Q1

My Source: 10.211.55.7; Me as a **Subscribe Client** 3310-u6325688
My Destination:52.15.211.32; It's the Broker(**Server**)
ACK/ SYN-ACK/SYN in Q1 are sent via TCP protocol.

QoS 0 (At most once)

```
mosquitto_sub -h comp3310.ddns.net -u students -P 33106331 -t counter/slow/q0 -i 3310-u6325688 -q 0
```

Screen Shot

ip.addr==52.15.211.32 & ip.addr==10.211.55.7

No.	Time	Source	Destination	Protocol	Length	Info
8	5.745367802	10.211.55.7	52.15.211.32	TCP	74	57378 → 1883 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1051923393 TSecr=0 WS=128
9	6.033237956	52.15.211.32	10.211.55.7	TCP	62	1883 → 57378 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=1460 WS=2
10	6.033303412	10.211.55.7	52.15.211.32	TCP	54	57378 → 1883 [ACK] Seq=1 Ack=1 Win=29312 Len=0
11	6.033487767	10.211.55.7	52.15.211.32	MQTT	103	Connect Command
12	6.033726127	52.15.211.32	10.211.55.7	TCP	54	1883 → 57378 [ACK] Seq=1 Ack=50 Win=32768 Len=0
13	6.287756233	52.15.211.32	10.211.55.7	MQTT	58	Connect Ack
14	6.287805151	10.211.55.7	52.15.211.32	TCP	54	57378 → 1883 [ACK] Seq=50 Ack=5 Win=29312 Len=0
15	6.288003581	10.211.55.7	52.15.211.32	MQTT	76	Subscribe Request
16	6.288188975	52.15.211.32	10.211.55.7	TCP	54	1883 → 57378 [ACK] Seq=5 Ack=72 Win=32768 Len=0
17	6.524899437	52.15.211.32	10.211.55.7	MQTT	59	Subscribe Ack
18	6.566909260	10.211.55.7	52.15.211.32	TCP	54	57378 → 1883 [ACK] Seq=72 Ack=10 Win=29312 Len=0
19	6.851823985	52.15.211.32	10.211.55.7	MQTT	78	Publish Message
20	6.851867277	10.211.55.7	52.15.211.32	TCP	54	57378 → 1883 [ACK] Seq=72 Ack=34 Win=29312 Len=0
21	7.101803701	52.15.211.32	10.211.55.7	MQTT	76	Publish Message
22	7.101847155	10.211.55.7	52.15.211.32	TCP	54	57378 → 1883 [ACK] Seq=72 Ack=56 Win=29312 Len=0
23	8.082193573	52.15.211.32	10.211.55.7	MQTT	76	Publish Message
24	8.082246103	10.211.55.7	52.15.211.32	TCP	54	57378 → 1883 [ACK] Seq=72 Ack=78 Win=29312 Len=0
25	9.004551456	52.15.211.32	10.211.55.7	MQTT	76	Publish Message
26	9.004600028	10.211.55.7	52.15.211.32	TCP	54	57378 → 1883 [ACK] Seq=72 Ack=100 Win=29312 Len=0
27	9.037254024	10.211.55.7	52.15.211.32	TCP	54	57378 → 1883 [FIN, ACK] Seq=72 Ack=100 Win=29312 Len=0
28	9.037580825	52.15.211.32	10.211.55.7	TCP	54	1883 → 57378 [ACK] Seq=100 Ack=73 Win=32768 Len=0
29	9.430019769	52.15.211.32	10.211.55.7	TCP	54	1883 → 57378 [FIN, ACK] Seq=100 Ack=73 Win=32768 Len=0
30	9.430086858	10.211.55.7	52.15.211.32	TCP	54	57378 → 1883 [ACK] Seq=73 Ack=101 Win=29312 Len=0

How hand shake work

(1)entry No.8 - entry No.10: all packets are sent by TCP protocol. 3 hand shake: Client sends a SYN packet to Server, once server receives client's STN; server sends SYN-ACK packet to client, once client receives SYN-ACK packet; client sends ACK packet to server. Once the server receives ACK, the TCP socket connection is established.
(2)entry No.11 - No.26: firstly client sends "CONNECT" and "SUBSCRIBE" via MQTT protocol, each one will let client receive ACK from sever. Once the subscription success, server will "Publish Message" to client continuously via MQTT protocol, client sends ACK to server when itself receives messages of the topic "counter/slow/q0".
(3)entry No.27 - No.30: It shows the TCP connection is in termination stage, which means client no long subscribes the "counter/slow/q0".

What it implies message duplication

Nothing relate to message duplication in QoS0, once sent by sender, once forget.

What it implies message order

There is no order need.

