MQTT Report

Wenjun Yang u6251843

1.

In the mqtt protocol, how the mqtt messages get published by the broker are quite different when the subscriber connects to the broker by different QoS (0, 1, 2). Here I will explain the similarities and differences when you subscribe with different QoS.

No matter what QoS is, the client will send the connect request to the broker firstly and if the broker receives that message, it will send an acknowledgement back to tell the client that the client has connected to the server. Then the client will choose a specific topic, which is “counter/slow/…” here and send a subscribe request to the broker, similarly to connecting, the broker will send an acknowledgement back to confirm that the client has successfully subscribed to the topic. The operations above are almost the same even though you change the value of QoS, they are the preparation before the broker publishes messages to the client.

When QoS is 0, the message will be published by broker only once, the message won’t be acknowledged by the client or stored and redelivered by the broker. It implies that it is impossible to receive a duplicate message (DUP flag is true) since even if the message is lost, the broker won’t send it again. At the same time, there is no order for the message when QoS is 0. The message order is determined by messageId in the packet. In reality, I didn’t find the messageId in the packet when viewing the wireshark. Hence, we can deduce that QoS 0 will be chosen if we have a very stable connection between client and broker, or we don’t care whether one or more messages are lost, which means the data is not that crucial or the data is sent at short intervals (MQTT essential, 2017).

When QoS is 1, the message will be published by the broker firstly and if the client receives that message, it will send an acknowledgement back. This guarantees that the message will be delivered at least once to the client. However, the message can be delivered more than once, which causes the message duplication. Thus, it is possible for client to receive a duplicate message when QoS is 1. When you view the information in the publish packet, you will find that an additional one called message identifier appears and the value of it is the arriving order of packet, which indicates the messageId determines the order of message. I think the order of message will be helpful to adjust the order of messages. We will choose QoS = 1 if we need to get every message but we need to guarantee that our application can tolerate duplicates and process them accordingly (MQTT essential, 2017).

When QoS is 2, the message will be published by the broker firstly and if the client receives, it will send a “Publish Received” message back to the broker to confirm that the message has arrived. Then, the broker will respond to “Publish Received” message by sending “Publish Release”. In the end, if the client receives the message, it will send “Publish Complete” to end the handshake. There are no duplicate messages in this case since both sides can be sure that the message has been delivered and broker also knows about it, this guarantees that the client will only receive the message once. Almost the same as the situation that QoS is 1, the message order is determined by messageId here. We will use QoS 2 when our application has to get every message but cannot deal with the duplicate messages. Additionally, our application may not extremely care about the speed of service since the QoS 2 is the slowest one when it compared to the other two ones.

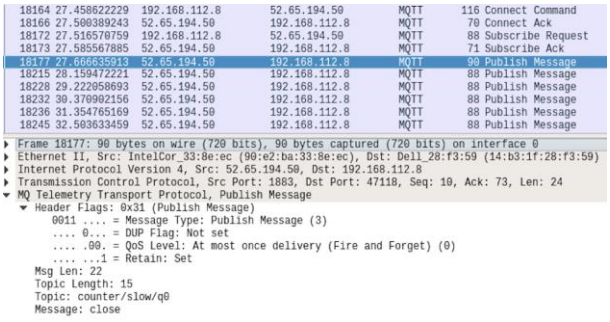

Figure 1: Handshake when QoS is 0

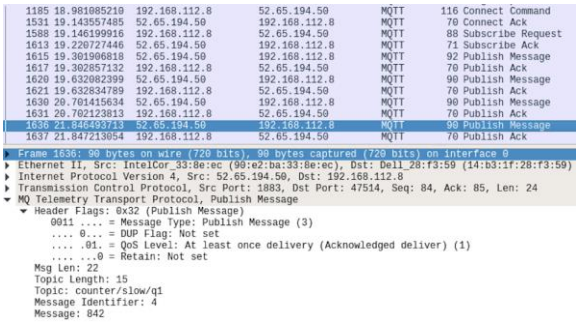

Figure 2: Handshake when QoS is 1

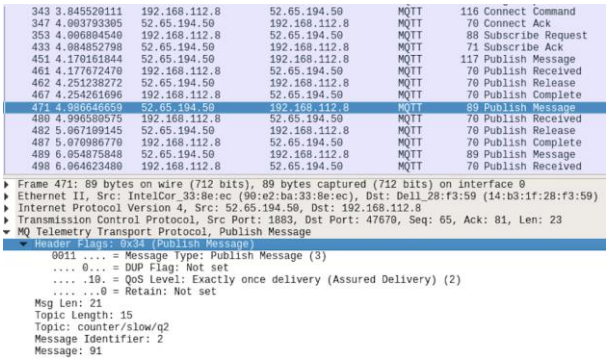

Figure 3: Handshake when QoS is 2

2. (a)

I listened to the three different QoS, each for 20 minutes. After collecting and analysing the data, I get some conclusions as follows:

For QoS 0, the rate of messages I received is 100% (6246 messages per minute). Additionally, I didn't find any message loss, duplicated messages and out-of-order messages.

