**MQTT and CoAP exercises**

March 7 & 14, 2024

**Part 1: MQTT**

Open a terminal window and change to the directory with the files for this exercise with the command:  
**cd ~/MQTT-CoAP-Exercise**

**Task 1.0 (Verify connectivity – Does not have to be documented in the exercise report):**

In the terminal: Execute the command **sudo ./netgenerate.sh** that creates a small *virtual* network using Linux network namespaces with three hosts (H1, H2, and H3) connected to a switch, as illustrated in Figure 1, which also shows the IPv4 addresses for the hosts. The command also opens an Xterm window on each host so you can execute commands on a specific host. When the virtual network is not needed anymore, it can be deleted by executing **sudo clearnet.sh**



Figure 1 – Switched network with three hosts.

Choose an arbitrary host and (in that host’s Xterm window) check if you can ping the other two hosts with the command **ping -c1 <ip-address>**

Note that the virtual network “disappears” if the virtual machine is restarted, so you will need to execute the **sudo ./netgenerate.sh** command again after a restart of the VM and.

**Task 1.1 (Basic MQTT connections):**

In host H1’s Xterm window, type **ip addr** and verify that one of its interfaces has an IP address in the **10.0.0.0/24** range. Type the following command in H1’s Xterm window to start the MQTT-broker on this host: **sudo mosquitto -c ./mosquitto.conf**

Start Wireshark in the VM’s normal[[1]](#footnote-1) terminal window ( **sudo wireshark** ) and start a capture on the **veth11** interface[[2]](#footnote-2). It might be a good idea to set the display filter (just below the toolbar at the top of the window) to **mqtt** as shown in Figure 2:

**A screenshot of a computer

Description automatically generated**

Figure 2 – Wireshark with **mqtt** as display filter.

Task 1.1.a: In H2’s Xterm window, start an MQTT-subscriber:

* **mosquitto\_sub -v -h <H1’s IP-address> -t '#'**

Stop the client again after a few seconds with Ctrl-C in the Xterm window, switch to the Wireshark window and stop the capture. You should have captured five MQTT packets; list these packets and explain their purpose.

Task 1.1.b: Start a new capture with Wireshark and restart the subscriber client again (as above), but keep it running for 2-3 minutes – what do you observe in Wireshark? Can you relate the additional messages compared to the first capture and their intervals to one of the first packets?

Task 1.1.c: Start a new Wireshark capture. In H3’s Xterm window, publish a message with:

* **mosquitto\_pub -h <H1’s IP-address> -t Topic1 -m "Hello World!"**

Verify that your message appears in H2’s Xterm window. Switch to Wireshark and stop the capture. List and explain the MQTT packets related to the publishing of the message. Do the MQTT “Publish Message” contents correspond to the message string of the **mosquitto\_pub** command? (Hint: Check the “raw” bytes of the message in Wireshark – if the ‘Packet Bytes’ section of Wireshark’s window is not shown, make sure that ‘Packet Bytes’ is selected in the View menu.)

**Task 1.2 (Message QoS):**

MQTT supports three different “QoS”-levels related to message delivery, but only the default QoS level (level 0) was used in Task 1. A non-default QoS-level can be specified for both the MQTT-subscriber and the MQTT-publisher by appending the option **-q <qos-level>** (where **<qos-level>** is **1** or **2** ) to the command. For each non-default QoS level, capture MQTT traffic with Wireshark as in Task 1 and explain the difference between the different QoS levels 0, 1, and 2 (It may be helpful to consult section 4.3 in the official MQTT-specification – see https://mqtt.org).

Also, capture and explain the MQTT traffic if the publisher uses QoS-level 0 but the subscriber has requested QoS-level 2.