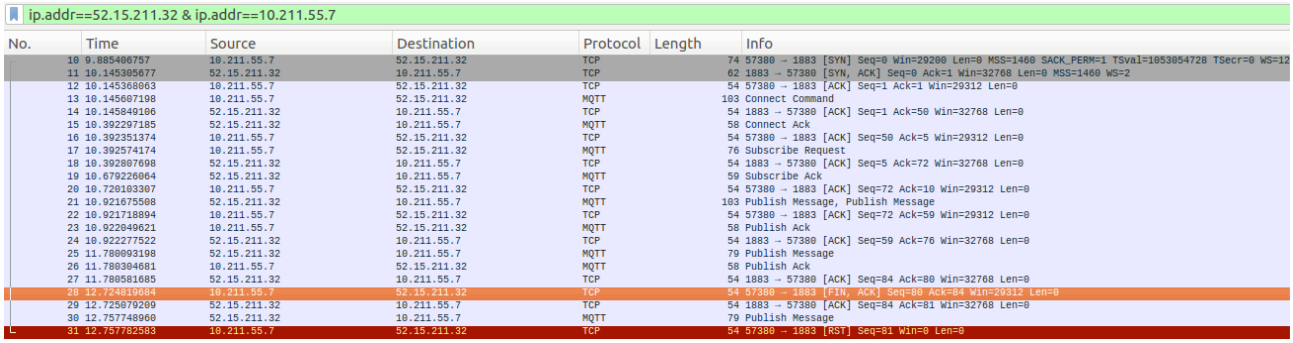
The circumstances of using QoS 0

When the internet is fast and reliable, the messages can be received without loss. When clients do not need full messages and store the published messages on server.

QoS 1 (At least once)

```
mosquitto_sub -h comp3310.ddns.net -u students -P 33106331 -t counter/slow/q1 -i 3310-u6325688 -q 1
```

Screen Shot



ip.addr==52.15.211.32 & ip.addr==10.211.55.7

No.	Time	Source	Destination	Protocol	Length	Info
10	9.885406757	10.211.55.7	52.15.211.32	TCP	74	57380 → 1883 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1053054728 TSecr=0 WS=12
11	10.145305677	52.15.211.32	10.211.55.7	TCP	62	1883 → 57380 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=1460 WS=2
12	10.145368063	10.211.55.7	52.15.211.32	TCP	54	57380 → 1883 [ACK] Seq=1 Ack=1 Win=29312 Len=0
13	10.145607198	10.211.55.7	52.15.211.32	MQTT	103	Connect Command
14	10.145849106	52.15.211.32	10.211.55.7	TCP	54	1883 → 57380 [ACK] Seq=1 Ack=50 Win=32768 Len=0
15	10.392297185	52.15.211.32	10.211.55.7	MQTT	58	Connect Ack
16	10.392351374	10.211.55.7	52.15.211.32	TCP	54	57380 → 1883 [ACK] Seq=50 Ack=5 Win=29312 Len=0
17	10.392574174	10.211.55.7	52.15.211.32	MQTT	76	Subscribe Request
18	10.392807698	52.15.211.32	10.211.55.7	TCP	54	1883 → 57380 [ACK] Seq=5 Ack=72 Win=32768 Len=0
19	10.679226064	52.15.211.32	10.211.55.7	MQTT	59	Subscribe Ack
20	10.720103307	10.211.55.7	52.15.211.32	TCP	54	57380 → 1883 [ACK] Seq=72 Ack=10 Win=29312 Len=0
21	10.921675508	52.15.211.32	10.211.55.7	MQTT	103	Publish Message, Publish Message
22	10.921718894	10.211.55.7	52.15.211.32	TCP	54	57380 → 1883 [ACK] Seq=72 Ack=59 Win=29312 Len=0
23	10.922049621	10.211.55.7	52.15.211.32	MQTT	58	Publish Ack
24	10.922277522	52.15.211.32	10.211.55.7	TCP	54	1883 → 57380 [ACK] Seq=59 Ack=76 Win=32768 Len=0
25	11.780093198	52.15.211.32	10.211.55.7	MQTT	79	Publish Message
26	11.780304681	10.211.55.7	52.15.211.32	MQTT	58	Publish Ack
27	11.780581685	52.15.211.32	10.211.55.7	TCP	54	1883 → 57380 [ACK] Seq=84 Ack=80 Win=32768 Len=0
28	12.724819084	10.211.55.7	52.15.211.32	TCP	54	57380 → 1883 [FIN, ACK] Seq=80 Ack=84 Win=29312 Len=0
29	12.725079209	52.15.211.32	10.211.55.7	TCP	54	1883 → 57380 [ACK] Seq=84 Ack=81 Win=32768 Len=0
30	12.757748960	52.15.211.32	10.211.55.7	MQTT	79	Publish Message
31	12.757782583	10.211.55.7	52.15.211.32	TCP	54	57380 → 1883 [RST] Seq=81 Win=0 Len=0

How hand shake work

(1)entry No.10 - entry No.12: all packets are sent by TCP protocol. It's the "TCP 3 hand shake" which has the same procedure as QoS0.

(2)entry No.11 - No.19: firstly client sends "CONNECT" and "SUBSCRIBE" via MQTT protocol, each one will let client receive ACK from sever. Then the subscription success.

