COMP6330 Assignment 3 – MQTT

Yujie Qiu
u6551675

1.

The Quality of Service (QoS) level in MQTT is an agreement between the sender and the receiver which describes the guarantee of a certain message's delivery.

There are 3 QoS levels in MQTT. QoS 0 sends message only once, while QoS 1 would send message multiple times until acknowledged. Differ from QoS 0 and 1, QoS 2 requires both sender and receiver to engage in handshakes, so the only one copy's delivery can be guaranteed.

An MQTT connection covers connect, publish/subscribe, and disconnect.

a. Connect

No.	Time	Source	Destination	Protocol	Length	Info
5	2.871960	192.168.2.101	3.17.27.45	TCP	66	49601 → 1883 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
6	3.160802	3.17.27.45	192.168.2.101	TCP	66	1883 → 49601 [SYN, ACK] Seq=0 Ack=1 Win=62727 Len=0 MSS=1452 SACK_PERM=1 WS=128
7	3.160918	192.168.2.101	3.17.27.45	TCP	54	49601 → 1883 [ACK] Seq=1 Ack=1 Win=132096 Len=0
8	3.161141	192.168.2.101	3.17.27.45	MQTT	103	Connect Command
9	3.468216	3.17.27.45	192.168.2.101	TCP	60	1883 → 49601 [ACK] Seq=1 Ack=50 Win=62720 Len=0
10	3.468218	3.17.27.45	192.168.2.101	MQTT	60	Connect Ack

Figure 1.1 screenshot

As it shown in Figure 1.1, the client first sent Synchronize Sequence Numbers (SYN) from 192.168.2.101 to the broker, which later sent SYN and Acknowledge Character (ACK) back. The client would then send an ACK to the broker. After TCP connection was done, the client sent MQTT's connect command to the broker, and the broker would return the client an ACK. MQTT's connection was done then.

b (1). q0

No.	Time	Source	Destination	Protocol	Length	Info
11	3.468317	192.168.2.101	3.17.27.45	MQTT	76	Subscribe Request (id=1) [counter/slow/q0]
12	3.761420	3.17.27.45	192.168.2.101	MQTT	60	Subscribe Ack (id=1)
13	3.802509	192.168.2.101	3.17.27.45	TCP	54	49601 → 1883 [ACK] Seq=72 Ack=10 Win=132096 Len=0
14	4.080310	3.17.27.45	192.168.2.101	MQTT	78	Publish Message [counter/slow/q0]
15	4.120103	192.168.2.101	3.17.27.45	TCP	54	49601 → 1883 [ACK] Seq=72 Ack=34 Win=132096 Len=0
16	4.470733	3.17.27.45	192.168.2.101	MQTT	77	Publish Message [counter/slow/q0]
17	4.511704	192.168.2.101	3.17.27.45	TCP	54	49601 → 1883 [ACK] Seq=72 Ack=57 Win=131840 Len=0

Figure 1.2 screenshot

As it mentioned above, QoS 0 sends message only once. The broker only needs to publish message, without any ACK from the client.

b (2). q1

No.	Time	Source	Destination	Protocol	Length	Info
8	1.100061	192.168.2.101	3.17.27.45	MQTT	76	Subscribe Request (id=1) [counter/slow/q1]
9	1.398268	3.17.27.45	192.168.2.101	MQTT	60	Subscribe Ack (id=1)
10	1.438091	192.168.2.101	3.17.27.45	TCP	54	52481 → 1883 [ACK] Seq=72 Ack=10 Win=132096 Len=0
11	1.715478	3.17.27.45	192.168.2.101	MQTT	80	Publish Message (id=1) [counter/slow/q1]
12	1.715763	192.168.2.101	3.17.27.45	MQTT	58	Publish Ack (id=1)
13	2.040528	3.17.27.45	192.168.2.101	TCP	60	1883 → 52481 [ACK] Seq=36 Ack=76 Win=62720 Len=0
14	2.237432	3.17.27.45	192.168.2.101	MQTT	79	Publish Message (id=2) [counter/slow/q1]
15	2.237987	192.168.2.101	3.17.27.45	MQTT	58	Publish Ack (id=2)
16	2.513454	3.17.27.45	192.168.2.101	TCP	60	1883 → 52481 [ACK] Seq=61 Ack=80 Win=62720 Len=0
17	3.241001	3.17.27.45	192.168.2.101	MQTT	79	Publish Message (id=3) [counter/slow/q1]

Figure 1.3 screenshot

QoS 1 would send message multiple times until acknowledged. In other word, there is a handshake between client and broker, as Figure 1.3 shows.

