COMP3310/ENGN3539/etc Assignment 3 – Testing MQTT

Introduction:

- This assignment is worth 15% of the final mark
- It is due by **Tuesday 20 May 2025 23.55 AEST**
- Late submissions will not be accepted, except in special circumstances. *Any extensions should be requested well before the due date, with appropriate evidence. Please use the extension-request link on wattle and not direct emails.*
- *This is another evolving experiment for us, any issues/corrections will be announced via wattle.*

Assignment 3

MQTT is the most common open IoT protocol being deployed today. It uses a publisher/subscriber model, allowing for an almost-unbounded number of sources to publish information, each at their own rate, and subscribers to receive information as desired. As such, it is designed to provide high-performance communication mechanisms, with minimal delays and excellent scaling performance. We'll use it to monitor the performance of some imaginary system: say counting the total kilograms of minerals rushing by on several conveyor belts, that you can control. This assignment will look at the functionality and performance of the publishers, brokers, the network (potentially) and subscribers.

This is a coding, analysis and writing assignment. You may code in C/Java/Python or any programming language that a tutor can assess (hope that's enough for everyone), and yes, you may use MQTT and other helper libraries. The assessment will not rely solely on running on your code, but more on the data gathering and your analysis. However, we will review the code and may briefly test it against our own broker running in the usual lab-type environments or similar. You need to note in your report/code any libraries you are using.

Submitting

You will be submitting two things: your code and your analysis report. Note that there will be two submission links on the Wattle course-site:

1. Your code must be submitted to wattle as a zip file, with instructions in your report on how to compile/run the components as appropriate.
2. Your analysis report (pdf) must be submitted via TurnItIn on the wattle site, so ensure you quote and cite sources properly.

Outcomes

We're assessing your understanding of MQTT, as well as your code's functionality in subscribing/publishing to a broker, dealing with high message rates from a number of sources, measuring message performance and statistics of a networked application, and your insight to interpret what you're seeing. You will be exploring MQTT functionality, the quality-of-service (QoS) levels, describing how they work, why you would use different levels, and how they perform in real-world deployments given various publishers.

Resources

You will need to set up your own MQTT server/broker for you to connect to as per the specifications below, on your own computer, or it's even better if you have a separate computer to use for the broker and publishers. There are a number of free brokers to choose from, https://mqtt.org/software/ has a good list. Ideally you could use a cloud-hosted provider to see how a long-haul network impacts performance, but they get a bit thingy about lots of students doing performance testing against their systems…

You can test your broker works by subscribing to the $SYS/# topics, which describe the server, and it will help get you familiar with the information presented there - you will be using them for your analysis later.

Assignment programming:

You need to write two programs.

- **A Publisher**:
  - A Publisher will first subscribe (listen) to a set of 'request' topics, namely **request/qos**, **request/delay, request/messagesize** and **request/instancecount**. When it sees new values for these, it will start publishing accordingly. You probably also want a **request/go** topic as a trigger, to avoid the race condition when updating the other request/# topics.
  - You will have ten instances (or threads?) of a Publisher running at the same time, called *pub-01* to *pub-10*. These will help stress the broker (and network, if you have separate computers). The 'instancecount' will tell you how many publishers should be active, the rest should keep quiet.
  - Each Publisher will send a sequence of simple messages to the broker for 30 seconds. Each message will be of the form **counter:timestamp:xxx…xxx**, where:
    - **counter** is an incrementing value for each message (0, 1, 2, 3, …).
    - **timestamp** is the precise current time (a number, millisecond level, or better).
    - **xxx…xxx** is a string of repeated characters ("x") of length <messagesize>, which can have values of 0 (i.e. no 'x'), 1000, and 4000.
  - Each Publisher will publish those messages to the broker at a requested MQTT <QoS> level (0, 1 or 2), and with a requested <delay> between messages (either 0ms, or 100ms).
  - Each Publisher will publish to the topic **counter/<instance>/<qos>/<delay>/<message-size>**, so e.g. **counter/1/0/100/4000** is the topic for messages coming from *Pub-1* at *qos=0*, *delay=100 and message-size=4000*.
  - After it has finished its burst of messages, each Publisher should go back to listening to the 'request' topics for the next round of instructions.
  - *At 0ms delay, qos=0 and messagesize=0 each publisher should be able to publish very quickly, which should be many thousands of messages per second. At 100ms delay, it's just 10 messages per second, which should have no issues, and is useful to verify everything is working ok.*

- **An Analyser**: Who controls your Publishers? Your Analyser.
  - Your Analyser will start by subscribing to the relevant **counter** topic(s), then publishing to the **request/qos**, **request/delay, request/messagesize** and **request/instancecount** (and **request/go**?) topics, asking for some number of Publishers to deliver accordingly.
  - It will then listen to the specified **counter** topic(s) on the broker and take measurements as below to report statistics on the performance of the publisher/broker/network/analyser combination.
  - The measurements should be taken across the range of delay (0,100ms), publisher-to-broker QoS (0,1,2), message-size (0,1000,4000) and some instance-count values (1,5 and 10), so that you can compare them; things can get weird under load.
  - You will need to run the full suite of tests with each of the three QoS values for the broker-to-analyser subscription as well, as things can also get weirder when the Publisher and Subscriber have very different QoS. *You may need to disconnect and reconnect when changing the subscription QoS; this can be broker-specific behaviour.*
  - Yes, that's 3*2*3*3*3=162 separate tests, each taking 30sec. Statistics for performance analysis requires a reasonable amount of data, and in a real-world test you should collect a lot more! Fortunately your code can do it all for you.