**Task 1.3 (Wills):**

MQTT permits a publisher to set up a “will”, i.e., a message to be sent in case the publisher *unexpectedly* disconnects from the broker. In Tasks 1 and 2, the publisher disconnected normally immediately after a single message, but the publisher can also be instructed to repeat a message a specified number of times with a specific interval between the messages. Execute the following command in H3’s Xterm window and verify/document with Wireshark that a) the message is sent 5 times with 2 seconds intervals between each message, and b) that the publisher disconnects after the 5th message has been sent:

**mosquitto\_pub -h 10.0.0.1 -t Topic1 -m "Hello World!" --repeat 5 --repeat-delay 2**

Consult the documentation for the **mosquitto\_pub** command (found on <https://mosquitto.org>) and determine the extra arguments that must be added to the command so that the publisher creates a “will” with the text “Help me!” to be published to the topic “Wills” if it disconnects unexpectedly.

Start a Wireshark capture. Rerun the publisher command with these extra arguments and also the value 60 for the **--repeat-delay** option, so the command runs for at least 4 minutes. Before the publisher exits, execute (in a new VM-terminal window) the command (this will forcefully kill the publisher):  
**sudo pkill mosquitto\_pub**

Document in your report with the Wireshark capture and a screenshot of the subscriber’s Xterm window that the “will” is sent to the subscriber.

**Task 1.4 (Retained messages):**

Normally, an MQTT client that starts subscribing to a topic will not receive anything before a message is published to that topic. However, MQTT also supports “retained” messages. If a publisher indicates that messages published on a topic should be retained, the broker keeps a copy of the last published message on that topic that was specified to be retainer, so if a client starts subscribing to this topic, it will receive the retained message immediately.

Investigate two scenarios[[3]](#footnote-3), one with and one without retained messages, illustrating the difference between the scenarios, and document these with Wireshark captures. You may also need to consult the documentation for the **mosquitto\_pub** command for this task.

Note that if you want to “unretain” a message on the broker, you must publish an empty message (also called a null message) to be retained, i.e., use **-m ''** instead of **-m 'Hello world!'** as an argument for **mosquitto\_pub.**

**Part 2: CoAP**

**Task 2.0 (Verify connectivity – Does not have to be documented in the exercise report):**

Open a terminal window and execute the following commands (note the extra option to the netgenerate.sh script, which specifies that the simulated nodes use IPv6 addresses instead of the default IPv4 addresses):

* **sudo ./netgenerate.sh --ipv6**

The command creates a small *virtual* network using Linux network namespaces with three hosts (H1, H2, and H3) connected to a switch, as illustrated in Figure 2. The IPv6 addresses of the hosts are **fd00::1** for H1, **fd00::2** for H2, and **fd00::3** for H3. The command also opens an Xterm window on each host so you can execute commands on a specific host. If the virtual network is not needed anymore, it can be deleted by executing **sudo clearnet.sh**



Figure 2 – Switched network with three hosts.

Select an arbitrary host and (in that host’s Xterm window) check if you can ping the other two hosts with the command **ping -c1 fd00::x** (where x is 1, 2, or 3, depending on the host).

Note that the virtual network “disappears” if the virtual machine is restarted, so you will need to execute the **sudo ./netgenerate.sh --ipv6** command again after a restart of the VM.

**Task 2.1 (Basic CoAP):**

Start a CoAP server on host H1 with the command (in H1’s Xterm window):

**coap-server -A fd00::1**

Start Wireshark in the normal terminal window with **sudo wireshark** and start a capture on the **veth11** interface with a display filter set to **coap**.

Task 2.1.a: Use the CoAP client program in H2’s Xterm window to retrieve the default resource (‘**/**’) from the server with the command[[4]](#footnote-4) **coap-client coap://[fd00::1]/**

Stop the capture in Wireshark and examine the contents of the packets captured[[5]](#footnote-5):

* What version of CoAP is used?
* Does the message exchange use confirmable or non-confirmable messages?
* What is the purpose[[6]](#footnote-6) of the two options in the CoAP message sent from **fd00::1** to **fd00::02**?

Task 2.1.b: Start a new Wireshark capture and use the coap-client to retrieve the ‘**/async**’ resource with the command **coap-client coap://[fd00::1]/async**   
  
Stop the Wireshark capture. Explain the differences in the CoAP messages between the retrieval of the default resource (‘**/**’) in task 1a and the ‘**/async**’ resource in this task.

