TELE 6510: Project, 100 points total)

Questions:

1. Capture (and paste) wireshark traces between device and application.

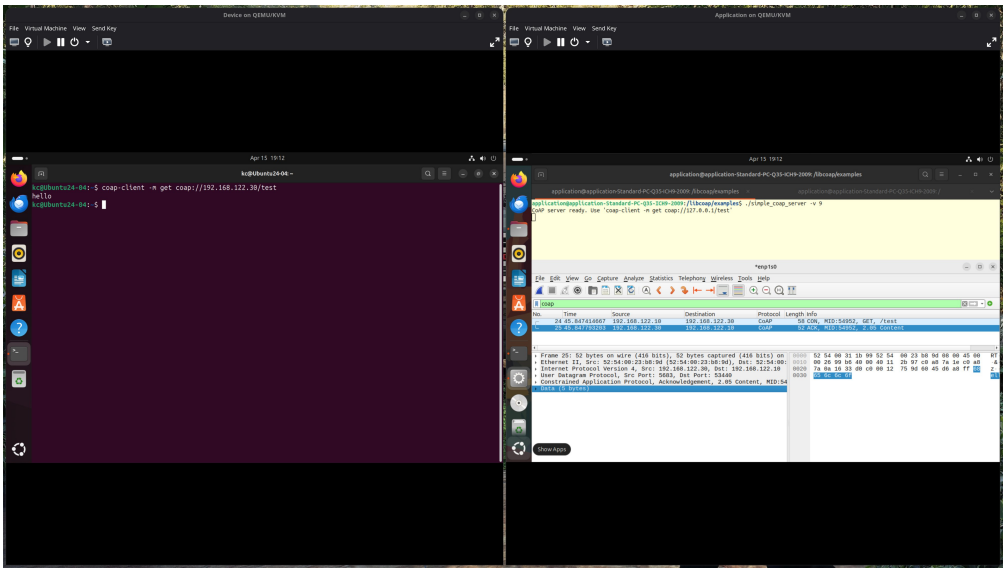


Figure 1

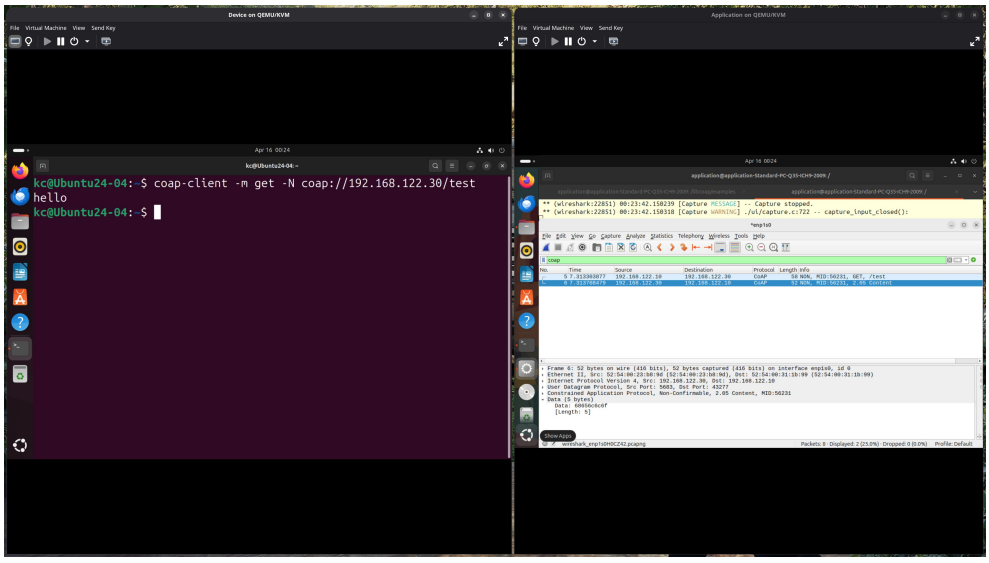


Figure 2

Figure 1, shows communication between the device and application using the CoAP Protocol. The client sends a confirmable request to the server. Figure 2 shows a non-confirmable request made to the server. I created a custom server with the test resource which hold a 5 byte payload.

