



Universidad de
Oviedo



ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

MÁSTER EN INGENIERÍA DE TELECOMUNICACIÓN

ÁREA DE TEORÍA DE LA SEÑAL Y COMUNICACIONES

**KILL-THE-PLC: IMPLEMENTATION AND PERFORMANCE
EVALUATION OF A DISRUPTIVE ROBOTIC CELL
ENVIRONMENT BASED ON 5G AND EDGE CLOUD
TECHNOLOGIES**

DAVID ARIAS-CACHERO RINCÓN

**TUTOR: IGNACIO RODRÍGUEZ LARRAD
SUPERVISOR EXTERNO: PREBEN MOGENSEN**

FECHA: JULIO 2023

This MSc thesis was written in collaboration with *Aalborg University*.



AALBORG UNIVERSITY

Abstract

Considering the fast technological advancements and population growth, the significance of smart production processes, Industrial Internet of Things (IIoT) and 5G communication systems in driving manufacturing enhancements cannot be overstated. By applying the paradigm of Industry 4.0, digital technologies are integrated within automation in manufacturing leading to more intelligent and interconnected systems. As a consequence, increased productivity, efficiency, flexibility, scalability, real-time monitoring and optimized production processes will be possible, increasing competitiveness in the global market. This work aims investigating, designing, and deploying an IIoT solution based on edge-cloud and 5G technologies that enables the migration of the control logic wired robotic cell towards a cloudified and wireless solution. This will be accomplished through a series of sequential objectives based on the application of the main IIoT protocols (MQTT and OPC-UA) in both Ethernet and 5G scenarios, starting from a basic solution and scaling up to a final concept where an operational robotic arm is controlled from a remote PLC executed in the edge-cloud and exchanging data over 5G.

The study first considers the analysis of the main MQTT and OPC-UA communication protocols and their associated architectures, including a reference testbed implementation and preliminary performance evaluation of the protocols in wired Ethernet settings. The analysis is later extended to 5G and edge-cloud by elaborating on the different Ethernet-5G operational integration aspects, and the performance of the protocols is re-evaluated in this scenario. From these studies, it was found that, in general, MQTT with QoS 0 performs better than OPC-UA in terms of closed control loop latency in both the Ethernet and 5G scenarios for both small and large packet sizes, independently of the underlying network load. From an implementation point of view, MQTT manages data payloads in a more efficient way than OPC-UA, which, together with the fact that the MQTT broker performance has a negligible impact on link performance, translates into potential superior scalability capabilities than OPC-UA PubSub, which is heavily-dependent on the configured scheduling period.

The final step of the project considers a real industrial solution based on an operational robotic cell composed by a robotic arm controlled from a PLC. Based on the previous learnings, the application is architected to be operated from a cloudified PLC located in an enterprise edge-cloud, exchanging control commands via MQTT and OPC-UA over private 5G. The design was implemented and validated in operational settings by analyzing the execution cycles times. It was observed that, 5G edge-cloud operation of the industrial use case was successful, providing the required flexibility and re-configurability, at expenses of an increased cycle elapsed time of 0.3-0.5 s.

The project has generated a great impact towards industrial partners. Future lines of work may include the integration of more robotic arms into the cell to further explore scalability and optimization of the IIoT protocols, as well as the re-evaluation of the performance over future 5G Releases with URLLC capabilities.

Resumen en español

Considerando los rápidos avances tecnológicos y el crecimiento de la población, no se puede subestimar la importancia de los procesos de producción inteligentes, el Internet Industrial de las Cosas (IIoT) y los sistemas de comunicación 5G en el impulso de las mejoras en la fabricación. Al aplicar el paradigma de la Industria 4.0, las tecnologías digitales se integran en la automatización de procesos, lo que conduce a sistemas más inteligentes e interconectados. Como consecuencia, es posible aumentar la productividad, eficiencia, flexibilidad, escalabilidad, monitoreo en tiempo real y la optimización de los procesos de producción, lo que aumentará la competitividad en el mercado global. Este trabajo tiene como objetivo investigar, diseñar e implementar una solución IIoT basada en tecnologías de edge-cloud y 5G que permita la migración de la lógica de control de una celda robótica cableada hacia una solución inalámbrica en la nube. Esto se logrará a través de una serie de objetivos secuenciales basados en la aplicación de los principales protocolos IIoT (MQTT y OPC-UA) en escenarios Ethernet y 5G, partiendo de una solución básica y escalando hacia un concepto final en el cual un brazo robótico operativo es controlado desde un PLC remoto ejecutado en el edge-cloud y comunicándose a través de 5G.

El estudio considera inicialmente el análisis de los principales protocolos de comunicación MQTT y OPC-UA así como sus arquitecturas asociadas, incluyendo una implementación de referencia en un entorno de pruebas y una evaluación preliminar del rendimiento de los protocolos en configuraciones cableadas. Después el análisis se amplía al ámbito de 5G y edge-cloud al detallar los diferentes aspectos de integración entre Ethernet y 5G, y volviendo a evaluar el rendimiento de los protocolos en este escenario. A partir de estos estudios, se encontró que en general MQTT con una configuración QoS 0 tiene un mejor rendimiento que OPC-UA en términos de latencia de bucle de control cerrado tanto en escenarios de Ethernet como de 5G, para tamaños de paquete pequeños como grandes, independientemente de la carga de red subyacente. Desde el punto de vista de la implementación, MQTT gestiona las cargas útiles de datos de manera más eficiente que OPC-UA, lo que, junto con el hecho de que el rendimiento del broker MQTT tiene un impacto insignificante en el rendimiento del enlace, se traduce en una capacidad de escalabilidad potencialmente superior en comparación con OPC-UA PubSub, que depende en gran medida del periodo de suscripción.

El último paso del proyecto considera una solución industrial real basada en una celda robótica operativa con un brazo robótico controlado desde un PLC. La aplicación se diseña para ser operada desde un PLC ubicado en un edge-cloud empresarial, intercambiando comandos de control a través de MQTT y OPC-UA sobre una red 5G privada. El diseño se implementó y validó en entornos operativos mediante el análisis de los tiempos de ejecución de los ciclos operacionales. Se pudo observar que el despliegue y configuración del escenario sobre 5G y edge-cloud fue exitoso, proporcionando la flexibilidad y reconfigurabilidad esperadas, a expensas de aumentar el tiempo de ciclo operacional en 0.3-0.5 s respecto al caso de Ethernet.

El proyecto ha generado gran interés e impacto sobre las empresas colaboradoras de esta iniciativa. En cuanto a las líneas futuras de investigación, se plantea la posibilidad de incluir más brazos robóticos para explorar la escalabilidad y la optimización de los protocolos utilizados así como la reevaluación del rendimiento en futuras Release de 5G que toleren las capacidades del URLLC (comunicaciones ultra fiables de baja latencia).

Acknowledgement

This master thesis is the result of a six month adventure where, despite experiencing moments of enjoyment, learning, stumbling, and tears, I am not the main character. I would like to take this moment to express my gratitude to all the people who have accompanied me along this journey, without whom it would be nearly impossible for me to be writing these lines.

First of all, I would like to thank Nacho and Preben for their trust, support, and guidance. Without them, I would not have been able to take full advantage of this opportunity. Thank you for teaching me that things can be done in a “different way”. I cannot continue without also mentioning my unofficial supervisor Weifan. I believed in my abilities until our paths crossed. Since then, I have stopped using Google.

As a bridge between the academic and personal aspects, I would like to mention my colleague, friend, and mentor, Ali. It’s incredible how quickly you can connect with someone when you share values, ideas, and goals.

On the personal side, I can’t of course forget about my sister Isabel, my parents and friends. Thank you for the unconditional support you gave me in every step I took before this one, because without that, I wouldn’t have boarded that plane to Aalborg (something I would have always regretted).

Finally, I would like to express my gratitude to you, Claudia. I have postponed mentioning you in these lines because I still haven’t found the right words (neither in English, Spanish, nor Danish) to express my gratitude for how you have behaved with me all these months. Only you know how valuable and important this has been for me, and you have proved it by being there every time I needed you. There is nothing you don’t already know.

To Claudia, who supported and taught me more than anyone else during this adventure.

Table of contents

Abstract	2
Resumen	3
Acknowledgement	4
Table of contents	6
List of figures	9
List of tables	11
Acronyms	16
1. Introduction	17
1.1. Motivation	17
1.2. Historical approach	17
1.2.1. Industry 4.0	18
1.3. Industrial networks	20
1.3.1. Ethernet	20
1.3.2. WiFi	20
1.4. 5G for industrial environments	21
1.4.1. Benefits of using 5G in industry	23
1.5. Edge-cloud computing for industrial environments	24
1.6. Thesis application scenario	25
1.6.1. Robotic cell	25
1.6.2. Industrial communication protocols	27
1.6.3. 5G	27
1.6.4. Edge-cloud and cloudification	28
1.7. Objectives	28
1.8. Thesis outline	28
2. IIoT control protocols	30
2.1. MQTT	32
2.1.1. Packet structure	32
2.1.2. Publish-Subscribe architecture	33
2.1.3. MQTT over TCP	37
2.1.4. MQTT-SN: MQTT over UDP	38
2.2. OPC-UA	40
2.2.1. Client-Server architecture	41
2.2.2. PubSub architecture	42
2.2.3. OPC-UA over TCP	43
2.3. Performance evaluation of IIoT protocols over Ethernet	46
2.4. Test KPIs, setups and configurations	46
2.4.1. Closed Control Loop Latency	46
2.4.2. Link outage	52
2.4.3. Broker processing time in MQTT	52

2.4.4.	Effective payload	54
2.5.	Performance results	56
2.5.1.	Closed Control Loop Latency	56
2.5.2.	Link outage	58
2.5.3.	Broker processing time in MQTT	59
2.5.4.	Effective payload	60
2.6.	Discussion of results	62
3.	Ethernet to 5G system integration	66
3.1.	5G architectures	66
3.2.	5G Box deployment	67
3.2.1.	Gateworks Newport GW6404 SBC	67
3.2.2.	SIM8262E-M2	68
3.2.3.	Hardware set up	68
3.2.4.	Software set up	70
3.3.	Performance evaluation of IIoT industrial protocols over 5G	73
3.3.1.	Performance test configurations	75
3.4.	Performance results	75
3.4.1.	Closed Control Loop Latency	75
3.4.2.	Link outage	77
3.5.	Discussion of results	79
4.	Operational robotic cell environment	84
4.1.	Robotic arms for industrial manufacturing	84
4.2.	Deployment of the robotic cells and integration with 5G edge cloud	85
4.3.	Operational cycle performance study in operational settings	88
4.4.	Operational performance results	90
4.4.1.	Operational Closed Control Loop Latency	90
4.4.2.	Operational cycle elapsed time	91
4.5.	Discussion of results	93
5.	Conclusions and future work	95
5.1.	Future work	96
	Impact of the project	97
	References	100
A.	Project structure	105
B.	Deployment of MQTT & MQTT-SN brokers	107
B.1.	Installation and configuration of Mosquitto broker	107
B.2.	Installation and configuration of RSMB broker	108
C.	Study of the payload	110
C.1.	OPC-UA	110
C.2.	MQTT	116

D. Closed Control Loop Latency plots	122
D.1. CCLL for MQTT over Ethernet: QoS comparison	122
D.2. CCLL for MQTT over Ethernet: Network load study	126
D.3. CCLL for MQTT-SN over Ethernet	128
D.4. CCLL for MQTT-SN over Ethernet: Network load study	131
D.5. CCLL for OPC-UA over Ethernet	132
D.6. CCLL for OPC-UA over Ethernet: Network load study	137
D.7. CCLL for MQTT over 5G: Testbed scenario	139
D.8. CCLL for MQTT over 5G: Network load study (testbed)	142
D.9. CCLL for MQTT-SN over 5G: Testbed scenario	144
D.10. CCLL for MQTT-SN over 5G: Network load study (testbed)	147
D.11. CCLL for OPC-UA over 5G: Testbed scenario	148
D.12. CCLL for OPC-UA over 5G: Network load study (testbed)	153

List of figures

1.1.	The four industrial revolutions.	18
1.2.	ISA-95 pyramid.	19
1.3.	Handover in WiFi.	21
1.4.	5G applications [12].	22
1.5.	Wired production chain.	24
1.6.	Current architecture of a typical wired robotic cell.	26
1.7.	Proposed architecture of a robotic cell using 5G and edge-cloud technologies.	26
2.1.	Segment fields of Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).	32
2.2.	Message Queuing Telemetry Transport (MQTT) packet structure.	32
2.3.	MQTT Publish-Subscribe architecture.	34
2.4.	Open Systems Interconnection (OSI) model.	34
2.5.	Quality of Service (QoS) 0 level.	35
2.6.	QoS 1 level.	36
2.7.	QoS 1 level: duplicated message.	36
2.8.	QoS 2 level.	37
2.9.	MQTT (TCP) connection packet flow.	38
2.10.	MQTT for Sensor Networks (MQTT-SN) (UDP) connection packet flow.	39
2.11.	OPC-UA Object model.	41
2.12.	Open Platform Communications Unified Architecture (OPC-UA) use case example.	42
2.13.	OPC-UA Connection establishment, data transmission and closing procedures.	45
2.14.	Ethernet testbed used for the study of MQTT.	47
2.15.	Real scenario of the Ethernet testbed used for the study of MQTT.	47
2.16.	Ethernet testbed used for the study of OPC-UA.	47
2.17.	Real scenario of the Ethernet testbed used for the study of OPC-UA.	47
2.21.	Flowchart of Closed Control Loop Latency (CCLL) in MQTT.	50
2.22.	Flowchart of CCLL in OPC-UA Client-Server.	50
2.23.	Flowchart of CCLL in OPC-UA PubSub.	51
2.24.	Testbed set up used during broker processing time study.	53
2.25.	Real scenario of the Ethernet testbed used for the study of MQTT's broker processing time.	53
2.26.	Wireshark inside the TCP/Internet Protocol (IP) model.	55
3.1.	5G architecture used in AAU 5G Smart Production Lab.	67
3.2.	Hardware needed for setting up one 5G Box.	68
3.3.	Subscriber Identity/Identification Module (SIM) card mounted on the USB3.0 board.	69
3.4.	SimCom modem mounted on the Gateworks's USB 3.0 port.	69
3.5.	Final set up of the 5G Box	69
3.6.	5G Box integration within AAU 5G Smart Production Lab network.	70
3.7.	MQTT testbed architecture integrated with the 5G edge cloud.	74
3.8.	OPC-UA testbed architecture integrated with the 5G edge cloud.	74
4.1.	Disassembled UR5 robotic arms	85

4.2. Robotic cell architecture integrated with 5G and edge-cloud	86
4.3. Final setup for the operational robotic cell	87
4.4. Diagram of the Operational CCLL.	90
4.5. Data acquisition process by the robotic arm.	90
4.6. Operational Closed Control Loop Latency (OPCCLL) over MQTT over Ethernet vs 5G.	92
4.7. OPCCLL over OPC-UA over Ethernet vs 5G.	92

List of tables

2.1. MQTT connect return code [32].	38
2.2. CCLL in MQTT for Testbed over Ethernet.	56
2.3. CCLL in MQTT-SN for Testbed over Ethernet.	56
2.4. CCLL in OPC-UA for Testbed over Ethernet.	57
2.5. Link outage in MQTT for Testbed over Ethernet.	58
2.6. Link outage in MQTT-SN for Testbed over Ethernet.	58
2.7. Link outage in OPC-UA for Testbed over Ethernet.	59
2.8. MQTT CCLL for different number of publishers.	60
2.9. Time elapsed in MQTT broker	60
2.10. Payload size comparison.	61
2.11. Summary of performance indicators over Ethernet.	65
3.1. CCLL for Testbed over 5G using MQTT.	76
3.2. CCLL for Testbed over 5G using MQTT-SN.	76
3.3. CCLL for Testbed over 5G using OPC-UA.	77
3.4. Link outage in MQTT for Testbed over 5G.	78
3.5. Link outage in MQTT-SN for Testbed over 5G.	78
3.6. Link outage in OPC-UA for Testbed over 5G.	78
3.7. Summary of performance indicators over 5G	81
3.8. Comparison of performance indicators over Ethernet and 5G	83
3.9. Performance difference over Ethernet and 5G	83
4.1. Payload, packet and frame sizes in operational settings.	89
4.2. Operational cycle elapsed time for the different tested configurations	91

Acronyms

3GPP 3rd Generation Partnership Project. 21, 68

5G NR 5G New Radio. 26

5G NSA 5G Non-Standalone. 66, 68

5G SA 5G Standalone. 66, 68

5GC 5G Core. 66

ACK Acknowledgement. 35

AMR Autonomous Mobile Robot. 21, 23

AP Access Point. 21

APN Access Point Name. 71

AR Augmented Reality. 22

ASCII American Standard Code for Information Interchange. 54, 60, 61, 114, 116–118

AVG Automated Guided Vehicle. 23

BAUP Bytes Actually Used for Payload. 55, 61

BRP Bytes Reserved for Payload. 55, 61

BS Base Station. 21

CCLL Closed Control Loop Latency. 10–12, 46–52, 56–60, 62, 64, 65, 73, 75–79, 81, 83, 87, 89, 90, 122

CDF Cumulative Distribution Function. 123–154

CLI Command Line Interface. 70

CPS Cyber-Physical Systems. 18

DL Downlink. 68, 75

DNAT Destination NAT. 71, 72

ECDF Empirical Cumulative Distribution Function. 52, 56, 63, 64, 79, 80, 82, 90

eMBB Enhanced Mobile Broadband. 22

eNB Evolved Node B. 66

EPC Evolved Packet Core. 66

EPP Effective Payload Percentage. 55, 61

- ERP** Enterprise Resource Planning. 19
- Gbps** Gigabit per second. 46, 68
- gNB** Next-generation Node B. 66
- GNSS** Global Navigation Satellite System. 68
- GPIO** General Purpose Input/Output. 68
- GSM** Global System for Mobile Communications. 71
- HMI** Human Machine Interface. 19, 40
- HSPA+** Evolved HSPA. 68
- I/O** Input/Output. 52, 85
- IIoT** Industrial Internet of Things. 3, 17, 18, 20, 21, 23, 24, 28–31, 51, 57, 62, 66, 79, 83, 91, 95
- IoT** Internet of Things. 22, 54
- IP** Internet Protocol. 10, 55, 70, 71
- KPI** Key Performance Indicator. 52, 88
- LAN** Local Area Network. 46, 70, 71
- LBT** Listen-Before-Talk. 20
- LTE** Long Term Evolution. 66, 68
- M2M** Machine to Machine. 18
- MAC** Medium Access Control. 39
- MES** Manufacturing Execution System. 19
- mMTC** Massive Machine-Type Communication. 22, 23
- MPLS** Multiprotocol Label Switching. 66, 67
- MQTT** Message Queuing Telemetry Transport. 10–12, 27, 31–35, 37–39, 41, 46–50, 52–65, 70, 71, 73, 74, 76–83, 85–93, 95, 97, 98, 105–107, 109, 110, 116, 117, 120–127, 139–143
- MQTT-SN** MQTT for Sensor Networks. 10, 12, 38, 39, 56–59, 62, 63, 76–80, 95, 105, 106, 108, 109, 128–131, 144–147
- MTL** Maximum Tolerable Latency. 52, 58, 59, 77–79
- MTU** Maximum Transmission Unit. 51

NAT Network Address Translation. 70–72

NFV Network Function Virtualization. 66

OPC Open Platform Communications. 40

OPC AE Open Platform Communications Alarms & Events. 40

OPC DA Open Platform Communications Data Access. 40

OPC HDA Open Platform Communications Historical Data Access. 40

OPC-UA Open Platform Communications Unified Architecture. 10–12, 27, 31, 40–52, 54–65, 71–74, 76–83, 85–89, 91–93, 95–98, 105, 106, 110–114, 132–138, 148–154

OPCCLL Operational Closed Control Loop Latency. 11, 88–92

OPCET Operational Cycle Elapsed Time. 89, 91

OSI Open Systems Interconnection. 10, 30, 34

PCIe Peripheral Component Interconnect Express. 68

PLC Programmable Logic Controller. 19, 25, 27, 28, 40, 41, 52, 74, 84, 86, 88, 89, 95, 105, 106

PP Payload Percentage. 55, 61

QoS Quality of Service. 10, 23, 33–37, 54, 56–60, 62, 64, 65, 75–79, 81, 87, 95, 126, 127, 131, 142, 143, 147

RAN Radio Access Network. 26

RSMB Realy Small Message Broker. 108

RTDE Real-Time Data Exchange. 89, 105, 106

SCADA Supervisory Control and Data Acquisition. 19, 40

SDN Software Defined Networks. 66

SIM Subscriber Identity/Identification Module. 10, 68, 69

SSH Secure Socket Shell. 72

TCP Transmission Control Protocol. 10, 30–35, 37–39, 43, 55, 56, 72, 108, 109

UDP User Datagram Protocol. 10, 30–32, 38, 39, 51, 56, 62, 75

UE User Equipment. 26, 67

UL Uplink. 68, 75

URLLC Ultra-Reliable Low Latency. 22

USB Universal Serial Bus. 68, 71

VR Virtual Reality. 22

WAN Wire Area Network. 66, 67

1. Introduction

1.1.- Motivation

Most of the wireless communications applications are visible on a daily basis thanks to the use of our smart devices but, how did these devices reach us? Everything that is tangible has been the result of a certain production process. Due to population growth and the fast advancement of technology, smart production processes, Industrial Internet of Things (IIoT) and 5G communication systems are really important when it comes to manufacturing enhancement. Industry 4.0 can be applied to an industrial scenario to achieve the following aspects:

- Efficiency improvement: through the use of IIoT and smart production processes, it is possible to improve efficiency by optimizing operations, tasks, reducing costs and waste.
- Quality improvement: with IIoT and 5G technologies is possible to track, collect data and monitor production to identify quality problems or even predict them before they become a bigger issue.
- Better decision-making: thanks to all the data collected by machines, sensors and connected devices in general, improved and more accurate decisions can be made due to data-driven models. This is particularly interesting at the time of identifying patterns, optimizing processes and making predictions.
- Safety improvement: sensors can also be used to study and monitor worker's safety in the factory using technologies such as computer vision which require the high bandwidth and low latency that 5G communication technologies offer.

1.2.- Historical approach

It is clear that before we knew about the concept of Industry 4.0, there were three previous important industrial revolutions [1]. The First Industrial Revolution began in the mid-18th and brought really important changes in the economical and social organisation. It was mainly triggered by the emergence of capitalism, mechanisation of production and the use of steam power. Although steam power had been known for years, it was its application for industrial purposes that achieved the paradigm shift.

Just over a century later and thanks to the discovery of electricity and advances in the creation of steel and chemicals, the society met The Second Industrial Revolution. Well known tycoon Henry Ford got closer to what we know about industrial processes nowadays thanks to the inclusion of conveyor belts in the mass production process used for his card company.

Moving closer in time, around the 1970's, mankind witnessed the birth of The Third Industrial Revolution whose drivers were partial automation and the use of electronic devices in manufacturing processes.

Last but not least, we are living in The Fourth Industrial Revolution which is characterised by the use of smart devices, digitalization, IIoT and Cyber-Physical Systems (CPS) among other causes. The following thesis will cover one of the main aspects of this huge and new ecosystem: the smart production processes.

In Figure 1.1 a small diagram illustrates the key concepts of each industrial revolution.

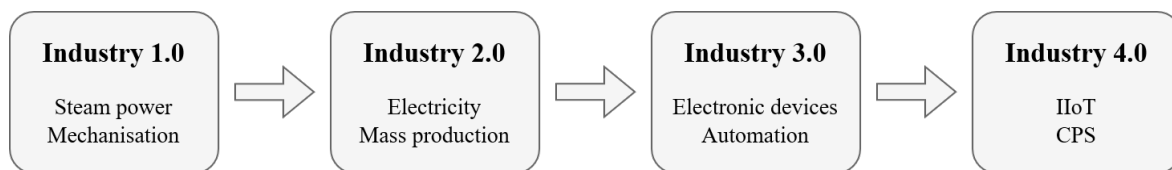


Figure 1.1: The four industrial revolutions.

1.2.1.- Industry 4.0

Industrial environments are one of the main players when it comes to the fast and disruptive evolution of the communication technologies. This new industrial revolution based on that technologies is called Industry 4.0. It focuses in Machine to Machine (M2M) communications so all those processes can be seen as data exchanges. In this way, physical devices are converted to data terminals whose main function is to gather information that combined with the power and benefits of cloud and egde-computing technologies, can lead to a deeper and more intelligent analysis that optimises operations.

If we stop to think about the scenarios that can be covered by industrial automation, the options are endless but the most known are mining, energy production, agriculture, logistics and manufacturing. The main thing they all have in common is the huge amount of devices connected in order to control their systems. That's why scalability, latency, reliability and security are key requirements when it comes to the deployment of these environments.

It is important to understand the ISA-95 (IEC 62264) [2] standard. This model labels the industrial automation elements into five main layers according to their functionality and time dependence (some of them are time-critical). It is depicted in Figure 1.2

- Layer 0 - Field level: IoT sensors and actuators of the factory. It is the physical production process.
- Layer 1 - Control level: the main task is to sense and manipulate the production processes. It is usually done using Programmable Logic Controllers (PLCs).
- Layer 2 - Supervisory level: the main tasks are the supervisory, monitoring and automated control of the production processes. Supervisory Control and Data Acquisition (SCADA) and Human Machine Interface (HMI) are used in this layer.
- Layer 3 - Planning level: here is where the Manufacturing Execution System (MES) takes place. It's main goal is to ensure effectiveness of the operations by maintaining records and optimizing the production process.
- Layer 4 - Management level: it's the top layer. Here the Enterprise Resource Planning (ERP) system is exploited.

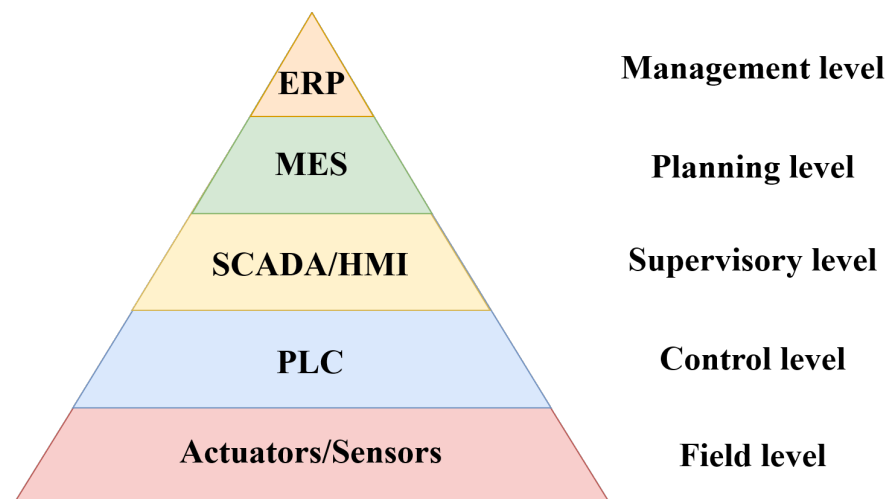


Figure 1.2: ISA-95 pyramid.

1.3.- Industrial networks

1.3.1.- Ethernet

Network architecture in today's industry is mainly Ethernet based [3]. This well known technology has been the principal component when it comes to real-time communications within all industrial automation levels. Although there are cable categories that cover really high bandwidth, speed, latency and compatibility demands such as the CAT6 [4], there is still some major disadvantages that have not been solved:

- **Mobility:** Ethernet cables are limited in terms of mobility. It is a clear disadvantage when it comes to scenarios where devices need to move around or be repositioned, such as in industrial manufacturing environments.
- **Set up complexity:** these cables can be difficult to manage in big industrial environments where there can be found a high density of devices.

1.3.2.- WiFi

Legacy industrial systems won't disappear with the introduction of new wireless communication systems to industry scene since not all scenarios will require ultra low latencies [5]. Known technologies such as WiFi are able to cover most of the requirements demanded by certain kind of applications but using them is not always the best option. One of main issues when it comes to the deployment of WiFi is related to its operation in unlicensed bands. Resources have to be shared and therefore interference from neighbour networks might be present.

Some mechanisms such as Listen-Before-Talk (LBT) are used to avoid collision between devices sharing the same frequency bands. It is the de-facto access mechanism in Wi-Fi. This is really important in the 2.4 and 5 GHz bands (although it also applies 6 and 60 GHz) since they are usually heavy congested due its adoption many general-purpose communication technologies such as Bluetooth [6] and Zigbee [7] and there is a limited number of available channels. These methods can have an impact on latency because they introduce a delay between the time a device wants to transmit data and the time when it actually begins to transmit. This delay is necessary in order to allow the device to ensure there is no one using the channel.

Another important point to consider in IIoT scenarios is the mobility of the devices since it is desired to ensure a full flexibility of the scenario. Handover behaviour is different when

using 5G or Wi-Fi. While 5G’s handover is managed by the own network, in Wi-Fi it is done by the Base Station (BS). It computes how and when to handle a roaming event at the time the devices leaves a certain area covered by a particular Access Point (AP). In the process of abandoning the current AP connection and linking to a different AP, there is a brief period of time in which the beacon is lost. This happens because Wi-Fi handover is “break before make” so the connection to the first AP is broken before establishing a new one to the second AP [8]. This is shown in Figure 1.3, where an Autonomous Mobile Robot (AMR) moves from the covering area of the AP 1 to the one covered by AP 2 finding itself without connection in the border region between both of them.

Even though there have been some advances in the latest versions of Wi-Fi such as Wi-Fi 6 (IEEE 802.11ax) [9] that target mobility enhancements, they are still not enough to fully meet the actual demanding IIoT requirements [10].

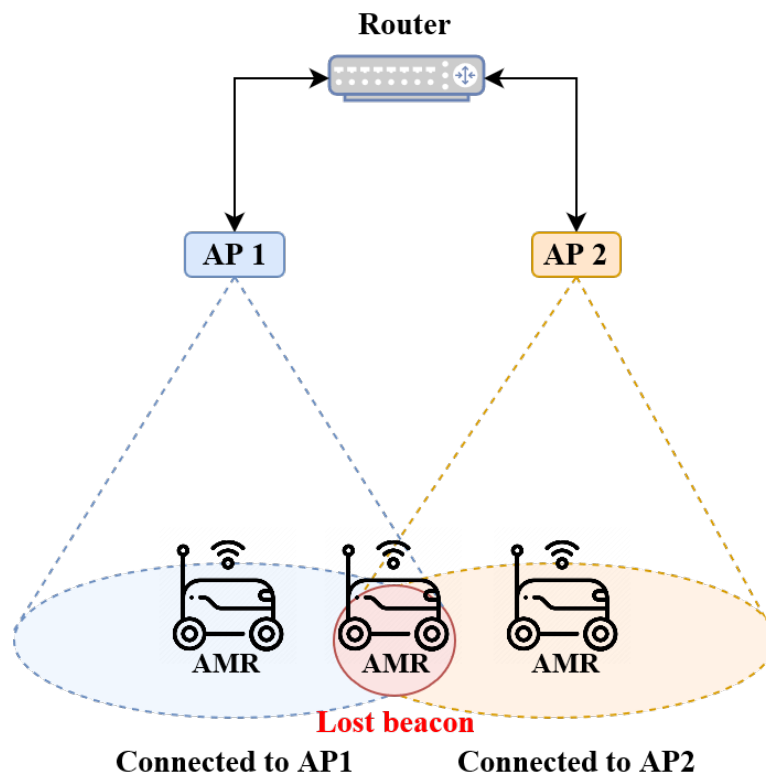


Figure 1.3: Handover in WiFi.

1.4.- 5G for industrial environments

5G technology supports different services with specific requirements. Three main categories defined by the 3rd Generation Partnership Project (3GPP) provide different capabilities and levels of performance for wireless communication systems [11]. Figure 1.4 displays

a common way of explaining the three main categories of 5G technology related to their most important features. Industry 4.0 goals mainly focused on Ultra-Reliable Low Latency (URLLC) due to the low latency requirements but as the blue square shows, in order to achieve them it is important to consider some other aspects included in Enhanced Mobile Broadband (eMBB) and Massive Machine-Type Communication (mMTC).

