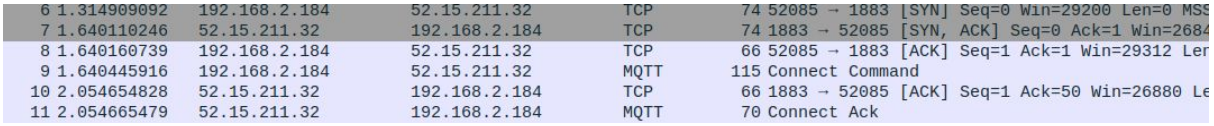
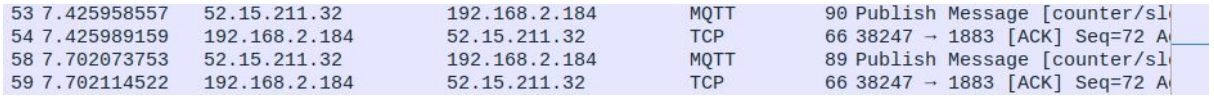
1.
Connection part: The TCP handshake and MQTT connection are the same in the three QoS levels:



6	1.314909092	192.168.2.184	52.15.211.32	TCP	74 52085 → 1883 [SYN] Seq=0 Win=29200 Len=0 MSS
7	1.640110246	52.15.211.32	192.168.2.184	TCP	74 1883 → 52085 [SYN, ACK] Seq=0 Ack=1 Win=2684
8	1.640160739	192.168.2.184	52.15.211.32	TCP	66 52085 → 1883 [ACK] Seq=1 Ack=1 Win=29312 Len
9	1.640445916	192.168.2.184	52.15.211.32	MQTT	115 Connect Command
10	2.054654828	52.15.211.32	192.168.2.184	TCP	66 1883 → 52085 [ACK] Seq=1 Ack=50 Win=26880 Le
11	2.054665479	52.15.211.32	192.168.2.184	MQTT	70 Connect Ack

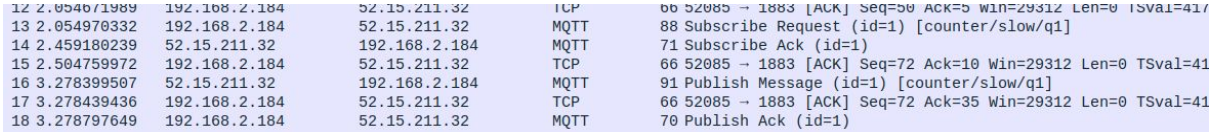
This is the TCP handshake and MQTT connection of counter/q0, counter/q1, counter/q2 Wireshark screenshot, and the first three packets are for TCP connection: First, the client sends SYN(192.168.2.157) request to the broker(52.15.211.32), then, the broker sent the acknowledge to the client, finally, the client sends the acknowledge. And the next is an MQTT packet which sent from the client to the broker, it is a connection request. Then the broker sent back the acknowledge which means the MQTT connection is accomplished.

publish and subscribe topic:
q0: it's a fire and forgot level, which means it doesn't need any acknowledge packet from the packet recipient. As the picture below, it only has “Publish Message” from broker to the subscriber.



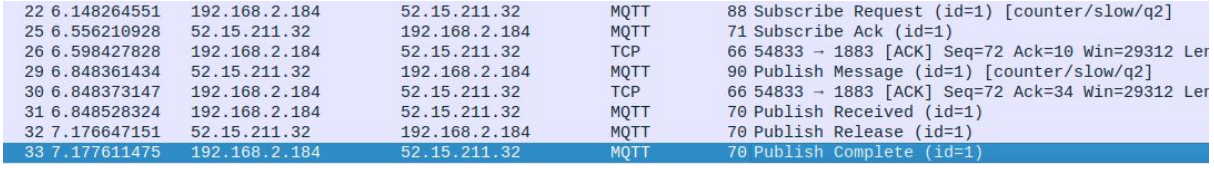
53	7.425958557	52.15.211.32	192.168.2.184	MQTT	90 Publish Message [counter/sl
54	7.425989159	192.168.2.184	52.15.211.32	TCP	66 38247 → 1883 [ACK] Seq=72 A
58	7.702073753	52.15.211.32	192.168.2.184	MQTT	89 Publish Message [counter/sl
59	7.702114522	192.168.2.184	52.15.211.32	TCP	66 38247 → 1883 [ACK] Seq=72 A

q1: It has acknowledged packets from the recipient to guarantee the packets received at least one time, but it would have duplicates until the publisher received the ack packet.So there is a handshake of Publish Message and ACK message.