2. What is the request/response delay at the application when sending a simple 10-byte (or less) readout ? (use wireshark timestamps, run 30 transaction and use the average) for uniform 0%, 3%, 5% and 10% packet loss introduced at the access side (up to you where). Make a table packet loss vs latency for confirmable and non-confirmable CoAP.

Solution:

- Ping test to 192.168.122.30(coap-server): 0% packet loss (6 packets sent, 6 received)

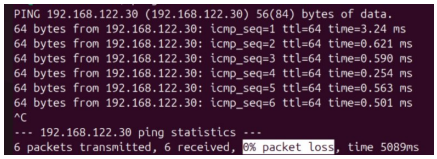


Figure 3

- Ping test to 192.168.122.30(coap-server): 5% packet loss:(38 packets sent, 36 received)

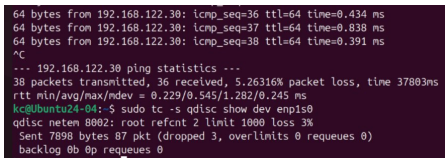


Figure 4

- Ping test to 192.168.122.30(coap-server): 10% packet loss:(10 packets sent, 9 received)

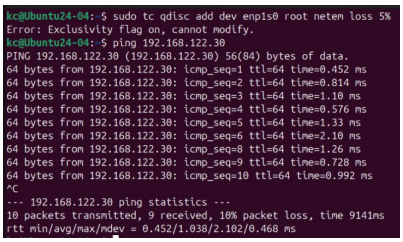


Figure 5

- Ping test to 192.168.122.30(coap-server):17% packet loss:(12 packets sent,10 received)

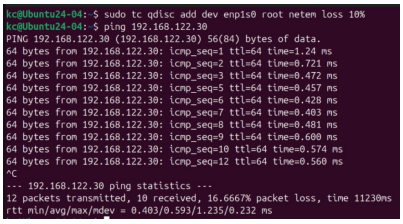
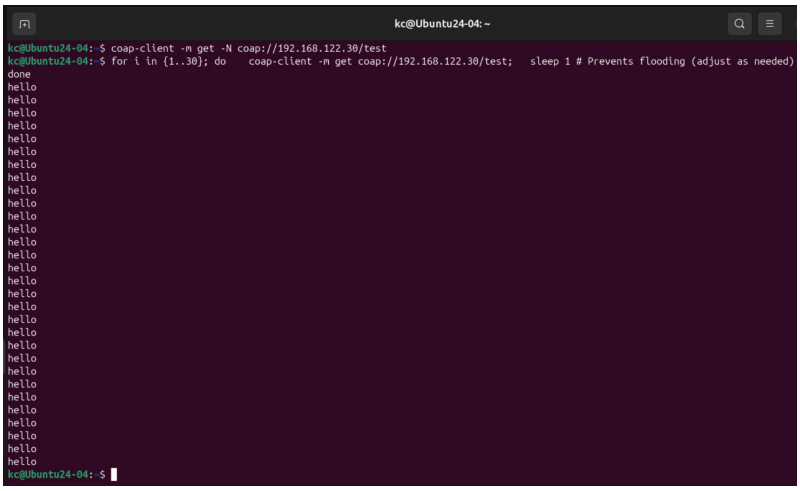
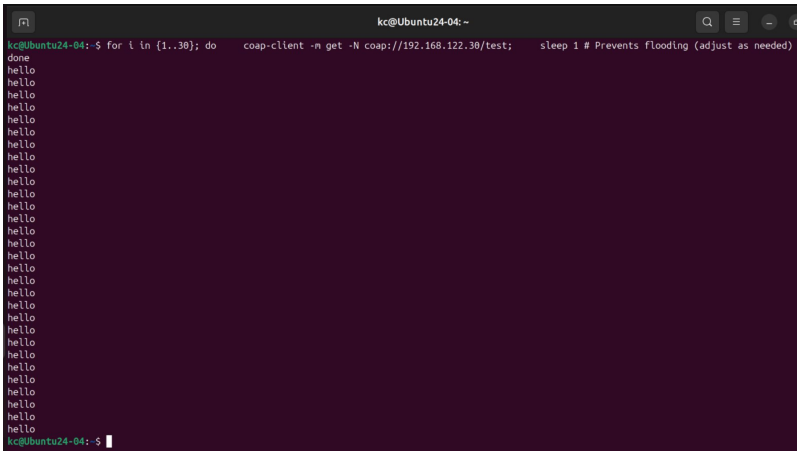