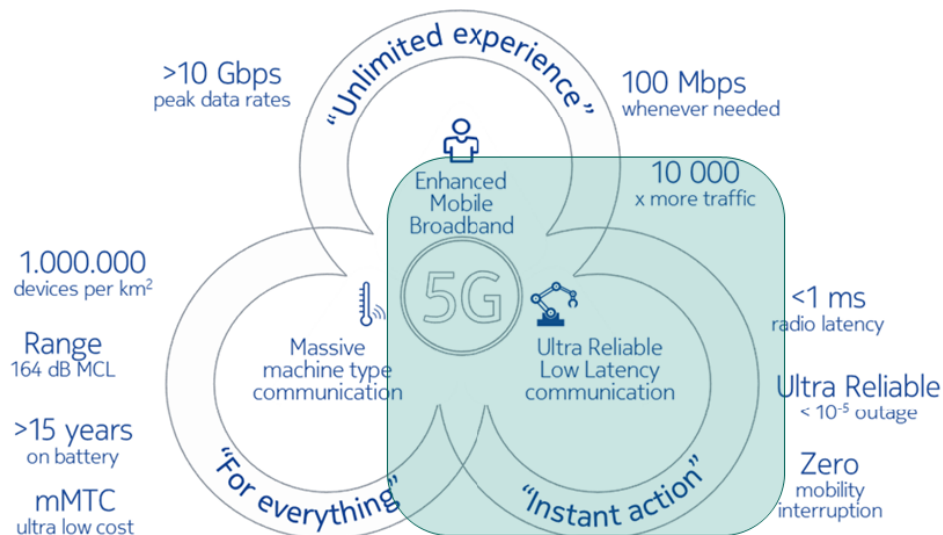


Figure 1.4: 5G applications [12].

- eMBB: this service category is designed for applications such as Virtual Reality (VR), Augmented Reality (AR), video streaming and cloud-based gaming due to their high-speed, high bandwidth and low latency requirements [13].
- URLLC: this service category is designed to provide (as the name implies) ultra-reliable and low-latency communications for scenarios such as autonomous systems, industrial automation or remote surgery. Release 15 sets a latency target as low as 1 millisecond while providing high reliability making it suitable for fast response times [13].
- mMTC: this latter category is designed to support a massive quantity of sensors, actuators and other Internet of Things (IoT) devices. The focus here is not on latency but on low data rates and low power consumption, making it a great candidate for applications that require long battery life [13].

The area highlighted in green shown in Figure 1.4 represents approximately the targets of industrial 5G which, in addition to requiring really low latencies (URLLC), also demands for large bandwidths (eMBB) that can be used for deploying VR and AR solutions as well as

a high number of devices (mMTC), particularly relevant when numerous IIoT devices and sensors need to communicate and collect data. It is important to emphasize that although not all requirements are provided at the same time, 5G is flexible by design allowing to configure multiple uses simultaneously.

1.4.1.- Benefits of using 5G in industry

Once the current industrial networks and 5G technology are introduced, it is time to highlight the advantages of using the latter one for industrial applications. Apart from all the supported use cases, 5G is scheduled and its handover offers the introduction of mobile elements within the industrial production environment and mobility support. Due to the capacity of 5G of handling a massive number of devices, flexibility, efficiency and cloudification are other interesting advantages offered by this technology. Another key feature of 5G is its operation in licensed spectrum which translates into that Quality of Service (QoS) can be guaranteed [5].

Due to the use of mobile devices, it is possible to achieve reconfiguration inside the industrial scenario. It is the ability of modifying the configuration of the environment in order to meet changing requirements or to correct issues. This allows the manufacturer to make changes in its industrial topology changing the roles or even the number of devices used to develop an specific task. Thus, if we have a wired production chain consisting of a controller and three machines as shown in the Figure 1.5, in case we wanted to reorder their elements so we could face a different production task it would be an issue because everything would have to be disconnected and reconnected again, and there could even be the possibility of getting compatibility problems between the different interfaces [5].

By using 5G this would not be a problem since we could swap between machines and reconfigure their tasks faster. We could even achieve a *plug and play* environment reducing the time and effort required for their setup and configuration.

Following this same idea, we run into another of the great advantages of using 5G: mobility support. It is a key feature that enables devices to seamlessly transition between different network environments. This can be used for Automated Guided Vehicles (AVGs) and AMRs as depicted in Figure 1.5. These devices are usually intended for material handling, transportation tasks and logistics processes.

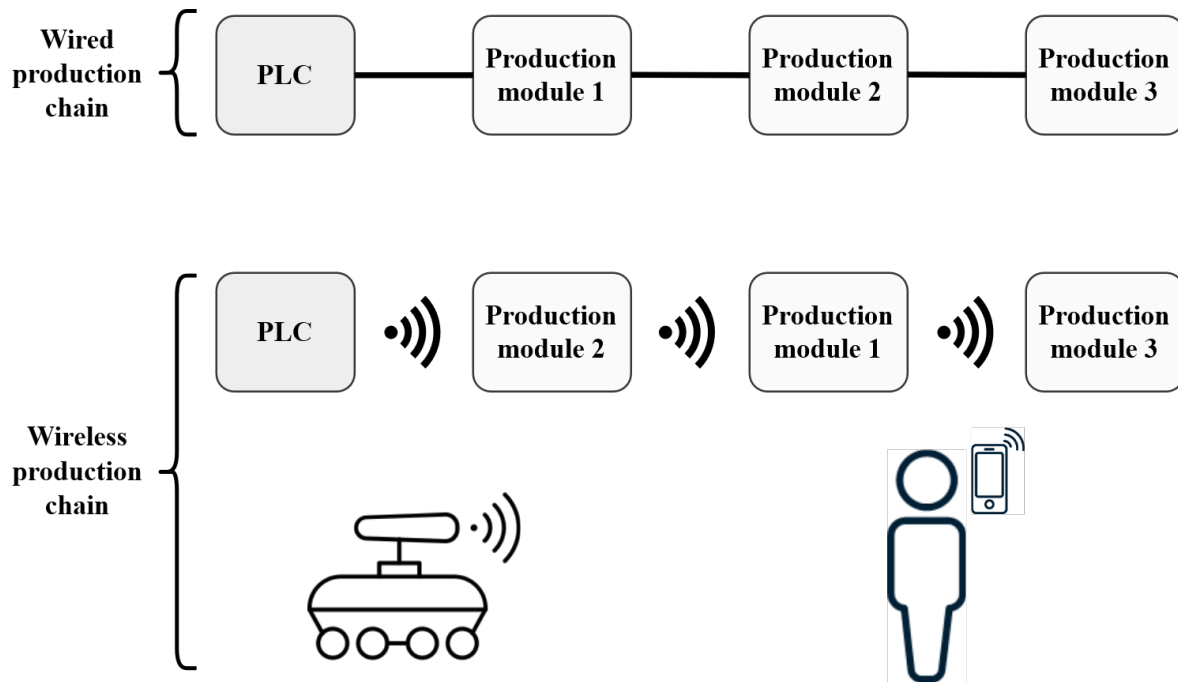


Figure 1.5: Wired production chain.

1.5.- Edge-cloud computing for industrial environments

Smart manufacturing and Industry 4.0 are characterized by the presence of interconnected systems and processes. Their output consists on real-time exchange of data that controls the events taking place inside the factory and their connection to the outside world. Due to the large number of devices that make up an intelligent manufacturing network, there is a massive amount of data that has to be collected, analyzed and used in order to create value and improve security, efficiency, costs and customer needs [14]. In order to do so, it is necessary to have high computational resources. Current trend achieves it by using two paradigms: cloud and edge-cloud computing [15].

Within a cloud industrial environment there is a data exchange between IIoT devices and the internet by making use of a centralized computing unit, the cloud. In these kind of scenarios where desired transmission delays are around a few milliseconds [16] and reliability, privacy and security of data is expected, this is not the most suitable option due to the greater distance and number of hops between where the data is produced and analyzed.

Moving the computing closer to the IIoT devices, results in the edge-cloud paradigm [17]. Here, acquired data is analyzed and processed in the network's edge. This architecture offers lower latencies and response timing due to the proximity between both ends therefore it is more suitable for real-time applications. Security and privacy is also enhanced since credentials and critical data could be treated and encrypted in the edge-cloud and only remaining

payload is sent across the internet [14].

The combination of edge-cloud and 5G technologies is an enabler for cloudification in industrial scenarios. This refers to the process of migrating computing resources that are currently used in local devices to cloud infrastructure and data centers so that it is possible to take advantage of the high capacity, low latency and reliability of those technologies. Typical cloudified application scenarios can range from storage or processing of data or control software to virtual replication of complex tasks and machines. The common benefits of cloudification are [18]:

- Elasticity: resources can be increased or decreased freely.
- Flexibility: new services or add-ons can be easily attached to the cloudified process leading to more custom solutions.
- Information exchange: it enables seamless information sharing and coordination among various manufacturing elements.

1.6.- Thesis application scenario

This last section of Chapter 1 gathers all important information needed in order to understand and segment the different concepts and goals of the thesis: relocating the logic of a PLC within an edge-cloud architecture.

1.6.1.- Robotic cell

The term robotic cell in manufacturing refers to a specific configuration or deployment of industrial robots and their supporting equipment within a production facility. Its design is thought with the objective of developing an specific manufacturing task or process which usually requires high levels of efficiency and automation [19].

Figure 1.6 displays a diagram of what a typical robotic cell architecture looks like in factories. As depicted in the figure, the robotic cell is composed by a robotic arm (sensor and actuator) and a PLC which controls the manufacturing action performed by the arm itself. Since these scenarios are mainly wired [5], they are therefore limited by the disadvantages explained in Section 1.3.1. The main research lines and work done in this project will deal with migrating the PLC from an on-site scenario to a cloudified one using 5G technology as depicted in Figure 1.7. The advantages of this deployment are given in Section 1.5.

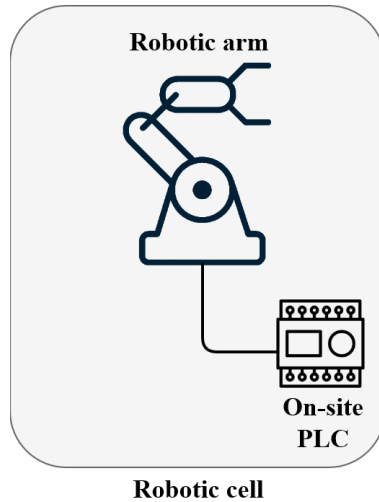


Figure 1.6: Current architecture of a typical wired robotic cell.

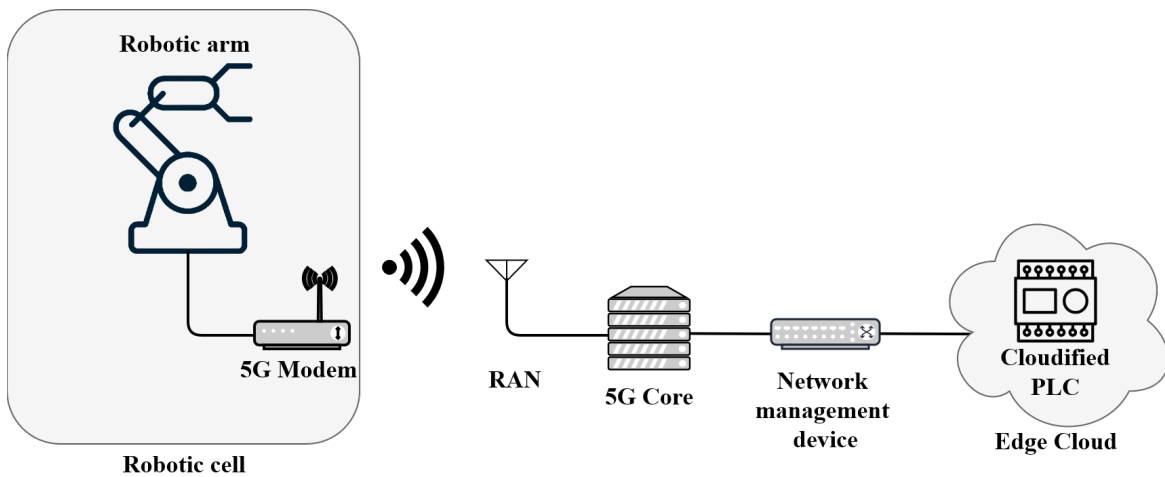


Figure 1.7: Proposed architecture of a robotic cell using 5G and edge-cloud technologies.

While the current scenario is only composed by two elements, the proposed one gathers the following:

- **5G Modem:** device that enables connectivity of the robotic arm to a 5G network. This kind of elements are also usually referred as User Equipment (UE) or 5G industrial UE (more specifically in this type of scenarios).
- **Radio Access Network (RAN):** part of the the cellular network that is in charge of managing the communication between the UE and the core infrastructure.
- **5G Core:** it is the heart of a 5G New Radio (5G NR) network. Its main tasks consist on providing access to the services, routing data and establishing secure and reliable communications between the end users and the network.

- Network management device: equipment used for interconnecting different networks.
- Edge Cloud: distributed computing system used to bring computational resources closer to generation, collection and consumption of data. Differs from the internet-based cloud in that the data processing happens closer to the user devices leading to lower latencies and higher efficiency.
- Cloudified PLC: software implementation of an on-site PLC running on a edge cloud that performs exactly the same operations as the on-site PLC. This will be used for controlling and communicating with the robotic cell due to the use of industrial communication protocols.

1.6.2.- Industrial communication protocols

Since the main communication needs in industrial ecosystems are mainly related to gathering telemetry data and control machines, sending and receiving data to/from devices is needed. This is typically done by using reference industrial communication protocols. They are standardized sets of rules and formats that enable devices and systems in industrial environments to exchange information effectively and reliably. In such scenarios where automated systems, complex machinery and processes are prevalent, these protocols must ensure seamless interoperability and coordination among different components [20].

The project will also investigate two prominent industrial communication protocols: Message Queuing Telemetry Transport (MQTT) and Open Platform Communications Unified Architecture (OPC-UA). Both are great candidates for the development of the project due to their widespread adoption, versatility, and relevance in modern industrial applications [21]. A thorough performance study and comparison of both protocols will be made in order to evaluate their behavior and characteristics when deployed over Ethernet and 5G technologies.

1.6.3.- 5G

As explained in Section 1.4.1, 5G holds great potential for industrial scenarios due to its unique capabilities and advantages. Exploring them and how they affect the behaviour of our robotic cell scenario is other of the main aspects to be analyzed in the project.

1.6.4.- Edge-cloud and cloudification

By embracing edge-cloud and implementing cloudification, organizations can leverage the benefits explained in Section 1.5, enabling the transformation of industrial processes, enhances operational efficiency, and empowers businesses to embrace digital transformation in the Industry 4.0 era.

1.7.- Objectives

The main objective of this project is:

- OBJ: To investigate, design, and implement a wireless industrial network solution to enable the control of a robotic cell environment from a remote PLC situated in the edge-cloud using 5G technology.

In order to achieve the main goal, a subset of partial objectives were set:

- OBJ1: Investigation of the main control network architectures for the different IIoT protocols and performance evaluation under realistic network conditions in both typical wired setups and in 5G wireless settings.
- OBJ2: Design and development of control network architectures for real-world operational robotic equipment based on the reference IIoT protocols, using 5G and edge-cloud technologies.
- OBJ3: Deployment and performance evaluation of the designed 5G edge-cloud control network solutions with real-world robotic equipment in operational conditions.

1.8.- Thesis outline

This document will be organized into five main chapters, each focusing on specific aspects of the project. The following are detailed below, accompanied by a brief description:

- Chapter 1: this introductory chapter provides an overview of the project, discussing the motivation behind it and presenting the current state of the oart of relevant technologies in the field of IIoT. It also outlines the objectives of the research and introduces

the selected case study.

- Chapter 2: it includes a deep theoretical and practical study of the main IIoT control protocols and architectures, as well as their performance evaluation over wired Ethernet.
- Chapter 3: this chapter describes a framework for Ethernet to 5G system integration and includes a performance evaluation of the IIoT protocols over 5G and edge-cloud technologies.
- Chapter 4: it includes a description of the operational robotic cell environment. Within this environment, a functional solution was designed, implemented, deployed, and tested, using 5G and edge-cloud technologies. The chapter provides an overview of the specific setup and infrastructure used for the development and execution of the solution, highlighting the integration of cutting-edge technologies to enhance robotic operations.
- Chapter 5: concludes the thesis with an outline of the main conclusions and some thoughts about potential future research lines.

In addition to the aforementioned elements, the thesis incorporates a section that elucidates the impact of the project and significance within the smart manufacturing ecosystem. Furthermore, it features four appendices which provide a more comprehensive understanding of some of the topics illustrated in the preceding chapters.

2. IIoT control protocols

When it comes to deploying an IIoT scenario, we need to make a deep study on which protocol we should use in order to get the most out of our application. How, when and in which way we need to collect, analyze and share the data are some of the most important variables at the time of choosing the protocol. Actually almost all the details that we need to evaluate are related to the information we want to transmit and receive between the different elements of the scenario. The most important concepts to consider are [22]:

- **Data collection:** it is important to choose an IIoT protocol that supports the types and amount of data that our application requires.
- **Data transmission:** the chosen IIoT protocol should handle the data transmission requirements of our application such as bandwidth and latency.
- **Data processing:** if our application requires a specific data processing such as real-time, it is important to adopt an IIoT protocol that can handle it.
- **Interoperability:** this is one of the most important features to cover when deploying an IIoT environment. Different devices and systems are thought to work together seamlessly so it is important to choose a protocol that ensures the communication between them regardless of the manufacturer or technology.
- **Reliability:** as in most cases, industrial environments need to ensure reliability as any downtime or communication error can lead to significant production losses, safety hazards or any other negative consequences. It is important to implement a protocol that ensures reliable data delivery even in the worst network conditions.
- **Security:** most valuable and vulnerable elements within an IIoT ecosystem are data and devices. The chosen protocol should include robust security features in order to ensure that our data and machines are protected from cyber threats and unauthorized users.

In addition to the above bullet points, from a pure communication-related point of view, apart from their specific application layer features, the IIoT protocols are dominated by their characteristics at transport layer. At this level, the two main characters of this Open Systems Interconnection (OSI) model layer are Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) and they can be compared using the following concepts [23]:

- **Reliability:** TCP offers reliability while UDP does not. TCP guarantees that data will be delivered to the receiver in an orderly and error-free way (it is possible since it adds sequence numbers to the packets). If for some reason any packets are damaged or lost during the transmission, TCP will take care of re-transmitting them. UDP's scenario is different since it does not guarantee an ordered delivery of the packets and does not offer any error-correction mechanisms.
- **Packet structure:** It is clear that since they are different protocols, their packet structure will vary. TCP includes more headers, acknowledgments, sequence numbers and other control information in order to ensure reliable data transmission while UDP is set in a smaller packet and therefore lighter.
- **Connection:** TCP is what is known as a connection-oriented protocol. This means that it has to establish an initial virtual connection between the sender and receiver before transmitting any data. UDP, on the other hand, is not connection-oriented so it just encapsulates the data into a datagram and sends it over the network.
- **Flow and congestion control:** TCP is able to regulate the amount of data that is being sent by implementing flow and congestion control mechanisms. UDP does not offer this feature.

Figure 2.1 shows the packet structure of both transport layer protocols where the above concepts can be corroborated. TCP segments include fields such as checksum, acknowledgement number and sequence number (among others) that lead to a reliable and ordered data transmission. Other important differences between them are the flags, windows size and urgent pointer found within a TCP segment. First ones are used to indicate specific status and control conditions of the communication. Window size is used by a TCP receiver to indicate how many bytes it desires to receive. Urgent pointer, as the name implies, is intended to advertise data that should be delivered as soon as possible [23].

Just as TCP and UDP are known as the main protocols of the transport layer, going up to the application layer within an industrial scenario, MQTT [24] and OPC-UA [25] are the two of the most important and used IIoT protocols.

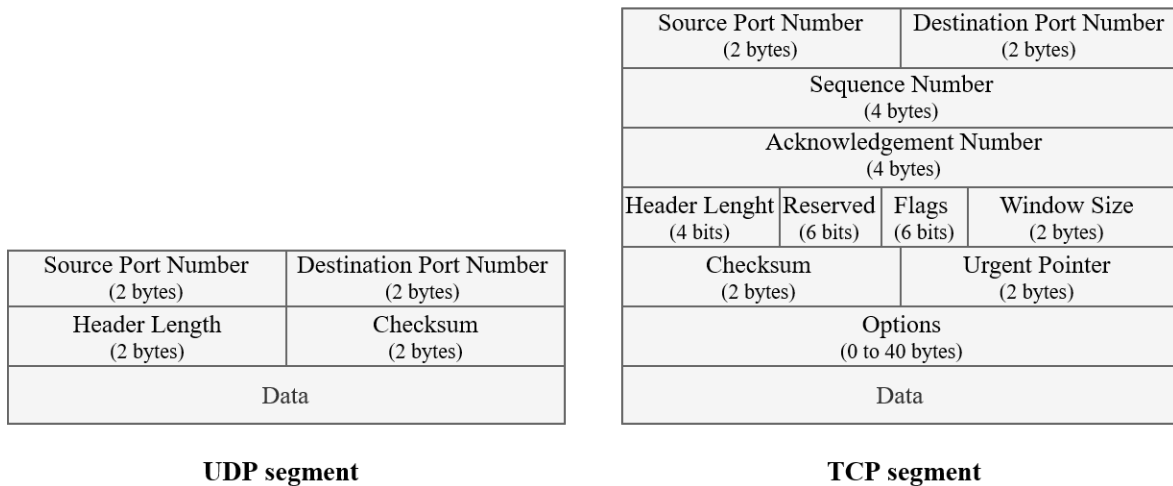


Figure 2.1: Segment fields of TCP and UDP.

2.1.- MQTT

MQTT is a very lightweight publish-subscribe application layer protocol mainly designed for low-bandwidth and constrained environments. It is used in a wide variety of scenarios including industrial automation, telecommunications, smart homes and even healthcare.

2.1.1.- Packet structure

As displayed in Figure 2.2 MQTT’s packet structure is made up of a fixed header, a variable-length header and the payload [26]. The fixed header gathers the control header and the packet length. This is always present in all MQTT message types. The variable-length header is not always present and it depends on the message type. Focusing on “PUBLISH” messages, MQTT’s packet size will increase depending on the name of the topic because its value and length will be stored in the variable-length header. MQTT standard defines what is the architecture of the packet but not which is the direct limit of “X” and “Y”. It only indicates that the maximum packet size is 256 MB. Finally, payload field will contain the data being sent (in case of an acknowledge packet it won’t be there since there is no data to send).

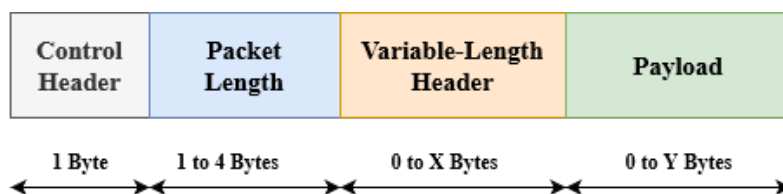


Figure 2.2: MQTT packet structure.

2.1.2.- Publish-Subscribe architecture

MQTT's communication happens between two main components: the client and the broker. An MQTT client is any application or device that is able to communicate with an MQTT broker using the MQTT protocol. A client can publish messages, subscribe to topics or both. They can be deployed on a wide range of devices such as sensors, supercomputers, micro-controllers, servers... There is also a wide variety of programming languages and frameworks that can be used in order to implement MQTT clients. In this project, Python will be used for this task.

On the other hand, an MQTT broker is a server whose role is to act as a central hub for communicating MQTT clients sending the messages to those who have subscribed to certain topics. MQTT brokers are able to manage thousands of concurrent connections while being highly available and scalable. Brokers can be deployed in different platforms such as edge devices, cloud or even dedicated servers. In the same way that there are many programming languages and frameworks to implement MQTT clients, there is also a wide variety of libraries to implement brokers, for example RabbitMQ [27], HiveMQ [28] or Mosquitto [29] (the one we will use for our project).

MQTT is publish-subscribe architecture based [26]. This is exemplified in Figure 2.3, where publishers send messages to a MQTT broker which then distributes them to the subscribers interested on those messages. Messages are published to a given topic, which is basically an identifier for the messages. Subscribers can subscribe to one or more topics, receiving the messages once they are sent to the broker by the publishers. The way topics are organised is hierarchical and they can be seen as a directory whose different levels are separated by a forward slashes.

Thus, given a robotic arm publishing to two topics called *arm/status* and *arm/temp*, subscribers interested in how the arm is performing would have to subscribe to that topic but in case they wanted to know the arm's temperature, they should have to subscribe to the topic *arm/temp*. MQTT clients (i.e publishers, subscribers or applications) are able to publish messages to the broker without the need of knowing the identity or location of other clients. It is the broker's task to route the messages to the subscribers interested on them.

One of the most important and noteworthy features of MQTT is what is known as QoS. It adds an extra level of reliability at application layer to guarantee the successful exchange of information between the publisher and the subscriber. MQTT can resend messages and guarantee their delivery even when transport layer protocol fails (for example when a TCP disconnection happens).

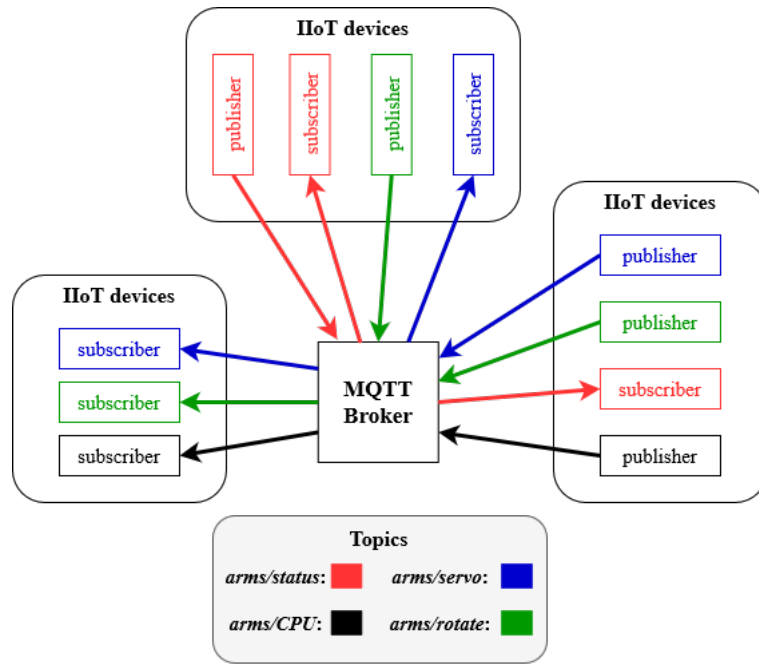


Figure 2.3: MQTT Publish-Subscribe architecture.

Figure 2.4 shows each layer of the OSI model [30] in which the physical connection (either wired or wireless) between the sending and receiving devices represented with a solid arrow happens at physical layer, while the logical connection of the rest of layers is represented with dashed arrows. As explained in the previous sub chapter, TCP offers reliability at transport layer and setting a QoS level on MQTT does it at application layer.

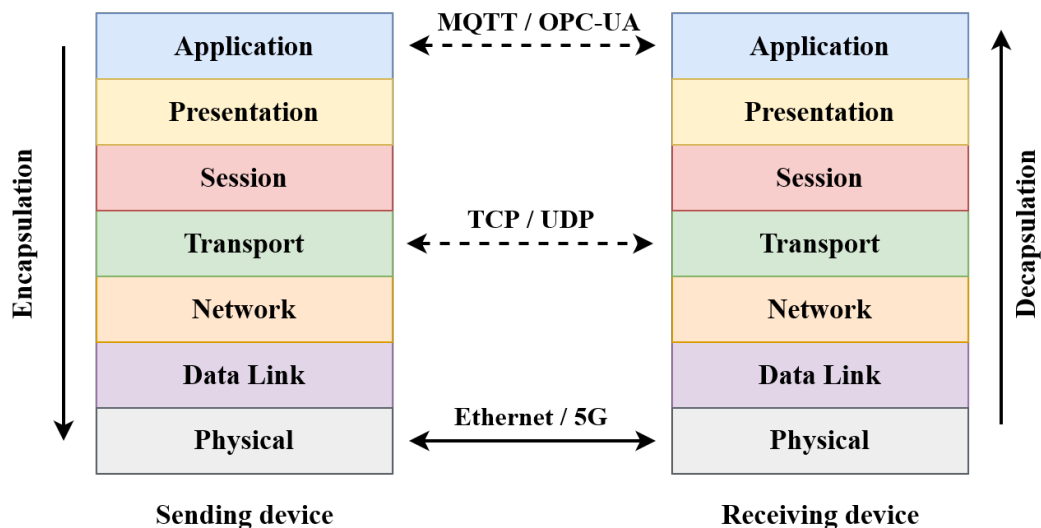


Figure 2.4: OSI model.

At the time of exchanging data between two clients using MQTT, there are two different segments in the communication from broker’s point of view: in the first one acts as receiver and in the second one as sender. When a client sends a message to the broker using a desired

QoS level the broker won't use this level when transmitting the message to the subscriber clients, it will use the one specified by each on them. That way, when a subscriber defines a lower QoS than the publishing client, the broker will send the message with the lowest QoS. MQTT supports the following levels of QoS [31]:

- QoS 0 (at most once): with this QoS level there is a best-effort delivery but there is no guarantee it is always delivered. There is no Acknowledgement (ACK) at application layer from the receiver side and the message is not stored so it can't be re-transmitted. This level is also called "fire and forget" and its reliability depends on the stability of the underlying TCP connection. With a stable TCP connection, there is a successful delivery of messages. However, if the connection is reset or closed, there is a risk that messages get lost. As it is illustrated in Figure 2.5, publisher sends a "PUBLISH" message to the broker who later will do the same with the subscriber.

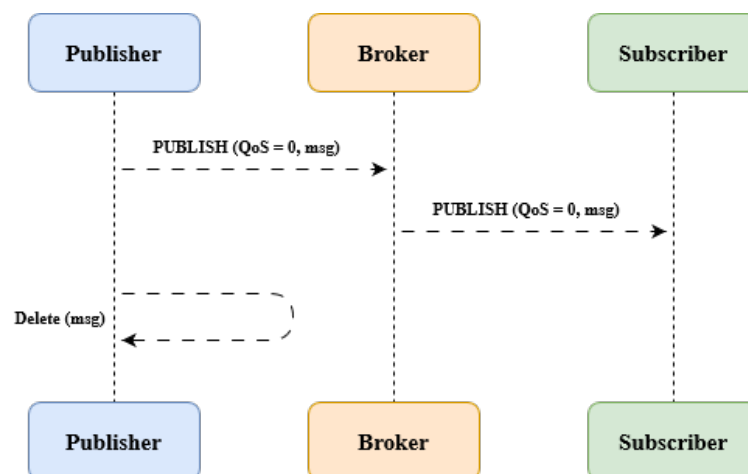


Figure 2.5: QoS 0 level.

- QoS 1 (at least once): this is the middle level of quality of service in MQTT. Here the message is delivered at least once to the receiver. The message is stored in the sender until it receives a "PUBACK" packet to check the status of sent data. In this QoS configuration there is a risk of sending a message multiple times. QoS level 1 guarantees that a message is delivered at least one time to the receiver. Figure 2.6 illustrates the entire process. There is a "PUBACK" on both sides of the communication (from broker to "Node 1" and "Node 2"). It is possible for a message to be sent or delivered multiple times. The sender uses Packet ID in each packet to match the "PUBLISH" packet to the corresponding "PUBACK" packet. If the sender does not receive a "PUBACK" packet in a reasonable amount of time, it will resend the "PUBLISH" packet. When a receiver gets a message with QoS 1, it can process it immediately. For example, if the receiver is a broker, the broker sends the message to all subscribing clients

and then replies with a “PUBACK” packet. If the packet does not reach destination, sender won’t receive “PUBACK” so it will re-transmit the message and that way the receiver will get it only once. There is an opposite case in which if the packet reaches destination but “PUBACK” doesn’t make it to the sender, it will figure out that initially the package didn’t arrived to the receiver, even though it did, so it will resend it again leading to a duplicated packet in destination. This whole process is displayed in Figure 2.7 where it is shown a “PUBLISH” packet whose Packet ID is 17 and its duplicated flag DUP is set to zero. It’s “PUBACK” message never reaches the sender so the next “PUBLISH” sets DUP flag to one.