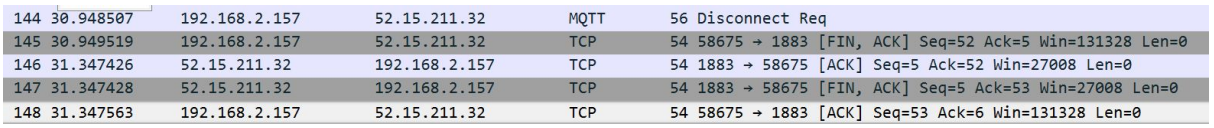
12	2.054671989	192.168.2.184	52.15.211.32	TCP	66 52085 → 1883 [ACK] Seq=50 Ack=5 Win=29312 Len=0 TSval=417
13	2.054970332	192.168.2.184	52.15.211.32	MQTT	88 Subscribe Request (id=1) [counter/slow/q1]
14	2.459180239	52.15.211.32	192.168.2.184	MQTT	71 Subscribe Ack (id=1)
15	2.504759972	192.168.2.184	52.15.211.32	TCP	66 52085 → 1883 [ACK] Seq=72 Ack=10 Win=29312 Len=0 TSval=41
16	3.278399507	52.15.211.32	192.168.2.184	MQTT	91 Publish Message (id=1) [counter/slow/q1]
17	3.278439436	192.168.2.184	52.15.211.32	TCP	66 52085 → 1883 [ACK] Seq=72 Ack=35 Win=29312 Len=0 TSval=41
18	3.278797649	192.168.2.184	52.15.211.32	MQTT	70 Publish Ack (id=1)

q2: the publish message of level 2 has 4 part: PUBLISH, PUBREC, PUBREL, PUBCOMP to guarantee the packet received exactly once. In the picture, they are Publish Message, Publish Received, Publish Release and Publish Complete.



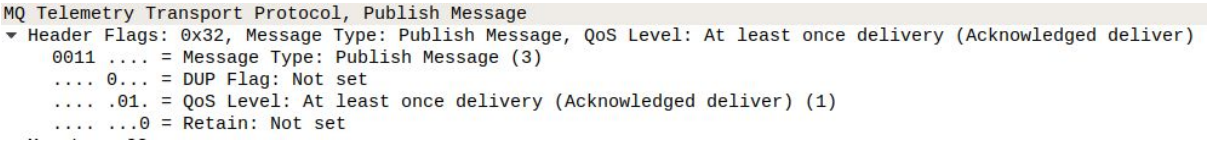
22	6.148264551	192.168.2.184	52.15.211.32	MQTT	88 Subscribe Request (id=1) [counter/slow/q2]
25	6.556210928	52.15.211.32	192.168.2.184	MQTT	71 Subscribe Ack (id=1)
26	6.598427828	192.168.2.184	52.15.211.32	TCP	66 54833 → 1883 [ACK] Seq=72 Ack=10 Win=29312 Len
29	6.848361434	52.15.211.32	192.168.2.184	MQTT	90 Publish Message (id=1) [counter/slow/q2]
30	6.848373147	192.168.2.184	52.15.211.32	TCP	66 54833 → 1883 [ACK] Seq=72 Ack=34 Win=29312 Len
31	6.848528324	192.168.2.184	52.15.211.32	MQTT	70 Publish Received (id=1)
32	7.176647151	52.15.211.32	192.168.2.184	MQTT	70 Publish Release (id=1)
33	7.177611475	192.168.2.184	52.15.211.32	MQTT	70 Publish Complete (id=1)

disconnection: First the client sent a disconnection request to the broker, then the broker sent ACK packet and a disconnection request, and the client sent back an ACK packet to broker eventually.

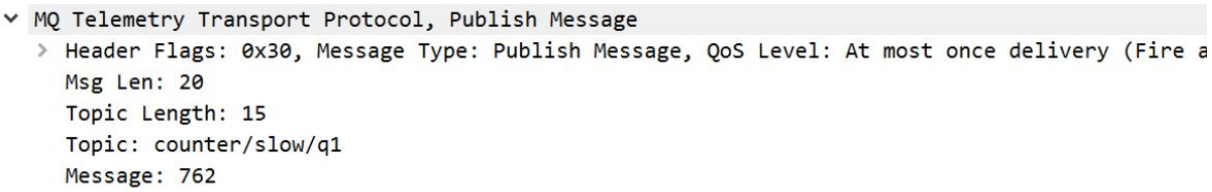


144	30.948507	192.168.2.157	52.15.211.32	MQTT	56 Disconnect Req
145	30.949519	192.168.2.157	52.15.211.32	TCP	54 58675 → 1883 [FIN, ACK] Seq=52 Ack=5 Win=131328 Len=0
146	31.347426	52.15.211.32	192.168.2.157	TCP	54 1883 → 58675 [ACK] Seq=5 Ack=52 Win=27008 Len=0
147	31.347428	52.15.211.32	192.168.2.157	TCP	54 1883 → 58675 [FIN, ACK] Seq=5 Ack=53 Win=27008 Len=0
148	31.347563	192.168.2.157	52.15.211.32	TCP	54 58675 → 1883 [ACK] Seq=53 Ack=6 Win=131328 Len=0