Figure 6

Average of 30 Confirmable iterations:



Average of 30 Non-Confirmable iterations:



Packet Loss	Confirmable Latency (ms)	Non-Confirmable Latency(ms)
0%	0.22	0.14
3%	0.29	0.13
5%	0.32	0.14
10%	0.36	0.15

From the table, the Confirmable latency is very negligible for 0%, 3%, 5% and 10% packet loss though it seams to be increasing, while the Non-Confirmable latency remains consistent and lower than confirmable latency, despite introducing different packet loss percentages.

I.2. Describe the messages that are exchanged between the device and the application.

Solution:

In figure 1, the client sends a Confirmable (CON) GET request to the server's /test resource (coap://192.168.122.30/test), confirmed by the command `coap-client -m get

coap://192.168.122.30/test` (MID: 54952). The server, running in verbose mode (`./simple_coap_server -v 9`), responds with an ACK containing a 2.05 Content response and payload "hello". This transaction completes in approximately 0.38ms, showcasing CoAP's efficiency over UDP. The consistent Message ID (54952) ensures transaction integrity, while the absence of a Token field indicates a basic implementation. CoAP's CON/ACK mechanism provides reliability without TCP overhead, confirmed by the 2.05 code for successful resource retrieval. Wireshark trace verifies proper protocol operation with server listening on port 5683 and processing requests correctly.

In figure 2, the client initiates the exchange by sending a Non-Confirmable (NON) GET request to the server's /test resource, as evidenced by the command coap-client -m get -N coap://192.168.122.30/test. This NON-Confirmable message (MID: 56231) does not require an acknowledgment because in CoAP Protocol, it is designed for best-effort delivery without the overhead and delay of acknowledgments or re-transmissions. The server, launched in verbose mode (./simple_coap_server -v 9), responds with a 2.05 Content response code and the payload "hello".

I.3. How does quality affect latency? Why?

Solution:
Examining the confirmable latency data shows a clear pattern: as packet loss increases (network quality decreases), the request/response time increases proportionally:
0% packet loss: 0.22ms latency
3% packet loss: 0.29ms latency (32% increase)
5% packet loss: 0.32ms latency (45% increase from baseline)
10% packet loss: 0.36ms latency (64% increase from baseline)