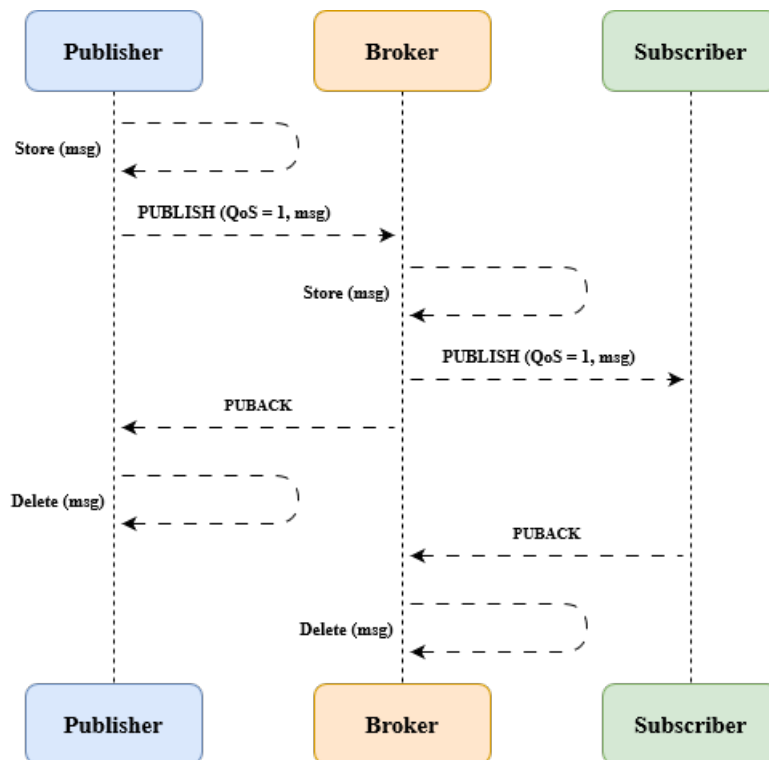


Figure 2.6: QoS 1 level.

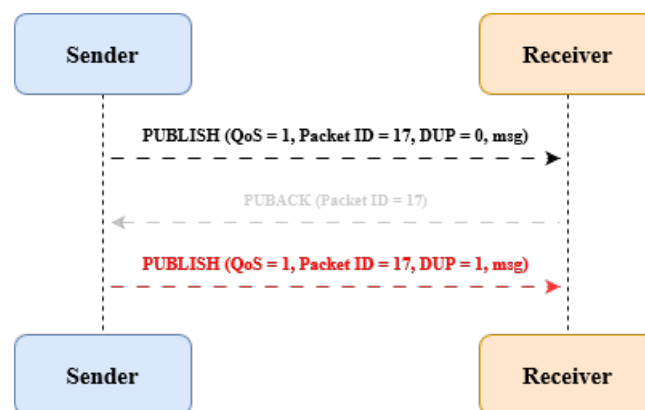


Figure 2.7: QoS 1 level: duplicated message.

- QoS 2 (exactly once): this is the highest level of QoS service in MQTT. It introduces the highest level of redundancy to guarantee the information is correctly transferred and the message is received only once. QoS 2 provides the safest scenario using a 4-way handshake (“PUBLISH”, “PUBREC”, “PUBREL” and “PUBCOMP” packets) that also leads to a relatively longer end-to-end delays. The whole flow of messages is shown in Figure 2.8.

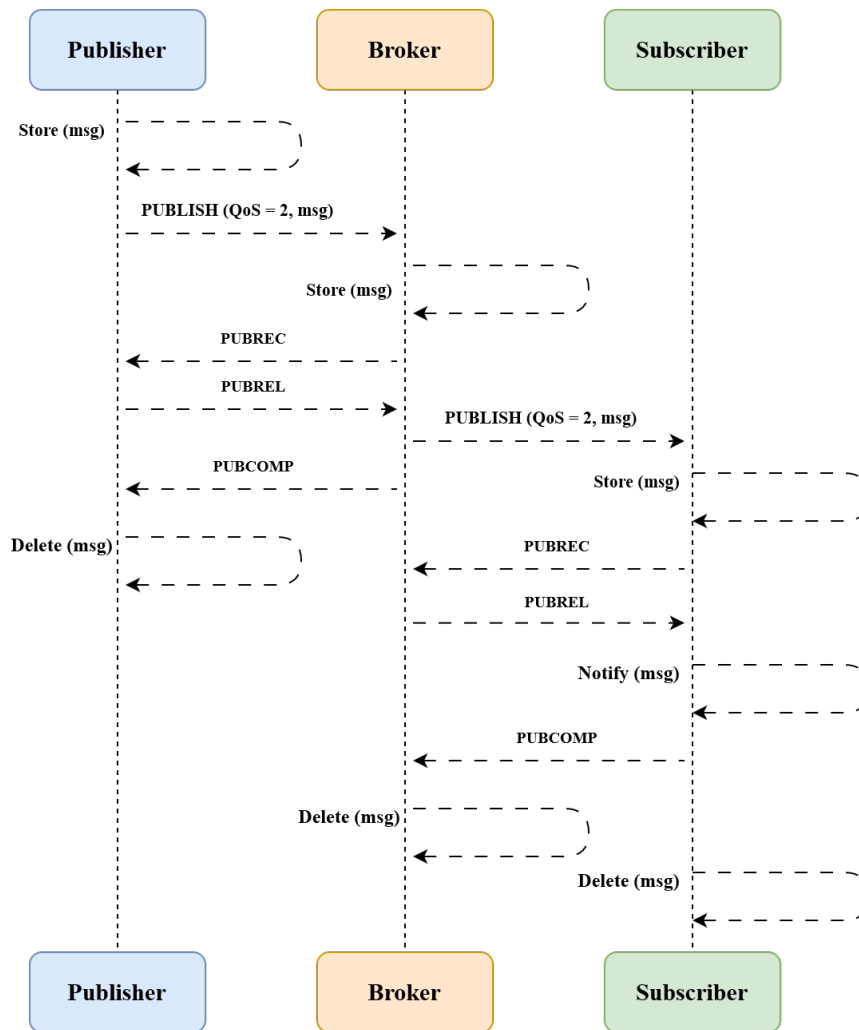


Figure 2.8: QoS 2 level.

2.1.3.- MQTT over TCP

Connection establishment of MQTT over TCP happens in two steps. First, the TCP 3-way handshake takes place. Once the virtual connection is set at transport layer, MQTT establishes its connection at application level using the “CONNECT” and “CONNACK” packets. This last packet includes a return code which is an important parameter at the time of developing and debugging MQTT applications. The whole connection process is displayed in Figure 2.9 while the most common return codes are gathered in Table 2.1.

Return Code	Meaning
0	Connection accepted
1	Server does not support the level of MQTT requested by the client
2	Client ID is correct in UTF-8 but is not allowed by the server
3	Network connection has been made but MQTT service is not available
4	Connection refused due to an error in MQTT credentials
5	Connection refused. Client is not authorized to connect
6 to 255	Reserved for future use cases

Table 2.1: MQTT connect return code [32].

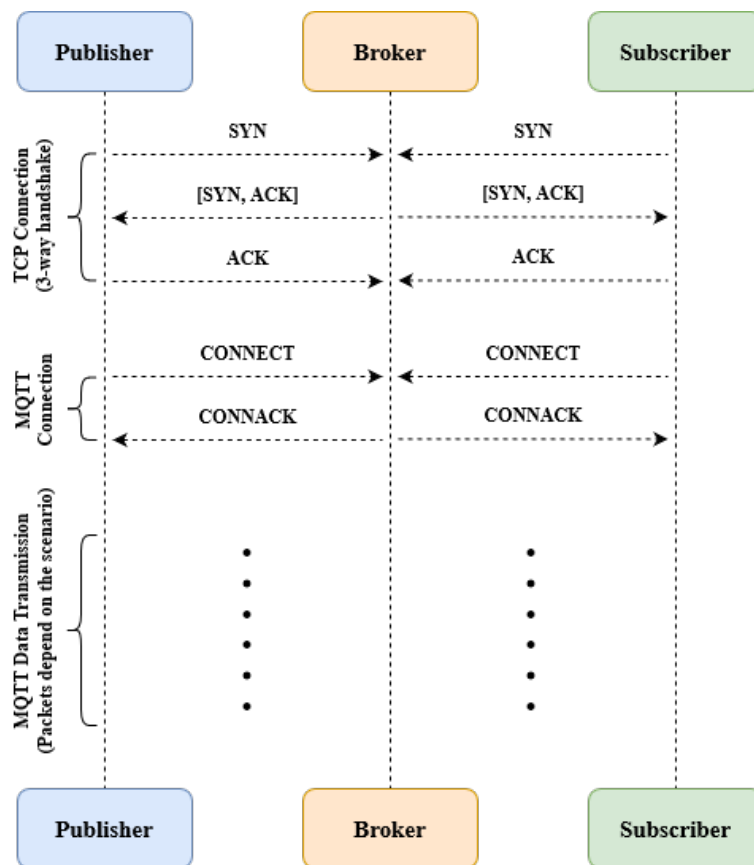


Figure 2.9: MQTT (TCP) connection packet flow.

2.1.4.- MQTT-SN: MQTT over UDP

MQTT for Sensor Networks (MQTT-SN) [33] was designed as a variation of the MQTT protocol thought to be used in constrained environments operating over various transport protocols including UDP. MQTT-SN accommodates specific requirements for devices with limited memory, processing power and network capabilities.

Using UDP as underlying transport protocol enables efficient communication scenarios where TCP's overhead is not feasible. Thus, it can reduce network bandwidth usage, communication latency and power consumption (key points studied in this project). An important thing to consider is the inability of detecting if data has been lost both at transport and application layer. Therefore, MQTT-SN should be used in use cases that do not require high sensitivity of closed-loop control unless there is assured reliability on physical and Medium Access Control (MAC) layers as it happens with 5G.

Main differences between MQTT and MQTT-SN are related to their connection establishment and payload size. While MQTT establishes a dedicated connection through the 3-way handshake explained above, MQTT-SN removes that need due to the use of UDP. Although the exact payload size limit in MQTT-SN varies depending on the specific implementation, it is common to transmit less data than in MQTT scenarios.

Figure 2.10 illustrates the exchange of messages between a publisher, broker and subscriber when using MQTT-SN. Although there is no connection establishment based on the underlying transport protocol, there is still an exchange of messages that set a connection at application level between the clients and the broker ("CONNECT" and "CONNACK" messages). This does not guarantee reliability given that in the case a packet is lost, UDP does not have built-in mechanisms for reliable and ordered delivery like TCP. This is important because it is not as critical to lose a "CONNECT" packet sent by a publisher to the broker as it is to lose a sample of data sent from a sensor.

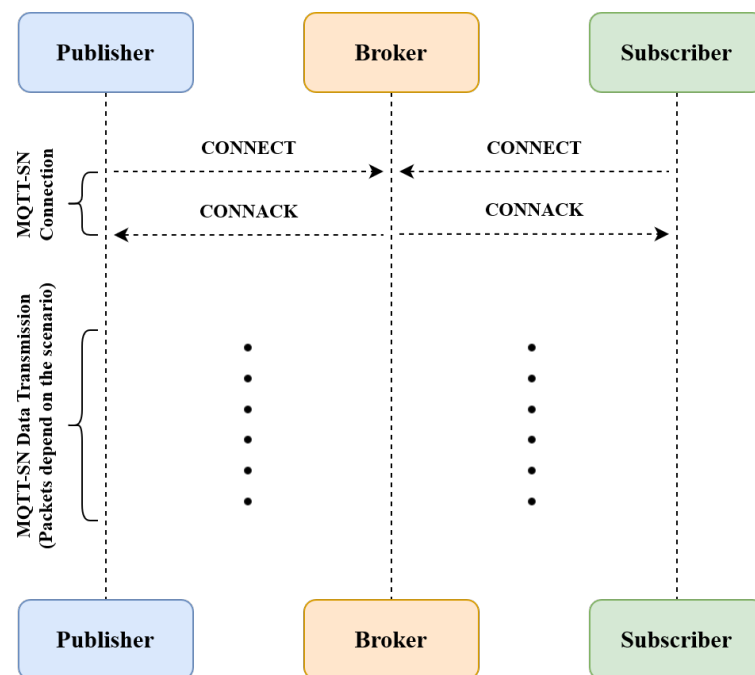


Figure 2.10: MQTT-SN (UDP) connection packet flow.

2.2.- OPC-UA

OPC-UA is one of the most famous and widely used data exchange standards for industrial applications. It was developed by the Open Platform Communications (OPC) Foundation, industry conglomerate that designs and maintains open connectivity standards focused on industrial automation [34]. This foundation has released different standards suitable for a big range of uses cases before developing OPC-UA. Some examples are Open Platform Communications Data Access (OPC DA) for data accessing, Open Platform Communications Historical Data Access (OPC HDA) for historical data and Open Platform Communications Alarms & Events (OPC AE) for accessing alarms and events [35].

The situation inside factories prior to the creation of the OPC Foundation was quite different: HMI and SCADA system collected, analysed and showed data from diverse devices by using several networks and protocols. This implied the need for manufacturers to create and distribute their own drivers so if there were changes in the specifications, the environment experienced malfunctioning communications. Solving those issues cost money and time as well as the need of stopping production or operations of some important parts of the industrial system. The biggest achieve from the OPC Foundation was the development of OPC-UA since it replaced the older single-purpose standards (there were not connection between values read with the OPC DA specification and those read with the OPC HDA standard). OPC-UA proves multi-platform support, web services, better security, scalability and data handling.

In OPC-UA there are two main roles: the client and the server. The client is the one that starts a connection to an OPC-UA server and requests data from its resources. An OPC-UA client can range from a hardware device such a PLC to a fully software application running on a personal laptop. On the other side of the communication, we can find the OPC-UA server which provides services and data to the clients. It is in charge of managing the resources on the system and making them accessible to the clients over the network. In addition to providing data, OPC-UA servers can also offer authentication and security services.

As depicted in Figure 2.11, the way data is organised inside an OPC-UA environment is defined by the OPC-UA's object model. It specifies data's structure and behavior as a hierarchical tree-like format. The OPC-UA's object represents a physical or logical device such as sensors, actuators, robotic arms, conveyor belts, PLCs... The OPC-UA object model is defined by the following concepts:

- Variables: it is the basic data element. In a manufacturing scenario it could represent the temperature or the status of a device.

- **Methods:** it is an action that can be applied on a variable or object inside a modelled system. They are used to change the value of variables or trigger certain actions in the environment. A common example would be a method used to close the gripper of a robotic arm.
- **Events:** within the OPC-UA standard, an event is a notification that is generated when a specific condition is achieved inside the system. They can be used whether to alert OPC-UA clients or to trigger a certain action inside the environment. An example of this would be a low battery event for a sensor inside a manufacturing plant.

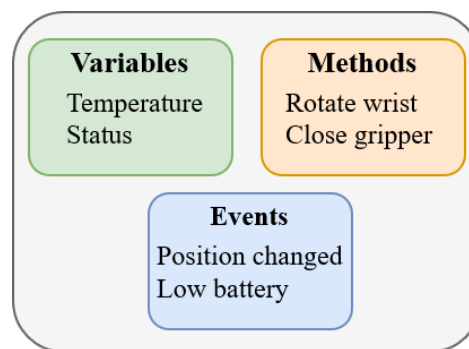


Figure 2.11: OPC-UA Object model.

From now on, it is important to highlight that OPC-UA is a whole architecture where the communication process is only a small part of it, unlike MQTT which is a lightweight messaging protocol. Data structure is more complex in OPC-UA and it is usually considered more capable of dealing with more sophisticated data and functionalities within industrial scenarios where devices from different manufactures coexist.

2.2.1.- Client-Server architecture

An OPC-UA ecosystem consists mainly of clients and servers [36]. Compared with MQTT, where data is not stored at any of the nodes, in OPC-UA data is saved in the servers. Clients can whether read and write data as well as subscribing to changes on it (this will be further explained in the next section). Although it is a fixed data architecture, users can freely define how and how many components of the object they desire to use in order to control their industrial environment. Figure 2.12 exemplifies an industrial environment in which some OPC-UA Clients control two different scenarios:

- **Robotic arms:** There is an OPC-UA Server installed in a PLC that is in charge of controlling two robotic arms from different manufacturers (KUKA [37] and UR [38]). Each of them will be represented inside the server as an object that contains different variables, methods and events.

- Lighting and ventilation systems: The OPC-UA Server installed here will take care of two light bulbs called *Light 1* and *Light 2* besides controlling the ventilation system of the factory.

OPC-UA clients can access to the data reading and writing it regardless of the device's manufacturer. Here we can once again understand the main advantage of using this standard in a factory since that way it is possible to switch any time to other devices if your application or other factors require it.

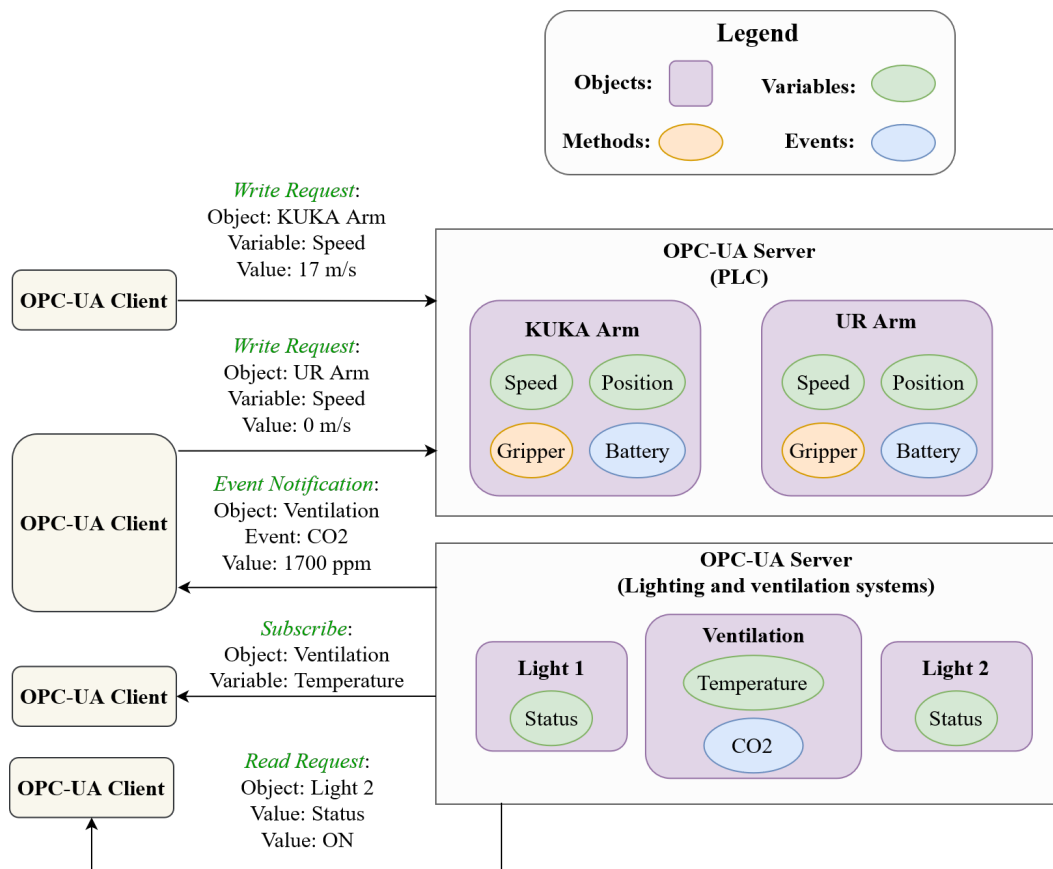


Figure 2.12: OPC-UA use case example.

2.2.2.- PubSub architecture

The other architecture offered by OPC-UA is known as PubSub model [36]. Here the OPC-UA clients subscribe to variables or events stored inside an OPC-UA object. The server notifies them every time the data has changed according a refreshing period specified by the clients. The clients don't have to ask for resources every time they want them as it happens in Client-Server, so this is a good model for applications in which several clients need to receive or access the same data. This leads to a reduction in the load on the server and the network.

The main differences between both OPC-UA communication models are:

- **Communication establishment:** in PubSub model the server starts the communication when it publishes data while in client-server model it is done by the client when it asks for resources.
- **Scalability:** Client-Server model is suitable for applications where there is a limited amount of clients while PubSub model fits uses cases with a large amount of clients.
- **Use of resources:** since PubSub model reduces the load on the server and network, resources are more optimized when using this architecture.

2.2.3.- OPC-UA over TCP

OPC-UA is a TCP-based technology [39]. Due to this, its connection establishment is based on a 3-way handshake in transport layer. Once this process is finished, OPC-UA connection starts with a “Hello” message used to specify the buffer sizes that the client supports. If the OPC-UA server accepts it, answers with an “Acknowledge” message completing the buffer negotiation. Thereafter, the client requests for a secure channel and a secure session are later created and activated. Once the OPC-UA connection establishment is done, data can be accessed from the client by requesting its reading or writing. Finally the connection is closed whenever the client requests. All these steps are illustrated in detail in Figure 2.13 where connection between Client 1 and Server follows a Client-Server model whereas connection between Client 2 and Server follows a PubSub model. Both TCP connection and OPC-UA connection establishment are the same but messages differ when it comes to OPC-UA data transmission and closing OPC-UA connection.

In the first case, the following messages are exchanged in a Client-Server model:

- **“Read Request”:** sent by an OPC-UA client to an OPC-UA server in order to request values of one or more variables.
- **“Read Response”:** sent by an OPC-UA server to an OPC-UA client in response to a Read Request message, containing the values of the requested variables.
- **“Write Request”:** sent by an OPC-UA client to an OPC-UA server in order to write one or more variables.
- **“Write Response”:** sent by an OPC-UA server to an OPC-UA client in response to a Write Request message, containing the status and result of the writing operation.

In a PubSub model, the following messages are exchanged:

- “Publish Request”: sent by an OPC-UA subscriber to an OPC-UA publisher in order to request new events or data.
- “Publish Response”: sent by an OPC-UA publisher to an OPC-UA subscriber in response to a Publish Request message, containing the data or information the subscriber asked for.

Finally, PubSub model exchanges four extra messages before closing the OPC-UA connection compared to Client-Server model:

- “Delete Monitored Items Request”: sent by an OPC-UA subscriber to an OPC-UA publisher in order to stop monitoring one or more items within a subscription.
- “Delete Monitored Items Response”: sent by an OPC-UA publisher to an OPC-UA subscriber in response to a Delete Monitored Items Request message.
- “Delete Subscriptions Request”: sent by an OPC-UA subscriber to an OPC-UA publisher in order to delete a given subscription.
- “Delete Subscriptions Response”: sent by an OPC-UA publisher to an OPC-UA subscriber in response to a Delete Subscriptions Response message, indicating the status of the unsubscribing process.

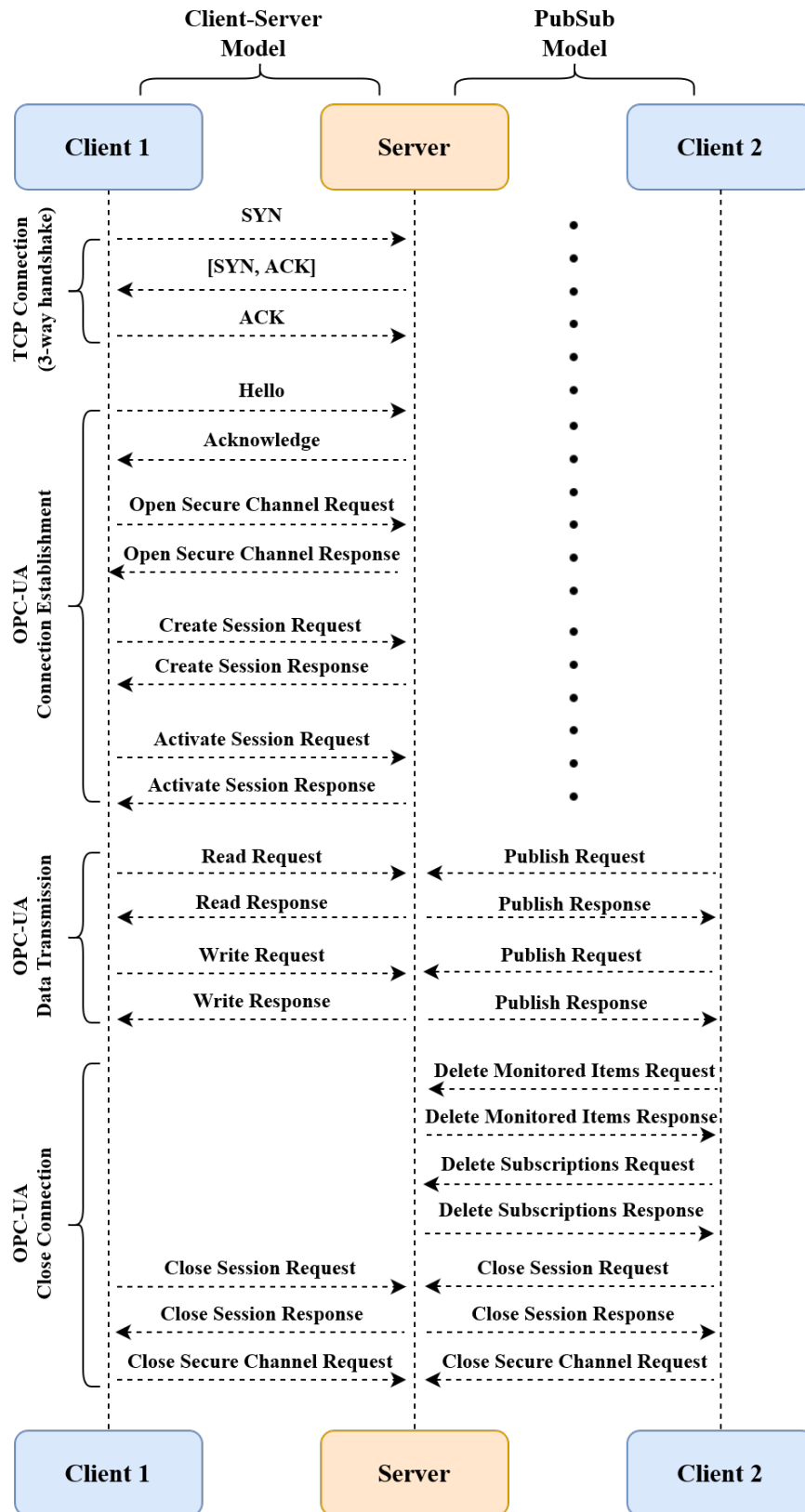


Figure 2.13: OPC-UA Connection establishment, data transmission and closing procedures.

2.3.- Performance evaluation of IIoT protocols over Ethernet

The first step for understanding how both protocols work and what is their performance, consists on the deployment of a testbed connected over wired Ethernet. The hardware used and their main technical specs are listed below:

- UP Squared Pro Board [40]: this 8 GB RAM and 64 GB of storage single-board computer is used to perform the roles of client and broker in the MQTT scenario and the roles of client and server in OPC-UA.
- Switch: the communication within the boards in the Local Area Network (LAN) is done by using a Zyxel GS-105B v2 switch [41]. This 5-port device can reach up to 2 Gigabit per second (Gbps) per port while offering low latency and error-free packet delivering.
- CAT 6 Ethernet cables: the UP Squared Boards are connected to the switch using CAT 6 Ethernet cables that provide around 1 Gbps of data transfer speed [4].

The specific implementation based on the described hardware for the different architectures to be evaluated is detailed in the next sections.

2.4.- Test KPIs, setups and configurations

The performance of both protocols will be studied based on three main variables: Closed Control Loop Latency (CCLL), link outage and effective payload size. For MQTT we will make an extra test in order to understand its broker processing time and how it affects latency.

2.4.1.- Closed Control Loop Latency

This test is used to understand how protocols behave in terms of latency. The main idea is to send a packet from “Node 1” to “Node 2” who will finally re-transmit it back to “Node 1”. Two timestamps will be calculated in Node’s 1 side: one just before sending the packet and another just after receiving it. It is important to remember that MQTT’s testbed has three boards (“Node 1”, broker and “Node 2”) whereas OPC-UA’s has only two. The architectural diagram of both testbeds is displayed in Figures 2.14 and 2.16, while a picture of the real setup is displayed in Figures 2.15 and 2.17, for MQTT and OPC-UA, respectively.

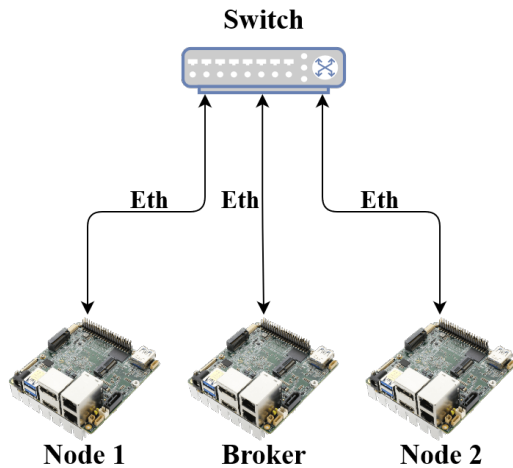


Figure 2.14: Ethernet testbed used for the study of MQTT.

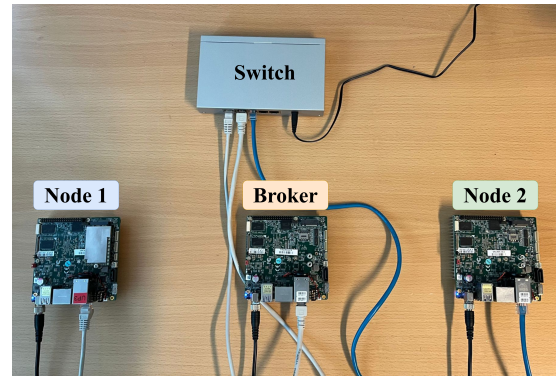


Figure 2.15: Real scenario of the Ethernet testbed used for the study of MQTT.

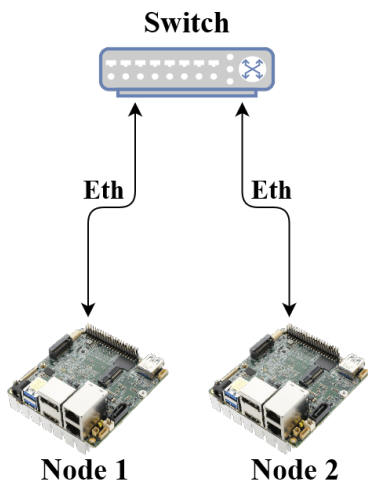


Figure 2.16: Ethernet testbed used for the study of OPC-UA.

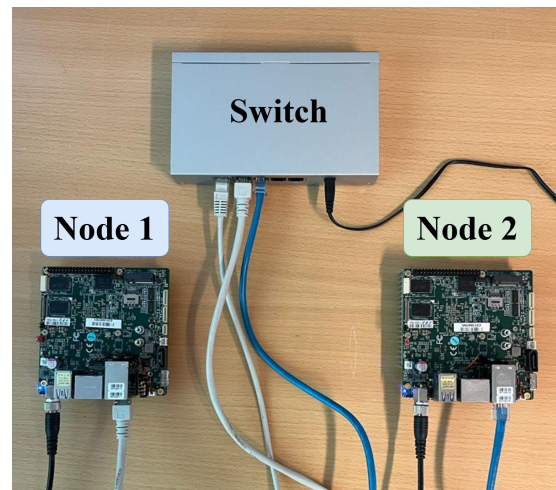


Figure 2.17: Real scenario of the Ethernet testbed used for the study of OPC-UA.

The following figures illustrate how the CCLL is measured within the different scenarios and protocols. Packets are depicted as envelopes with the following colour coding: blue for packets sent by “Node 1”, orange for packets sent by the MQTT broker and green for packets sent by “Node 2”.

Figure 2.18 shows the path followed by the data at the time of measuring the CCLL in a MQTT scenario. In the “STEP1”, “Node 1” measures “Timestamp 1” and publishes data to the topic “request”. “STEP 2” happens just after the broker receives the packet from “Node 1” and sends it to “Node 2”, who is subscribed to that topic. Once “Node 2” receives data from the broker, it retransmits it to the topic “response”, to which “Node 1” is subscribed

and resulting to “STEP 3”. Finally, the broker sends the packet received from “Node 2” to “Node 1”, which measures “Timestamp 2” after receiving the packet and closes the loop.