(3)entry No.21 - No.27: server will "Publish Message" to client continuously via MQTT protocol, client sends ACK to server when itself receives messages of the topic "counter/slow/q1"; after the TCP ACK, client also sends "Publish ACK" to server, then server sends TCP ACK back to client. Such as entry No.21, server keep sending "Publish message" to client until got TCP back.

(4)entry No.28 - No.31:It shows the TCP connection is in termination stage, which means client no long subscribes the "counter/slow/q1". In the end, when server sends unexpected "Publish Message" to client, client will send reset packet with the RST bit to server.

What it implies message duplication

Cause QoS1 guarantee client and server to receive messages. If the "Publish ACK" is not sent, the duplicated "Publish Message" will be sent continuously. The DUP flag will be sent. For the subscriber, it will get duplications.

What it implies message order

The duplication message may make the messages received by subscriber be not unordered.

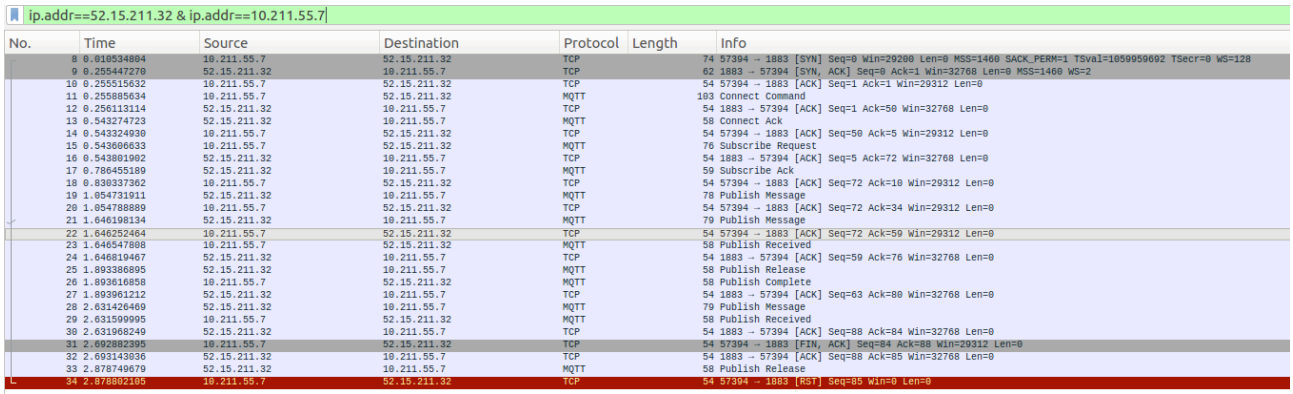
The circumstances of using QoS 1

When you must have every messages from broker and you don't care duplication messages.

QoS 2 (Exactly once)

```
mosquitto_sub -h comp3310.ddns.net -u students -P 33106331 -t counter/slow/q2 -i 3310-u6325688 -q 2
```

Screen Shot



ip.addr==52.15.211.32 & ip.addr==10.211.55.7

No.	Time	Source	Destination	Protocol	Length	Info
8	0.010534804	10.211.55.7	52.15.211.32	TCP	74	57394 → 1883 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1059959692 TSecr=0 WS=128
9	0.255447270	52.15.211.32	10.211.55.7	TCP	62	1883 → 57394 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=1460 WS=2
10	0.255515632	10.211.55.7	52.15.211.32	TCP	54	57394 → 1883 [ACK] Seq=1 Ack=1 Win=29312 Len=0
11	0.255885634	10.211.55.7	52.15.211.32	MQTT	103	Connect Command
12	0.256113114	52.15.211.32	10.211.55.7	TCP	54	1883 → 57394 [ACK] Seq=1 Ack=50 Win=32768 Len=0
13	0.543274723	52.15.211.32	10.211.55.7	MQTT	58	Connect Ack
14	0.543324930	10.211.55.7	52.15.211.32	TCP	54	57394 → 1883 [ACK] Seq=50 Ack=5 Win=29312 Len=0
15	0.543606633	10.211.55.7	52.15.211.32	MQTT	76	Subscribe Request
16	0.543801902	52.15.211.32	10.211.55.7	TCP	54	1883 → 57394 [ACK] Seq=5 Ack=72 Win=32768 Len=0
17	0.786455189	52.15.211.32	10.211.55.7	MQTT	59	Subscribe Ack
18	0.830337362	10.211.55.7	52.15.211.32	TCP	54	57394 → 1883 [ACK] Seq=72 Ack=10 Win=29312 Len=0
19	1.054731911	52.15.211.32	10.211.55.7	MQTT	78	Publish Message
20	1.054788889	10.211.55.7	52.15.211.32	TCP	54	57394 → 1883 [ACK] Seq=72 Ack=34 Win=29312 Len=0
21	1.646198134	52.15.211.32	10.211.55.7	MQTT	79	Publish Message
22	1.646252464	10.211.55.7	52.15.211.32	TCP	54	57394 → 1883 [ACK] Seq=72 Ack=59 Win=29312 Len=0
23	1.646547808	10.211.55.7	52.15.211.32	MQTT	58	Publish Received
24	1.646819467	52.15.211.32	10.211.55.7	TCP	54	1883 → 57394 [ACK] Seq=59 Ack=76 Win=32768 Len=0
25	1.893386895	52.15.211.32	10.211.55.7	MQTT	58	Publish Release
26	1.893616858	10.211.55.7	52.15.211.32	MQTT	58	Publish Complete
27	1.893961212	52.15.211.32	10.211.55.7	TCP	54	1883 → 57394 [ACK] Seq=63 Ack=80 Win=32768 Len=0
28	2.631426469	52.15.211.32	10.211.55.7	MQTT	79	Publish Message
29	2.631599995	10.211.55.7	52.15.211.32	MQTT	58	Publish Received
30	2.631968249	52.15.211.32	10.211.55.7	TCP	54	1883 → 57394 [ACK] Seq=88 Ack=84 Win=32768 Len=0
31	2.692882395	10.211.55.7	52.15.211.32	TCP	54	57394 → 1883 [FIN, ACK] Seq=84 Ack=88 Win=29312 Len=0
32	2.693143036	52.15.211.32	10.211.55.7	TCP	54	1883 → 57394 [ACK] Seq=88 Ack=85 Win=32768 Len=0
33	2.878749679	52.15.211.32	10.211.55.7	MQTT	58	Publish Release
34	2.878892105	10.211.55.7	52.15.211.32	TCP	54	57394 → 1883 [RST] Seq=85 Win=0 Len=0