For level one, if the recipient didn't receive the message then the sender would send it again with a duplicate(DUP) flag, the screenshot below shows the UDP flag(but still not set in the screenshot).



Also, in our situation, our publisher uses a counter which steadily increases as the message content, so the number in the payload can also use to find a duplicate. If the number in the payload appears more than once(from different packets' payload), it indicates the duplicate. For the packet order, as our publisher use a counter which steadily increases as the message content, and it can also indicate the order of publishing packets. So the number in the payload can represent the order:



For QoS level 0, there is no acknowledge packet and no retransmission and storage, so I would choose it when I need a high speed but don't care about the guarantee and don't need message queuing, such as some audio real-time system.
For QoS level 1, this level guarantees the recipient receives the message at least once, and it will have duplicated message, I would use it to send some security message in a security system such as entrance sensor.
For QoS level 2, this level guarantees the recipient receives the message only once, so it is the safest but lowest one, I would use it to send some message which would cause mistakes when received more or less than once, such as the billing system.

2.
c) Subscribing to $SYS topics can gain some information about the broker. The message from the publisher to clients has two parts 1) from the publisher to the broker 2) from the broker to the publisher. So subscribing to $SYS topics can make it more clear about where and why cause the loss/duplicate/out-of-ordered rates in the processing. For the loss/duplicate/misorded-rates, these all vary with different QoS levels, so I would like to subscribe it with the 6 topics(fast and slow are seperated and each with the $SYS topics) at the same time to find out the statement of the broker. So I subscribe the $SYS topics with each topic at the same time when collecting the statistics, and then I can make a comparison with the result of loss/duplicate/out-of-ordered rates to find whether it is any correlation such as more active clients leads to a higher loss rate.

I subscribe to the topic:$SYS/broker/load/publish/sent/5min which means the moving average number of publishing messages(per one minute) sent by the broker in 5 minutes. I want to see whether it would have a higher rate of loss/duplicate/out-of-ordered rates messages when the broker publishes with higher rates. Because sending more messages per minute means the broker sends messages more frequently and it is a big challenge for the buffer to handle too many messages, it means it's more possible to make errors in the traffic.

$SYS/broker/heap/current size The current size of the heap memory is used by the broker. If the number is too big, it means there is too much workload for the broker to handle and it is more possible to make errors. So I want to see whether it is more possible to cause loss/duplicate/out-of-ordered rates when it has a bigger current size of heap in use. But I didn't receive any message from that topic, that is because this topic may be unavailable depending on compile time options.

$SYS/broker/clients/active The number of clients that are connected. If the number is too big, it means there is more pressure on the Internet and the broker needs to send more messages to more clients at the same time, and it is also a big challenge for the broker to buffer too many messages to a queue, it is more possible to make errors. I want to see whether the rate of loss/duplicate/ou-of-ordered rates when the active client goes high.

I listen to the 6 topics 3 times(fast and slow topics are separated and both with the $SYS topics to record the statistic). Then I would compare the changes. The results are:

	First time	Second time	Third time
slow/q0:			
loss rate:	0.0	0.0	0.0
duplicate rate:	0.0	0.0	0.0
misorder rate:	0.0	0.0	0.0
slow/q1:			
loss rate:	0.0	0.0	0.0
duplicate rate:	0.0	0.0	0.0
misorder rate:	0.0	0.0	0.0
slow/q2:			
loss rate:	0.0	0.0	0.0
duplicate rate:	0.0	0.0	0.0
misorder rate:	0.0	0.0	0.0
sys_slow_5min:	34263.57	31856.15	31947.64
sys_slow_current_size:	[ ]	[ ]	[ ]
sys_slow_active:	[20,22,21]	[24, 23,22]	[17,19]
fast/q0:			
loss rate:	0.0	0.0	0.0
duplicate rate:	0.0	0.0	0.0
misorder rate:	0.0	0.0	0.0
fast/q1:			
loss rate:	0.908	0.9	0.938
duplicate rate:	0.0	0.0	0.0

misorder rate:	0.0	0.0	0.0
fast/q2:			
loss rate:	0.914	0.91	0.943
duplicate rate:	0.0	0.0	0.0
misorder rate:	0.0	0.0	0.0
sys_fast_5min:	33543.73	30419.37	37426.64
sys_fast_current_size:	[ ]	[ ]	[ ]
sys_fast_active:	[28,27,26]	[21,20]	[27,28,29]