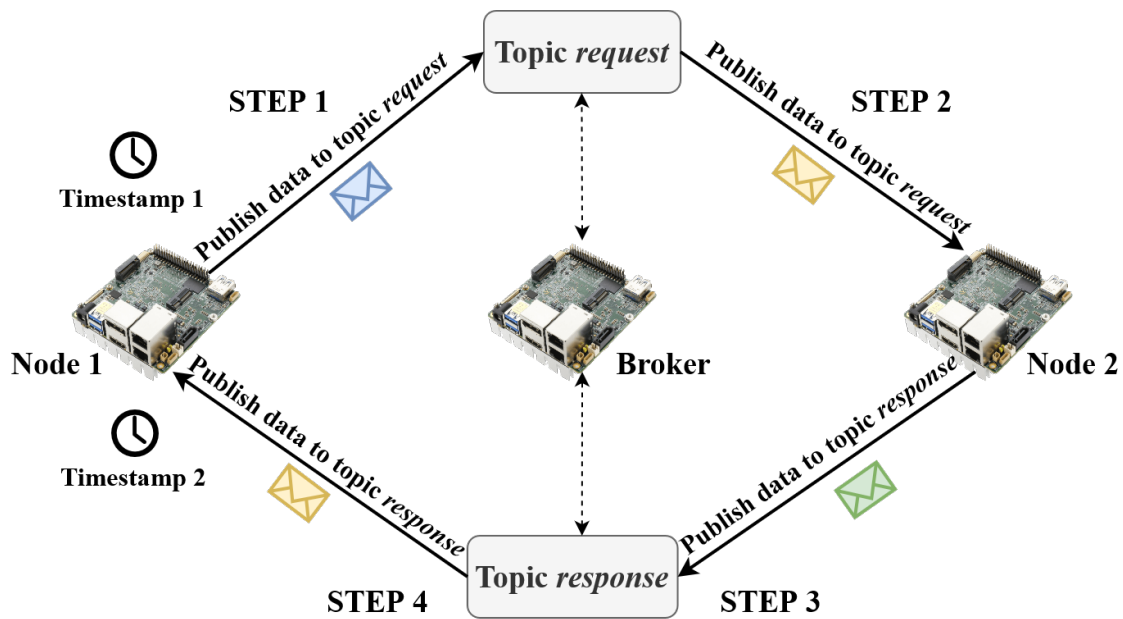


Figure 2.18: Steps of the CCLL for MQTT using the testbed.

When it comes to OPC-UA, the process is different depending on whether we are using Client-Server or PubSub architectures. In the first scenario, the way of computing CCLL is by following the steps shown in Figure 2.19. “Node 1” measures “Timestamp1” just before reading the variable stored in “Node 2”. Once it knows its value, the next step is to write a new value on it and right after that “Node 1” measures ‘Timestamp2’. This way, a closed loop packet can be generated since the packet goes from “Node 2” to “Node 1” during the reading process and from “Node 1” to “Node 2” during the writing.

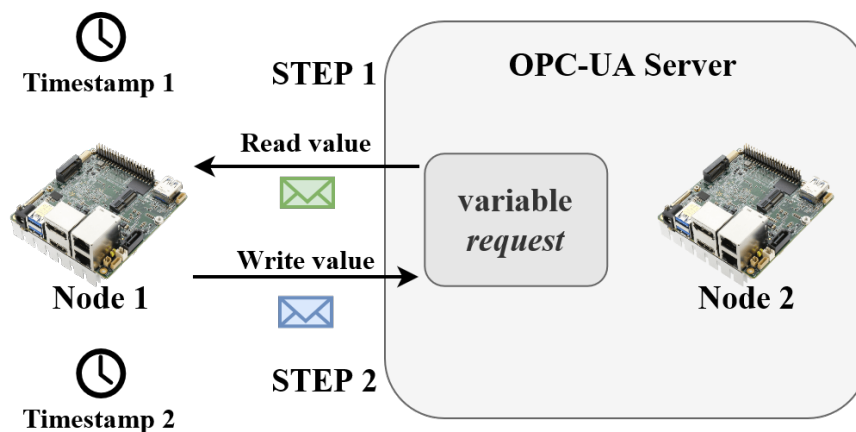


Figure 2.19: Steps of the CCLL for OPC-UA Client-Server using the testbed.

The computing of CCLL for a OPC-UA PubSub scenario depicted in Figure 2.20, follows the same idea proposed for MQTT but it differs in some details. Since this protocol has no broker, the closed loop starts on “Node 1” and ends on “Node 2”, without going through a third device. Since in OPC-UA’s address space there are not topics but variables, two of them will be created in “Node 2” since it will play the role of an OPC-UA’s Server. The aim is that both protocols are studied as similarly as possible, so the variables will be called “request” and “response”.

Instead of publishing and subscribing to topics, when it comes to OPC-UA PubSub, the nodes change and detect changes in variables. That way, “STEP 1” begins when “Node 1” measures ‘Timestamp 1’ and changes the value of the variable “request” inside “Node 2”s. Once the “Node 1” acts over ‘request’, a notification is triggered to “Node 2” who detects the change finishing “STEP2”. Just after that, “Node 2” changes the value of “response” within “STEP3”. Finally, the loop is closed on “Node 1” who detects this last change carried out by “Node 1” and finally measures “Timestamp 2”.

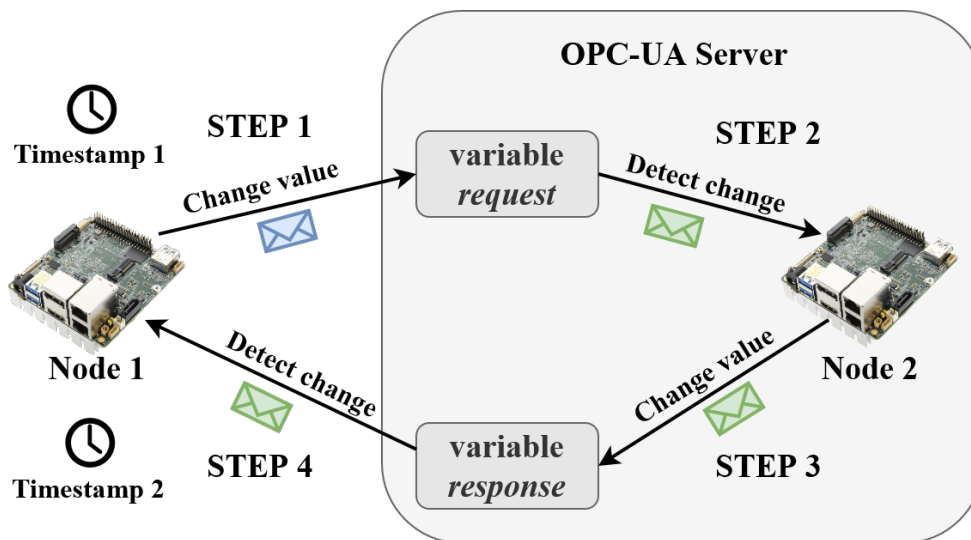


Figure 2.20: Steps of the CCLL for OPC-UA PubSub using the testbed.

For both MQTT and OPC-UA, its value is obtained by subtracting “Timestamp1” from “Timestamp2” as indicated in Equation 2.1 and according to the timestamp references in Figures 2.21, 2.22 and 2.23.

$$\text{CCLL [ms]} = \text{Timestamp2 [ms]} - \text{Timestamp1 [ms]} \quad (2.1)$$

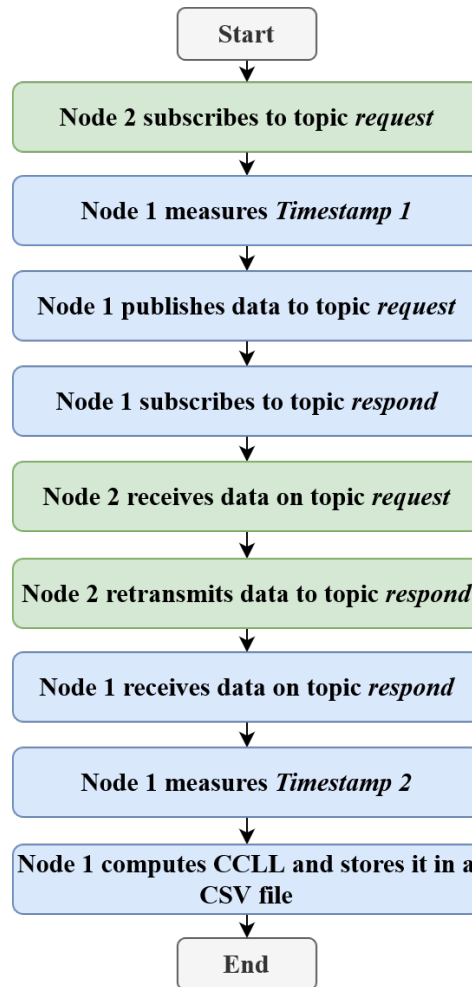


Figure 2.21: Flowchart of CCLL in MQTT.

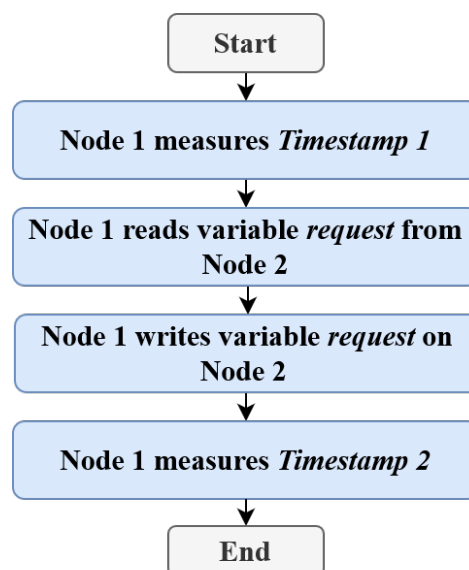


Figure 2.22: Flowchart of CCLL in OPC-UA Client-Server.

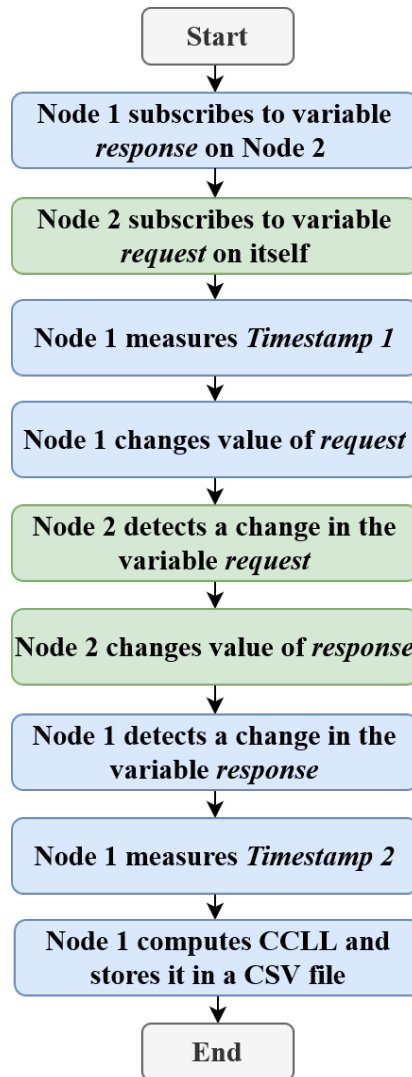


Figure 2.23: Flowchart of CCLL in OPC-UA PubSub.

Results of the CCLL will be studied by measuring it 500 times for different payload sizes and network load levels. 2 and 1300 bytes will be used in order to understand how both protocols behave when dealing with small and near Maximum Transmission Unit (MTU) packet sizes. Load on the network will be generated sending UDP packets from “Node 1” to “Node 2” and vice versa with an approximate bitrate of 0, 25, 50, 50, 75 and 100% of the total bandwidth of the link by using the open-source network testing tool iPerf3 [42]. This way it will be possible to test how the different IIoT communication protocols perform over variable network conditions and how different levels of external traffic affect CCLL.

From the 500 samples of CCLL we will be able to analyze the results statistically by using the Empirical Cumulative Distribution Function (ECDF). The formula of this function follows the expression written in Equation 2.2 where the indicator function $I(X_i \leq x)$ is equal to 1 if the i -th CCLL sample X_i is less or equal to x . The factor $\frac{1}{n}$ is used to normalize the sum of the indicator function values.

$$\hat{F}(x) = \frac{1}{n} \sum_{i=1}^n I(X_i \leq x) \quad (2.2)$$

Results will be based on the minimum, P50 or median, P90 or 90th percentile and maximum values so it will be possible to gain a complete understanding of the CCLL's samples range.

2.4.2.- Link outage

This Key Performance Indicator (KPI) is highly related to CCLL. We define link outage as the situation where a package would not be valid for a certain application based on a Maximum Tolerable Latency (MTL). This is a typical parameter used in industry as a malfunctioning threshold in operational machinery. In many cases, if the MTL is exceeded, the manufacturing process is stopped and an alarm is issued to capture the attention of the human operator [5]. For example, if our MTL is 10 ms, a packet with CCLL higher than that value would be seen as if it had never reached destination so its functionality would have never been performed.

MTL values of 10, 100, and 1000 milliseconds will be studied in order to develop this test since they are representative of MTL in industrial processes with different communication requirements. MTL level of 10 ms represents a very quick Input/Output (I/O) robotic link while the 1000 ms threshold represents the current MTL levels in general PLC-orchestrated production modules [5].

2.4.3.- Broker processing time in MQTT

In addition to the architecture of both protocols, the main difference between OPC-UA and MQTT is the existence of a broker in the latter one. Since it is an additional hop in the communication path, the broker will introduce extra latency to our application. It is our task to study how it behaves under different conditions in order to know and understand how much extra time it adds.

Two additional devices called “Node 3” and “Node 4” were deployed in our testbed as illustrated in Figure 2.24. They were set up using the same boards specified in Section 2.3. A picture of the real set up of this new testbed is exemplified in Figure 2.25.

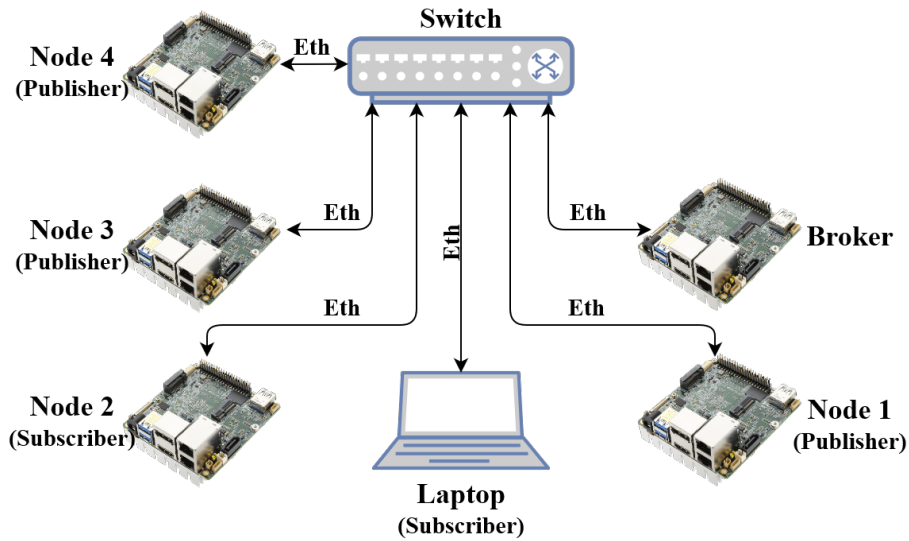


Figure 2.24: Testbed set up used during broker processing time study.

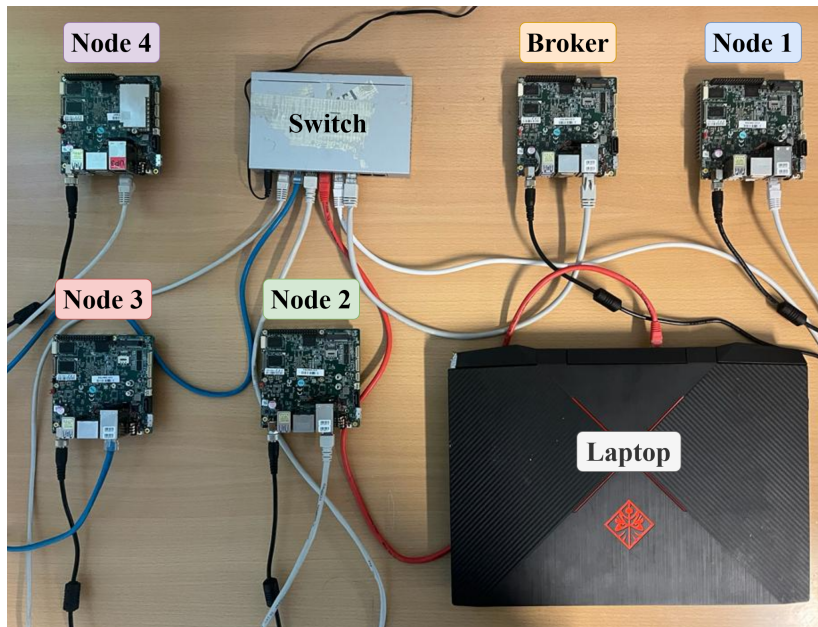


Figure 2.25: Real scenario of the Ethernet testbed used for the study of MQTT’s broker processing time.

Each of the new nodes runs 100 threads that publish a packet every 5 milliseconds to a laptop subscribed to both of them. This way, we are left with the following scenarios:

- Scenario 1: 1 publisher. “Node 1” publishes to “Node 2”.
- Scenario 2: 101 publishers. “Node 1” and 100 threads of “Node 3” publish to “Node 2”.
- Scenario 3: 201 publishers. “Node 1”, 100 threads of “Node 3” and 101 threads of “Node 4” publish to “Node 2”.

In the first scenario, the broker has to deal with 1 publisher. In the the second one it has to manage 101 publishers, and in the third one 201 publishers. This way we are covering the performance from one sensor network to a high density IoT scenario.

Another experiment that will be made in this test is the study of the time that elapses since the broker receives a message from a publisher and publishes it to a subscriber. This will be made by studying the different packets of the different MQTT QoS configurations when there is a data exchange between “Node 1” and “Node 2”. The scenarios in the list shown above are also considered in this analysis.

2.4.4.- Effective payload

A deep study of the packet structure of both protocols will be made by focusing on the comparison of their payload. Different scenarios will be considered in order to analyse how effective are the protocols at the time of storing data.

OPC-UA offers a complex architecture that is able to distinguish from different built-in data types by default. They are part of the OPC-UA information model and grant a standardized data-exchange between OPC-UA clients and servers. There are 25 built-in data types [43] but only Int64, String, Float and Boolean will be considered in our study.

On the other hand, MQTT primary deals with payload data as binary or plain text [24]. In the first case, its data is treated as a sequence of bytes used to represent any form of binary information. In the second case, data is treated as a set of characters that can be used to transmit commands, sensor readings, numerical values... In this last scenario, data is encoded using American Standard Code for Information Interchange (ASCII).

In order to understand the results presented in this chapter, a detailed extensive study can be found in Appendix B.

Messages exchanged by higher layer protocols such as OPC-UA and MQTT are encapsulated in frames. They are link-layer based and are transmitted over physical media like Ethernet cables. Assuming this, all upper-layer protocols (Internet Protocol (IP), TCP and MQTT for example) are encapsulated in an Ethernet frame. As shown in Figure 2.26, frame size will be the total byte value that WireShark captures in each packet transmitted through the network whereas MQTT/OPC-UA packet size will be the amount of bytes used on to transmit the actual data excluding any transport or network layer headers.

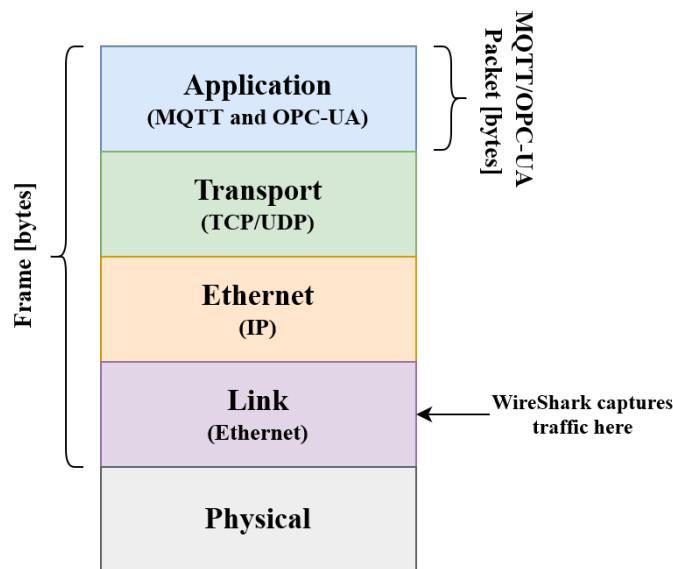


Figure 2.26: WireShark inside the TCP/IP model.

Payload size in both protocols will be compared using the following parameters:

- Bytes Reserved for Payload (BRP): this value indicates the number of bytes used in the application packet for storing the payload.
- Bytes Actually Used for Paylaod (BAUP): this value indicates the number of bytes from among the BRP used to store the payload and do not contain zeros.
- Payload Percentage (PP): percentage of the frame size reserved for the payload.

$$PP [\%] = \frac{BRP [\text{bytes}]}{\text{Frame size} [\text{bytes}]} \times 100 \quad (2.3)$$

- Effective Payload Percentage (EPP): percentage of BRP that are actually used to store the data.

$$EPP [\%] = \frac{BAUP [\text{bytes}]}{BRP [\text{bytes}]} \times 100 \quad (2.4)$$

2.5.- Performance results

This section summarizes the main statistical results of the CCLL and link outage performance tests for the different configurations of the MQTT and OPC-UA protocols in the cabled Ethernet testbed versions as well as the effective payload study and the MQTT broker management. For further details on the full statistics, all ECDF plots are given in Appendix D.

2.5.1.- Closed Control Loop Latency

As indicated in Table 2.2, QoS 0 exhibits a lower CCLL than other cases. This is particularly noticeable at P50 and higher percentile levels. For example, at P50, for 1300B, QoS 0 results in 3-4 ms, while QoS 1 and QoS 2 result in 7-8 and 23-27 ms, respectively. In general, the CCLL performance of MQTT seems stable with respect to network load. However, interestingly, the CCLL performance for 2B packet sizes is worse than the one at 1300B for QoS 1 and QoS 2. The exact cause of this effect will not be further studied, as it is not relevant for this project itself; but we speculate that it is probably due to the switch in the setup, which might be filtering and handling short MQTT packages with lower priority than larger ones.

		0 % load				50 % load				100 % load			
		Min	P50	P90	Max	Min	P50	P90	Max	Min	P50	P90	Max
2 B	QoS 0	3.32	3.77	3.91	8.33	2.17	3.02	3.27	6.88	1.67	2.46	2.78	5.64
	QoS 1	2.46	33.86	66.57	93.69	2.55	44.57	80.95	123.04	2.21	33.68	68.55	105.05
	QoS 2	10.57	53.47	91.72	130.34	6.66	59.39	91.62	131.55	8.98	38.90	56.22	134.88
1300 B	QoS 0	3.58	3.97	4.13	8.36	2.34	3.21	3.63	6.15	2.36	3.10	3.45	7.01
	QoS 1	3.23	7.36	11.87	15.93	2.61	8.73	10.53	88.69	3.04	8.83	10.02	85.82
	QoS 2	9.18	26.56	42.41	133.98	5.55	24.16	39.01	86.68	7.10	22.63	34.79	89.07

Table 2.2: CCLL in MQTT for Testbed over Ethernet.

In MQTT-SN, the MQTT protocol version over UDP, the performance for both packet sizes is stable and similar for all network loads. CCLL values, which are summarized Table 2.3, are approximately 3 ms at median level, similar to those from QoS in MQTT over TCP; however, maximum values in MQTT-SN are approximate half from the ones in MQTT, due to fact that UDP transmissions does not implement any reliability mechanism, resulting in a contained latency at the risk of experiencing packet loss.

		0 % load				50 % load				100 % load			
		Min	P50	P90	Max	Min	P50	P90	Max	Min	P50	P90	Max
2 B	QoS 0	2.53	3.05	3.28	3.56	1.81	2.85	3.47	4.44	2.06	2.61	2.91	3.50
1300 B	QoS 0	2.67	3.47	3.86	4.18	2.27	3.14	3.84	4.72	2.45	3.02	3.24	5.60

Table 2.3: CCLL in MQTT-SN for Testbed over Ethernet.

In the case of OPC-UA, Table 2.4 shows that Client-Server architecture offers higher CCLL values for both payload sizes with P50 values in the range 11.5-12.5 ms and P90 values around 12-13.5 ms. Results obtained using the PubSub architecture are worse than for the Client-Server case as the configured scheduling/transmission period increases. For PubSub scenario of 1 ms the performance for both 2B and 1300B packet sizes at the P50 and P90 values are in the range of 6-8 ms and 7-8 ms, respectively. Incrementing the PubSub period to 10 and 50 ms, results in increased CCLL latency as compared to the 1 ms period. CCLL is 9 ms higher at P50 and 10-15 ms higher at P90 for the 10 ms period with respect to the 1 ms one. This increases further in 30-40 ms at P50 and 50-60 ms for the 50 ms period. The effect of the different payload size is small for all PubSub configured refreshing periods. Although low CCLL values are expected for our IIoT applications it is important to find a seek a balance at the time of setting the OPC-UA PubSub refreshing period since small values overload the network and over time lead to situations that are the opposite of those originally intended. For a given OPC-UA PubSub period and payload size, CCLL values are very similar for all network load levels.

		0 % load				50 % load				100 % load			
		Min	P50	P90	Max	Min	P50	P90	Max	Min	P50	P90	Max
2 B	Client Server	9.47	11.82	11.96	14.45	8.97	11.35	11.69	14.20	8.86	12.28	12.56	15.23
	PubSub 1ms	6.14	7.63	8.04	14.12	6.18	7.33	8.10	9.72	5.62	6.19	6.85	7.96
	PubSub 10ms	9.75	14.43	18.51	21.12	8.99	15.19	19.34	21.11	6.48	10.90	15.25	17.98
	PubSub 50ms	30.65	55.49	71.93	83.04	28.35	54	72.08	82.45	9.02	34.17	54.94	60.79
1300 B	Client Server	9.49	12.16	12.32	15.26	9.53	11.48	12.09	18.81	8.84	12.51	13.52	18.01
	PubSub 1ms	6.76	7.89	8.31	13.25	6.26	7.22	7.92	11.58	5.85	6.34	6.89	8.94
	PubSub 10ms	8.53	17.44	24.09	26.95	8.92	14.65	18.67	20.85	11.47	16.57	21.03	23.04
	PubSub 50ms	41.68	68.31	87.37	95.03	39.11	66.21	84.99	91.26	42.39	68.40	87.47	93.88

Table 2.4: CCLL in OPC-UA for Testbed over Ethernet.

Based on all collected results, the following summary observations can be made:

- **MQTT-SN offers the best CCLL performance, similar to that of MQTT QoS 0, at the price of reduced reliability reliability.**
- **MQTT QoS 0 offers better CCLL performance than QoS 1 and QoS 2.**
- **MQTT performs better for large packets than for small packets.**
- **OPC-UA PubSub CCLL performance depends heavily on the configured PubSub refreshing period. In general, the higher the period, the higher the experienced CCLL.**
- **OPC-UA Client-Server CCLL performance for both packet sizes is similar to OPC-UA PubSub with a refreshing period of 10 ms.**

2.5.2.- Link outage

The CCLL results described in the previous section are now put in perspective of the target MTL values. Table 2.5 gathers link outage results for MQTT. With QoS levels greater than 0, link outage becomes an issue for MTL values of 10 ms, both in 2 and 1300B. First payload size experiences around 88-94% of link outage when QoS 1 is used while bigger payloads lead to the half of it approximately (42-55%). If we apply QoS 2 level, 2B payload packets grow up to around 100% of link outage whereas 1300B do it up to 95% approximately. Moving to higher MTL values, link outage is negligible in all scenarios, with an exception in some isolated cases where it is less than 5%. Gathering all results, it seems that network load does not increase link outage whereas bigger payloads tend to show lower values. This makes sense in perspective of the results in Table 2.2.

		0 % load			50 % load			100 % load		
		10 ms	100 ms	1000 ms	10 ms	100 ms	1000 ms	10 ms	100 ms	1000 ms
2 B	QoS 0	0%	0%	0%	0%	0%	0%	0%	0%	0%
	QoS 1	93.8%	0%	0%	88.4%	0%	0%	92.2%	0%	0%
	QoS 2	99.6%	2.4%	0%	99.8%	2.4%	0%	100%	4%	0%
1300 B	QoS 0	0.4%	0%	0%	0.2%	0%	0%	0%	0%	0%
	QoS 1	47.4%	0%	0%	55.6%	0%	0%	42.2%	0%	0%
	QoS 2	93.8%	0%	0%	94.6%	0%	0%	96.4%	0%	0%

Table 2.5: Link outage in MQTT for Testbed over Ethernet.

Table 2.6 depicts link outage results for MQTT-SN. There is no scenario in which link outage exists. It is an expected result since there are not CCLL values greater than 10 ms (see Table 2.3). It is important to highlight that even though these could be seen as the best link outage results until now, there is no guarantee of reliability in this protocol.

		0 % load			50 % load			100 % load		
		10 ms	100 ms	1000 ms	10 ms	100 ms	1000 ms	10 ms	100 ms	1000 ms
2 B	QoS 0	0%	0%	0%	0%	0%	0%	0%	0%	0%
1300 B	QoS 0	0%	0%	0%	0%	0%	0%	0%	0%	0%

Table 2.6: Link outage in MQTT-SN for Testbed over Ethernet.

Table 2.7 illustrates link outage results for OPC-UA over Ethernet. Client-Server model offers around 100% for MTL of 10 ms both in small and big payload sizes. By moving to higher MTL values, it is possible not to suffer from link outage. In PubSub model, link outage starts to be noticeable only when high scheduling periods are used. Based on the entirety of the results, it is possible to assert that network load and payload size do not affect link outage.

		0 % load			50 % load			100 % load		
		10 ms	100 ms	1000 ms	10 ms	100 ms	1000 ms	10 ms	100 ms	1000 ms
2 B	Client Server	99.6 %	0 %	0 %	99.4 %	0 %	0 %	99.6 %	0 %	0 %
	PubSub 1ms	0.2 %	0 %	0 %	0 %	0 %	0 %	0 %	0 %	0 %
	PubSub 10ms	0.2 %	0 %	0 %	0 %	0 %	0 %	0 %	0 %	0 %
	PubSub 50ms	100 %	0 %	0 %	100 %	0 %	0 %	99 %	0 %	0 %
1300 B	Client Server	99.8 %	0 %	0 %	99.4 %	0 %	0 %	99 %	0 %	0 %
	PubSub 1ms	0.2 %	0 %	0 %	0.2 %	0 %	0 %	0 %	0 %	0 %
	PubSub 10ms	0.2 %	0 %	0 %	0 %	0 %	0 %	0 %	0 %	0 %
	PubSub 50ms	100 %	0 %	0 %	100 %	0 %	0 %	100 %	0 %	0 %

Table 2.7: Link outage in OPC-UA for Testbed over Ethernet.

In terms of link outage analysis, the following observations serve as a summary:

- **MQTT provides reliable support for MTLs of 1s and above for all QoS levels, independently of packet size and network load.**
- **MQTT provides reliable support for MTLs of 100 ms and above for QoS 0 and QoS 1, independently of packet size and network load.**
- **MQTT provides reliable support for MTLs of 10 ms and above for QoS 0, independently of packet size and network load.**
- **MQTT-SN can support MTLs of 10 ms and above at expenses of some potential packet loss.**
- **OPC-UA Client-Server and PubSub provide reliable support for MTLs of 100 ms and above independently of packet size and network load.**

2.5.3.- Broker processing time in MQTT

Table 2.8 shows statistics regarding the CCLL for the different MQTT QoS configurations with increasing number of publishers and scenarios. Results were obtained without any other extra load (apart from the one created by the multiple publishers themselves). For all QoS, MQTT CCLL appears to be independent on the number of the publishers. Further, the observed values are well aligned with those previously presented in Table 2.2. This means that broker capacity is not a limiting factor in MQTT, at least when simultaneously operating 201 or less publisher sessions.

MQTT Configuration	Scenario	Min	P50	P90	Max
QoS 0	1 publisher	2.86 ms	3.78 ms	4.13 ms	7.72 ms
	101 publishers	2.33 ms	3.17 ms	3.66 ms	4.34 ms
	201 publishers	2.24 ms	2.83 ms	2.29 ms	5.18 ms
QoS 1	1 publisher	2.57 ms	13.86 ms	44.7 ms	88.01 ms
	101 publishers	2.2 ms	14 ms	44.8 ms	88.9 ms
	201 publishers	2.47 ms	14.65 ms	44.3 ms	86.97 ms
QoS 2	1 publisher	9.18 ms	38.56 ms	78.32 ms	134.44 ms
	101 publishers	8 ms	45.45 ms	77.65 ms	132.3 ms
	201 publishers	13.16 ms	44.65 ms	68.46 ms	130.29 ms

Table 2.8: MQTT CCLL for different number of publishers.