How hand shake work

(1)entry No.8 - entry No.10: all packets are sent by TCP protocol. It's the "TCP 3 hand shake" which has the same procedure as QoS0.

(2)entry No.11 - No.18: firstly client sends “CONNECT” and “SUBSCRIBE” via MQTT protocol, each one will let client receive ACK from sever. Then the subscription success.
(3)entry No.19 - No.30: server will “Publish Message” to client via MQTT protocol, client sends ACK to server when itself receives messages of topic “counter/slow/q2”; after the TCP ACK, client also sends “Publish Received” to server, then server sends TCP ACK back to client. The “Publish Release” will be also sent by server to client, next, client sends “Publish Complete”, another TCP ACK can also be sent from server to client. Not every time the TCP will be sent from server or from client.
(4)entry No.31 - No.34:It shows the TCP connection is in termination stage, which means client no long subscribes the “counter/slow/q2”. In the end, when server sends unexpected “Publish Message” to client, client will send reset packet with the RST bit to server.

What it implies message duplication

It’s guaranteed by message ID. So, cannot be duplicated.

What it implies message order

It doesn’t guarantee order. Since the duplication message will appear later.

The circumstances of using QoS 2

If you wanna get one-time reliable message.

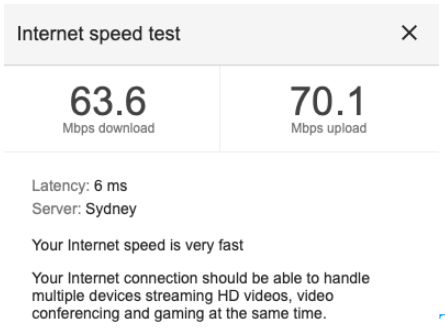
Q2

(a)

Queue: input message info by O(1). It's used to store counter messages in order
HashMap: input by O(1), search by O(1), delete by O(1). It's used to check loss

Running Methods

The MQTT status code is from HiveMQ
Network: Public Wifi



Test link

You will see the MQTT subscribe output every 10 seconds.

The rate of messages you receive *[messages/second]*

RECV Rate = *(Total Number of messages received) / [(TimeStamp at N) - (TimeStamp at 0)]*

The rate of message loss you see *[percentage]*

Get the smallest number *S*, and largest number *L* from actually received messages
Generate an expected payloads set *A* from the *S* to *L*.
Actual received payload set *B*.
Total Number of message loss = sizeOf(A-B)
Loss percentage = *(Total Number of message loss) / (Total Number of messages received)*

The rate of duplicated messages you see *[percentage]*

The duplication is measured by paho-mqtt "dup". If the "dup" != 0, then the dup counter increment by 1.
Duplicate percentage = *(Total Number of message duplicate) / (Total Number of messages received)*

The rate of out-of-order messages you see *[percentage]*

If int(current message payload) < int(last message payload), then the OOO counter increment by 1.
OOO percentage = *(Total Number of message OOO) / (Total Number of messages received)*

The mean Inter-message-gap and gap-variation

Mean Inter-message-gap = *(Sum of message gap) / (Total Number of messages gap)*

Std Inter-message-gap = $\sqrt{\frac{\sum (G - \bar{G})^2}{n}}$, G: gap, G bar: Mean Inter-message-gap