For QoS 1, the average rate of messages received is 76.3% (4643 messages per minute), the median value is 77.7% (4647 messages per minute). The average rate of messages loss is 23.7%, the median value is 22.3%. There were no duplicated messages and out-of-order messages.

For QoS 2, the average rate of messages received is 37.2% (2234 messages per minute), the median value is 37.5% (2234 messages per minute). The average rate of messages loss is 62.8%, the median value is 62.5%. There were no duplicated messages and out-of-order messages.

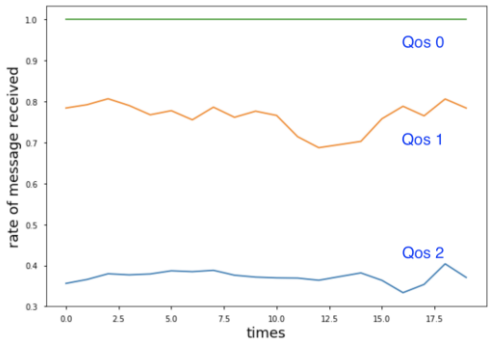

Figure 4: message received rate in different QoS

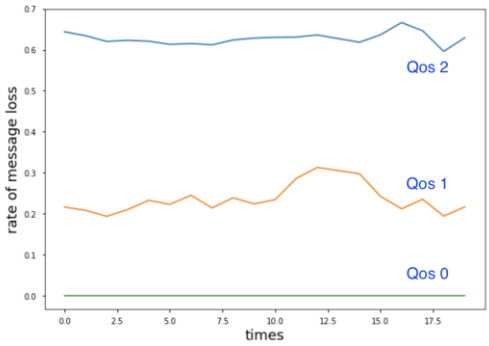

Figure 5: message loss rate in different QoS

(b)

I subscribed to the $SYS/broker/load/messages/#, $SYS/broker/load/publish/# and

$SYS/broker/heap/current each for 5 minutes and collect statistics for analysing the relation of data in these topics and rate of messages loss.

For the topic $SYS/broker/load/messages/#, I listened to two sub-topics, namely received and sent. The graph I got is listed left below (Figure 6). The blue column is the data in the topic $SYS/broker/load/messages/received/1min while the orange column is the data in the topic $SYS/broker/load/messages/sent/1min. I found the message loss rate doesn't depend on the number of MQTT messages received by the broker since when the number of MQTT messages received increases constantly, the message loss rate does not grow a lot as well. However, I found a positive correlation between the message loss rate and the number of MQTT messages sent by the broker, the message loss rate almost follows the trend of the number of MQTT messages sent by the broker. This indicates that the more messages sent by broker, the more messages lost during the transmitting process.

For the topic $SYS/broker/load/publish/#, there are three sub-topics in all. The blue column is the data in the topic $SYS/broker/load/publish/received/1min, the orange column is the data in the topic $SYS/broker/load/publish/dropped/1min and the green column is the data in the topic $SYS/broker/load/publish/sent/1min (Figure 7). In this case, I found that almost the same as the situation in the messages, the number of publish messages received by broker doesn't change dramatically but the message loss rate will change according to the tendency of the number of publish messages dropped by broker and the number of publish messages sent by broker, which means when the number of messages dropped or sent by broker expands, the message loss rate will increase too.

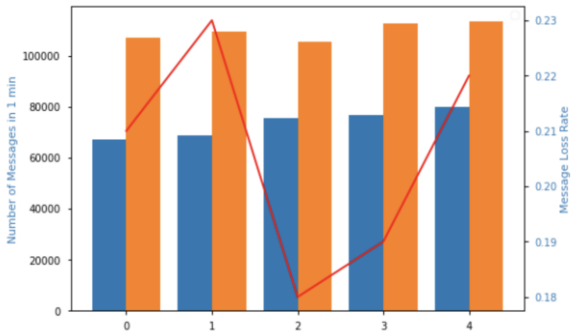

Figure 6: number of messages received/sent and message loss rate

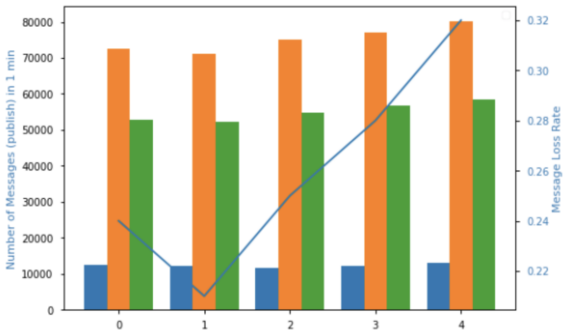

Figure 7: number of publish messages received/sent/ dropped and message loss rate

For the topic $SYS/broker/heap/current, I also seemed to find a positive relation between heap size and the message loss rate, which means the more heap memory is used by mosquito currently, the more message loss rate emerges (Figure 8).