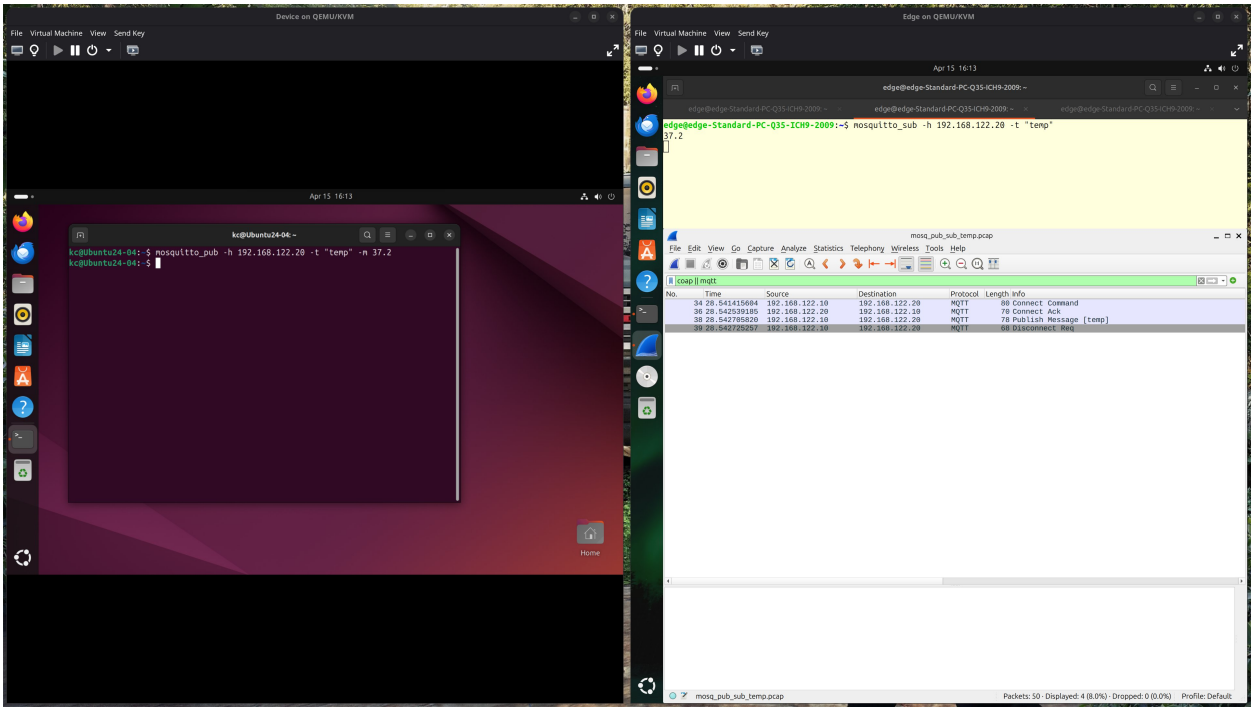
This consistent increase occurs because of specific protocol mechanisms that handle unreliable networks which include: **Packet re-transmission:** When a packet is lost, the protocol must detect this loss and resend the data. Each re-transmission adds delay proportional to the packet loss rate. **Timeout calculations:** Higher packet loss forces longer wait times before determining if a packet was truly lost rather than just delayed. **Acknowledgment processing:** With increased packet loss, more resources are devoted to processing and managing acknowledgments, adding overhead to each transaction. The data demonstrates how these mechanisms create a direct relationship between network quality and latency - each 1% increase in packet loss contributes approximately 0.014ms additional latency due to these recovery processes.

Non-confirmable (NON) latency remains relatively consistent despite increasing packet loss because NON messages lack a re-transmission mechanism. When a NON packet is lost, the sender doesn't resend it. Latency measurements typically track the transit time of successfully delivered packets. As long as underlying network conditions like congestion are stable, the delivery time for the packets that do arrive remains similar, unaffected by the loss of other packets. In contrast, confirmable (CON) messages re-transmit upon loss, leading to increased latency with higher packet loss.

II. Goal: CoAP + MQTT

Questions:

1. Capture (and paste) wireshark traces between device and application.



2. What is the delay between the device and the application when sending a simple 10-byte (or less) readout ? (use wireshark timestamps, run 30 transaction and use the average) for uniform 0%, 3%, 5% and 10% packet loss introduced at the access side (up to you where). Make a table packet loss vs latency for confirmable CoAP and each quality level at most once, at least once and exactly once of MQTT.

Solution:
Packet loss for CoAP Confirmable similar to *figure 3 – figure 6* above.

Packet loss(%)	CoAP Confirmable (avg ms)	MQTT QoS 0 – At Most Once (avg ms)	MQTT QoS 1 – At Least Once (avg ms)	MQTT QoS 2 - Exactly Once (avg ms)
0	0.24	0.26	0.31	0.47
3	0.27	0.27	0.42	0.83
5	0.31	0.27	0.54	1.02
10	0.34	0.27	0.97	1.88

Findings:

- **CoAP Confirmable:** Single ACK model, similar to MQTT QoS 1. Re-transmits if ACK isn't received. Lightweight, designed for lossy networks. Latency increases linearly, but modestly with packet loss.
- **MQTT QoS 0 (At Most Once):** No retries, no ACKs.Latency stays low and nearly constant. Packet loss might just drop the message entirely.
- **MQTT QoS 1 (At Least Once):** Requires PUBACK, so retry happens if lost. Packet loss triggers re-transmission delay.latency increased moderately with loss.
- **MQTT QoS 2 (Exactly Once):** Full handshake: PUBLISH → PUBREC → PUBREL → PUBCOMP. Packet loss causes multiple retries at different stages. Significant increase in latency under loss observed.