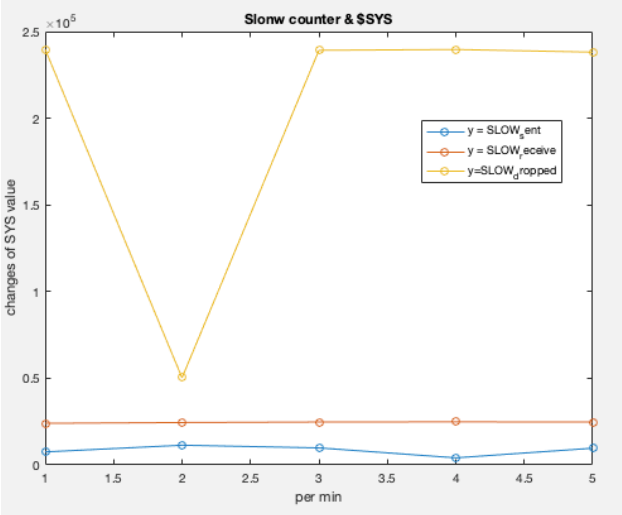
2(b) & 2(c)

I use mqtt-spy to subscribe the $SYS topics by another client ID: 3310-u6325688SYS

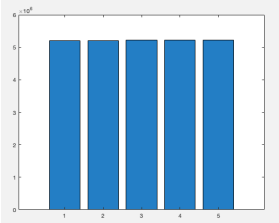
```
MQTT_TOPIC_SYS = [("$SYS/broker/load/publish/sent/1min"),
                  ("$SYS/broker/load/publish/received/1min"),
                  ("$SYS/broker/load/publish/dropped/1min"),
                  ("$SYS/broker/heap/current"),
                  (“$SYS/broker/clients/active")]
```

I run my code and the mqtt-spy at the same time, then my timestamp will help me find the data from the mqtt-spy