My comparison is the change between the three results(in three runs) of the same rate(fast/low) and level. It can be seen that the loss rate is higher when all the related number of $SYS topics goes high which is as my exception above(but I received no message from heap/current size). And the duplicate and out-of-order rate are all 0 in the three runs.

After comparing and observing the loss rate and the gap-mean/gap-variation in fast topics, it can find out that higher gap-mean/gap-variation may have a higher loss rate of packets.
Related results:
studentreport/u6683953/fast/0/gap:5.139
studentreport/u6683953/fast/0/gvar:38.006
studentreport/u6683953/fast/0/loss:0.0
studentreport/u6683953/fast/1/gap:92.153
studentreport/u6683953/fast/1/gvar:205.277
studentreport/u6683953/fast/1/loss:0.932
studentreport/u6683953/fast/2/gap:11.06
studentreport/u6683953/fast/2/gvar:232.509
studentreport/u6683953/fast/2/loss:0.943

3.
I subscribe to the three levels of the same rate(fast/slow) at the same time(without $SYS topics) and load the counter number and the timestamp of each packet into a list. Then using the list to calculate each measurement.

For the total number of messages I received, I calculate the length of the whole list. And then calculate the rate of it by dividing the time(300 seconds). It indicates the frequency of receiving messages of each topic. It can be seen in the result that the three low topics received the message at the same rate. The rates of receiving messages in fast counter topics are much higher. The rate in fast counters is different because of the loss/duplicate/out-of-order in the traffic.

For the loss message, the definition is: the number that received is not steadily increased because the publishing message is a steadily increasing counter. So I exclude the duplicate

and sot the list. (duplicate is not loss message and it would disturb the calculation) Then I compare each number with its previous one to decide whether it has a loss. (The max number to a new beginning is a specific one need to be excluded) And the loss number is subtraction. (sequence:1236, loss number=6-3-1=2) Then calculate the rate by dividing the total number of the original list(without loss and duplicate) to get the loss rate. This measurement can reflect the performance and the capacity of handling the traffic of each level and publishing rate. It can be seen in the result that the loss rate in slow counter messages which is 0, and the rate increases as the QoS level goes high in the fast counter message.

For the duplicate number: the Duplicate message is the message received more than once which can only happen in QoS level1. The definition of duplicate number=original list length - without duplicate list length. Then it can calculate the duplicate rate by dividing the number of receiving packets. Duplicate is not good messages which can waste bandwidth and many other resources in the traffic. So it can also reflect the performance and the capacity of handling the traffic. It can be seen in the result that there is no duplicate in the traffic of all 6 topics.

For the out of order, if the number is not increasing to the previous one(the max number to 0 is a specific one), it is out-of-order. If the list contains the max number---> 0, then excluding it as a specific one. Then I compare each number with its previous one to decide whether it is out of order and calculate the rate of that by dividing the number of receiving packets. This measurement also reflects the performance and capacity of handling the traffic because out-of-order is also a bad message in the traffic. It can be seen in the result that there is no out-of-order message in the traffic of all 6 topics.

For the message gap, I record the timestamp of every packet into a list, and then calculate the time gap between each packet, and calculate the mean and the standard deviation. This measurement also reflects the frequency of receiving messages and the variation of it which can reflect the stability in the traffic. It can be seen in the result that the average time gaps are approximately the same in the slow counter, and the time gaps of the fast counter message are much smaller which indicate a higher frequency. Because of more loss message, the variation is bigger in the fast counter messages. And among the three levels in the fast counter messages, the gap means/gap variation also bigger with a bigger loss rate.

P.S. The network was super unstable and bad in my accommodation. So the results of my code always changed of each run, I tried my best to pick one to publish, but it’s still not perfect. So I make more words to illustrate my algorithm and expectation of measurements, and I hope that can make up the unperfect results that I published to the broker.

After observing the results, it can be obviously seen that the rate of publishing message also affects the performance of the traffic. So I also want to publish the average rate of loss/duplicate/misordered of the slow and fast counter messages. Although it can also be concluded from the existing publish, it's not straightforward with different QoS levels and counter rates. It's better to make an average number to indicate the slow/fast counter. It can be more straightforward to observe the influence of the rate of publishing messages.