Task 2.1.c: Start a new capture in Wireshark and use the CoAP client program again to retrieve both the default resource (as in task 1a) and the ‘**/async**’ resource (as in task 1b), but now with the option ‘**-N**’ between the command and the URI.

Explain the difference in terms of the CoAP messages in tasks 2.1.a and 2.1.c and this task.

**Task 2.2 (Dynamic resources):**

Stop the server (press **Ctrl-C** when H1’s Xterm window has focus). Restart the server with the extra option

**‑d 999** that permits a client to define and set a value for up to 999 dynamic resource.

Task 2.2.a: Start a new capture in Wireshark. Use the coap-client in H2’s Xterm window to define a resource with the name ‘**/C34351**’ (i.e., the URI will be **coap://[fd00::1]/C34351** ) and the value “**DTU’s best course**”. You will need to use additional options for the CoAP client compared to task 2.1 to define a) the action/method that creates/instantiates a dynamic resource on the server, b) the value to be stored in the resource, and c) the type of information for the value. Please consult the documentation[[7]](#footnote-7) for the coap-client to determine which options are required and how they are used. Document that the dynamic resource is created with the capture from Wireshark.

Task 2.2.b: Use the client to retrieve the resource with URI **coap://[fd00::1]/C34351** again. Does it contain the expected value?

Task 2.2.c: Now use the client to delete the resource and verify afterwards that it has been deleted.

**Task 2.3 (Observing a resource):**

Start a new wireshark capture. Use the client to *subscribe* (Hint: consult the client documentation again) to the URI **coap://[fd00::1]/time** for 15 seconds. Stop the Wireshark capture and explain[[8]](#footnote-8) the CoAP packets captured.

**Task 2.4: (Publish/Subscribe with CoAP):**

Design a setup with one server and two clients that resembles an MQTT network, i.e., with one client acting as a publisher that (in this exercise manually) “publishes” new content to a dynamic resource of your choosing on the server and the other client acting as a subscriber that “subscribes” to this dynamic resource for a suitable amount of time.

List the commands you use to make this setup and document with a Wireshark capture and screenshots that the setup works as expected. Note: You should probably run the server command on H1 (and capture the coap traffic on interface **veth11**) and run the clients on H2 and H3.

**Task 2.5:**

Stop the CoAP server in H1. Start *two* new CoAP-servers on hosts H1 and H2, respectively, with the command: **coap‑server ‑g ff05::fd**

Start a Wireshark capture on interface **veth31**. Execute the following command in H3’s Xterm window: **coap-client -N coap://[ff05::fd]/**

Stop the Wireshark capture and explain the CoAP packets captured, including:

* What is the type of MAC address used as the destination MAC address in the CoAP packet from the client?
* Why is the ‘-N’ option needed in this scenario?

1. I.e., not in any of the Xterm windows of the hosts. [↑](#footnote-ref-1)
2. The **veth11** interface is the switch’s interface towards host H1 [↑](#footnote-ref-2)
3. Neither scenario should use wills, i.e., the options for wills identified in task 3 should not be used in task 4. [↑](#footnote-ref-3)
4. Note that IPv6 addresses must be enclosed in square brackets, ‘[‘ and ‘]’. [↑](#footnote-ref-4)
5. You can ignore that Wireshark describes an option as “Unknown Option” – this is because the installed version of Wireshark does not “recognize” all possible CoAP message options. [↑](#footnote-ref-5)
6. See the RFC-document for CoAP (RFC-7252): <https://www.rfc-editor.org/rfc/rfc7252> [↑](#footnote-ref-6)
7. See <https://libcoap.net/doc/reference/4.3.4/man_coap-client.html> (or the PDF-files on DTU Learn) [↑](#footnote-ref-7)
8. RFC-7641 (<https://www.rfc-editor.org/rfc/rfc7641>) might be relevant to consult. [↑](#footnote-ref-8)