b (3). q2

No.	Time	Source	Destination	Protocol	Length	Info
50	19.704373	192.168.2.101	3.17.27.45	MQTT	76	Subscribe Request (id=1) [counter/slow/q2]
51	20.207160	3.17.27.45	192.168.2.101	MQTT	60	Subscribe Ack (id=1)
52	20.247789	192.168.2.101	3.17.27.45	TCP	54	52589 → 1883 [ACK] Seq=72 Ack=10 Win=132096 Len=0
53	20.743757	3.17.27.45	192.168.2.101	MQTT	80	Publish Message (id=1) [counter/slow/q2]
54	20.744407	192.168.2.101	3.17.27.45	MQTT	58	Publish Received (id=1)
55	21.309460	3.17.27.45	192.168.2.101	MQTT	79	Publish Message (id=2) [counter/slow/q2]
56	21.309757	192.168.2.101	3.17.27.45	MQTT	58	Publish Received (id=2)
57	21.807431	3.17.27.45	192.168.2.101	MQTT	60	Publish Release (id=1)
58	21.808414	192.168.2.101	3.17.27.45	MQTT	58	Publish Complete (id=1)

Figure 1.4 screenshot

QoS 2 is more complicated comparing with QoS 0 and QoS 1. As it shown in Figure 1.4, after the broker published the message, the client returned the broker a Publish Received. Then the broker would send back a Publish Release. The client would send a Publish Complete as soon as it received the Release message. So generally, QoS 2 would take a longer time comparing with QoS 0 and 1.

c. Disconnect

No.	Time	Source	Destination	Protocol	Length	Info
118	12.161062	192.168.2.104	3.17.27.45	MQTT	56	Disconnect Req
119	12.161191	192.168.2.104	3.17.27.45	TCP	54	52427 → 1883 [FIN, ACK] Seq=74 Ack=208 Win=131072 Len=0
120	12.164220	3.17.27.45	192.168.2.104	MQTT	76	Publish Message [counter/slow/q0]
121	12.164291	192.168.2.104	3.17.27.45	TCP	54	52427 → 1883 [RST, ACK] Seq=75 Ack=230 Win=0 Len=0
122	12.374064	3.17.27.45	192.168.2.104	TCP	60	1883 → 52427 [ACK] Seq=230 Ack=74 Win=62720 Len=0
123	12.374142	192.168.2.104	3.17.27.45	TCP	54	52427 → 1883 [RST] Seq=74 Win=0 Len=0

Figure 1.5 screenshot

All 3 QoS disconnected in the same way, as Figure 1.5 demonstrates.

As it discussed above, QoS 0 sends message only once. The stability of connection between sender and receiver need to be promised to minimize the loss. In other words, QoS 0 should be applied when connection can be guaranteed, or some message loss is acceptable. Therefore, I would apply it when data is not that important, environmental sensor, for example.

QoS 1 guarantees message's arrival, and it is faster than QoS 2. However, duplicate messages may occur as messages are published multiple times in QoS 1, which means subscribers may receive same messages. A typical use case of QoS 1 is voice sensor.

QoS 2 guarantees all message arrive exactly once, but it is much slower than QoS 0 and 1. It can be used in charging system.

2.

a). I subscribed to each of the 6 counters 3 times a day for 3 days, as shown in the following table.

	recv	loss	dupe	ooo	gap	gvar
counter/slow/q0	0.997	0.003	0	0	1004	796
counter/slow/q1	0.997	0.003	0	0	1004	429
counter/slow/q2	0.99	0.003	0	0	1006	612
counter/fast/q0	78.377	0	0	0	13	75
counter/fast/q1	17.387	0.477	0	0	57	120
counter/fast/q2	9.303	0.766	0	0	61	129

Figure 2.1

	recv	loss	dupe	ooo	gap	gvar
counter/slow/q0	0.993	0	0	0	1006	324
counter/slow/q1	0.993	0	0	0	1006	346
counter/slow/q2	0.99	0	0	0	1003	3762
counter/fast/q0	204.217	0	0	0	5	40
counter/fast/q1	26.047	0.854	0	0	26	94

counter/fast/q2	5.693	0.001	0	0	175	225

Figure 2.2

	recv	loss	dupe	ooo	gap	gvar
counter/slow/q0	0.997	0.003	0	0	1001	546
counter/slow/q1	0.997	0.003	0	0	1003	318
counter/slow/q2	0.997	0.003	0	0	1004	428
counter/fast/q0	92.483	0	0	0	11	92
counter/fast/q1	31.567	0.657	0	0	11	58
counter/fast/q2	16.0	0.824	0	0	18	74

Figure 2.3

i). The average rate of messages received across the period is measured by

$$recv = \frac{messages_{recv}}{t}$$

where t is the total listening time. In this case, we set timer for 300 seconds, as it required in the question. Ideally, it is expected that:

$$recv_{slow/q2} < recv_{slow/q1} < recv_{slow/q0} \ll recv_{fast/q2} < recv_{fast/q1} < recv_{fast/q0}$$

Experimental results show that $recv_{slow} < recv_{fast}$. Although it is hard to tell the difference among $recv_{slow/q0}$, $recv_{slow/q1}$ and $recv_{slow/q2}$ from the figure, it is clear that $recv_{fast/q2} < recv_{fast/q1} < recv_{fast/q0}$.