Insights:

1. CoAP Confirmable and MQTT QoS 1 offer similar latency, both faster than MQTT QoS 2.
2. MQTT QoS 2 is the most reliable, followed closely by QoS 1 and CoAP Confirmable, while QoS 0 is the least reliable.
3. CoAP is more lightweight and efficient, especially in low-power or lossy networks.

II.2. Describe the messages that are exchanged between the device and the application.

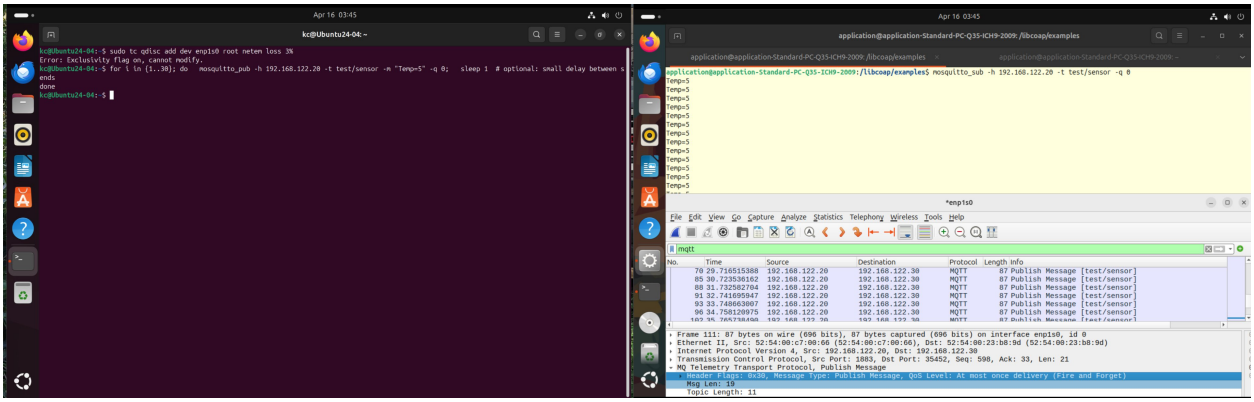


Figure 7: MQTT QoS 0 - At Most Once

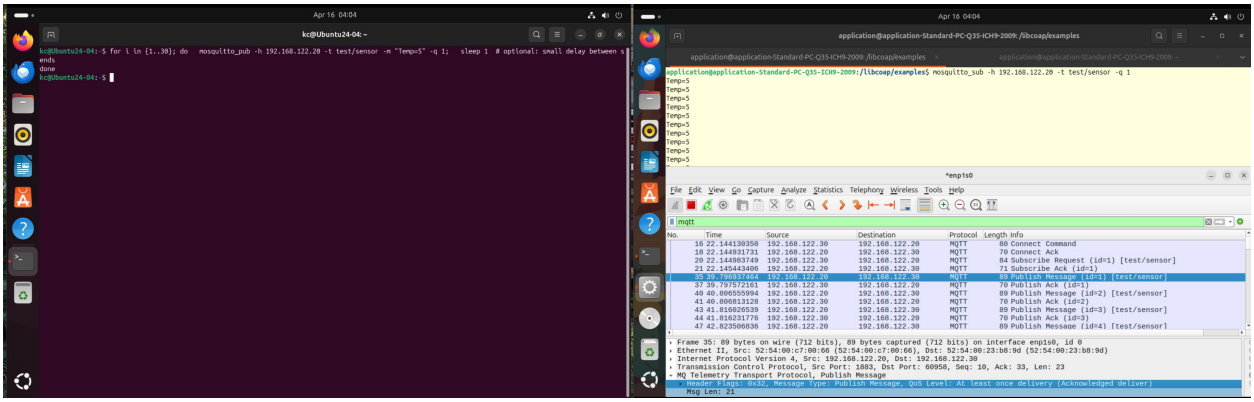


Figure 8: MQTT QoS 1 - At Least Once

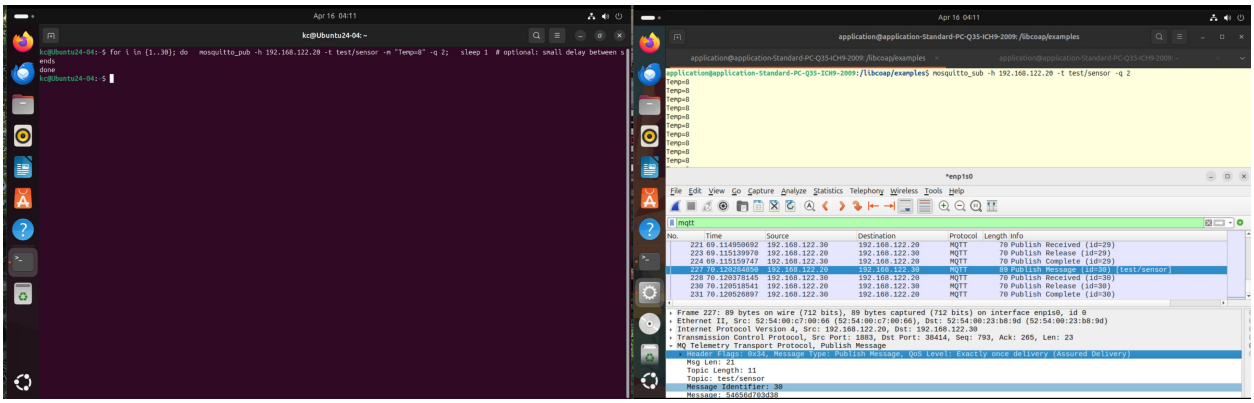


Figure 9: MQTT QoS 2 - Exactly Once

In **CoAP Confirmable**, the device starts by sending a confirmable GET request to the application, usually to request some data. The application responds with an acknowledgment (ACK) that includes the requested information. If the device doesn't receive the ACK, it will automatically retry after a short delay. This makes CoAP confirmable reliable, even on lossy networks, while keeping things lightweight.

With **MQTT QoS 0**, the device simply sends a publish message without expecting any response. This is the fastest option but also the least reliable, since the message could be lost and there's no retry.

MQTT QoS 1 use a 2-step handshake - (PUBLISH MESSAGE), and adds reliability by requiring an acknowledgment (PUBLISH ACK) from the receiver. If the device doesn't get the PUBLISH ACK, it re-sends the message, which means the receiver might get duplicates, but the message won't be lost.