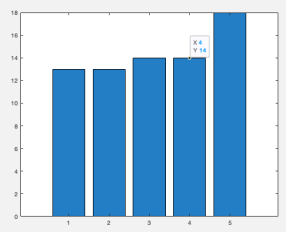
Slow Counter



heap/Current



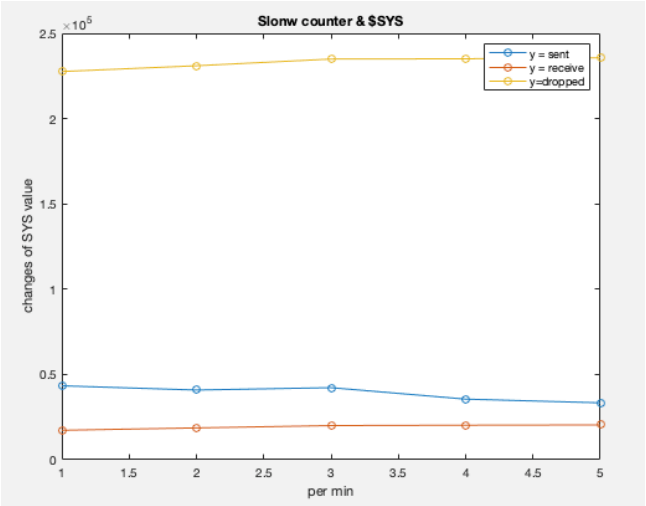
clients/Active



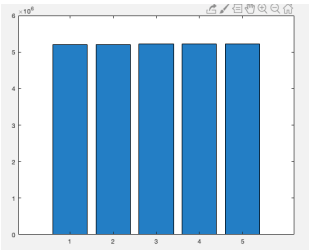
```
{'qos': '0', 'recv': '0.9899(messages/second)', 'loss': '0.0%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '1010.2325(milliseconds)', 'gvar': '151.5587(milliseconds)'
{'qos': '1', 'recv': '0.9899(messages/second)', 'loss': '0.0%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '1010.2311(milliseconds)', 'gvar': '151.5636(milliseconds)'
{'qos': '2', 'recv': '0.9908(messages/second)', 'loss': '0.0%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '1009.3309(milliseconds)', 'gvar': '144.6533(milliseconds)'
Disconnected: 0
2019-05-30 07:10:53.462003
{'qos': '0', 'recv': '0.9855(messages/second)', 'loss': '0.0%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '1014.6862(milliseconds)', 'gvar': '146.746(milliseconds)'}
{'qos': '1', 'recv': '0.9855(messages/second)', 'loss': '0.0%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '1014.6926(milliseconds)', 'gvar': '146.7713(milliseconds)'}
{'qos': '2', 'recv': '0.991(messages/second)', 'loss': '0.0%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '1009.0424(milliseconds)', 'gvar': '137.759(milliseconds)'}
Disconnected: 0
2019-05-30 07:11:56.063582
{'qos': '0', 'recv': '0.9885(messages/second)', 'loss': '0.0%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '1011.6735(milliseconds)', 'gvar': '158.8241(milliseconds)'}
{'qos': '1', 'recv': '0.9885(messages/second)', 'loss': '0.0%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '1011.6708(milliseconds)', 'gvar': '158.8267(milliseconds)'}
{'qos': '2', 'recv': '0.988(messages/second)', 'loss': '0.0%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '1012.1767(milliseconds)', 'gvar': '144.8643(milliseconds)'}
Disconnected: 0
2019-05-30 07:12:58.490855
{'qos': '0', 'recv': '0.9924(messages/second)', 'loss': '0.0%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '1007.6354(milliseconds)', 'gvar': '156.898(milliseconds)'}
{'qos': '1', 'recv': '0.9924(messages/second)', 'loss': '0.0%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '1007.6355(milliseconds)', 'gvar': '156.9024(milliseconds)'}
{'qos': '2', 'recv': '0.991(messages/second)', 'loss': '0.0%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '1009.0921(milliseconds)', 'gvar': '144.3599(milliseconds)'}
Disconnected: 0
2019-05-30 07:14:00.338195
{'qos': '0', 'recv': '0.9914(messages/second)', 'loss': '0.0%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '1008.6299(milliseconds)', 'gvar': '151.4099(milliseconds)'}
{'qos': '1', 'recv': '0.9915(messages/second)', 'loss': '0.0%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '1008.6232(milliseconds)', 'gvar': '151.4201(milliseconds)'}
{'qos': '2', 'recv': '0.9908(messages/second)', 'loss': '0.0%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '1009.3322(milliseconds)', 'gvar': '142.965(milliseconds)'}
Disconnected: 0
2019-05-30 07:15:03.011000
```

When Server dropped significantly at the second minute, the gap mean for the QoS0, QoS1 is bigger than other minutes. There is no loss, dup or ooo in slower counter. The heap for the 5 minutes are steady. The active client is higher in the last minutes, however, it doesn't influence the message gap mean. But, for the $SYS/broker/load/publish/sent/, it has the same status changes as the message gap mean. The sent action of Server may influence the message gap mean.

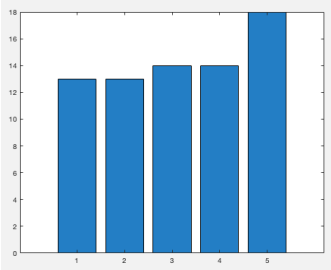
Fast Counter



Current



Active



{'qos': '0', 'recv': '131.7623(messages/second)', 'loss': '0.0%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '7.5894(milliseconds)', 'gvar': '37.038(milliseconds)'}
{'qos': '1', 'recv': '15.413(messages/second)', 'loss': '87.5251%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '64.8805(milliseconds)', 'gvar': '131.5307(milliseconds)'}
{'qos': '2', 'recv': '14.0124(messages/second)', 'loss': '88.6043%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '71.3653(milliseconds)', 'gvar': '133.8977(milliseconds)'}
Disconnected: 0
2019-05-30 08:35:47.177313

{'qos': '0', 'recv': '160.7512(messages/second)', 'loss': '0.0%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '6.2208(milliseconds)', 'gvar': '32.8507(milliseconds)'}
{'qos': '1', 'recv': '15.5802(messages/second)', 'loss': '89.7802%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '64.1839(milliseconds)', 'gvar': '132.5932(milliseconds)'}
{'qos': '2', 'recv': '13.9646(messages/second)', 'loss': '90.8275%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '71.6095(milliseconds)', 'gvar': '144.039(milliseconds)'}
Disconnected: 0
2019-05-30 08:36:48.674254

{'qos': '0', 'recv': '171.1217(messages/second)', 'loss': '0.0%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '5.8438(milliseconds)', 'gvar': '31.4625(milliseconds)'}
{'qos': '1', 'recv': '16.448(messages/second)', 'loss': '89.7816%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '60.7978(milliseconds)', 'gvar': '127.1223(milliseconds)'}
{'qos': '2', 'recv': '14.0803(messages/second)', 'loss': '91.2166%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '71.0214(milliseconds)', 'gvar': '143.4819(milliseconds)'}
Disconnected: 0
2019-05-30 08:37:49.937882

{'qos': '0', 'recv': '169.5449(messages/second)', 'loss': '0.0%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '5.8981(milliseconds)', 'gvar': '31.3955(milliseconds)'}
{'qos': '1', 'recv': '15.8309(messages/second)', 'loss': '90.02%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '63.1674(milliseconds)', 'gvar': '146.1914(milliseconds)'}
{'qos': '2', 'recv': '13.9(messages/second)', 'loss': '91.2123%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '71.9424(milliseconds)', 'gvar': '148.3157(milliseconds)'}
Disconnected: 0
2019-05-30 08:38:50.982063