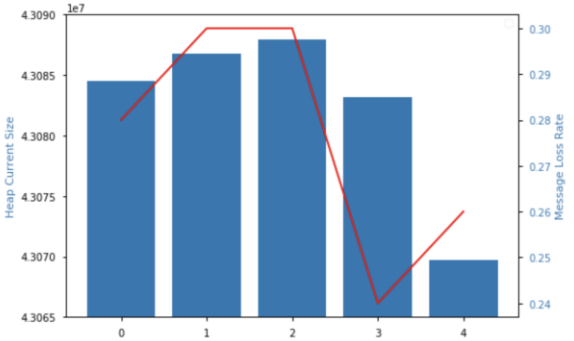

Figure 8: heap current size and message loss rate

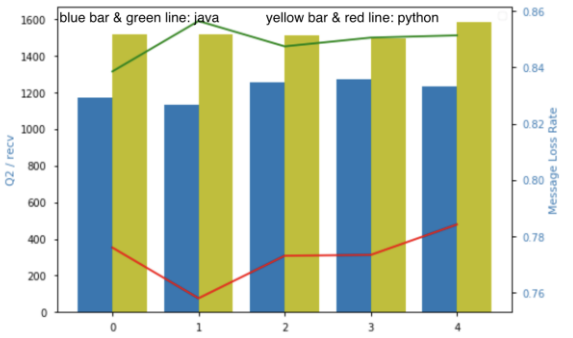

Figure 9: received messages and message loss rate In java and python at similar times

3.

This part is shown in the code.

4.

In this section, I collected five groups of data from the topic studentreport/# which are published by other students at similar times and all the data is achieved when QoS is 2. The graph is listed right above. The blue bar represents best 1-min rate of messages received by java while the yellow bar is from python.

(a)

From the graph (Figure 9), I found that the data is inconsistent although it was collected at similar times. The gap in each group is around 200-300 messages and the group which receives more messages in 1 min seems to own a lower message loss rate. After calculation, I concluded that the 'ideal' number of messages is almost the same, which means the broker is 'fair' to every client, but the 'actual' number of messages different clients receive is various. I guessed that this phenomenon may be caused by the various network connections and network quality.

(b)

The client in python works better than that of java over 1-min interval. The client in python can receive 300 more messages than the java one and the message loss rate is reduced 7% when it is compared to java client.

5.

(a)

i. CPU

The CPU on the brokers and clients may not be fast enough to process the incoming messages or connections, this will lead to message loss. Take the Mosquiito broker as an example, when the number of subscriber is greater than 128k, the broker cannot deal with any incoming connections. This broker utilizes only one CPU core, which reduce the CPU usage a lot. With more CPU cores, each core may handle dispatched connections and locally process MQTT logic (Pipatsakulroj, 2017).

ii. Network

When many thousands of sensors publish to a lot of subscribers, if the quality of network is quite bad, for example, the bandwidth is not large enough, network congestion may occur. If the packets are impossible to be sent to destination, they will be dropped, which causes the message loss during the transmitting process.

iii. Memory

When the broker accepts a great number of messages, the memory size may not be large enough to hold these messages if necessary. For example, if the majority of messages set the retained flag, which means the messages need to be stored in the memory of broker, it is possible that the broker collapses. Another scenario is that messages with QoS 1/2 must be stored until the client accepts the message successfully, the broker may run out of memory.

(b)

The messages with QoS 0 don't have to be stored. This could reduce the pressure of memory. At the same time, one handshake only consists of one process, the client publishes messages to broker or the broker publishes messages to client. The number of communications is less than that of messages with QoS 1/2. I think this may make the CPU process less information and the network congestion may not be as serious as the condition of messages with QoS 1/2.

Conversely, the messages with QoS 1/2 need to be stored. Additionally, the duplicate messages in QoS 1 and multiple communications between broker and clients in QoS 2 can make network congestion even worse and increase the pressure to CPU since it has to deal with more messages.

(C)

The results of actual quantified differences between QoS levels are consistent with actual effects when QoS levels deal with challenges. The clients subscribe with QoS 1/2 receives fewer messages and the message loss rate is higher while the clients subscribe with QoS 0 have no message lost and get more messages. I think it does not deviate the analysis in part (b), which means frequent communications between broker and clients put great pressure on network and CPU and the requirement that messages should be stored in broker until the clients receive it is a great challenge to the memory.

Reference

"MQTT essential part 6", (2017), Available at: https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels (Accessed 21 May 2018).

Pipatsakulroj. W, (2017), "muMQ: A Lightweight and Scalable MQTT Broker", Available at: https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels (Accessed 21 May 2018).