Once we have studied how the number of publishers affects the CCLL, it is time to study the MQTT broker processing time in detail (elapsed time between the broker receiving a message from a publisher and transmitting it to the subscriber). As summarized in Table 2.9, the processing time is in the range 0.1-0.6 ms for all QoS levels.

MQTT Configuration	Min	P50	P90	Max
QoS 0	0.27 ms	0.33 ms	0.43 ms	0.51 ms
QoS 1	0.27 ms	0.39 ms	0.40 ms	0.55 ms
QoS 2	0.14 ms	0.33 ms	0.38 ms	0.58 ms

Table 2.9: Time elapsed in MQTT broker

It is possible to partially conclude the following:

- **MQTT’s broker processing time is stable and appears to be independent of the number of connections managed for any level of QoS.**
- **MQTT processing time was found to be below 0.6 ms for all tested configurations.**

2.5.4.- Effective payload

It is important to remember that OPC-UA has built-in data types so the protocol is able to distinguish between integer, boolean and string values (among others) before sending the data whereas MQTT sends all data as string. Thus, rows “Int (ASCII)” and “Float (ASCII)” of MQTT refer to the scenarios in which those data types are sent as text. In order to perform a better comparison and make a deeper study of both protocols, two algorithms were design and applied so it was able to send integer and floating data as hexadecimal values. Results obtained in these last two scenarios appear “Int (HEX)” and “Floating (HEX)” rows. The steps followed in order to perform this study and get the results shown in Table 2.10 are detailed explained in Section C.

Protocol	Data type	Payload	Frame size	Packet size	BRP	BAUP	EPP	PP	
MQTT	Int (ASCII)	123456789	84 bytes	18 bytes	9 bytes	9 bytes	100 %	10.71 %	
	Int (HEX)	123456789	79 bytes	13 bytes	4 bytes	4 bytes	100 %	5 %	
	Float (ASCII)	3.14159	83 bytes	16 bytes	7 bytes	7 bytes	100 %	8.43 %	
	Float (HEX)	3.14159	79 bytes	13 bytes	4 byte	4 byte	100 %	5 %	
	String	'x'		76 bytes	10 bytes	1 byte	1 byte	100 %	1.32 %
'Claudia'			82 bytes	16 bytes	7 byte	7 byte	100 %	8.54 %	
OPC-UA	Int64	Min value	181 bytes	97 bytes	8 bytes	8 bytes	100 %	4.42 %	
		Max value	181 bytes	97 bytes	8 bytes	8 bytes	100 %	4.42 %	
		123456789	181 bytes	97 bytes	8 bytes	4 bytes	50 %	2.21 %	
	Bool	True	174 bytes	90 bytes	1 byte	1 byte	100 %	0.57 %	
		False	174 bytes	90 bytes	1 byte	1 byte	100 %	0.57 %	
	String	'x'		178 bytes	95 bytes	1 byte	1 byte	100 %	0.56 %
		'Claudia'		184 bytes	100 bytes	7 bytes	7 bytes	100 %	3.8 %

Table 2.10: Payload size comparison.

In terms of frame size, OPC-UA considers around 129 % and 124 % larger sizes than MQTT for integer value payload of “123456789” (using HEX encoding) and string value “Claudia” respectively. If we go deeper and only focus on packet size for the same examples, MQTT’s packet is only around 21.5 % of frame size for the integer case whereas OPC-UA grows up to 54 %. In the string scenario, both results are 19.5 % and 54 % approximately. BRP values are the same in both protocols for those cases but if we use the default ASCII encoding for MQTT it grows from 4 to 9 bytes, which means that it needs more twice the space for sending the same payload. This shows that the idea of MQTT as an lightweight protocol seems to make sense and reflects the impact of implementing extra encoding solutions for MQTT in addition to the default one.

For both MQTT and OPC-UA, EPP is 100 %, except in the case of OPC-UA Int64 numeric payload which is only 50 %. This is due to the fact that OPC-UA assigns 8 bytes to store the value regardless of whether they are filled or not. That is why if a value different from the maximum or minimum, payload efficiency decreases because those reserved bytes that are not used will be stored as zeros. Finally, in terms of PP, MQTT tends to have higher values than OPC-UA because the proportion of payload size within the packet frame is bigger for that protocol.

Based on the analysis of the previous results, the following summary observations can be derived:

- **Frame and packet sizes in MQTT are considerably smaller at the time of transmitting the same payload.**
- **MQTT is more efficient when it comes to storing the payload as it offers higher PP values.**

- **OPC-UA offers more built-in data types than MQTT, which facilitates the treatment of data in transmission and reception. This comes at the cost of increased packet/frame size.**

2.6.- Discussion of results

Previous sections of this chapter gather a deep investigation and understanding of both protocols and their architectures in a wired scenario. This gives a partial answer to OBJ1.

In terms of implementation, the MQTT solution was found to be simpler to design and deploy due to the wide availability of online resources and the low complexity of the protocol architecture and the structure of its packets. Moving to its UDP version MQTT-SN, its deployment was just as easy but the library used to it was still under development so although this protocol stands out for having the best CCLL and link outage results, it is better to keep it as an extra and interesting case of study and not as a candidate for further and more complex scenarios due to the big amount of crashes, unknown errors and lack of reliability, something crucial in industrial applications that must be highly considered from the very first moment within this discipline. On the other hand, OPC-UA implementation was found to be the most difficult one starting from the complexity at the time of understanding how the whole OPC-UA architecture works.

Figures 2.27, 2.28 and 2.29 illustrate selected CCLL statistics, comparing relevant performance results exhibited by the different IIoT protocols over Ethernet considering large packet sizes (1300 B) and the highest level of network load (100%). Thus, these results can be interpreted as a performance boundary reference.

In terms of wired communication performance, Figure 2.27 depicts how both MQTT QoS 0 and MQTT-SN show a stable and similar trend with median values of approximately 3-4 ms. Half of the MQTT QoS 1 CCLL performance values follow a similar trend to QoS 0 with levels of approximately 5 ms, however, the other half exhibits values of 25 ms and higher. MQTT QoS 2 is the worst case in terms of CCLL, reaching values that can be larger than 50 ms.

As depicted in Figure 2.28, OPC-UA Client-Server and OPC-UA PubSub with 1 ms publishing rate behave both very stably. However, OPC-UA Client-Server presents 6 ms increased CCLL performance as compared to OPC-UA PubSub with 1 ms period. Increasing the PubSub scheduling period to 10 and 50 ms results in a less stable and slower CCLL performance, with values of up to 25 and 100 ms, respectively.

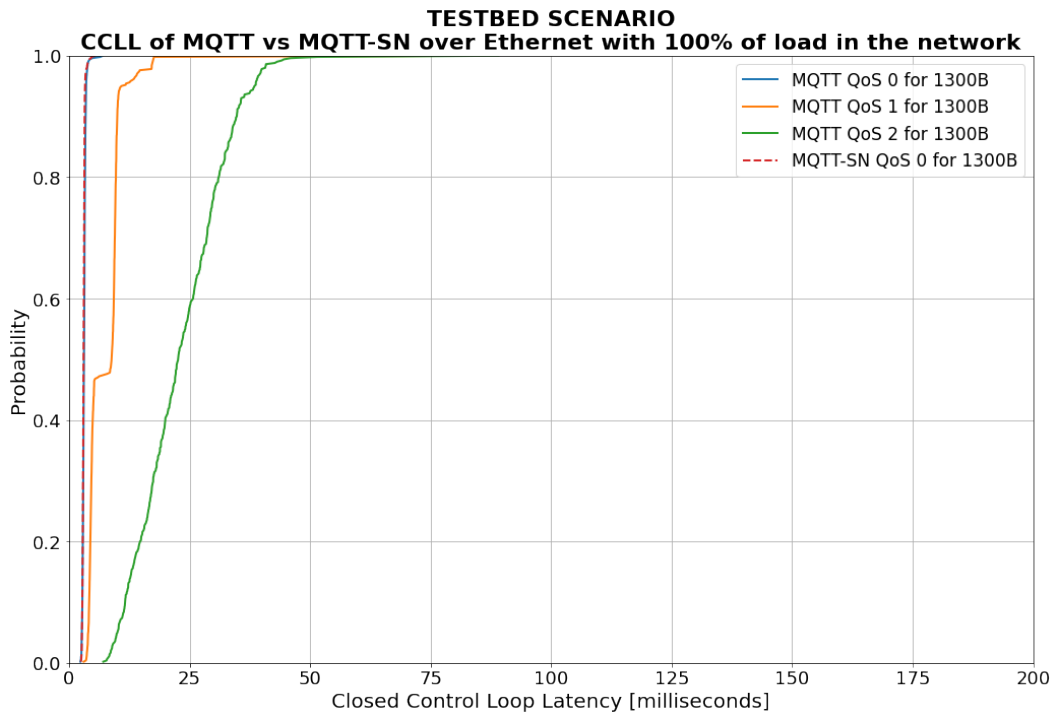


Figure 2.27: ECDF for MQTT vs MQTT-SN over Ethernet for packets with 1300 B payload and 100% of network load.

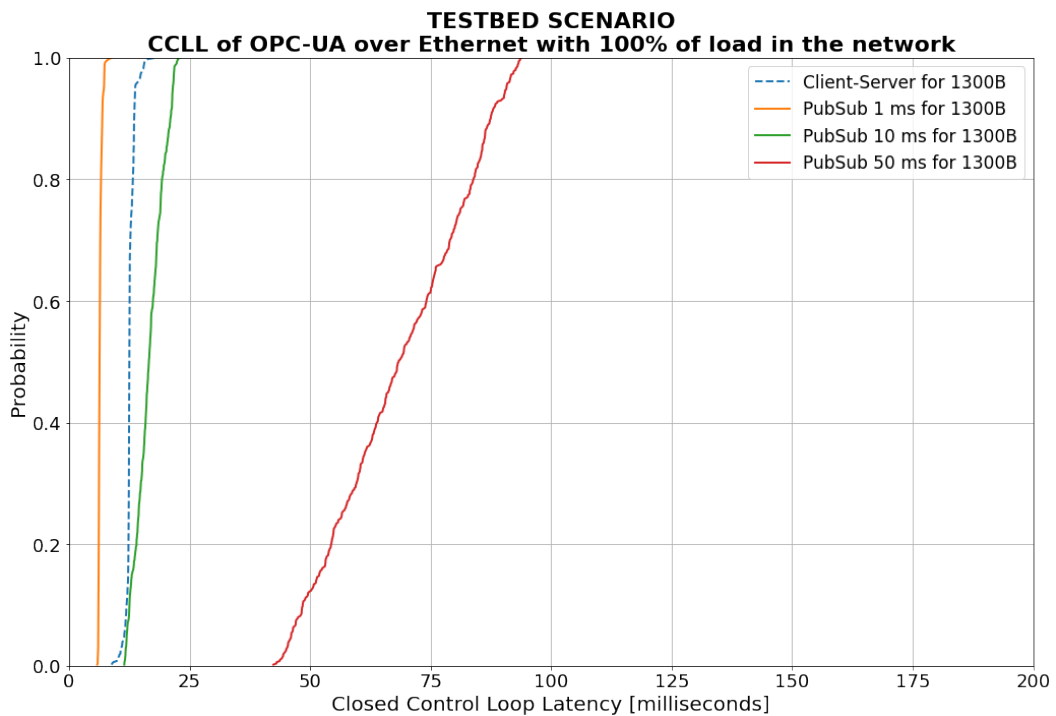


Figure 2.28: ECDF for OPC-UA Client-Server and OPC-UA PuBSub over Ethernet for packets with 1300 B payload and 100% of network load.

Finally, Figure 2.29 serves a final conclusion to this chapter with a comparison between MQTT and OPC-UA CCLL performance. In particular, MQTT QoS 0 and OPC-UA PubSub with scheduling rate of 10 ms have been chosen for benchmarking. The motivation for this choice will be further explained in details in Chapter 4, as it is related to the implementation of the protocols to control real robotic hardware, but for the moment they can just be taken as two representative configurations in the analysis. For the comparison, the following metrics are defined:

$$O_{MQTT} [\text{ms}] = P50_{MQTT-QoS0_{1300B}} [\text{ms}] - P50_{MQTT-QoS0_{2B}} [\text{ms}] \quad (2.5)$$

$$O_{OPC-UA} [\text{ms}] = P50_{OPC-UA-10ms_{1300B}} [\text{ms}] - P50_{OPC-UA-10ms_{2B}} [\text{ms}] \quad (2.6)$$

$$\Delta_{2B} [\text{ms}] = P50_{OPC-UA-10ms_{2B}} [\text{ms}] - P50_{MQTT-QoS0_{2B}} [\text{ms}] \quad (2.7)$$

$$\Delta_{1300B} [\text{ms}] = P50_{OPC-UA-10ms_{1300B}} [\text{ms}] - P50_{MQTT-QoS0_{1300B}} [\text{ms}] \quad (2.8)$$

O_{MQTT} and O_{OPC-UA} , both in ms, provide a quick insight on the operational CCLL performance range expected in the MQTT and OPC-UA selected configurations at P50 level, respectively. Δ_{2B} and Δ_{1300B} quantify the performance difference between MQTT QoS 0 and OPC-UA PubSub 10 ms at P50 level for 2B and 1300B payload, respectively.

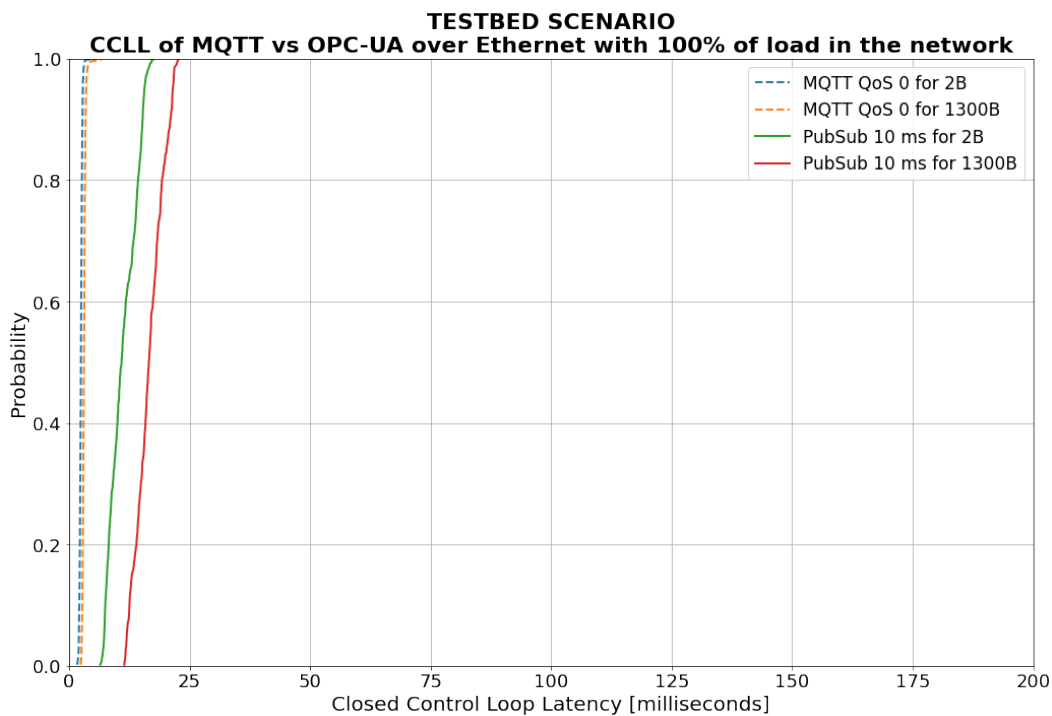


Figure 2.29: ECDF for MQTT vs OPC-UA over Ethernet for 100% of network load.

Table 2.11 collects the defined metrics for wired Ethernet performance. In summary, MQTT QoS 0 with 1300 B is 26 % slower than with 2B. For OPC-UA PubSub 10 ms, 2B payloads perform approximately 34 % faster than 1300 B. Comparing MQTT and OPC-UA, MQTT QoS 0 performs around 78 % and 81 % faster than OPC-UA 10 ms in terms of CCLL considering packets sizes of 2B and 1300B, respectively.

Metric	Value [ms]
O_{MQTT}	0.64
O_{OPC-UA}	5.67
Δ_{2B}	8.48
Δ_{1300B}	13.47

Table 2.11: Summary of performance indicators over Ethernet.

Thus, the following observation can be made:

- **MQTT CCLL performance is 80 % better than OPC-UA for the reference selected configurations.**

3. Ethernet to 5G system integration

Now that the IIoT control communication architectures and protocols have been analyzed, the next step considers integration of these with 5G technology in order to replace the main Ethernet wired connections, enabling the flexibility and reconfigurability explained in Section 1.4.1.

3.1.- 5G architectures

Two of the main components of a mobile network are the base stations and the core. 4G Long Term Evolution (LTE) base stations are known as Evolved Node B (eNB) whereas 5G ones are called Next-generation Node B (gNB). Evolved Packet Core (EPC) is the core of a LTE network. 5G Core (5GC) is the one corresponding to a 5G network [44].

Depending on how the above elements are combined, different 5G architectures can be achieved. The two main 5G deployments are known as 5G Non-Standalone (5G NSA) and 5G Standalone (5G SA) [44]. The main characteristic of 5G NSA is that the control plane occurs always over the 4G eNBs, while the data plane is steered over the 5G gNBs. It considers both eNB and gNB connected to EPC. With 5G NSA, network operators are able to provide the benefits of 5G technology such as Software Defined Networks (SDN), Network Function Virtualization (NFV) and network slicing. On the other hand, in 5G SA, only gNBs are connected to a 5GC. In this case both the control and data plane are over the 5G gNB interfaces. This way, operators can free themselves from legacy LTE components and install 5G base stations in areas without LTE coverage.

Figure 3.1 shows a high-level architecture of the AAU 5G Smart Production Lab [45]. It offers both 5G NSA and 5G SA architectures and it can be segmented into three main parts [46]:

- 5G Network: this network is deployed in a 5G SA architecture composed by a gNB, private 5GC and an enterprise edge cloud.
- Wire Area Network (WAN): this second network is deployed in a 5G NSA configuration using Telenor's public core, located a few km from the lab. In this case, the edge cloud service can still be accessed to the core via Multiprotocol Label Switching (MPLS).

- Internet: is used to provide general internet access to the UE and edge cloud within the lab; but also to connect to the Telenor public core via an alternative IPsec connection different from the dedicated MPLS WAN tunnel.

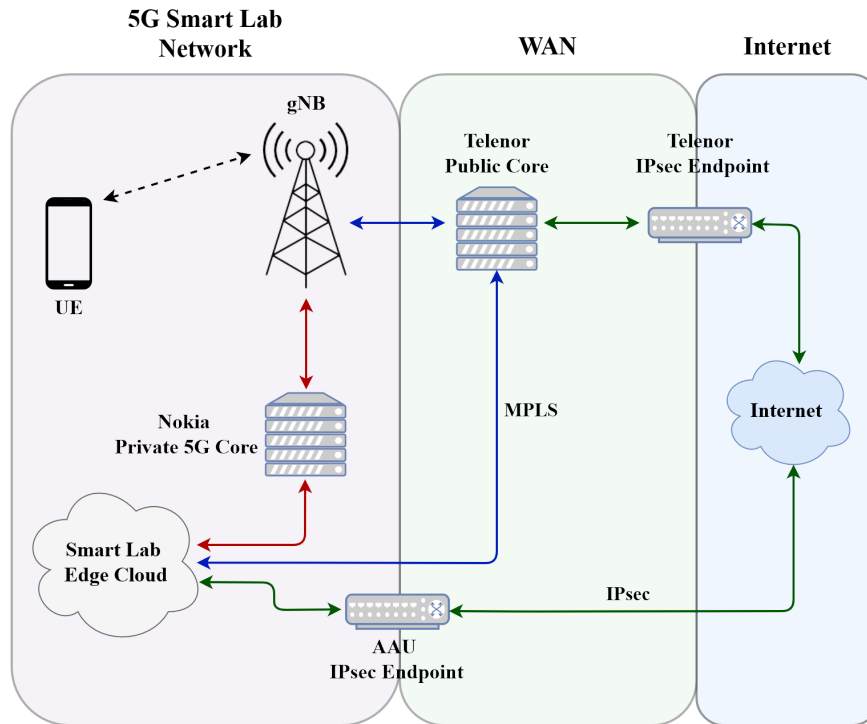


Figure 3.1: 5G architecture used in AAU 5G Smart Production Lab.

3.2.- 5G Box deployment

The network element used to integrate our devices within the 5G network is called 5G Box. It mainly consists of a Gateworks Newport GW6404 SBC [47] and a SIMCom SIM8262E-M2 modem [48]. The first one acts as the main computing unit of the 5G Box whereas the second one is allows the connection to the 5G network.

3.2.1.- Gateworks Newport GW6404 SBC

As explained above, this device is the “brain” of the 5G Box. Its tasks include managing software tools and controlling the modules connected to it. Its main technical specs include:

- Quad Core 1.5GHz ARM SoC.
- 2GB DDR4.
- 8GB eMMC Flash Memory.
- GPS and CAN support.

3.2.2.- SIM8262E-M2

This is a multi-mode wireless module that provides 5G/LTE/Evolved HSPA (HSPA+) connectivity. It is designed following 3GPP Release 16 and supports 5G NSA and 5G SA up to 3.4 Gbps of data transfer in Downlink (DL) and 1 Gbps in Uplink (UL). It also provides Global Navigation Satellite System (GNSS) technologies such as GPS, GLONASS, Galileo and BeiDou. Among its interfaces, it is worth mentioning Peripheral Component Interconnect Express (PCIe), USB 3.1 and General Purpose Input/Output (GPIO).

3.2.3.- Hardware set up

For the complete set up of the 5G Box, the following items will be needed:

- 1 Gateworks Newport GW6404 SBC boards.
- 4 5G/4G/LTE blade dipole antennas.
- 4 U.FL to SMA female connectors.
- 2 SIMCom SIM8262E-M2 modems.
- 2 USB 3.0 boards.
- 2 NOKIA SIM cards.

Figure 3.2 shows the hardware needed for the deployment one 5G Box. The whole process will be explained for the deployment of one device so in order to configure the others it would be enough to repeat it. The first step consists on mounting the SIM card in the “SIM1” slot of the USB 3.0 board as shown in Figure 3.3. Once it is done, we can connect the antennas to the U.FL ports of the SIM8262E-M2, connect the modem to the USB 3.0 board, and mount all on the USB3.0 slot of the Gateworks as illustrated in Figure 3.4.

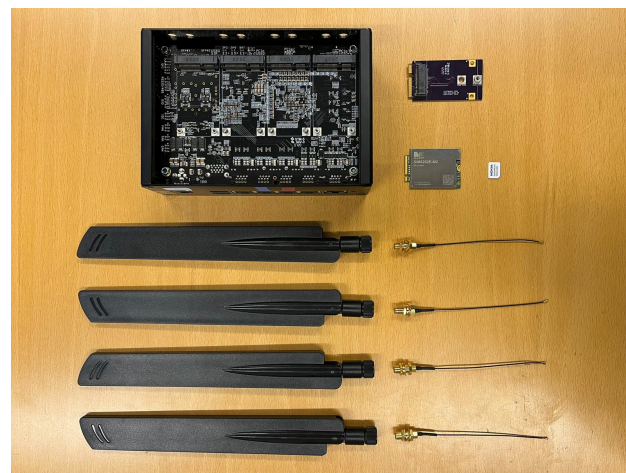


Figure 3.2: Hardware needed for setting up one 5G Box.

Finally we mount the antennas connecting them to the SMA connectors of the 5G Box.

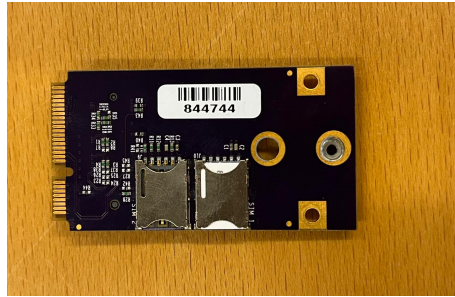


Figure 3.3: SIM card mounted on the USB3.0 board.

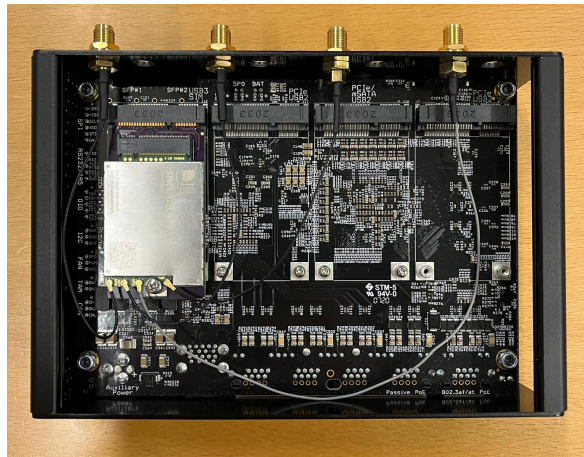


Figure 3.4: SimCom modem mounted on the Gateworks's USB 3.0 port.

The completed setup is shown in Figure 3.5.



Figure 3.5: Final set up of the 5G Box

Figure 3.6 illustrates how the 5G Box is integrated in the AAU 5G Smart Production Lab environment. Since it will be the boundary between the LAN where “Node 2” will be deployed and the 5G network. Edge cloud will be in its own network so in order to achieve communication between “Node 2” and “Node 1” it will be necessary to set up some Network Address Translation (NAT) rules as explained in next section.

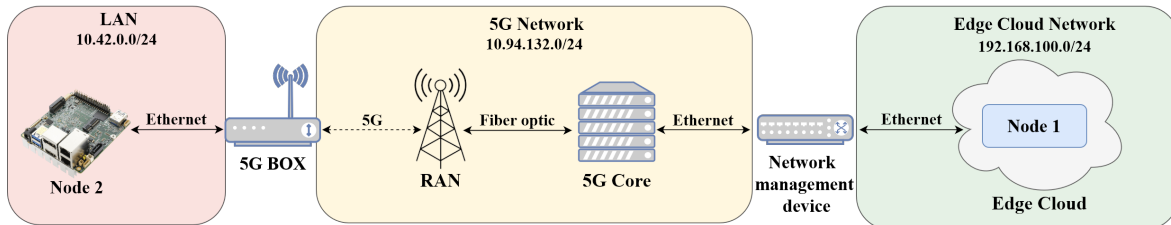


Figure 3.6: 5G Box integration within AAU 5G Smart Production Lab network.

The enterprise edge-cloud implemented in the AAU 5G Smart Production Lab consists on 12 servers [17]. Although it is possible to access the internet from inside the lab, it won't be necessary for our application. We will only need to go until the edge of the network. One of the servers will be used to install a Linux-based virtual machine in which “Node 1” and MQTT broker will be deployed. The technical specs of the server are the following:

- 56 CPU Logical cores.
- 125 GiB of RAM.
- 932 GiB of storage.

From the resources listed above, the virtual machine will use 32 CPU logical cores, 60 GiB of RAM and 128 GiB of storage.

3.2.4.- Software set up

The modem's firmware is Linux based so its setup will be made using a Command Line Interface (CLI). In order to access it, it is necessary to power on the modem and connect the laptop to the blue Ethernet port (Figure 3.5) so it is possible to check if we are getting an IP in the subnet 10.42.0.0/24. Using PuTTY [49], we will connect to the modem via 10.42.0.1 (it is the gateway's IP) Once we are logged in the modem, we have to create a new 5G *NetworkManager* profile so we can access the network. The command is specified below:

```
1 $ nmcli c add type gsm ifname cdc-wdm0 con-name nsa-aau5g-dave apn
2 internet
```

There we find the following parameters:

- “type”: we specify the connection as Global System for Mobile Communications (GSM).
- “ifname”: the name of the network interface used for the communication. In our case, *cdc-wdm0* is a generic USB communication interface used for mobile broadband devices, such as modems and cellular routers.
- “con-name”: name of the connection profile.
- “apn”: used for specifying the Access Point Name (APN).

With the connection successfully created, we can activate it by running the following command:

```
1 $ nmcli c up sa-aau5g-dave
```

It is important to know that since the “Node 2” is in the modem’s LAN, there is a need of using NAT to communicate with the “Node 1” because this last one is in the edge-Cloud and without configuring it there is no way both of them know each other as depicted in Figure 3.6.

The next step is to connect “Node 2” to the red Ethernet port of the 5G modem and check which IP is given. In our case, we are getting 10.42.0.233. Once we know “Node’s 2” IP, it is time to create the Destination NAT (DNAT) rules. They will be written inside FireHOL’s configuration file. FireHOL [50] is a Linux tool that provides an easy-to-use interface for configuring and managing firewall rules. It offers a simple syntax that translates the rules and policies into “iptables” or “nftables” commands

We need these rules because since we will be launching an iPerf3 server to study both protocols performance over loaded network links and an OPC-UA server to study the protocol itself, we would not be able to access “Node 2” without redirecting the traffic. There is no need of writing a rule for MQTT because the broker is deployed in the Edge cloud side. This means that when “Node 2” sends the request to the MQTT broker, the connection is set so DNAT is not needed later in order to send messages to “Node 2” from outside its network.

In order to do edit the configuration file, we have to run the following command:

```
1 $ nano /etc/firehol/firehol.conf
```

Then the following configuration needs to be applied:

```
1 dnat4 to 10.42.0.233:4840 dst 10.94.132.8 dport 4840
2 dnat4 to 10.42.0.233:22 dst 10.94.132.8 dport 22
3
```

```
4 interface eth2 config_iface
5     client all accept
6     server all accept
7
8 interface wwan0 5g
9     client all accept
10    server all accept
11
12 router eth-to-5g inface eth2 outface wwan0
13     masquerade
14     client all accept
15     server all accept
16
17 router 5g-to-eth inface wwan0 outface eth2
18     masquerade
19     client all accept
20     server all accept
```

Each of the first two lines specify a different DNAT rule for the “Node 2”. The first one is used for the OPC-UA protocol which will listen on the TCP port 4840 and the second one is dedicated to the Secure Socket Shell (SSH) protocol which uses its known TCP port 22.

Below the TCP rules, the network interfaces are defined. The “eth2” is the one that joins “Node 2” and the 5G modem. ”wwan0” is the wireless interface used to connect the 5G modem to the 5G core.

The last two sections are used to define the routing rules for the traffic that flows between the interfaces. There is one for the data going from “eth2” to “wwan0” and vice versa. With the “masquerade” option we enable NAT.

Once all this is done, we will find ourselves in the situation where OPC-UA packets coming from the edge-loud won’t reach the server in “Node 2” even though we have just wrote a DNAT rule to solve it. This happen because FireHOL and Network Manager use different firewall configurations that leads to a conflict between their iptables. FireHOL uses the “legacy” version of iptables whereas Network Manager uses the “nft” version. When executing Firehol configuration, it can’t clear the rules generated by Network Manager so it won’t work. There are two possible ways of solving this issue.

The first option is to set the iptables to “nft” version in the Gateworks as follows:

```
1 $ sudo update-alternatives --set iptables /usr/sbin/iptables-nft
2 $ sudo update-alternatives --set ip6tables /usr/sbin/ip6tables-nft
3 $ update-alternatives --set arptables /usr/sbin/arptables-nft
```



```
4 $ sudo update-alternatives --set ebtables /usr/sbin/ebtables-nft
```

Once the iptables are set to the “nft” version, we need to create the following script called “delete_all_ip_tables_rules.sh”:

```
1 for table in $(cat /proc/net/ip_tables_names)
2 do
3     iptables -t $table -F
4     iptables -t $table -X
5 done
```

The first command inside the loop flushes all the rules in the iptables of the system and the second one deletes any custom rules. It is important to reboot the system before running the script. Once it is rebooted, we can execute it. By running the script, all the iptables rules on the system will be cleared. From now on, we can use OPC-UA since our iptables are reset and running only one configuration.