Finally, **MQTT QoS 2** is the most reliable, using a four-step handshake (PUBLISH MESSAGE, PUBLISH RECEIVED, PUBLISH RELEASE, PUBLISH COMPLETE) to make sure the message is delivered exactly once. This avoids duplicates, but it takes longer due to the extra messages.

II.3. How does core based MQTT compare against HTTP for same network conditions ?

MQTT and HTTP serve different objectives, although under the same network conditions, core-based MQTT typically outperforms HTTP, particularly in low-bandwidth or unstable situations. MQTT is a lightweight publish-subscribe protocol that enables devices to communicate quickly and efficiently. It employs a persistent TCP connection and has minimal message sizes, making it perfect for real-time IoT applications. HTTP, on the other hand, has a request-response approach, which involves opening and closing a new connection for each transaction, resulting in overhead and latency.

MQTT outperforms HTTP in terms of latency and bandwidth because it eliminates the recurring handshake and header overhead that HTTP requires. MQTT also supports several Quality of Service (QoS) levels, enabling for consistent delivery even on packet-loss networks. HTTP lacks built-in reliability mechanisms and often requires additional application-level management.

Overall, MQTT is more efficient, responsive, and dependable for scenarios requiring frequent, little data exchanges, such as sensor updates or device connectivity, whereas HTTP is better suited to less frequent, bigger data transfers, such as web surfing or file downloads.