ii). The rate of message loss is measured by

$$loss = \frac{messages_{total} - messages_{recv}}{messages_{total}}$$

where $messages_{total}$ refers to messages should we have seen, and $messages_{recv}$ refers to messages actually received. It is assumed that all messages are continuous numbers. In this case, total messages should have seen can be calculated by the distance between the first message and the last message. For example, if the message list covers 3, 4, 6, 7, 8, 10, then the numbers of the message should be 10 - 3 + 1 = 8, and the loss should be $(8 - 6) \div 8 = 0.25$. Ideally, it is expected that:

$$loss_{slow/q2} < loss_{slow/q1} < loss_{slow/q0} \ll loss_{fast/q2} < loss_{fast/q1} < loss_{fast/q0}$$

However, we can hardly see any losses. Although there are some losses recorded, it seems that the losses are randomly distributed. We would discuss this in the later $SYS part.

iii). The rate of duplicated messages over any 10 seconds is measured by

$$dupe = \frac{messages_{dupe_{in\_10s}}}{messages_{10s}}$$

In this case, we first put all messages into a certain list, and record their timestamps in another list. First, we choose a random timestamp t_1, and then we find another timestamp t_2 which satisfies the condition $t_1 + 10 = t_2$. Then, we find these two messages, and calculate the dupe rate with above formula.

It is expected that $QoS\ 1$ should have the largest dupe rate, while $QoS\ 0$ and $QoS\ 2$ should have less. However, after several days' observation, no duplicate message was found.

iv). The rate of out-of-order messages is measured by

$$ooo = \frac{messages_{ooo}}{messages_{total}}$$

For instance, if we got [1,2,4,3,5,6], then 3 is considered as an out-of-order message, and the ooo

rate is 0.167.

It is expected that that $QoS\,2$ should have the least dupe rate, while $QoS\,0$ and $QoS\,1$ should have larger. However, after several days' observation, no mis-ordered message was found.

v). The mean inter-message-gap and gap-variation

$$gap = \frac{1}{n}\sum_{i=1}^{n}(timestamp_{i+1} - timestamp_i)$$

where n refers to the gap numbers.

In this case, we recorded message's timestamp, and calculate gaps by subtraction. It is expected that

$$gap_{fast/q0} < gap_{fast/q1} < gap_{fast/q2} \ll gap_{slow/q0} < gap_{slow/q1} < gap_{slow/q2}$$

And the results are consistent with the expectation.

b. I subscribed to $SYS/broker/load/…, $SYS/broker/heap/… and $SYS/broker/clients/connected.

(i). Subscribing to $SYS/broker/load/publish/sent/+ would return the moving average of the number of publish messages sent by the broker.

(ii). $SYS/broker/heap/current size and $SYS/broker/load/heap/maximum size should return the current size or the largest amount of the heap memory. But I did not get any messages from $SYS/broker/heap/current size or $SYS/broker/load/heap/maximum size.

(iii). $SYS/broker/clients/connected returns the number of currently active clients.

The experimental result turns out that the more publish messages sent by the broker and the more active clients are, the greater loss would be. However, as Figure 2.1 – 2.3 show, I haven't found any duplicate messages or out-of-order messages, which makes the relationship between $SYS topics and duplicate/mis-ordered rates unclear.

3.

a. what (cpu/memory/network) performance challenges there might be, from the source publis hing its messages, all the way through the network to your subscribing client.

The biggest challenge for MQTT is performing reliable data transmission with limited bandwidth and hardware resources. The number of subscribers, for instance, can be an important factor which may influent the performance. Challenge for network performance occurs when there are lots of subscribers, as individuals may have very different network situation. Other factors include the publish frequency, QoS type and so on.

b. how an irregular publisher has different issues to a very-frequent publisher

A very-frequent publisher can bring potential challenges. More resources are required if messages are published with a high frequency. In other words, sending messages to subscribers frequently may take lots of CPU and memory resources. Besides, publishing messages frequently can also be a challenge when it comes to the traffic issue.

c. how the different QoS levels may help, or not, in dealing with the challenges

As we've discussed, different QoS level sends data in different way. Applying different QoS level with respect to different usages may save lots of resources. QoS 0 sends message only once. It has a higher data rate and higher loss compared with other QoS levels. Hence all high-frequency data and real-time data where loss some data is acceptable can use QoS 0.

QoS 1 sends message multiple times until acknowledged by the client. Generally, QoS 1 can be used when delivery need to be guaranteed, for example, some events or commands.

QoS 2 guaranteed the safety of the message but is much slower. It is reported that QoS 2 has nearly

50% more overhead than QoS 1 messages[1], which means it needs more resources and bandwidth compared with other levels. In other words, resources may be saved by replacing QoS 2 with QoS 1 and a message filter to avoid duplicate messages.

[1] https://blog.usejournal.com/how-to-optimize-data-usage-over-mqtt-792abebd2cd1