COMP3310/ENGN3539/etc Assignment 3 – Testing MQTT

Introduction:

- This assignment is worth 12.5% of the final mark
- It is due by **Tuesday 21 May 23.55 AEST (Canberra time)**
- 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 rather than direct emails.*

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 a conveyor belt, 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.

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** and **request/instancecount**. When it sees new values for these, it will start publishing accordingly.
    - You will have 5 instances of a Publisher running at the same time, called *pub-1* to *pub-5*. These will help stress the broker (and network, if you have separate computers). The 'instancecount' will tell you how many publishers should be active, while the rest should keep quiet.
    - Each Publisher will send a sequence of simple message to the broker, namely an incrementing counter (0, 1, 2, 3, …). It will publish those messages to the broker at a requested MQTT QoS level (0, 1 or 2), and with a requested delay between messages (0ms, 1ms, 2ms, 4ms) for 60 seconds.
    - Each Publisher will publish to the topic **counter/<instance>/<qos>/<delay>**, so e.g. **counter/1/0/4** is the messages coming from *Publisher-instance-1* at *qos=0* and *delay=4*.
    - After it has finished its 60sec burst of messages, each Publisher should go back to listening to the 'request' topics for the next round of instructions.
    - *At 0ms delay and qos=0 your publisher should be able to publish very quickly, potentially many thousands of messages per second.*

- **An Analyser**: Who controls your Publishers? Your Analyser.
    - Your Analyser will start by publishing to the **request/qos**, **request/delay** and **request/instancecount** 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 (below) to report statistics on the performance of the publisher/broker/network combination.
    - The measurements should be taken across the range of delay (4), QoS (3), and instance-count (5) values as above, so that you can compare them; things can get weird under load.
    - Run it with all three QoS values for the Broker->Analyser subscription as well; things can get weird when the Publisher and Subscriber have very different QoS. *You may need to disconnect and reconnect when changing the subscription QoS.*
    - Yes, thats 3*3*4*5=180 tests, each taking 1min. Fortunately your code could do it all for you.

Analysis and Reporting

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

1. **Start the Publishers, then run your Analyser.** Have the Analyser tell the broker what you want the Publisher(s) to send, and record data for 60sec as below.
   *Tips: (i) only print to screen for debugging, not while 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.*

    a. Collect statistics, for each instance-count/delay/QoS combination, to measure over the 60sec period:
        i. The total average rate of messages you actually receive from all publishers across the period *[messages/second]*.
        ii. The rate of message loss you see *[percentage]*.
            (how many messages did you see, versus how many should you have seen)
        iii. The rate of any out-of-order messages you see *[percentage]*
            (i.e. how often do you get a smaller number after a larger number)
        iv. The median inter-message-gap you see, compared to the requested delay *[milliseconds]*.
            Only measure for actually consecutive counter-value messages, ignore the gap if you miss any messages in between.

    b. While measuring the above also
        i. Subscribe to and record the $SYS/# measurements, and identify what, if anything, on the broker do any loss/misordered-rates correlate with. *(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 of the $SYS measurements, to reflect when you actually put the broker under load)*

2. In your report: *[around 4-5 pages of text for all of Q2 and Q3, plus any charts]*
    a. 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]*
    b. Summarise your measurements, in suitable table form, and simple charts, to compare the impact of different delays and QoS combinations. Explain what you expected to see, especially in relation to the different QoS levels, and whether your expectations were matched.
    c. Describe what correlations of measured rates with $SYS topics you expect to see and why, and whether you do, or do not.

3. Consider the broader end-to-end (internet-wide maybe) network environment, in a situation with millions of sensors publishing to thousands of subscribers. Explain in your report *[around 1 page]*
    a. What performance challenges there might be when using MQTT in general for 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.

Bonus questions:

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

1. What happens if the Analyser modifies the request topics at the same time the Publisher sees the changed values – how could you ensure it doesn't start prematurely? Does your code do this?
2. How does it run with 10, or more, publishers with qos=0, delay=0ms? This will be subject to your computer's performance.
3. Try to run this on a public broker, across the internet. What happens to the performance and stability of the results? Don't repeat all 180 tests, just try the 0ms-delay ones, as much as they let you.

Assessment

- Your code clarity, and documentation/comments (15%).
    - *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 (15%).
    - *Do your Analyser and Publisher work efficiently and as specified? Can your publishers publish messages at a high rate when the delay=0ms? Can the Analyser run a number of tests with limited human input?*
- Your analysis report addressing the questions above (70%)
    - *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 setups, and may see very different results.*