Data capture and Analysis

Once your code is working, you need to tackle the following:

Start the Publishers, then run your Analyser. Have the Analyser tell the broker what you want the Publisher(s) to send, and record data as below.

> *Tips: (i) only ever print to screen for debugging, not while actually measuring, otherwise it will slow your code down **a lot** and mess up your data. (ii) Use the counter values to tell you what messages are arriving, or are not arriving, to calculate the rates below, (iii) use the timestamps to calculate changes to the transmission delay.*

Collect statistics, for each instance-count/delay/QoS/message-size combination, to measure over the test period:

1. The total mean rate of messages you actually receive in total from all publishers across the period *[messages/second]*.

2. The average (per-publisher) rate of
    a. Any message loss *[percentage] (how many messages did you see, versus how many should you have seen)*
    b. Any out-of-order messages *[percentage] (i.e. how often do you get a smaller number after a larger number)*
    c. Any duplicate messages *[percentage] (i.e. how often do you see the same message published)*

3. The average (per-publisher) mean-inter-message-gap (timestamp difference) and standard-deviation between consecutive messages *[both in milliseconds]*. *Only measure for actually consecutive counter-value messages, ignore any values where you miss any messages in between.*

The assumption is that all publishers will be basically identical, but some measurements only make sense per-publisher, and you can then average those measurements. You can do the calculations within your code, during/after each test burst, which is the most efficient, or save to memory/disk and then calculate with another tool.

4. While measuring the above also subscribe to and record the $SYS/# measurements, and identify what, if anything, on the broker do any loss/misordered/duplicates correlate with. You can do this with a separate client or your own code. *(Look at measurements under e.g. 'load', 'heap', 'active clients', 'messages'; anything that seems relevant. See e.g. https://mosquitto.org/man/mosquitto-8.html for ideas. Be aware of the timing and frequency of the $SYS measurements, to reflect when you actually put the broker under load. If running your own broker you may be able to configure the frequency of $SYS reports.)*

Reporting

In your report: *[around 4-5 pages of text, plus any diagrams and charts]*

1. Subscribe to some broker 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 of a pub/sub (filter for mqtt), and briefly explain how each QoS-level 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. *[around 0.5 page of text]*

2. Summarise your measurements from above, in suitable table form, and with simple charts, to compare the impact of different message sizes, delays, instance-counts and QoS combinations. Explain what you expected to see, especially in relation to the different QoS levels, and whether your expectations were matched. Also describe what correlations of measured rates with $SYS topics you expected to see and why, and whether you do, or do not.

3. Consider the broader end-to-end (internet-wide) network environment, in a situation with millions of sensors publishing frequently to thousands of subscribers. Explain in your report *[around 1 page]*
    a. What performance challenges might be when using MQTT for extremely high volumes of messages, from the sources publishing their messages, all the way through the network and broker to your subscribing client application. If you lose messages, where might they be lost, and why? Think about links, routers, memory/buffers, cpus, etc.
    b. How the different QoS levels may help, or not, in dealing with the challenges.
    c. Why 'retaining' messages would be a bad idea in this context.

Bonus questions:

Some extension questions to tackle, if you want and have time, for up to 10% bonus marks.

1. Show your results with 20, or more(?!), publishers with qos=0, delay=0ms. This will be subject to your computer's performance.
2. Run a test to see if there is a performance difference between your analyser subscribing to an explicit list of topics (counter/A/B/C) and subscribing to a top-level wildcard topic (counter/#). Why would it make a difference?
3. Try to run your code using a public broker, across the internet. What happens to the performance and stability of the results? Don't repeat all the tests, just try the 0ms-delay ones, as much as they let you – some are throttled. And perhaps be prepared to be kicked off…

Assessment

We'll be marking on (with weighting) out of 30 marks:

- Your code clarity, and code-documentation/comments (20%).
    - *Could somebody else pick this up, debug it or add features? Do you explain your approach and choices you made?*
- Your code subscribing, properly publishing/listening to the broker, and collecting data (20%).
    - *Do your Analyser and Publisher work efficiently and as specified? Can your publishers publish messages at a high rate when the delay=0ms?*
- Your analysis report addressing the questions above (60%)
    - *Have you done the wireshark? Did you neatly summarise the statistics you collected, to highlight anything interesting? Have you considered the whole workflow of data/messages from the publisher process to the analyser process, and all the various protocol overheads involved? Be aware, all students will have different set-ups, and may see very different results.*