The second way of solving the iptables conflict is by setting their version to “legacy”, rebooting the modem and finally running the command “sudo firehol try”. We can set the desired iptables version as follows:

```
1 $ sudo update-alternatives --set iptables /usr/sbin/iptables-legacy
2 $ sudo update-alternatives --set ip6tables /usr/sbin/ip6tables-legacy
3 $ update-alternatives --set arptables /usr/sbin/arptables-legacy
4 $ sudo update-alternatives --set ebtables /usr/sbin/ebtables-legacy
```

Using any of both options will lead to a working environment where the 5G Box is connected to the 5G network and ready to operate industrial traffic connections to and from the edge cloud.

3.3.- Performance evaluation of IIoT industrial protocols over 5G

Now that we know how to integrate devices into the 5G network and the edge-cloud, the same performance evaluation done for the cabled Ethernet case, can be repeated over 5G and edge cloud. Given that the broker processing time in MQTT and payload comparison of both protocols has already been done over Ethernet, (and the same results also apply to the 5G edge cloud case), only CCLL and link outage will be addressed in this case. Edge-cloud is now included in the scheme of both testbeds and it will play different roles depending on

the protocol so both testbed architectures for OPC-UA and MQTT will be different from the ones shown in Chapter 2.

For MQTT protocol, both the broker and “Node 1” will be running on the edge-cloud instead of being deployed in UP Squared Boards. Thus, “Node 1” representing a Cloudified PLC, will enjoy the advantages explained in Section 1.6. Running the broker on the Edge Cloud will provide scalability, mobility, and elasticity as explained in Section 1.4.1. “Node 2” will still be running on the same device used for the Ethernet performance study. Figure 3.7 details the testbed architecture for MQTT over 5G.

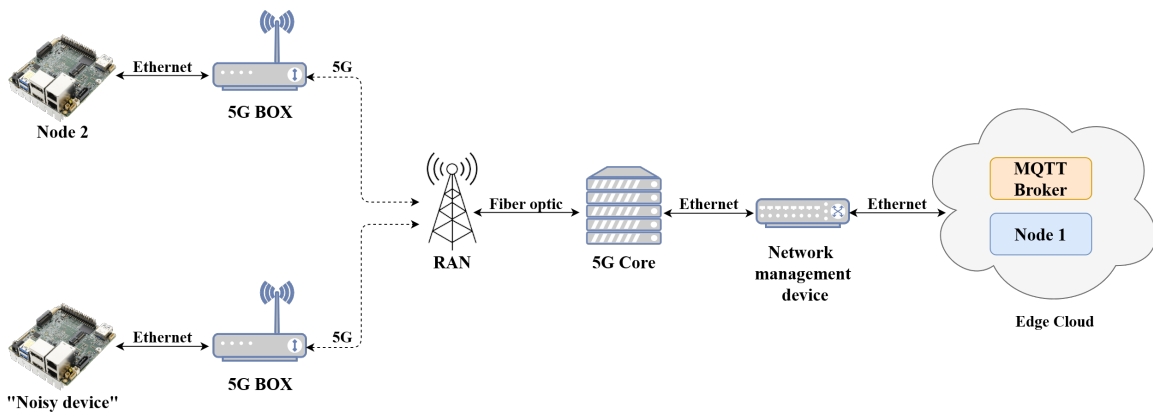


Figure 3.7: MQTT testbed architecture integrated with the 5G edge cloud.

Since OPC-UA is not broker-based, Node 1 will be the only process running on the Edge Cloud and will be subscribed to the OPC-UA server, which will still be deployed in “Node 1” board as shown in Figure 3.8.

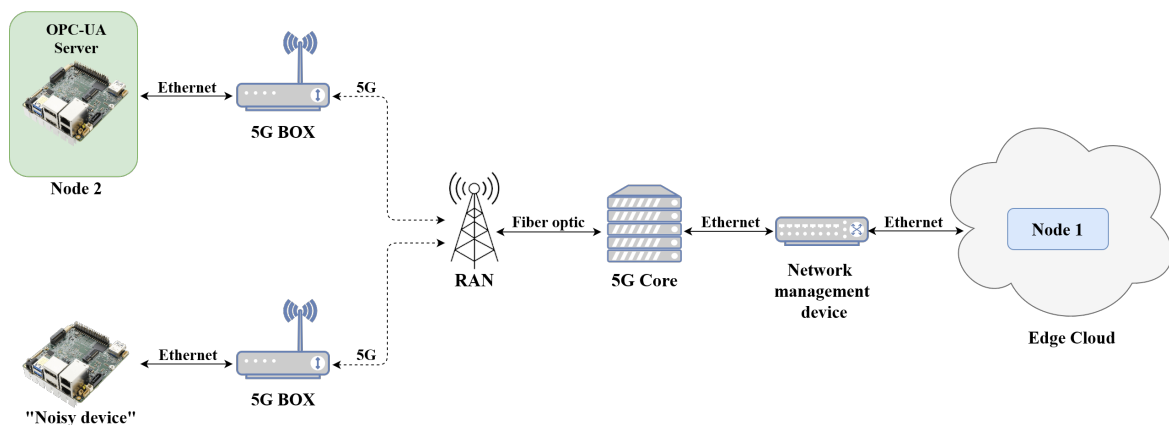


Figure 3.8: OPC-UA testbed architecture integrated with the 5G edge cloud.

In order to achieve load on the 5G network, an external device that introduces noisy traffic will be needed. The specifications of this component are the same as “Node 2”. The 5G Box used to connect it to the 5G network is the one discussed in Section 3.2.

The 5G network of the AAU 5G Smart Production Lab does not offer the same bandwidth over UL and DL. In order to develop these performance tests in terms of network load, the first step is to measure both channel bandwidths by using running the tool iPerf3 in the “noisy device” and edge-cloud. It was found that the UL channel offers 98.5 Mbps whereas the DL channel reaches 550 Mbps. Based on these values, the following main network load scenarios are proposed:

- 0 % load: UL and DL bandwidths will be fully used to perform the CCLL study.
- 50 % load: 49.25 Mbps out of 98.5 Mbps of the UL channel will be used for sending noisy traffic. 275 Mbps out of 550 Mbps of the DL channel will be used for the same purpose.
- 100 % load: the entire UL and DL bandwidths will be used for sending noisy traffic. This can be considered the worst scenario.

3.3.1.- Performance test configurations

Although we are now working on a different physical layer technology, CCLL will be measured on both protocols using the same payload sizes (2B and 1300B), network load levels and applying the same algorithms explained in Section 2.4.1. The main difference occurs at the “Node 1” level, which, as mentioned in the previous section, will be running on the edge cloud. Since 5G has asymmetrical bandwidths, the load on the network is triggered on two different links: an iPerf3 client is set on the “noisy device” and an iPerf3 server on the edge-cloud. We send UDP traffic from the client to the server loading the UL channel. In order to load the DL channel, an iPerf3 client is set on the edge cloud whereas the “noisy device” acts as an iPerf3 server receiving UDP traffic from the edge-cloud. This way both links are loaded simultaneously, and by changing the amount of traffic based on percentages explained in Section 3.3, we are able to achieve different network conditions.

3.4.- Performance results

3.4.1.- Closed Control Loop Latency

As depicted in Table 3.1, QoS 0 in a 2B scenario offers lower CCLL values than other cases. Its P50 and P90 are around 12 ms and 15 ms respectively. Keeping the same QoS configuration but increasing payload size up to 1300B leads to P50 and P90 values of 47 ms and 54-56 ms. Moving to QoS level 1 and 2B payloads, we face P50 and P90 values of 31

ms and 47-62 ms. Studying the same QoS with a bigger payload, shifts P50 and P90 values to 52-55 ms and 64-67 ms. Finally, analyzing QoS 2 with 2B of payload reflects P50 and P90 figures of 98-107 ms and 116-124 ms. Upgrading to a bigger payload size results in P50 and P90 values of around 133-138 ms and 157-165 ms. CCLL for MQTT over 5G is not load-dependant but it is payload-dependant since 1300B leads to higher CCLL results in all QoS configurations.

		0 % load				50 % load				100 % load			
		Min	P50	P90	Max	Min	P50	P90	Max	Min	P50	P90	Max
2 B	QoS 0	7.57	11.90	15.37	20.17	8.49	11.52	15.57	23.96	8.13	12.04	15.24	21.24
	QoS 1	13.49	31.52	61.99	92.59	12.58	31.07	46.95	92.16	9.43	30.99	61.34	122.56
	QoS 2	62.20	107	123.93	139.78	61.64	107	122.75	153.35	61.39	97.75	115.95	138.42
1300 B	QoS 0	20.72	47.29	55.68	77.65	24.39	46.63	54.44	75.25	30.89	46.54	54	95.57
	QoS 1	26.21	55.03	64.52	92.67	21.63	51.80	63.97	92.16	21.86	54.08	67.09	91.94
	QoS 2	77.97	138.36	157.47	186.17	77.39	133.23	164.62	199.81	76.99	137.88	158.33	199.36

Table 3.1: CCLL for Testbed over 5G using MQTT.

Table 3.2, illustrates CCLL results for MQTT-SN over 5G. Payloads of 2B lead to P50 and P90 values of around 13 ms and 16ms whereas payloads of 1300B causes P50 and P90 figures of 22-24 ms and 24-24.5 ms approximately. It takes around 5 ms longer to a 1300B payload packet to complete the closed loop compared to a 2B payload. The overall performance is stable and unaffected by the network load.

		0 % load				50 % load				100 % load			
		Min	P50	P90	Max	Min	P50	P90	Max	Min	P50	P90	Max
2 B	QoS 0	8.40	13.02	16.01	19.76	7.72	12.94	15.90	21.1	8.25	13.24	16.59	21.75
1300 B	QoS 0	13.86	23.95	27.43	36.04	11.29	23.77	28.35	35.65	13.20	22.01	24.61	28.17

Table 3.2: CCLL for Testbed over 5G using MQTT-SN.

Finally, the CCLL performance results for OPC-UA over 5G are gathered in Table 3.3. Client-Server model offers P50 of around 32 ms and P90 of 35-37 ms for 2B of payload. Increasing the payload to 1300B results in CCLL values 8-10 ms higher. Within PubSub model, there is a big variation between the different configurations. Starting from the lowest payload and scheduling period, we get P50 values of 21-23 ms and P90 values of 25 ms. Keeping that same period but increasing the payload size keeps us in similar ranges of CCLL. Differences start to be noticeable when we increase the scheduling period. For 10 ms and 2B of payload, P50 and P90 are now around 26 and 30 ms whereas for 1300B they are in the vicinity of 30 ms and 35-40 ms. It is not until the highest studied period when we find the biggest variation between PubSub scenarios. 2B and 50 ms of scheduling lead to 50-80 ms of P50 and 70-90 ms of P90 whereas 1300B increases those values up to 70-97 ms and 88-120 ms. As with most of the analyzed cases, OPC-UA CCLL performance is quite stable and the impact of the 5G network load is small. The wide variations suffered when using high scheduling periods such as 50 ms, could be due to the timing of the notification. When an OPC-UA client subscribes to a variable with such an update rate, there is a larger waiting

window possibility, which increases the probability of having delays closer to the configured PubSub period.

		0 % load				50 % load				100 % load			
		Min	P50	P90	Max	Min	P50	P90	Max	Min	P50	P90	Max
2 B	Client Server	25.01	31.95	35.19	46.98	24.80	32.22	36.60	41.94	25.97	32.47	36.57	45.40
	PubSub 1ms	15.10	22.73	25.24	29.35	14.83	20.58	25.03	30.59	15.15	22.85	25.29	32.99
	PubSub 10ms	20.22	25.73	30.52	38.14	17.61	25.21	28.29	32.60	18.20	25.35	28.64	33.44
1300 B	PubSub 50ms	22.73	50.04	70.71	83.20	55.22	80.13	100.80	110.39	43.57	72.17	90.59	98.02
	Client Server	29.90	42.06	47.91	63.03	29.83	40.17	45.04	52.20	27.14	41.56	45.10	51.95
	PubSub 1ms	15.18	23.40	28.33	35.62	14.98	25.29	28.51	35.52	17.49	25.16	28.94	33.68
	PubSub 10ms	22.80	30.37	35.38	43.45	22.99	33.33	38.13	43.56	22.77	30.41	39.59	48.14
	PubSub 50ms	35.46	68.29	88.32	98.68	68.31	95.16	117.77	125.28	65.13	97.74	120.01	128.55

Table 3.3: CCLL for Testbed over 5G using OPC-UA.

After analyzing all the CCLL data collected, the following summary observations can be highlighted:

- **MQTT-SN over 5G depends on payload size and performs better for small packets.**
- **MQTT QoS 0 performs better than QoS 1 and QoS 2 also over 5G.**
- **Differently from the Ethernet case, MQTT over 5G performs better for small packets than for large packets.**
- **Also over 5G, OPC-UA PubSub CCLL performance depends heavily on the configured PubSub refreshing period. In general, the higher the period, the higher the experienced CCLL.**

3.4.2.- Link outage

The collected data in Table 3.4 demonstrates that link outage presents a challenge for MQTT over 5G, specifically for MTL targets of 10 ms. This issue persists across all QoS configurations and payload sizes. If we move to a MTL of 100 ms, only QoS 2 becomes problematic and high payloads are more affected (98%) than small payloads (47-62%). Similar outage level values are achieved over 5G for all network load levels.

		0 % load			50 % load			100 % load		
		10 ms	100 ms	1000 ms	10 ms	100 ms	1000 ms	10 ms	100 ms	1000 ms
2 B	QoS 0	89.8 %	0 %	0 %	85.4 %	0 %	0 %	90.8 %	0 %	0 %
	QoS 1	100 %	0 %	0 %	100 %	0 %	0 %	99.8 %	0.4 %	0 %
	QoS 2	100 %	56.8 %	0 %	100 %	62 %	0 %	100 %	46.8 %	0 %
1300 B	QoS 0	100 %	0 %	0 %	100 %	0 %	0 %	100 %	0 %	0 %
	QoS 1	100 %	0 %	0 %	100 %	0 %	0 %	100 %	0 %	0 %
	QoS 2	100 %	98 %	0 %	100 %	98 %	0 %	100 %	98.8 %	0 %

Table 3.4: Link outage in MQTT for Testbed over 5G.

Table 3.5 depicts link outage results for MQTT-SN over 5G. Differently from the Ethernet case, outage levels are now observed over 5G for the MTL target of 10 ms for both payload sizes. This would add negatively to the fact that this protocol does not offer underlying reliability.

		0 % load			50 % load			100 % load		
		10 ms	100 ms	1000 ms	10 ms	100 ms	1000 ms	10 ms	100 ms	1000 ms
2 B	QoS 0	91.8 %	0 %	0 %	92.4 %	0 %	0 %	94.8 %	0 %	0 %
1300 B	QoS 0	100 %	0 %	0 %	100 %	0 %	0 %	100 %	0 %	0 %

Table 3.5: Link outage in MQTT-SN for Testbed over 5G.

The results for link outage in OPC-UA over 5G are presented in Table 3.6. In this case, it is noticeable the fact that there is full outage for MTL targets of 10 ms. There are some isolated cases for 50 ms of scheduling period in which it is possible to observe link outage with a MTL of 100 ms. This phenomenon can be related to the pseudo-stochastic behavior of CCLL on this kind of scenarios as explained in the previous section. Link outage for OPC-UA over 5G appears to be independent of network load and packet size.

		0 % load			50 % load			100 % load		
		10 ms	100 ms	1000 ms	10 ms	100 ms	1000 ms	10 ms	100 ms	1000 ms
2 B	Client Server	100 %	0 %	0 %	100 %	0 %	0 %	100 %	0 %	0 %
	PubSub 1ms	100 %	0 %	0 %	100 %	0 %	0 %	100 %	0 %	0 %
	PubSub 10ms	100 %	0 %	0 %	100 %	0 %	0 %	100 %	0 %	0 %
	PubSub 50ms	100 %	0 %	0 %	100 %	14 %	0 %	100 %	0 %	0 %
1300 B	Client Server	100 %	0 %	0 %	100 %	0 %	0 %	100 %	0 %	0 %
	PubSub 1ms	100 %	0 %	0 %	100 %	0 %	0 %	100 %	0 %	100 %
	PubSub 10ms	100 %	0 %	0 %	100 %	0 %	0 %	100 %	0 %	0 %
	PubSub 50ms	100 %	0 %	0 %	100 %	36.6 %	0 %	100 %	40.4 %	0 %

Table 3.6: Link outage in OPC-UA for Testbed over 5G.

To summarize the link outage analysis, the following observations can be made:

- **MQTT over 5G provides reliable support for MTLs of 1s and above for all QoS levels, independently of packet size and network load.**

- **MQTT over 5G can support MTLs of 100 ms and above only when using QoS 0 and QoS 1.**
- **MQTT-SN over 5G can support MTLs of 100 ms and above at expenses of some potential packet loss.**
- **OPC-UA provide reliable support for MTLs of 100 ms and above independently of packet size and network load for scheduling periods below 50 ms.**

3.5.- Discussion of results

The preceding sections of this chapter provide an in-depth exploration and comprehension of the MQTT and OPC-UA IIoT protocols their architectures, and performance within a 5G context. Along with the details presented in Chapter 2 for the wired Ethernet scenario, OBJ1 is completed.

Since the algorithms and libraries used for the performance tests were the same used in a wired setting, the conclusions about complexity of the deployment of the different solutions over 5G and edge-cloud is similar: MQTT and MQTT-SN were easier to implement than OPC-UA. Over 5G and edge cloud technologies higher latencies and link outages than over the wired Ethernet setup are experienced. Due to the asymmetrical bandwidth and how is the access to the medium, it is important to find a balance in the OPC-UA PubSub scheduling period. If set to low values (high scheduling rate), the 5G network could congest rapidly, whereas if set to high values (low publishing period) it will result in large CCLL values. This did not happen over wired Ethernet.

As in the previous chapter, selected CCLL statistics are presented as summary for the 5G scenarios, comparing relevant performance results of the different IIoT protocols. Figures 3.9 and 3.10 display the ECDFs of CCLL considering the different protocols, for large payload sizes (1300 B) and the highest level of network load (100%).

Figure 3.9 depicts how MQTT-SN exhibits a very stable performance with an approximately constant CCLL of 25 ms. Differently from the Ethernet case, the MQTT QoS 0 CCLL performance is now approximate double than for MQTT-SN. MQTT QoS 1 is close to that of QoS 0 for 30% of the cases, but in the other 70% it can reach values of up to approximately 75-100 ms. MQTT QoS 2 is the worst scenario, with very variable performance and CCLL values reaching close to 200 ms.

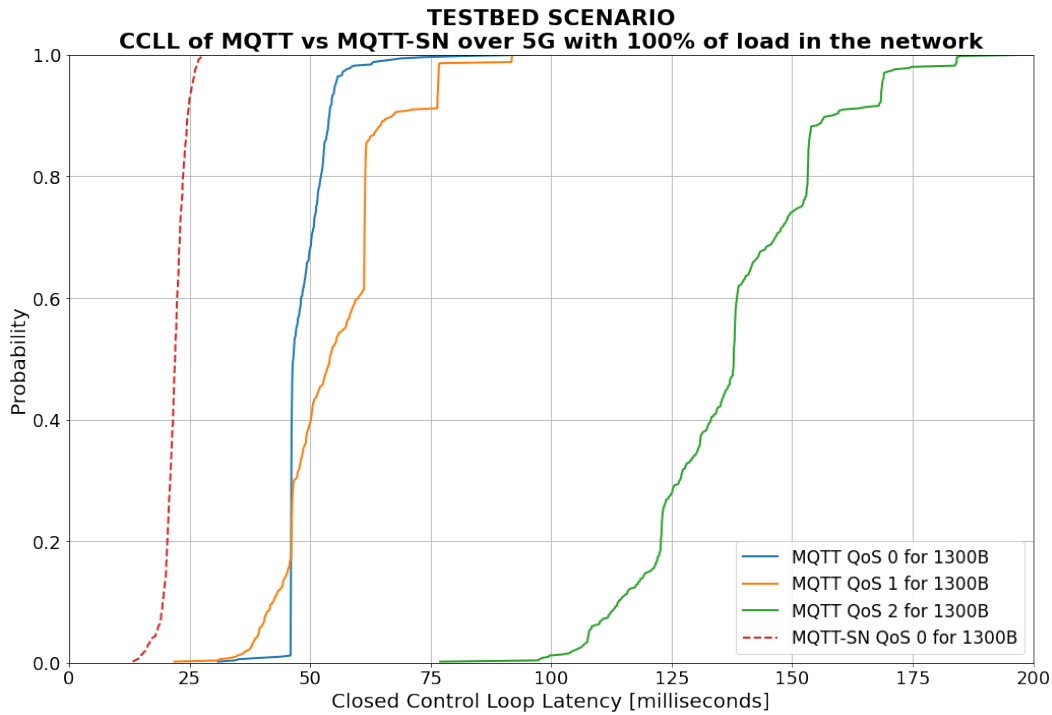


Figure 3.9: ECDF for MQTT vs MQTT-SN over 5G for packets with 1300 B payload and 100 % of network load.

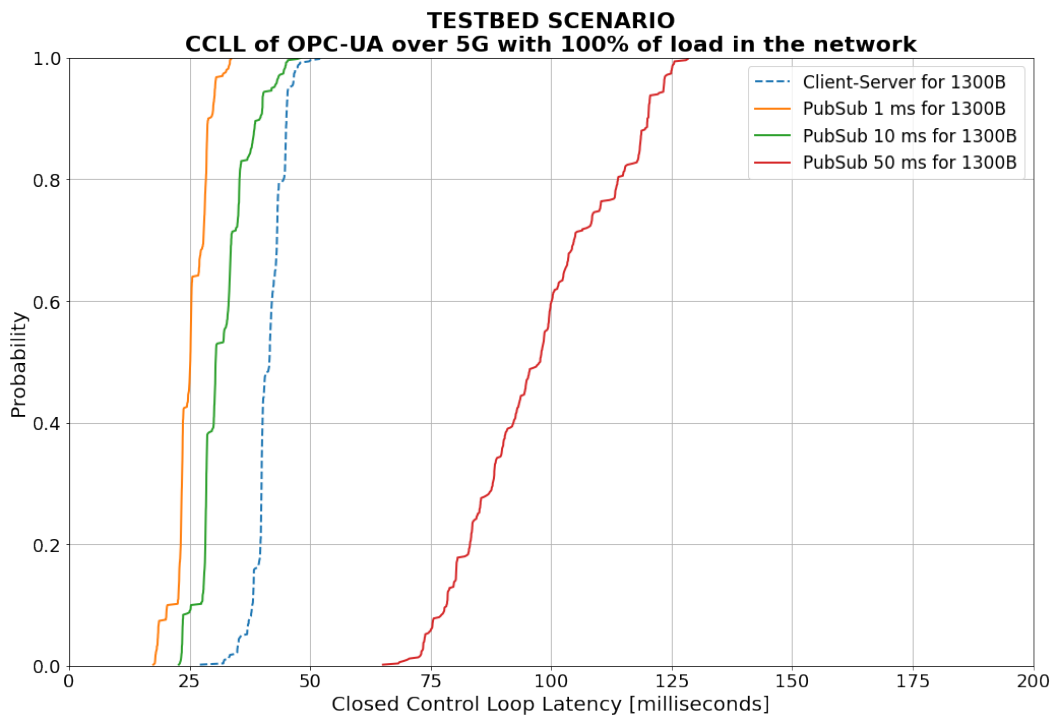


Figure 3.10: ECDF for OPC-UA over 5G for packets with 1300 B payload and 100 % of network load.

In Figure 3.10 it is noticeable how OPC-UA Client-Server and PubSub with scheduling periods 1 and 10 ms present very stable CCLL performance, being the best the PubSub 1 ms case with CCLL values of approximately 25 ms. The PubSub 10 ms case presents an increase of approximately 5 ms as compared to the PubSub 1 ms case. OPC-UA Client-Server is increased in further 10 ms as compared to the PubSub 10 ms case. OPC-UA PubSub with publishing rate 50 ms is far in terms of CCLL performance from the previous cases reaching values of up to 150 ms.

Figure 3.11 and Table 3.7 illustrate and summarize the comparison between MQTT and OPC-UA performance over 5G, based on the MQTT QoS 0 and OPC-UA PubSub with scheduling rate of 10 ms previously selected for reference benchmarking. For the 5G case in particular, MQTT QoS 0 with 1300 B is 286.5 % slower than with 2B. For OPC-UA PubSub 10 ms, 2B payloads perform 16.6 % faster than 1300 B. Comparing MQTT and OPC-UA, MQTT QoS 0 performs 52.5 % faster and 53 % slower than OPC-UA 10 ms in terms of CCLL considering packets sizes of 2B and 1300 B, respectively.

Metric	Value [ms]
O_{MQTT}	34.5
O_{OPC-UA}	5.06
Δ_{2B}	13.31
Δ_{1300B}	-16.13

Table 3.7: Summary of performance indicators over 5G

Based on the previous, the following observation can be made:

- **Over 5G, MQTT CCLL performance is 52.5 % better than OPC-UA for the reference selected configurations when payload size is 2B. With big payload sizes (1300B) it is 53 % slower.**

To conclude this chapter, a comparison of the CCLL performance of MQTT and OPC-UA over wired Ethernet and 5G is done. Figures 3.12 combines the information from Figure 2.29 (for the Ethernet) and 3.11 (for the 5G case). Similarly, Table 3.8 combines the information from Table 2.11 (for the Ethernet case) and Table 3.7 (for the 5G case). From the figure it is observed that, in general, the CCLL performance in the Ethernet cases is more stable than in the 5G case. It is also observed that 5G performance is well bounded below 50 ms, except in the case of MQTT QoS 0 ms with 1300 B, where CCLL values can reach up to 125 ms. This is 75-90 ms higher than for all other protocol configurations.

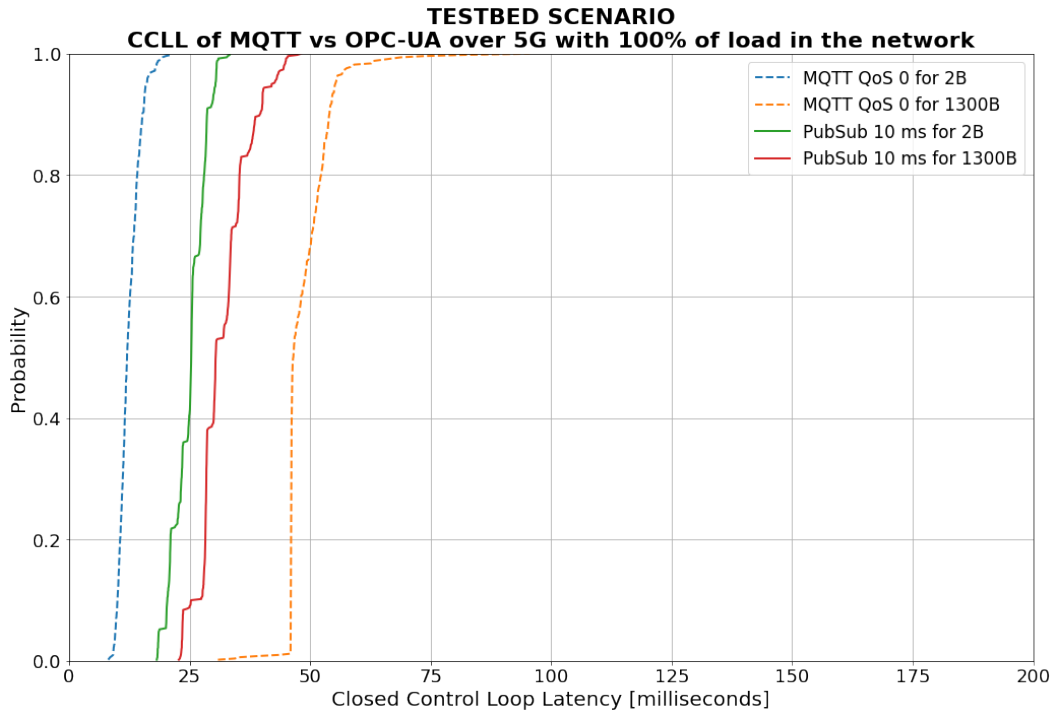


Figure 3.11: ECDF for MQTT vs OPC-UA over 5G for 100% of network load.

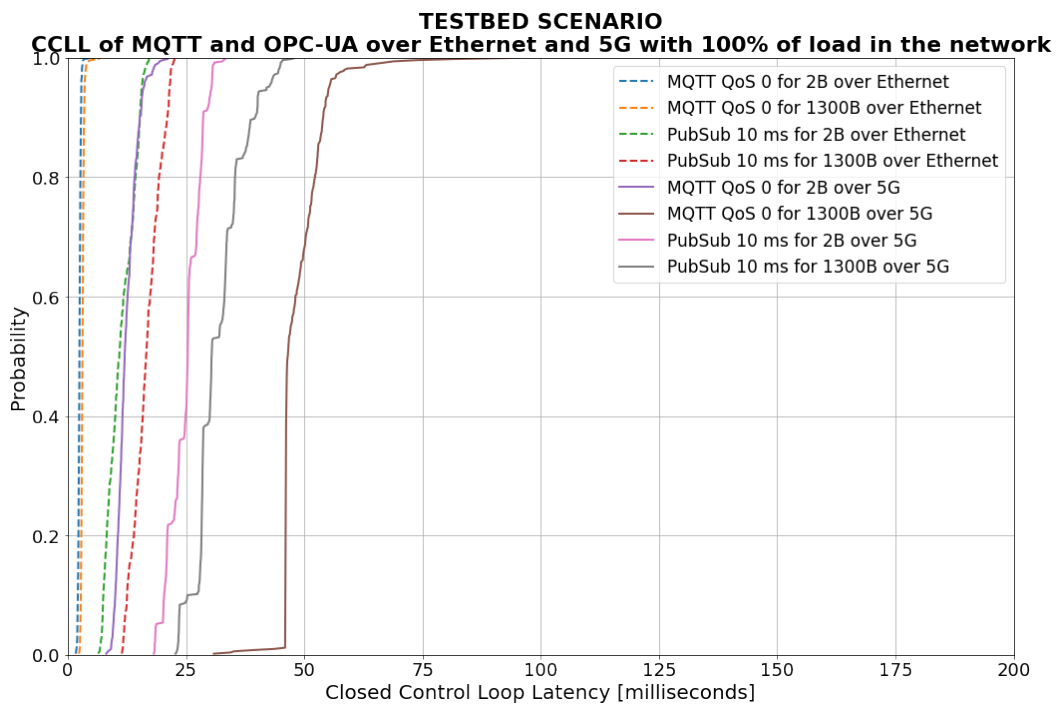


Figure 3.12: ECDF for MQTT and OPC-UA over Ethernet and 5G with 100% of network load.

Metric	Ethernet Value [ms]	5G Value [ms]
O_{MQTT}	0.64	34.5
O_{OPC-UA}	5.67	5.06
Δ_{2B}	8.48	13.31
Δ_{1300B}	13.47	-16.13

Table 3.8: Comparison of performance indicators over Ethernet and 5G

To quantify the CCLL performance difference between the Ethernet and 5G cases, the following metrics are defined:

$$\Theta_{2B} [\text{ms}] = \frac{1}{2} \left\{ (P50_{MQTT-QoS02B5G} - P50_{MQTT-QoS02BEth}) [\text{ms}] + (P50_{OPC-UA-10ms2B5G} - P50_{OPC-UA-10ms2BEth}) [\text{ms}] \right\} \quad (3.1)$$

$$\Theta_{1300B} [\text{ms}] = \frac{1}{2} \left\{ (P50_{MQTT-QoS01300B5G} - P50_{MQTT-QoS01300BEth}) [\text{ms}] + (P50_{OPC-UA-10ms1300B5G} - P50_{OPC-UA-10ms1300BEth}) [\text{ms}] \right\} \quad (3.2)$$

Θ_{2B} and Θ_{1300B} , therefore, quantify the average performance difference between the IIoT protocols being operated over Ethernet and 5G, computed at P50 level. These values are collected in Table 3.9. As detailed, the performance of the IIoT MQTT and OPC-UA protocols is degraded in 13.41 %, on average, when operated over 5G as compared to Ethernet.