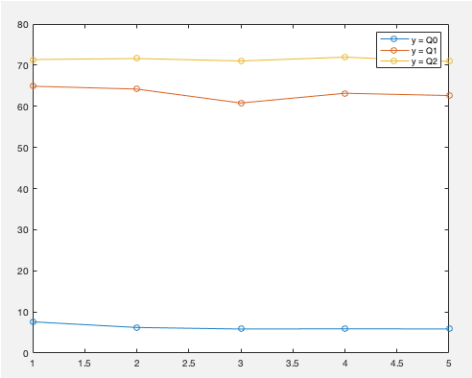
{'qos': '0', 'recv': '170.8002(messages/second)', 'loss': '0.0%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '5.8548(milliseconds)', 'gvar': '31.1072(milliseconds)'}
{'qos': '1', 'recv': '15.9713(messages/second)', 'loss': '90.0621%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '62.6124(milliseconds)', 'gvar': '144.4357(milliseconds)
{'qos': '2', 'recv': '14.1035(messages/second)', 'loss': '91.2059%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '70.9043(milliseconds)', 'gvar': '147.7502(milliseconds)
Disconnected: 0
2019-05-30 08:39:51.972723

In this fast counter, the loss happened in QoS1,QoS2 over the 5 mins. There is a “up” trend for server dropping the messages, the loss for the 2 levels also has the “up” trend during the period. The amount of server “sent” has decrease trend, at the same time, the message gap also decrease over the 5 mins. There is no dup or ooo happened over the 5 mins.

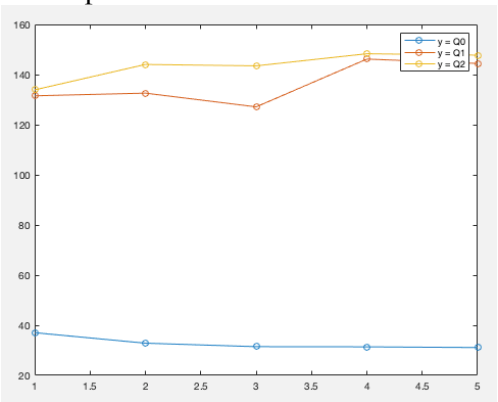
Summary:

I expect to see the server heap, and the changes of activated clients will affect the loss, however, the clients’ connection will not affect too much the loss. I also expected the server’s receive could influence the message gaps, there is no clue shows this.

Mean-Gap



Std-Gap



But after observing the two counters (slow & fast), the changes of server's sent [$SYS/broker/load/publish/sent/1min] do impact the the message gap mean. The client side is slow to receive messages.

Q3

Summary of measurement

slow q0 recv=0.9762(messages/second)
slow q0 loss=0.0%
slow q0 dupe=0.0%
slow q0 ooo=0.0%
slow q0 gap=1024.3348(milliseconds)
slow q0 gvar=94.9343(milliseconds)
---
slow q1 recv=0.9762(messages/second)
slow q1 loss=0.0%
slow q1 dupe=0.0%
slow q1 ooo=0.0%
slow q1 gap=1024.3348(milliseconds)
slow q1 gvar=94.9321(milliseconds)
---
slow q2 recv=0.9772(messages/second)
slow q2 loss=0.0%
slow q2 dupe=0.0%
slow q2 ooo=0.0%
slow q2 gap=1023.2869(milliseconds)
slow q2 gvar=83.2434(milliseconds)
---
fast q0 recv=125.2546(messages/second)
fast q0 loss=0.0%
fast q0 dupe=0.0%
fast q0 ooo=0.0%
fast q0 gap=7.9837(milliseconds)
fast q0 gvar=37.6453(milliseconds)
---
fast q1 recv=16.0984(messages/second)
fast q1 loss=86.9826%
fast q1 dupe=0.0%
fast q1 ooo=0.0%
fast q1 gap=62.1178(milliseconds)
fast q1 gvar=122.1576(milliseconds)
---
fast q2 recv=14.3988(messages/second)
fast q2 loss=88.3784%
fast q2 dupe=0.0%
fast q2 ooo=0.0%
fast q2 gap=69.45(milliseconds)
fast q2 gvar=129.1986(milliseconds)

Explanation of broker\server's performances in QoS 0

The server just send and forget, there is no duplications case in all of above records. Since there is no needs to ACK, the recv rate is the highest in 3 levels. Same reason, the message gap is the smallest in the 3 levels. There are no duplication.

Explanation of broker\server's performances in QoS 1

The server make the message gap smaller than QoS2. Even the receive rate can be larger than QoS2.

Explanation of broker\server's performances in QoS 2

Compared to the first 2 levels, QoS2 has shown its complex ACK can make the server have lower performance. There is no duplications.

Summary:
I use Python 3.7 to run under the public WIFI network environment.
According above measurements, the performance of server are relevant to the levels. Except the loss happened in both QoS2, and QoS3.

I tried use another python2.7 to run the same subscribing, there is no loss in the QoS2 and QoS3. Therefore, the reason caused the loss may incurred by the coding language or its version.

