtt application configurations.

References

Team, T.H. (2015). *Quality of Service 0,1 & 2 - MQTT Essentials: Part 6*. [online] www.hivemq.com. Available at: https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/ [Accessed 20 May 2023].

Steve (2022). *MQTT- Which QOS should you Use? 0,1,2*. [online] Available at: http://www.steves-internet-guide.com/mqtt-which-qos-to-use/ [Accessed 20 May 2023].

Cheong, S. (2022). *MQTT (Quality of Service) QoS levels explained*. [online] Cedalo. Available at: https://cedalo.com/blog/understanding-mqtt-qos/ [Accessed 20 May 2023].

Gheorghe-Pop, I.-D., Kaiser, A., Rennoch, A. and Hackel, S. (2020). A Performance Benchmarking Methodology for MQTT Broker Implementations. *2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. [online] doi:https://doi.org/10.1109/qrs-c51114.2020.00090.

Eclipse Mosquitto. (2019). *Mosquitto man page*. [online] Available at: https://mosquitto.org/man/mosquitto-8.html.