Metric	Value [ms]
Θ_{2B}	12.01
Θ_{1300B}	14.8

Table 3.9: Performance difference over Ethernet and 5G

From this analysis, it can be concluded the following:

- **For the reference selected configurations, CCLL performance is 13.41 % inferior over 5G than over wired Ethernet.**

4. Operational robotic cell environment

4.1.- Robotic arms for industrial manufacturing

Historically, the idea of a factory was conceived as a large noisy building full of workers who using different tools performed repetitive, dangerous and demanding tasks on a daily basis in order to assemble or manufacture a specific item. Things have changed a lot since then, and due to the automation of the industry, today some of those jobs are carried out by complex machines such as robotic arms. Their adoption only leads to advantages that directly or indirectly have a positive impact on people's working and life conditions [2]. The main enhancements that robotic arms bring to the industrial scenario are:

- **Efficiency:** robotic arms are able to carry out tasks faster and more accurate than humans (in some cases).
- **Quality:** by using robotic arms, there is a significant reduction in the risk of making mistakes. This encourages the manufacturing of higher quality products.
- **Flexibility:** robotic arms can be programmed to develop a big range of different tasks in a short period of time whereas humans are usually specialized in a limited amount of jobs. This is very useful when it comes to the changing market and customer needs.
- **Costs:** investing in a robotic arm will lately save long-term costs.
- **Safety:** factories can be hostile and hazardous environments so using robotic arms to complete certain tasks such as holding toxic chemicals or working under extreme temperatures leads to an improvement in safety.

The concept of robotic cell that will be leveraged in this project consists on an operational robotic arm connected to an on-site PLC via Ethernet (Figure 1.6). Its standard configuration limits its flexibility, mobility and reconfigurability (among other disadvantages). Therefore, migrating the PLC capabilities to the edge-cloud (Figure 1.7) will be the focus since it brings many advantages, as explained in Section 1.5

4.2.- Deployment of the robotic cells and integration with 5G edge cloud

With the architectural design and performance study of both protocols over Ethernet and 5G already done, the next step is to apply the learnings in a realistic operational robotic cell environment. While the previous cabled Ethernet and wireless 5G edge cloud testbeds considered simplified computing nodes as end points for the evaluation of the different MQTT and OPC-UA architectures, here real-world industrial-grade equipment is used to validate the previous designs.

In this case, two identical robotic cells were deployed, based on commercial UR5e robotic arms [51] from the Danish company Universal Robots (UR) [38]. The UR5e arm weights around 21 kg and can handle payloads up to 5 kg. It has six joints (base, shoulder, elbow and three wrists) allowing it to move in a lot of directions.

Each one of the robotic cells will be made up of one Universal Robots UR5 [51], an OEM Control Box [52] and a Polyscope [53] as depicted in Figure 4.1. The Polyscope is a touch-screen tablet device with which an operator can control, manage and supervise the robotic arm through an intuitive and functional GUI. By using the Polyscope, it is possible to create programs, change the position of the robot and monitor its performance without the need of programming skills. The control box is the “brain” of the robotic arm. It is a dedicated computer that contains all the software and hardware needed to control the movement and behavior of the arm. This computer has several I/O ports that communicate with the UR5 sensors and actuators.



Figure 4.1: Disassembled UR5 robotic arms

The main technical features of the OEM Control Box are the following:

- Operating System: Linux based.
- CPU: Intel Atom E3845 (4 cores).
- RAM: 2GB.
- Storage: 8GB.

The robotic arm will use a power and data transmission cable. The Polyscope will use its own custom cable connected to the OEM Control box whereas the 5G Box will be connected to the UR5e using a CAT6 Ethernet cable. Finally, the OEM Control Box will be connected to the power line. Figure 4.2 shows the integrated architecture of our operational robotic cells unified with the 5G network and the edge cloud. In this final complete engineered solution, the edge cloud hosts the PLC functionality (“Cloudified PLC”) together with the MQTT and OPC-UA control sides of the logic, as well as the MQTT broker functionality. At the robotic cell side, the OEM control box hosts the user device side of the protocols.

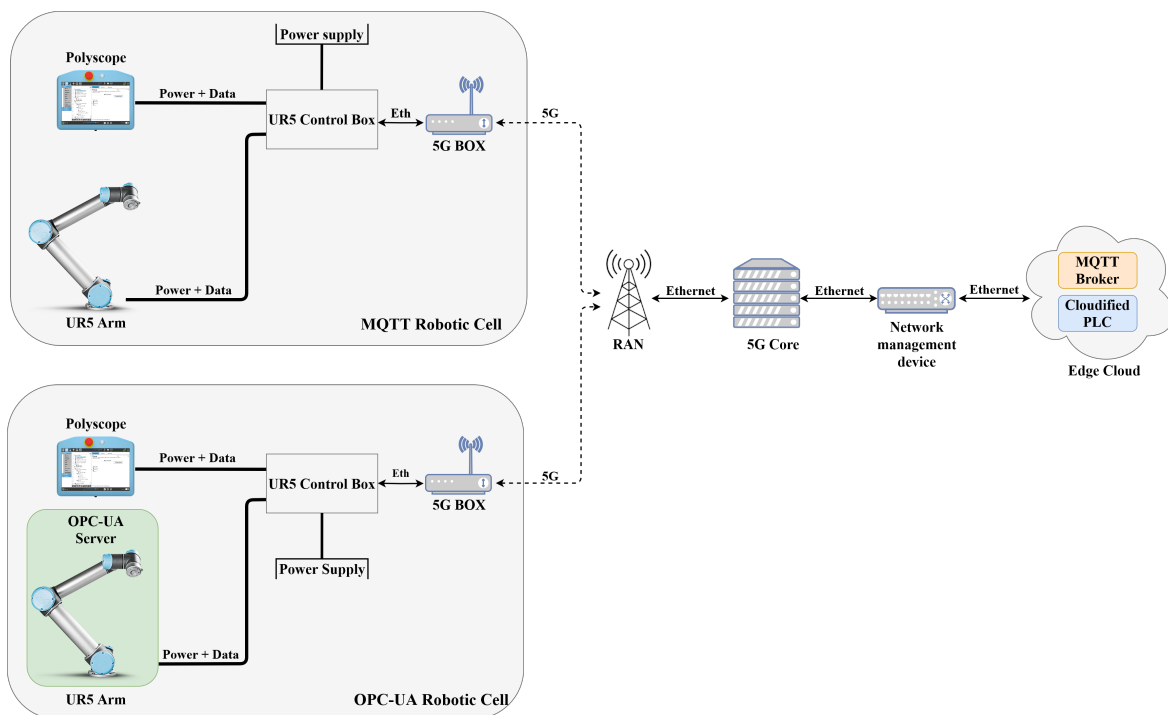


Figure 4.2: Robotic cell architecture integrated with 5G and edge-cloud

Each of the robotic cells will be used to study a different protocol. It is important to highlight that the UR5 inside OPC-UA Robotic Cell will act as an OPC-UA Server (which is the same role played by “Node 2” in Chapters 2 and 3) when running the performance tests. Figure 4.3 illustrates the real scenario once it is fully deployed. It is possible to distinguish the two robotic cells, each one composed by an UR5e robotic arm, OEM Control box, 5G Box and Polyscope.



Figure 4.3: Final setup for the operational robotic cell

Specifically, for the MQTT robotic cell, MQTT QoS 0 is used. In the case of the OPC-UA robotic cell, OPC-UA PubSub with scheduling rate 10 ms is configured. The reason for choosing this specific version of the protocols for the operational implementations is that MQTT QoS 0 delivered the best CCLL performance in both the testbed tests over wired Ethernet and 5G. Choosing QoS 0 will also avoid all unnecessary overhead and redundant exchange of packets introduced by the QoS 1 and QoS 2 configurations. In the case of OPC-UA, the refreshing period of 10 ms is chosen as it provides a good trade-off between the refresh rate of the protocol and the 5G scheduling configuration. Moreover, the physical movements of the robot are not as fast as 1 ms, and therefore a more relaxed period is suitable in this specific industrial use case. Further, it is desirable to avoid large CCLL performances as the ones introduced with the 50 ms period. The choice of PubSub rate was also validated by the vendor of the robot as it matches closely the sensor-actuator rate of the operational robotic arm.

Based on the above, OBJ2 is fulfilled and a number of observations and recommendations can be made:

- **Integrating 5G and edge-cloud with the operational equipment was at the same level of difficulty as in the testbed case.**

- For the operational equipment chosen, MQTT was easier to implement and integrate than OPC-UA.
- In current operational OPC-UA-capable equipment, the server is typically deployed at the robotic device side.
- Current operational OPC-UA-capable equipment has limited computing capabilities, therefore configured OPC-UA PubSub rates should be balanced.

4.3.- Operational cycle performance study in operational settings

Now that a real industrial scenario has been deployed, the MQTT and OPC-UA behavior will be studied in operational setting based on a functional solution designed to move the robotic arms in different directions inside a predefined routine. In this operational implementation, the payload of the packets transmitted via MQTT and OPC-UA over 5G and edge-cloud will contain the data related to the control and steering of the robotic arm (from PLC to robotic arm) and to the current position and status of the robotic arm (from the robotic arm to the PLC).

The precise packet size will be determined by the protocols themselves and will not be predefined to static values as in the previous preliminary evaluations from Chapters 2 and 3. No extra load will be applied to the 5G network for this evaluation in operational settings. This is not a limitation of the study, since it was already demonstrated that the MQTT and OPC-UA are stable, robust and scalable over 5G, so similar results should be observed in loaded 5G scenarios. When addressing a realistic operational industrial use case, as the robotic cells considered in this project, it is important to analyze the performance, not only from the communications point of view, but also taking into account manufacturing KPIs as the following:

- Operational Closed Control Loop Latency (OPCCLL): defined as the time that takes to the arm to move from a certain position to a new one after receiving the order from the cloudified PLC. This can be seen as the elapsed time for a given movement, including different latency components: L_{ComE2R} is the latency of the command which travels from the edge-cloud to the robotic arm and $ETCM$ (Elapsed Time for Current Movement) is the time it takes the robotic arm to accomplish the movement requested by the edge-cloud once it receives the command.

$$OPCCLL \text{ [ms]} = L_{ComE2R} \text{ [ms]} + ETCM \text{ [ms]} \quad (4.1)$$

- Operational Cycle Elapsed Time (OPCET): defined as the total elapsed time of one cycle of the robotic cell. In this case one operational cycle is defined by 10 different sequential movements of the robotic arm (5 in one direction and 5 in the opposite, to return to the initial home position).

Each of the robotic cells will be operated with a different protocol so it is possible to compare the overall robotic cell performance over MQTT and OPC-UA simultaneously under the exact same operational conditions. Each of the movements performed by the robotic arm is triggered by the exchange of data with the Cloudified PLC on the edge.cloud. Although the initial CCLL study was focused on small (2B) and big (1300B) payload sizes, in this case payload is subject to the targets of the operational use case. From PLC to robotic arm, the payload contains the position that the robotic arm joints need to apply in order to rotate. From the robotic arm to the PLC, the previous values are replaced with the current status values from the robot. The exact format of the payload is the same in both directions: a 6-element array of high resolution floats with angular information in radians. Table 4.1 shows the average payload, packet and frame sizes in the exchange of information in operational conditions.

Protocol	Payload size	Packet size	Frame size
MQTT	134B	150B	210B
OPC-UA	120B	402B	455B

Table 4.1: Payload, packet and frame sizes in operational settings.

Figure 4.4 illustrates the whole process of measuring the OPCCLL in both MQTT and OPC-UA. The cloudified PLC, which is subscribed to the robotic arm, measures “Timestamp 1” and just after, it publishes a command whose payload contains the new position the robotic arm has to move to. Once the robotic arm (which is subscribed to the edge-cloud) receives the command, it decodes the payload and applies that configuration to its joints by using Real-Time Data Exchange (RTDE) protocol [54]. It is a proprietary protocol from Universal Robots used to transmit data between an external client and the robotic arm actuators as shown in Figure 4.5. In the meantime the robotic arm is moving from its current position to the new one, it publishes its joints values to the edge-cloud every 5 ms. During this time slot the cloudified PLC does not publish, it just waits until one of the messages sent by the arm fits a threshold of radians compared to the new position. Once it happens, it understands that the arm’s current position is the new one so it continues to send the next position. This whole process is repeated for every movement. OPCCLL is calculated by subtracting “Timestamp 2” from “Timestamp 1”, just as explained in Equation 4.1.

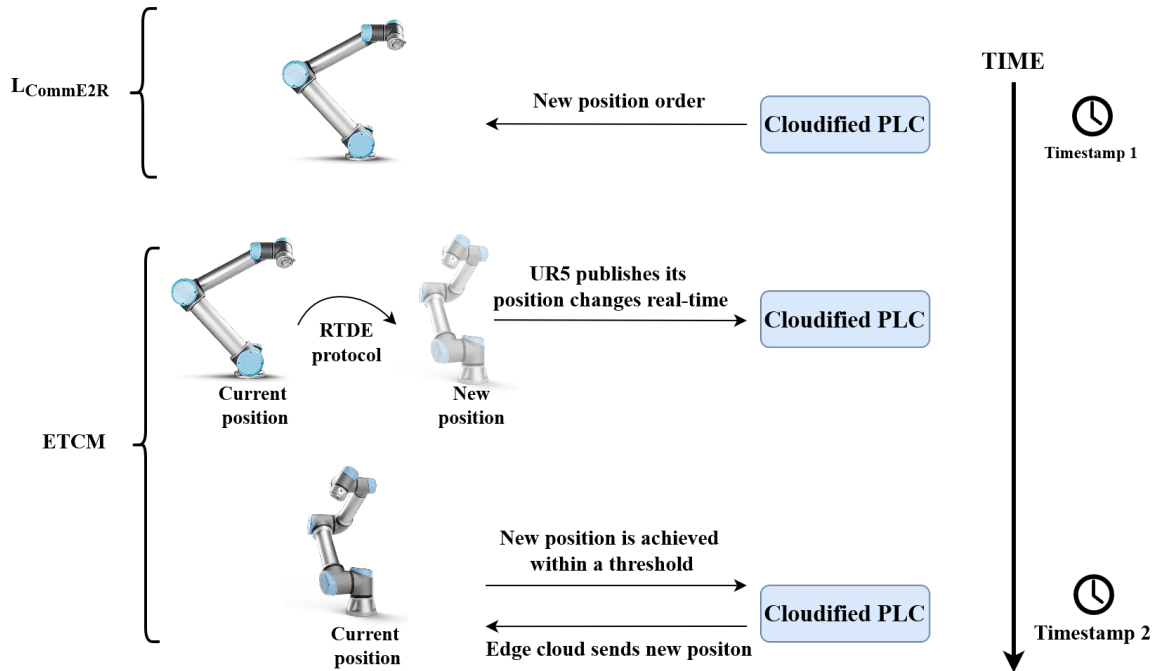


Figure 4.4: Diagram of the Operational CCLL.

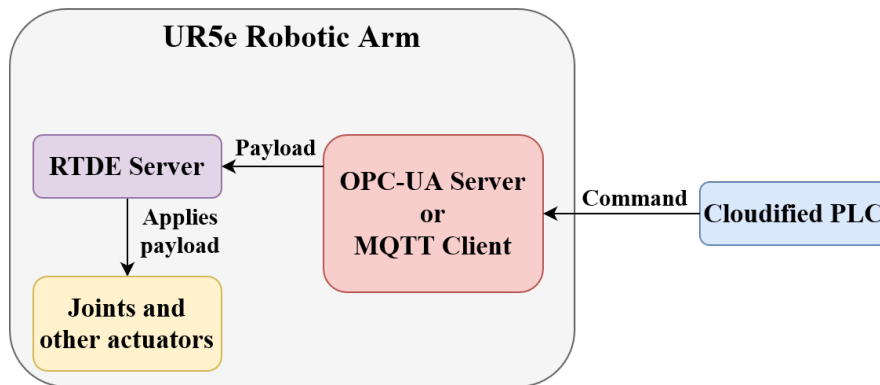


Figure 4.5: Data acquisition process by the robotic arm.

The exact same setup architecture was also evaluated with wired Ethernet connection between the edge-cloud and the OEM controller, in order to have the performance baseline for reference robotic cell operation in real-world settings.

4.4.- Operational performance results

4.4.1.- Operational Closed Control Loop Latency

In Figure 4.6, the ECDFs of the OPCCLL for the operational MQTT use case over Ethernet and 5G. The ECDFs present 5 steps. This is due to the 5 different movements of the robot (two of them are very similar in terms of latency and thus, they overlap). Over Ethernet, the

MQTT-controlled movements exhibit the approximate values of 12, 296, 312, 588, and 771 ms. Over 5G, these values are slightly increased to 12, 333, 372, 620, and 800 ms. The difference in values is due to the fact that some movements require big transitions, while others are just a rotation of a small number of radians. As observed, in this case, Ethernet provides a more stable performance than 5G.

In the OPC-UA case, displayed in Figure 4.7, the values of the movements are 11, 338, 357, 630 and 815 ms for Ethernet. For 5G, the latency values for the 5 movements are: 27, 365, 385, 650, and 835 ms. In this case, there is an outlier step with higher latency values over 1 s, which means that in this OPC-UA operational evaluation over 5G, some of the movements exhibited such latency.

Overall, the OPCCLL performance of MQTT over 5G is 33 ms slower as compared with Ethernet. In the OPC-UA case, this difference is 23 ms. Considering both protocols, the performance of the operational control of the robotic arms based on IIoT communication protocols is, on average, 28 ms slower over 5G edge-cloud than over reference Ethernet. This can be used as a partial conclusion:

- **The movements of the robot in operational conditions are delayed by approximately 25 ms when operated over 5G as compared to Ethernet.**

4.4.2.- Operational cycle elapsed time

To conclude the analysis, Table 4.2 collects the OPCET performance results for the different protocols and underlying communication technologies. In order to generate statistics, the operational cycle of the different MQTT and OPC-UA robotic cells considering the 10 robotic arm movements is executed 40 times. To quantify the difference in OPCET when operating over 5G and Ethernet, Γ is defined as follows:

$$\Gamma [s] = OPCET_{5G} [s] + OPCET_{Eth} [s] \quad (4.2)$$

Protocol	Scenario	Ethernet OPCET [s]	5G OPCET [s]	Θ [s]
MQTT	QoS 0	3.96 ± 0.007	4.34 ± 0.09	0.38
OPC-UA	PubSub 10ms	4.38 ± 0.05	4.88 ± 0.15	0.5

Table 4.2: Operational cycle elapsed time for the different tested configurations

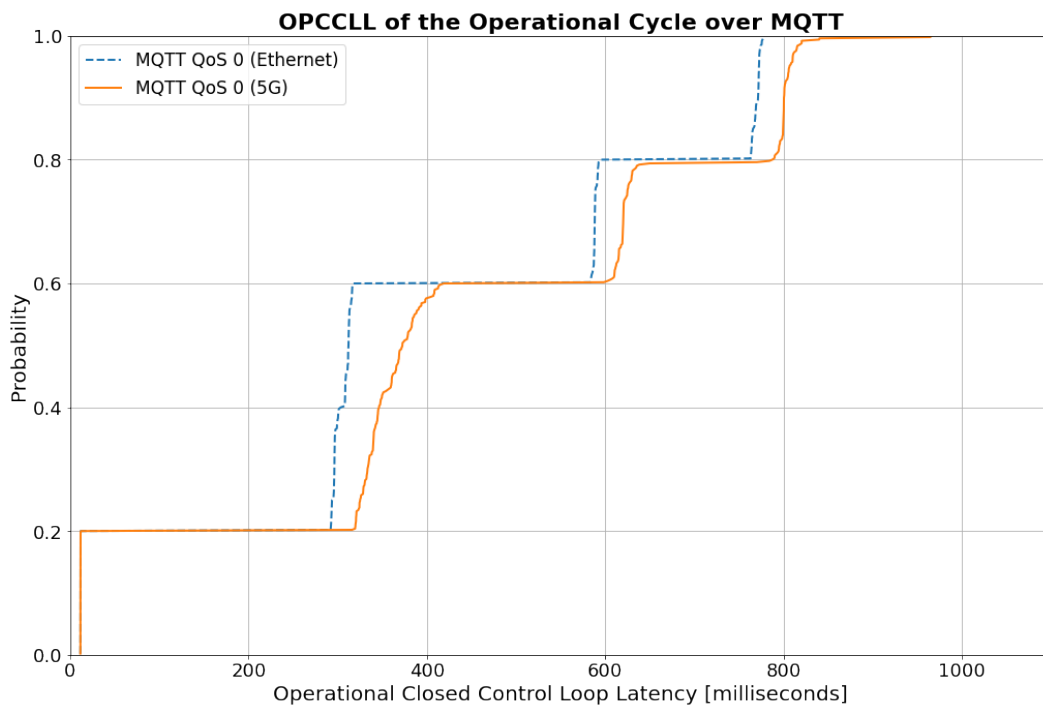


Figure 4.6: OPCCLL over MQTT over Ethernet vs 5G.

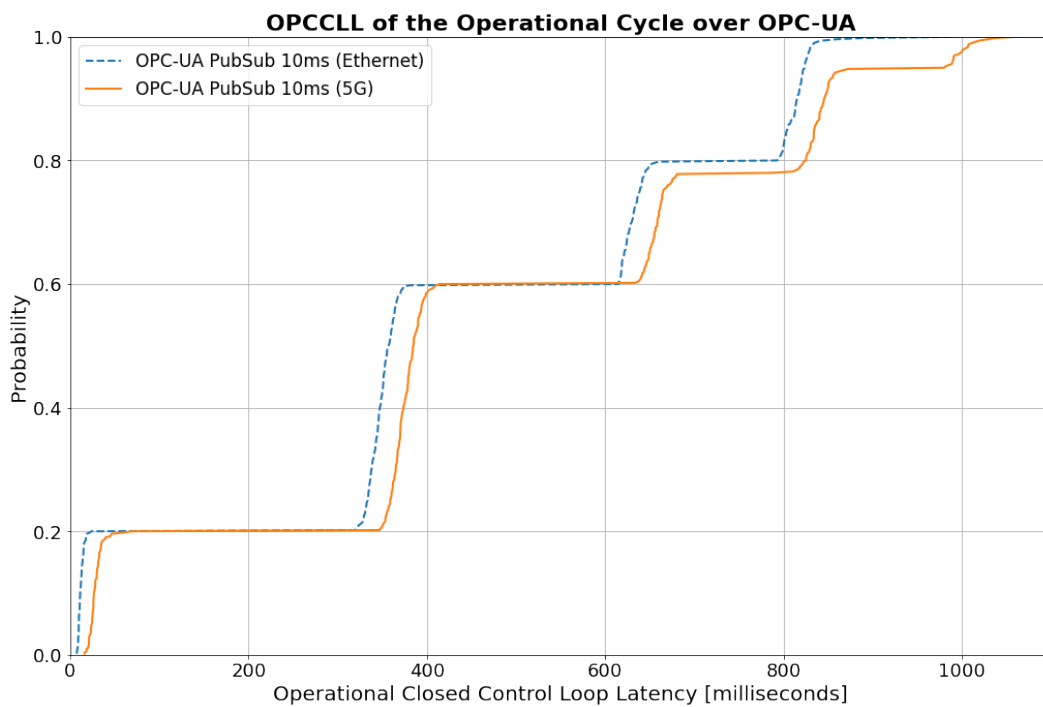


Figure 4.7: OPCCLL over OPC-UA over Ethernet vs 5G.

The OPCET values indicated in the table are reported as median \pm standard deviation in seconds. The results indicate that in terms of manufacturing KPI (OPCET), the cycle time of the robotic cell is incremented by 0.38-0.5 s when operating the robotic arm over 5G instead of Ethernet. This increment is due to the combined effect of all elements described along this document: communication technology, computing capabilities, and robotic hardware capabilities. For our specific reference robotic cell use case, the cycle time is increased from 3.96-4.38 s to 4.34-4.88 s.

As detailed in Section 1.4.1, this slightly worsened performance comes with other associated benefits such as the possibility of rapid reconfiguration of the cell and increased flexibility.

It is possible to draw the following conclusions:

- **In operational conditions, MQTT performs approximately 0.4 s better than OPC-UA, both over Ethernet and 5G.**
- **In operational conditions, the performance of MQTT is more stable than OPC-UA, both over Ethernet and 5G.**
- **In operational conditions, the performance of the robotic cell is delayed by approximately 0.4-0.5 ms when operated over 5G as compared to Ethernet.**

4.5.- Discussion of results

In order to complete this chapter, it was necessary to research the design and development of control network solutions based on industrial robotic architectures using MQTT and OPC-UA protocols, as well as the edge-cloud paradigm, connecting each node through 5G technology. Furthermore, it was also necessary to deploy the proposed solution and study its performance with real industrial equipment to evaluate the behavior of each layer as the final scenario was partially built. With this, we can consider the partial objectives OBJ2 and OBJ3 satisfactorily achieved since their outcome has resulted in added value to the current state of the wired robotic cells.

From the above results, and the development of the whole project, it can be concluded that, for our reference industrial use case, the MQTT solution, based on QoS 0 configuration, has been easier to design and deploy as compared to the OPC-UA PubSub with scheduling rate 10 ms. Further, the MQTT performance in operational conditions has been also superior,

providing 12% shorter operational cycle times than the glsopcu one when operating over 5G and edge-cloud technologies.

In terms of operational performance, the following conclusion is derived:

- **In operational conditions, robotic cycle times can be up to approximately 10% longer with OPC-UA than with MQTT, both over Ethernet and 5G.**

5. Conclusions and future work

This thesis presented the work done towards designing, implementing and testing the control communication of schemes of an operational industrial robotic cell environment based on 5G and edge-cloud technologies. In order to achieve that, an initial in-depth study of the main capabilities, architectures, and configuration aspects of the two most significant IIoT communication protocols (MQTT and OPC-UA) was done, considering different Ethernet and 5G scenarios. A wireless solution was proposed by exploring the control requirements set to overcome the limitation of current wired industrial manufacturing robotic cells. The main objective of the project has been successfully achieved by implementing a functional 5G wireless solution capable of steering a robotic arm from a cloudified PLC, without any operational distinction from the reference wired scenario beyond specific performance results.

Each one of the subobjectives presented in Section 1.7 has also been completed leading to a number of interesting outcomes and learnings:

- MQTT-SN is not a suitable protocol for controlling industrial machinery since it does not offer reliability, which is a must within manufacturing due to the big amount of risky processes that take place inside a factory. Nevertheless, it stands out as the protocol with the most favorable performance outcomes.
- The proposed solution for MQTT, both over Ethernet and 5G, was the easiest to design, deploy, and test in addition to offering the best performance results. This is advantageous for manufacturers aiming to generate value with new plug-and-play-based products focused on smart manufacturing since it does not require highly skilled personnel to integrate them.
- The performance differences between MQTT and OPC-UA were not very large, but they are significant enough to be considered in respect to the manufacturing processes and their associated operational cycle times. Faster data exchange results in shorter operational cycles, leading to increased production or execution of specific processes.
- Despite the introduction of an additional delay of approximately 0.35 to 0.5 ms when using 5G edge-cloud in the operation of the robotic cell, it remains advantageous in terms of reconfiguration, mobility, scalability, elasticity, and other factors emphasized consistently throughout the project. For example, the operational cycle elapsed time for MQTT QoS 0 over Ethernet lasted around 3.96 seconds while in the case of 5G it

reached 4.34 seconds.

- OPC-UA offers a more complex architecture and data structure that requires higher computing resources; however, can be advantageous when interconnecting various devices within a factory to achieve more sophisticated solutions beyond the scope of this thesis.

In conclusion, the development of this project has served to understand, analyze, and propose a roadmap to overcome current architectural limitations of reference industrial use cases in terms of communication and computing demands and possibilities. This thesis document can be relevant to telecom engineering professionals and industrial manufacturing professionals working in the domain of IT/OT integration design and deployment of vertical connectivity solutions for high-level control in industrial automation scenarios.

5.1.- Future work

Given the interest and impact of the results obtained from this project, it is proposed to continue exploring new horizons, such as the integration of additional robotic arms, aiming to study the scalability of the protocols, specially when operated over 5G and edge-cloud technologies. Based on the performance evaluations reported in this thesis, and potentially some new other, optimization of the Industrial Internet of Things (IIoT) protocols could be explored, applying compression methods, for example. Another interesting proposal would be to evolve the robotic cell to a multi-robotic arm one, integrating the operation of more than one robot. When doing this, synchronization of the robotic arms is paramount so that they can operate coordinately. This would allow for a deeper exploration of real-time communications between devices and opens the door to many exciting applications within the realm of smart manufacturing. Other slightly more distant but equally interesting proposal involves the re-evaluation of the observed performance with upgraded 5G versions based on the new Releases, including features such as Ultra-Reliable Low Latency (URLLC) capabilities and harnessing the power of data-driven decisions based on artificial intelligence in order to fully exploit the benefits of 5G and edge-cloud in operational industrial scenarios.

Impact of the project

The results obtained in this project were attractive to several companies and were used in several lectures, presentations, demos, and two scientific papers, which are currently in preparation.

The results in this project have attracted the interest of the following companies:

- Universal Robots (UR) [38]: innovative robotics company and world leading manufacturer of collaborative industrial robots (cobots). In particular, dedicated weekly meetings were held with their department of Technology and Innovation in Robotics and Integration, in order to discuss 5G integration into UR products and share the insights of the MQTT and OPC-UA implementations and performance.
- NOKIA [55]: Finnish multinational vendor specialized in information technology, telecommunication networks, and consumer electronics. In this case, the Standardization Department, was the one following the development of the project, as empirical performance results are always important in future 5G/6G standardization processes. A number of dedicated meetings based on the findings of the project were held to discuss the performance results of the IIoT protocols (specially OPC-UA) over private industrial 5G. Some of the results and finding might be forwarded and discussed in details with 5G-ACIA and the OPC Foundation.
- Grundfos [56]: world-leading manufacturer of water pumps and related equipment, based in Denmark. The project attracted the attention of the Engineering Department for Factories of the Future, as they already had experiences with private 5G and IIoT protocols. Face-to-face conversations happened at multiple events.
- Intelligent Systems A/S [57]: Danish company expert in software development for automated logistic solutions and integration and control systems for large-scale robot and automation systems. They implement control networks based on PLCs with MQTT support. Face-to-face conversations happened with their CEO and staff from their Engineering and software architecture teams during lab work and at multiple events.
- Technicon [58]: Danish company that delivers complete automation solutions across many industries. Face-to-face conversations happened with staff from their Implementation team during lab work and at multiple events around the use of MQTT for control of robotic entities.

In terms of demos and presentations of the project at industrial events, the following activities are reported:

- 5G-ROBOT Deep Dive Lecture: I delivered a 20 minutes lecture towards industrial and academic partners in the 5G-ROBOT Danish national project [59] with focus on MQTT and OPC-UA performance result over 5G.
 - Presenter: David Arias-Cachero Rincón
 - Date and location: 14th April 2023, remote over MS Teams.
- Presentation at Electronics of Tomorrow 2023 (EOT 2023) [60]: EOT is the main Danish conference and knowledge hub for the entire electronics and technology industry. My supervisor presented slides describing architectures, images, and videos of the robotic arms setups controlled via OPC-UA and MQTT over 5G.
 - Presenter: NOKIA Bell Labs Principal Scientist and AAU Professor Preben Mogenssen
 - Date and location: 11th May, Herning (Denmark).
- 5G-ROBOT Demo: I showcased the final implementation of my project with two operational robotic arms controlled with the different protocols, in front of an audience of over 30 people from different Industry sector, together with a Senior SW Development Engineer from UR, who put emphasis on the more robotic commercial aspects.
 - Presenter: David Arias-Cachero Rincón
 - Date and location: 2nd June 2023, AAU 5G Smart Production Lab, Aalborg (Denmark).

Further, the following scientific publications containing selected results of this project are planned:

- Paper 1: “Operational 5G Edge Cloud-Controlled Robotic Cell Environment based on MQTT and OPC UA”, with full focus on the main findings of this project (targeted as conference paper for one of the main international robotic conferences such as IROS, ICAR or the like.)
 - List of author affiliations: University of Oviedo (main), Aalborg University, UR, and Yildiz Technical University FBE.
- Paper 2: “5G-based Zero Touch Production”, where relevant OPC-UA and MQTT performance results will be included as part of the concept and description of a fully-automated wireless robotic production environment (targeted as journal paper for Q1/Q2

outlets with focus on industrial manufacturing or the like.)

- List of author affiliations: University of Oviedo (main), Aalborg University, NO-KIA, and FESTO.