The 2.7 version python code in folder: ass27.py in 5 seconds

```
total msg received:656

{'loss': '0.0%', 'qos': '0', 'ooo': '0.0%', 'gvar': '32.3508(milliseconds)', 'gap': '6.9689(milliseconds)', 'dupe': '0.0%', 'recv': '143.4949(messages/second)'}
Qos
total msg received:79

{'loss': '0.0%', 'qos': '1', 'ooo': '0.0%', 'gvar': '98.5744(milliseconds)', 'gap': '57.9097(milliseconds)', 'dupe': '0.0%', 'recv': '17.2683(messages/second)'}
total msg received:69

{'loss': '0.0%', 'qos': '2', 'ooo': '0.0%', 'gvar': '106.7731(milliseconds)', 'gap': '58.0007(milliseconds)', 'dupe': '0.0%', 'recv': '17.2412(messages/second)'}
```

The 3.7 version python code in folder: Assign3.py in 5 seconds

```
Last login: Thu May 30 16:06:14 on ttys001
/Users/remosy/.anaconda/navigator/a.tool ; exit;
(base) Remosy1:~ remosy$ /Users/remosy/.anaconda/navigator/a.tool ; exit;
(Demo) bash-3.2$ python3 /Users/remosy/Desktop/COMP3310RR/Assignment3/Assign3.py
Connected to Broker: comp3310.ddns.net
ON_SUBSCRIBE %d %s 1 (0, 1, 2)
Disconnected: 0
(Demo) bash-3.2$ python3 /Users/remosy/Desktop/COMP3310RR/Assignment3/Assign3.py
Connected to Broker: comp3310.ddns.net
ON_SUBSCRIBE %d %s 1 (0, 1, 2)
{'qos': '0', 'recv': '184.9843(messages/second)', 'loss': '0.0%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '5.4059(millisecond
s)', 'gvar': '30.5981(milliseconds)'}
{'qos': '1', 'recv': '16.0664(messages/second)', 'loss': '74.2515%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '62.2417(millise
conds)', 'gvar': '118.0064(milliseconds)'}
{'qos': '2', 'recv': '15.0514(messages/second)', 'loss': '69.3878%', 'dupe': '0.0%', 'ooo': '0.0%', 'gap': '66.4391(millise
conds)', 'gvar': '126.8313(milliseconds)'}
Disconnected: 0
(Demo) bash-3.2$
```

Q4

A: what (cpu/memory/network) performance challenges there might be, from the source publishing its messages, all the way through the network to your subscribing client

The millions of sensor will be the publisher client, and the thousand of publisher will be the subscriber client. It means the broker(server) will receive millions of messages from the sensor, the subscriber will subscribe and receive tons of messages from the server, both two receivers(server and subscriber) will have the scalability challenge. To transmit/receive such massive data, “last mile” techniques are also important and challenge to receivers(server and subscriber). Sensor will only publish its own data to server, there is no big challenge to sensors.

Scalability Challenge:
The receiver will need to have enough memory to receive all of the messages, especially if there are some publishers’ messages are marked by “restain”, the worst case the server will down frequently; also the receivers cannot get full of messages, because of the memory is located fully. The receiver ’s CPU will need to be steady, not to reach over 100% CPU, when the server CPU cannot handle the publishers, or the subscriber CPU cannot solve, there will be a lot of messages lost or delayed by server/subscribers slowly handling.

Last Mile Challenge:
There should be a type of networking architecture for the last mile. Most of time, it depends on what the sensors, the subscribers are, because there are wireless and wired connection methods. Sometimes, the last mile technique is expensive, such as using current 5G. The way is limited by price or more other factors. But in this case, the large upload speed and large download speed should be considered firstly.

B: how the different QoS levels may help, or not, in dealing with the challenges

QoS0:
It’s not reliable, the receivers will not get missed messages, since the “send and forget”, there will be no duplications. If receivers wanna trade off memory, CPU with reliability, then QoS0, is a good choices. Less network congestion will make.

QoS1:
It really depends receivers. QoS1 will make more massages when there is delay/missing issues. It doesn’t help receivers when performance issues existed, cause of the physical CPU/memory limitation, QoS1 makes it worse.

QoS2:
It won’t make duplications to receivers, so it’s good to memory. However, the duplicated messages still need to be forwarded to receivers several times, the CPU still need to recognise the message ID. And QoS will have longer and more reliable message delivery than QoS1, it will make slower workload.

C: how it compares (or not) with the actual quantified differences between QoS levels you measured as part of this assignment

In my measurement, there are two different counters. Therefore, if this situation slowly make messages, the QoS0 - QoS2 will have no loss, duplication, out of order. Then compare to the fast counter, if this situation

make fast messages, then the loss will easily see in the QoS1, and 2. The complex handshake procedures for these two levels may influence the receivers' equipment or application busier, and lost messages by application(such as python3 will have loss, but in python 2.7 will have 0 loss) itself. In no matter what, the QoS0 still have good results than the other 2 levels.