For the performance of handling each level of traffic, it depends on the loss/duplicate/misordered number, if it is higher means it handles worse. As the results which are shown above, slow topics are all good, when it comes to the fast topics, the performance is worse as the QoS level goes up. So the measurement tells me when the message published at a high rate, the capacity of transport messages decreases as the increase of QoS level.

4.
a)MQTT is lightweight and the packet can be very small so it can work on lower bandwidth and fewer computer resources. It can be used in almost any number of devices in a proper way. Therefore, for the challenge of the network in our scenario, simply knowing the number of publishers and clients is hard to measure the challenge. There are several factors closely related to the measurement of the challenge which can't be ignored, such as each packets' size(bigger payload in the packets brings more challenge), which QoS level publisher and broker use(higher level needs more resources), the frequency of publishing(more frequent brings more challenge).

When it has thousands of subscribers in MQTT but millions of publishers in a situation, so the challenge of the network between publishers and the broker and the network between broker and subscribers are different. When the number of devices is big, developer usually choose the enterprise MQTT broker like HiveMQ or mosquito
which have a much better capacity than common computers and can handle much more devices than our requirements(refer to the HiveMQ 10-mio-benchmark-paper), so it's no need to worry about the memory or CPU challenges in the broker if we send packets in common size and frequency. But the subscribers don't have such high capacity as the broker. There might be millions of packets sent from broker to one subscriber in a short time. So the challenge between broker and clients is more serious.

For the publisher, if the packet is too big(maximum 256MB) such as it has too many topics to make the payload very huge(maybe one million) and it would make the packets in a huge size. When the packets are published at the same time by millions of publishers to the broker(possible to happen), it would bring a big challenge to the bandwidth. And for the persisting session in subscription, if the packets are too big and too many, it would also bring a challenge to the broker's memory to wait for clients' reconnection. Also for the high frequency of publishing(such as more than 100 per seconds), it would need more resources(CPU usage and memory) for the broker to buffer the message into a queue or sent too much data to subscribers because the number of packets might be super huge(might be more than 1 million publishing message * 100 per second). Moreover, the level of QoS also affects the number of packets. Higher level needs more packets to guarantee the receiving(more details in section b). The challenge above also happened and amplified between broker and client, the subscribers have to receive almost the same number of packets as the broker but with much weak capacity of CPU and memory.

But MQTT is designed to apply in the extreme scenarios like ours. The challenge can be alleviated in several ways, such as compressing the content, using a lower level of QoS,

setting fewer flags(like clean sessions or retain flag) to reduce the workload of CPU and memory and tuning an appropriate frequency of publishing.

b)For the QoS level 0, it doesn't need to store and resent packet, so it can save many resources in the scenario with a large number of devices. But it can't guarantee the safety because it can cause loss. As our publishers are sensors, we don't know what the security level it needs. If the loss can be tolerated, the level 0 would be the best to transfer with a large number of publishers and clients.

For QoS level 1, the publisher would wait for the acknowledge packet, and before receiving the acknowledge message, the sender needs to store and keep publishing the packet, so level1 needs to write in the disk. So this level would not help to save memory. Due to the additional acknowledge packet and the duplicate messages, it would need more bandwidth in level1.

For level2, it is the safest level, but it needs more packets to guarantee the packets received only once. (4 handshakes when publishing a message) So this level needs the most resources of bandwidth and memory.

And if the clients set the clean session(only in level1 and 2), the broker needs to store the packets until the client reconnects. Also, and retain message would also occupy the broker's memory.

c)As the measurement I used in the assignment, when the publishing rate goes high, the rate of loss also goes high. And when the message published at a high rate, the capacity and performance of transport messages decrease as the increase of QoS level. So that can be also worked in this situation. When the publishers and clients are too many, tunning the publishing rate and the choice of QoS level are important. It can bring a huge improvement and help the traffic goes much better. My plan is to use the same measurements of loss/duplicate/out-of-order into the prototype of our situation and test it from a high rate of publishing to lower rates and observe the results of measurements, then tuning the publishing rate into an appropriate one which has an acceptable rate of loss/duplicate/out-of-order. Also, use the same method to choose a QoS level. It can be helpful to find a balance point between effective(high-rate publishing) and secure(higher QoS level).

Reference

[1]HiveMQ,"10,000,000 MQTT Clients HiveMQ Cluster Benchmark Paper ", https://www.hivemq.com/downloads/hivemq-10-mio-benchmark.pdf, 2017