References

- [1] M. Xu, J. M. David, S. H. Kim, *et al.*, “The fourth industrial revolution: Opportunities and challenges,” *International journal of financial research*, vol. 9, no. 2, pp. 90–95, 2018.
- [2] Brandl, Dennis and Johnsson, Charlotta”, “Beyond the Pyramid: Using ISA95 for Industry 4.0 and Smart Manufacturing.” (<https://www.isa.org/intech-home/2021/october-2021/features/beyond-the-pyramid-using-isa95-for-industry-4-0-an>), 2021. Accessed on: February 20, 2023.
- [3] M. Wollschlaeger, T. Sauter, and J. Jasperneite, “The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0.,” *IEEE industrial electronics magazine*, vol. 11, no. 1, pp. 17–27, 2017.
- [4] J. A. Kay, R. A. Entzminger, and D. C. Mazur, “Industrial Ethernet-overview and best practices,” in *Conference Record of 2014 Annual Pulp and Paper Industry Technical Conference*, pp. 18–27, IEEE, 2014.
- [5] I. Rodriguez, R. S. Mogensen, A. Schjørring, M. Razzaghpour, R. Maldonado, G. Berardinelli, R. Adeogun, P. H. Christensen, P. Mogensen, O. Madsen, *et al.*, “5G swarm production: Advanced industrial manufacturing concepts enabled by wireless automation.,” *IEEE Communications Magazine*, vol. 59, no. 1, pp. 48–54, 2021.
- [6] “Bluetooth technology website.” (<https://www.bluetooth.com/>). Accessed on: February 20, 2023.
- [7] N. A. Somani and Y. Patel, “Zigbee: A low power wireless technology for industrial applications,” *International Journal of Control Theory and Computer Modelling (IJCTCM)*, vol. 2, no. 3, pp. 27–33, 2012.
- [8] P. Roshan and J. Leary, *802.11 Wireless LAN fundamentals*. Cisco press, 2004.
- [9] IEEE, “IEEE 802.11ax,” Tech. Rep. 802.11ax-2019, IEEE Standards Association, 2019.
- [10] A. Fink, R. S. Mogensen, I. Rodriguez, T. Kolding, A. Karstensen, and G. Pocovi, “Empirical performance evaluation of EnterpriseWi-Fi for IIoT applications requiring mobility,” in *European Wireless 2021; 26th European Wireless Conference*, pp. 1–8, VDE, 2021.

- [11] J. Navarro-Ortiz, P. Romero-Diaz, S. Sendra, P. Ameigeiras, J. J. Ramos-Munoz, and J. M. Lopez-Soler, “A survey on 5G usage scenarios and traffic models,” *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 905–929, 2020.
- [12] I. Rodriguez, “5G for Industries,” in *Wireless Communications and Technologies for 5G/6G Networks*, (Granada), 2023.
- [13] Z. Zhu, X. Li, and Z. Chu, “Three major operating scenarios of 5G: eMBB, mMTC, URLLC,” *Intell. Sens. Commun. Internet Everything*, vol. 1, pp. 15–76, 2022.
- [14] J.-H. Huh and Y.-S. Seo, “Understanding edge computing: Engineering evolution with artificial intelligence,” *IEEE Access*, vol. 7, pp. 164229–164245, 2019.
- [15] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.
- [16] Z. Li, X. Zhou, and Y. Qin, “A survey of mobile edge computing in the industrial internet,” in *2019 7th International Conference on Information, Communication and Networks (ICICN)*, pp. 94–98, IEEE, 2019.
- [17] López, Melisa and Damsgaard, Sebastian Bro and Kabaci, Akif and Zhang, Weifan and Sharma, Himanshu and Bisheh, Sepideh Valiollahi and Rodriguez, Ignacio and Mogenssen, Preben E, “Towards the 5G-Enabled Factories of the Future,” in *IEEE International Conference on Industrial Informatics*, IEEE, 2023.
- [18] D.-H. Luong, H.-T. Thieu, A. Outtagarts, and Y. Ghamri-Doudane, “Cloudification and autoscaling orchestration for container-based mobile networks toward 5G: Experimentation, challenges and perspectives,” in *2018 IEEE 87th Vehicular Technology Conference (VTC Spring)*, pp. 1–7, IEEE, 2018.
- [19] M. W. Dawande, H. N. Geismar, S. P. Sethi, and C. Sriskandarajah, *Throughput optimization in robotic cells*, vol. 101. Springer Science & Business Media, 2007.
- [20] Pereira, Carlos E and Neumann, Peter, “Industrial communication protocols,” *Springer Handbook of Automation*, pp. 981–999, 2009.
- [21] Rocha, Murilo Silveira and Sestito, Guilherme Serpa and Dias, Andre Luis and Turcato, Afonso Celso and Brandão, Dennis, “Performance comparison between OPC UA and MQTT for data exchange,” in *2018 Workshop on Metrology for Industry 4.0 and IoT*, pp. 175–179, IEEE, 2018.
- [22] A. Hazra, M. Adhikari, T. Amgoth, and S. N. Srirama, “A comprehensive survey on interoperability for IIoT: taxonomy, standards, and future directions,” *ACM Computing Surveys (CSUR)*, vol. 55, no. 1, pp. 1–35, 2021.

- [23] F. T. AL-Dhief, N. Sabri, N. A. Latiff, N. Malik, M. Abbas, A. Albader, M. A. Mohammed, R. N. AL-Haddad, Y. D. Salman, M. Khanapi, *et al.*, “Performance comparison between TCP and UDP protocols in different simulation scenarios,” *International Journal of Engineering & Technology*, vol. 7, no. 4.36, pp. 172–176, 2018.
- [24] “MQTT: The Standard for IoT Messaging.” <https://mqtt.org/mqtt-specification/>. Accessed on: March 16, 2023.
- [25] “OPC Foundation.” <https://opcfoundation.org/about/opc-technologies/opc-ua/>. Accessed on: February 25, 2023.
- [26] M. Husnain, K. Hayat, E. Cambiaso, U. U. Fayyaz, M. Mongelli, H. Akram, S. Ghanfar Abbas, and G. A. Shah, “Preventing MQTT vulnerabilities using IoT-enabled intrusion detection system,” *Sensors*, vol. 22, no. 2, p. 567, 2022.
- [27] “The RabbitMQ broker.” <https://www.rabbitmq.com/>. Accessed on: March 16, 2023.
- [28] “HiveMQ broker.” <https://www.hivemq.com/public-mqtt-broker/>. Accessed on: March 16, 2023.
- [29] “Mosquitto: an open source MQTT broker.” <https://mosquitto.org/>. Accessed on: March 16, 2023.
- [30] M. M. Alani, *Guide to OSI and TCP/IP Models*. 2014.
- [31] S. Lee, H. Kim, D.-k. Hong, and H. Ju, “Correlation analysis of MQTT loss and delay according to QoS level,” in *The International Conference on Information Networking 2013 (ICOIN)*, pp. 714–717, IEEE, 2013.
- [32] O.A.S.I.S, “MQTT version 3.1.1.” (<http://docs.oasis-open.org/mqtt/mqtt/v3>), 2014. Accessed on: April 25, 2023.
- [33] A. Stanford-Clark and H. L. Truong, “MQTT for Sensor Networks (MQTT-SN) protocol specification,” *International business machines (IBM) Corporation version*, vol. 1, no. 2, pp. 1–28, 2013.
- [34] S.-H. Leitner and W. Mahnke, “OPC UA–service-oriented architecture for industrial applications,” *ABB Corporate Research Center*, vol. 48, no. 61-66, p. 22, 2006.
- [35] M. H. Schwarz and J. Börcsök, “A survey on OPC and OPC-UA: About the standard, developments and investigations,” in *2013 XXIV International Conference on Information, Communication and Automation Technologies (ICAT)*, pp. 1–6, IEEE, 2013.

- [36] T. Hannelius, M. Salmenpera, and S. Kuikka, “Roadmap to adopting OPC UA,” in *2008 6th IEEE International Conference on Industrial Informatics*, pp. 756–761, IEEE, 2008.
- [37] KUKA, “KUKA: Industrial intelligence beyond automation.” (<https://www.kuka.com/>). Accessed on: April 25, 2023.
- [38] U. Robots, “Universal Robots: collaborative robotic automation.” (<https://www.universal-robots.com/>). Accessed on: April 25, 2023.
- [39] “Performance Analysis of OPC UA for Industrial Interoperability towards Industry 4.0, author=Ladegourdie, Marc and Kua, Jonathan,” *IoT*, vol. 3, no. 4, pp. 507–525, 2022.
- [40] “Up Squared Pro.” (<https://up-board.org/up-squared-pro/>). Accessed on: April 25, 2023.
- [41] “Zyxel GS-105B V2 switch.” (https://www.zyxelguard.com/datasheets/Switches/GS105b_108b_ds_09-11-2012.pdf). Accessed on: April 25, 2023.
- [42] iPerf Development Team, “iPerf3.” <https://iperf.fr/>, 2021. Accessed on: February 20, 2023.
- [43] “OPC-UA Built-in Types.” (<https://reference.opcfoundation.org/v104/Core/docs/Part6/5.1.2/>). Accessed on: April 25, 2023.
- [44] C. Cox, *An introduction to 5G: the new radio, 5G network and beyond*. John Wiley & Sons, 2020.
- [45] “AAU Smart Production Laboratory.” <https://www.smartproduction.aau.dk/Laboratory/>. Accessed on: March 16, 2023.
- [46] Damsgaard, Sebastian Bro and Segura, David and Andersen, Martin Fejrskov and Markussen, Søren Aaberg and Barbera, Simone and Rodríguez, Ignacio and Mogensen, Preben, “Commercial 5G NPN and PN Deployment Options for Industrial Manufacturing: An Empirical Study of Performance and Complexity Tradeoffs.” 2023.
- [47] “Newport GW6400 Rugged & Industrial Single Board Computer.” <https://www.gateworks.com/products/industrial-single-board-computers/octeon-tx-single-board-computers-gateworks-newport/gw6400-single-board-computer/>. Accessed on: March 16, 2023.
- [48] “SIM8262E-M2 5G .” <https://techship.com/products/simcom-sim8262e-m2-5g-sub6/>. Accessed on: March 16, 2023.

- [49] “PuTTY. The SSH client.” <https://www.putty.org/>. Accessed on: March 16, 2023.
- [50] “FireHOL. Linux firewalling and traffic shaping for humans.” <https://firehol.org/>. Accessed on: March 16, 2023.
- [51] “UR5 collaborative robot arm.” (<https://www.universal-robots.com/products/ur5-robot/>). Accessed on: May 1, 2023.
- [52] “OEM Control Box Installation guide.” (<https://www.universal-robots.com/download/manuals-e-series/installation-guides/oem-control-box/oem-control-box-installation-guide-english-e-series/>). Accessed on: May 1, 2023.
- [53] “Program in your language with Polyscope - Universal Robots.” (<https://www.universal-robots.com/products/polyscope/>). Accessed on: May 1, 2023.
- [54] Universal Robots, “Real-Time Data Exchange (RTDE) Guide.” Retrieved on July 7, 2023, N/A.
- [55] “NOKIA Corporation.” <https://www.nokia.com/>. Accessed on: July 11, 2023.
- [56] “Grundfos.” <https://www.grundfos.com/>. Accessed on: July 11, 2023.
- [57] “Intelligent Systems AS.” <https://www.intelligentsystems.dk/>. Accessed on: July 11, 2023.
- [58] “Technicon.” <https://technicon.dk/en/>. Accessed on: July 11, 2023.
- [59] Aalborg University, “5G-Robot – 5G Enabled Autonomous Mobile Robotic Systems,” n.d. Accessed on 6th July 2023.
- [60] “Experience EOT.” <https://www.eot-expo.com/>. Accessed on: July 11, 2023.
- [61] “IEEE Standard for Floating-Point Arithmetic.” (<https://standards.ieee.org/ieee/754/6210/>). Accessed on: April 25, 2023.
- [62] “Unicode - The world standard for text and emoji.” (<https://home.unicode.org/>). Accessed on: April 25, 2023.

A. Project structure

The code necessary to replicate this study is gathered in the folder “KillThePLC.rar”. Inside, the following directories and files can be found:

- **cloud-side**: folder containing all the code executed on the edge-cloud.
 - **ProtocolPerformance**: folder containing all the code used to study MQTT, MQTT-SN and OPC-UA performance over Ethernet and 5G from the edge-cloud side.
 - **mqttsnclient**: MQTT-SN library in Python.
 - **“1500bytes.txt”**: file used to build payloads for big packets.
 - **“Node_1_MQTT_PRFM.py”**: script used to study MQTT performance over Ethernet and 5G. It plays the role of “Node 1”.
 - **“Node_1_MQTTSN_PRFM.py”**: script used to study MQTT-SN performance over Ethernet and 5G. It plays the role of “Node 1”.
 - **“Node_1 OPCUA_Client_PRFM.py”**: script used to study OPC-UA Client-Server performance over Ethernet and 5G. It plays the role of “Node 1”.
 - **“Node_1 OPCUA_PubSub_PRFM.py”**: script used to study OPC-UA Pub-Sub performance over Ethernet and 5G. It plays the role of “Node 1”.
 - **RemoteControl**: folder containing all the code used to implement the operational control of the robotic arm from the edge-cloud side.
 - **rtde**: RTDE library in Python.
 - **“record_configuration.xml”**: RTDE configuration file.
 - **“control_loop_configuration”**: RTDE configuration file.
 - **“Cloud_PLC_MQTT_RemoteControl.py”**: script used to control the robotic arm from the edge-cloud over MQTT. It plays the role of the “Cloudified PLC”.
 - **“Cloud_PLC OPCUA_RemoteControl.py”**: script used to control the robotic arm from the edge-cloud over OPC-UA. It plays the role of the “Cloudified PLC”.
- **robot-side**: folder containing all the code executed on the robotic arm.
 - **ProtocolPerformance**: folder containing all the code used to study MQTT, MQTT-SN and OPC-UA performance over Ethernet and 5G from the robotic cell side.
 - **mqttsnclient**: MQTT-SN library in Python.
 - **“1500bytes.txt”**: file used to build payloads for big packets.

- **“Node_2_MQTT_PRFM.py”**: script used to study MQTT performance over Ethernet and 5G. It plays the role of “Node 2”
- **“Node_2_MQTTSN_PRFM.py”**: script used to study MQTT-SN performance over Ethernet and 5G. It plays the role of “Node 2”
- **“Node_2 OPCUA_Client_PRFM.py”**: script used to study OPC-UA Client-Server performance over Ethernet and 5G. It plays the role of “Node 2”
- **“Node_2 OPCUA_PubSub_PRFM.py”**: script used to study OPC-UA Pub-Sub performance over Ethernet and 5G. It plays the role of “Node 2”
- **RemoteControl**: folder containing all the code used to implement the operational control of the robotic arm from the robotic cell side.
 - **rtde**: RTDE library in Python.
 - **“record_configuration.xml”**: RTDE configuration file.
 - **“control_loop_configuration”**: RTDE configuration file.
 - **“Robotic_Cell_MQTT_RemoteControl.py”**: script used to receive and process commands sent from the Cloudified PLC over MQTT.
 - **“Robotic_Cell OPCUA_RemoteControl.py”**: script used to receive and process commands sent from the Cloudified PLC over OPC-UA.

B. Deployment of MQTT & MQTT-SN brokers

B.1.- Installation and configuration of Mosquitto broker

In order to replicate the part of this project related to the MQTT protocol it is important to follow the same steps at the time of installing and configuring the broker. We begin by running the following commands used to update and upgrade Linux-based system:

```
1 dave@AAU:~$ apt update && apt upgrade
```

Once our packages are up to their latest version, we install the Mosquitto broker:

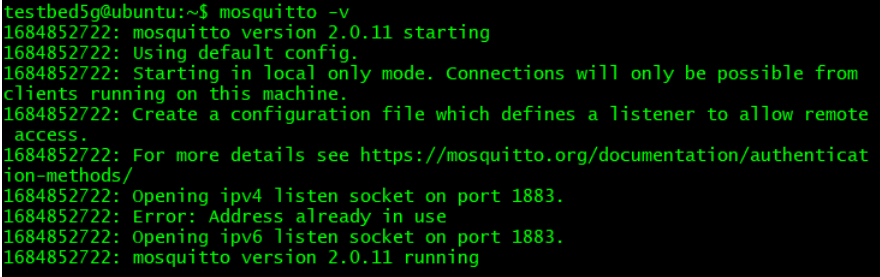
```
1 dave@AAU:~$ apt install -y mosquitto mosquitto-clients
```

Although the broker is intended to be run on a dedicated device or server that is always active, it is interesting that it starts automatically every time the server is restarted. The following command is needed for this purpose:

```
1 dave@AAU:~$ systemctl enable mosquitto.service
```

In order to run the broker once it is installed we need to execute the following command and the output of it should be similar to the one shown in figure B.1

```
1 dave@AAU:~$ mosquitto -v
```



```
festbed5g@ubuntu:~$ mosquitto -v
1684852722: mosquitto version 2.0.11 starting
1684852722: Using default config.
1684852722: Starting in local only mode. Connections will only be possible from
clients running on this machine.
1684852722: Create a configuration file which defines a listener to allow remote
access.
1684852722: For more details see https://mosquitto.org/documentation/authentication-methods/
1684852722: Opening ipv4 listen socket on port 1883.
1684852722: Error: Address already in use
1684852722: Opening ipv6 listen socket on port 1883.
1684852722: mosquitto version 2.0.11 running
```

Figure B.1: Output of the command 'mosquitto -v'.

If we take a closer look to figure B.1 it is possible to read the message 'Connections will only be possible from clients running on this machine'. This is not the scenario we are looking for since we need connections from other machines. This is solved by editing the configuration file 'mosquitto.conf' located in the directory '/etc/mosquitto/'. We need to apply the following instructions to it:

```
1 allow_anonymous false
```

```
2 listener 1883
3 password_file /etc/mosquitto/passwd
4 log_dest file /etc/mosquitto/mosquitto.log
```

With those statements we are telling the broker to establish authenticated connections in TCP port 1883 based on the credentials stored in the file 'passwd'. All incoming connections in which the username and password are not the ones written in that file, will be refused. Last line of the configuration file specifies the destination for the broker's log file.

The next step is to create both the 'passwd' and 'mosquitto.log' files. Our user will be 'testbed' and our password '1234'. That way, in the first one we write 'testbed: 1234'. In the second one there is no need to write anything.

We execute the following command to encrypt the password. The result should be similar to the one shown in figure

```
1 dave@AAU:~$ mosquitto_passwd -U passwd
```



Content of the file 'passwd' after encrypting it.

The final step is to restart the broker so all the above settings are applied:

```
1 dave@AAU:~$ systemctl restart mosquitto
```

B.2.- Installation and configuration of RSMB broker

Following the same idea as in the previous section, it is important to follow these steps at the time of installing the Realy Small Message Broker (RSMB) broker used for the MQTT-SN scenario. We start by creating a folder called 'RSMB' and cloning the broker's repository inside it:

```
1 dave@AAU:~$ git clone https://github.com/eclipse/mosquitto.rsmb.git
```

Once it is cloned, we run the following commands:

```
1 dave@AAU:~$ sudo apt install make
2 dave@AAU:~$ sudo apt install build-essential
```

The first command installs the 'make' package. It is used for compiling and building software projects. The second one installs the 'build-essential' package that includes tools and libraries needed at the time of building the project. By running these two orders we assure that the necessary dependencies and tools needed to compile and install the project are available in our system.

We go to '/RSMB/mosquitto.rsmb/rsmb/src' and execute the command:

```
1 testbed5g@ubuntu:~$ make
```

Its function is to follow instructions contained in a file called 'Makefile' which tells which commands and dependencies use to build the project. Once this process finishes, we can check if it has been successful by running the command:

```
1 dave@AAU:~$ echo $?
```

If the output is a zero, it means that the process has been executed successfully. Being that the case, there should be two new files called 'broker' and 'broker_mqtts'. The first one supports MQTT TCP version whereas the second one is the one we will use since it is MQTT-SN based. We create a file called 'conf_rsmb.txt' and add the following content:

```
1 trace_output protocol
2 listener 1884 INADDR_ANY mqtts
```

It is important to highlight that in this broker we are using the port 1884 so there is not a conflict with Mosquitto (it is listening in port 1883).

We apply this configuration file by running:

```
1 dave@AAU:~$ ./broker_mqtts conf_rsmb.txt
```

```
testbed5g@ubuntu:~/RSMB/mosquitto.rsmb/rsmb/src$ ./broker_mqtts conf_rsmb.txt
20230523 193736.416 CWNAN9999I Really Small Message Broker
20230523 193736.416 CWNAN9998I Part of Project Mosquitto in Eclipse
(http://projects.eclipse.org/projects/technology.mosquitto)
20230523 193736.416 CWNAN0049I Configuration file name is conf_rsmb.txt
20230523 193736.416 CWNAN0053I Version 1.3.0.2, May 20 2023 22:25:17
20230523 193736.416 CWNAN0054I Features included: bridge MQTTS
20230523 193736.416 CWNAN9993I Authors: Ian Craggs (icraggs@uk.ibm.com), Nicholas O'Leary
20230523 193736.416 CWNAN0300I MQTT-S protocol starting, listening on port 1884
```

Output after applying the configuration to the RSMB broker.

The last step in order to complete the MQTT-SN set up is to install and configure the clients. It has to be deployed in the Node1, Node2 for the Ethernet performance analysis and in the UR5 arm and Edge Cloud server for the 5G performance analysis. The library can be installed as:

C. Study of the payload

When analyzing packet size with WireShark, it is important to remember that the values the tool shows us in the main interface correspond to the total frame, which includes the payload as well as the encapsulating headers. To determine the size of the packet at a specific layer, it is necessary to open it and inside its content we will see a hierarchical representation of the different layers of the packet. Once we have located the desired layer (OPC-UA and MQTT in our case) we can find its size in the details panel.

C.1.- OPC-UA

OPC-UA has several built-in data types for its messages [43] but we are interested in the most common ones: Int64, Float, Bool and String.

Inside the OPC-UA packet, the field we need to focus on is called 'WriteValue'. It contains the following information:

- **NodeId**: identifier of the node to which the value will be written.
- **AttributeId**: the attribute of the node to which the value will be written.
- **IndexRange**
- **Value**: the data that will be written to the node. Inside it we have other fields such as its type and the data itself.

Int64 data type is used to represent integer values in the range -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807, both included. To test this built-in type, the minimum, maximum and other values will be sent in order to study how the payload behaves.

The minimum value is encoded in hexadecimal as 8000000000000000_{16} and it is stored using 8 bytes. The whole frame is 181 bytes while the OPC-UA packet size is 97 bytes. From those bytes, the field 'WriteValue' takes up 34 bytes. Figure ?? and subsequent figures highlight in blue the payload of the packet in this experiment.

```

  [0]: WriteValue
    NodeId: NodeId
      .... 0001 = EncodingMask: Four byte encoded Numeric (0x1)
      Namespace Index: 2
      Identifier Numeric: 3
      AttributeId: Value (0x0000000d)
      IndexRange: [OpcUa Null String]
    Value: DataValue
      > EncodingMask: 0x07, has value, has statuscode, has source timestamp
      Value: Variant
        Variant Type: Int64 (0x08)
        Int64: -9223372036854775808
        StatusCode: 0x00000000 [Good]
        SourceTimestamp: May 15, 2023 14:07:20.820717000 Romance Daylight Time
  
```

0000	02 00 00 00 45 00 00 89 6e 13 40 00 80 06 00 00E... n.@....
0010	ac 1b 14 cc ac 1b 14 cc fb a3 12 e8 a0 18 05 9a
0020	d1 a5 44 82 50 18 27 f4 70 87 00 00 4d 53 47 46	--D·P·'· p...MSGF
0030	61 00 00 00 06 00 00 00 0d 00 00 00 09 00 00 00	a.....
0040	09 00 00 00 01 00 a1 02 01 00 e9 03 42 47 12 cdBG..
0050	25 87 d9 01 09 00 00 00 00 00 00 00 ff ff ff ff	%.....
0060	a0 0f 00 00 00 00 00 01 00 00 00 01 02 03 00 0d
0070	00 00 00 ff ff ff ff 07 08 00 00 00 00 00 00 00
0080	80 00 00 00 00 42 47 12 cd 25 87 d9 01 00 00 00BG·-%.
0090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00a0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00b0	00 00 00 00 00

WriteValue fields when minimum Int64 value is sent.

If we send the maximum value, we also get a frame size of 181 bytes and an OPC-UA packet size of 97 bytes. 'WriteValue' field will have the same size as before. This time the payload is encoded as $7FFFFFFFFFFFFFFFFF_{16}$ using 8 bytes.

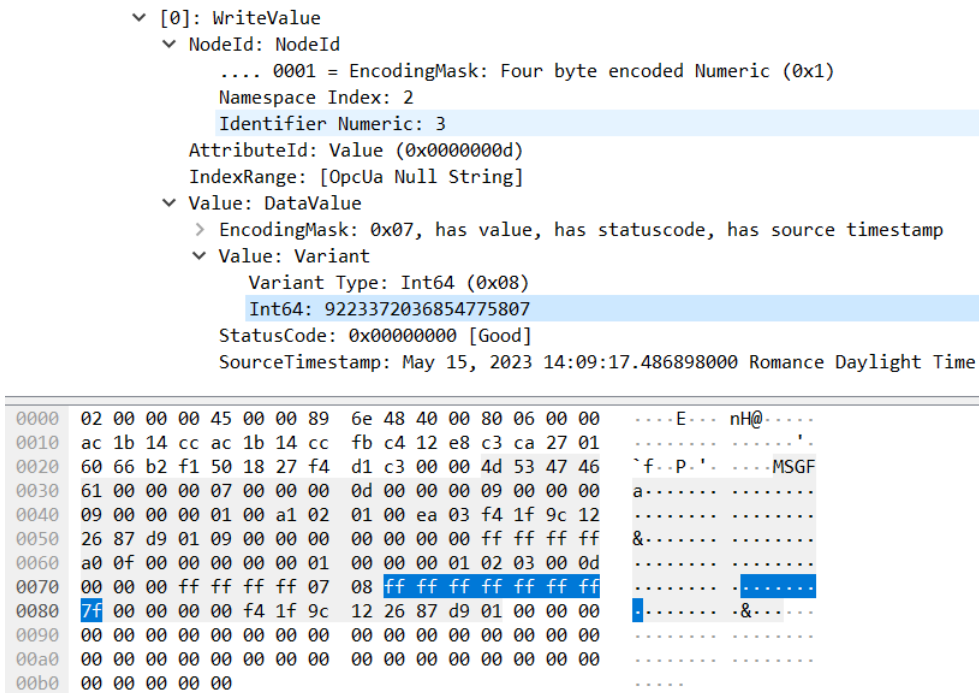


Figure C.1: WriteValue fields when maximum Int64 value is sent.

To conclude the study of the built-in type Int64, the value '123456789' will be sent. This can be considered as a small value compared to the constraints of the data type. In this case, the OPC-UA packet and the 'WriteValue' field sizes are the same as the two previous examples. The payload is encoded as 75BCD15₁₆ using 4 bytes while the other remaining bytes, circled in red, are used to store zeros.

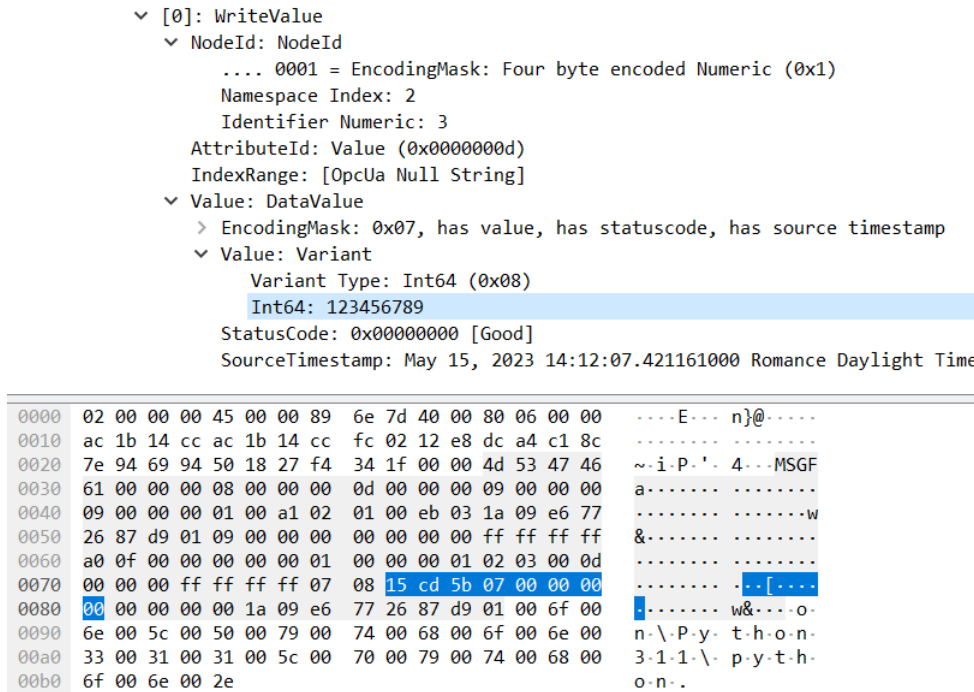


Figure C.2: WriteValue fields when '123456789' Int64 value is sent.

Boolean data type in OPC-UA is used to send two-state logical values, which represent a true or false. First one is encoded as 1_{16} and second one as 0_{16} . When sending a boolean true value, the frame size is 174 bytes and OPC-UA packet size is 90 bytes containing a 'WriteValue' field of 27 bytes. From those bytes, only one of them used to store the actual payload data as shown in figure C.3.

Otherwise, when sending a False value, OPC-UA packet and 'WriteValue' sizes are the same. Figure C.4 displays this scenario.

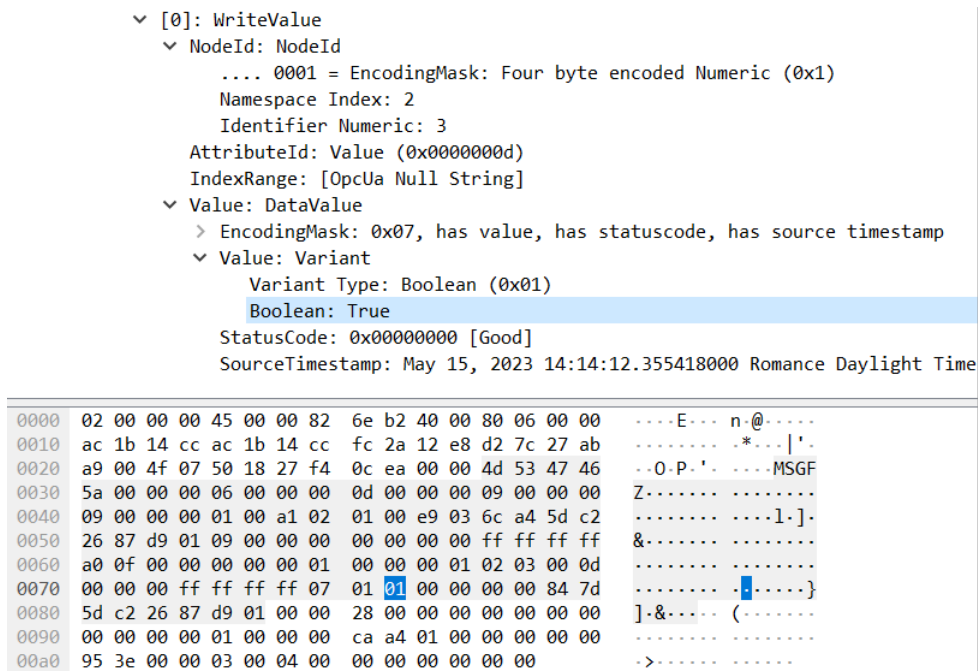


Figure C.3: WriteValue fields when Bool True value is sent.

OPC-UA uses the IEEE 754 Standard [61] for representing floating-point numbers. Figuring out how the packet size changes in this scenario is a difficult task to perform in a systematic way so we will therefore not include this built-in type our study.

The last data type that will be studied is the String. It is used to send a sequence of Unicode [62] characters. OPC-UA uses ASCII encoding to store the data in the packet. The smallest amount of data that can be sent in this case is one character. If we transmit the value 'x', we get a frame of 178 bytes and an OPC-UA packet size of 95 bytes from which 'WriteValue' field uses 31 bytes. The character is encoded in ASCII as the value '78' so it only needs 1 byte to be stored as shown in figure C.5. It is important to highlight that in this built-in data type, OPC-UA uses 4 bytes to indicate the length of the string as circled in red in the same figure.

In order to see how the packet size varies when sending a bigger amount of information, we will transmit the word 'Claudia'. The total size of the frame grows up to 184 bytes and the OPC-UA packet is 100 bytes. The field 'WriteValue' is 37 bytes long in which our data is stored in ASCII as 436C6175646961 using 7 bytes.

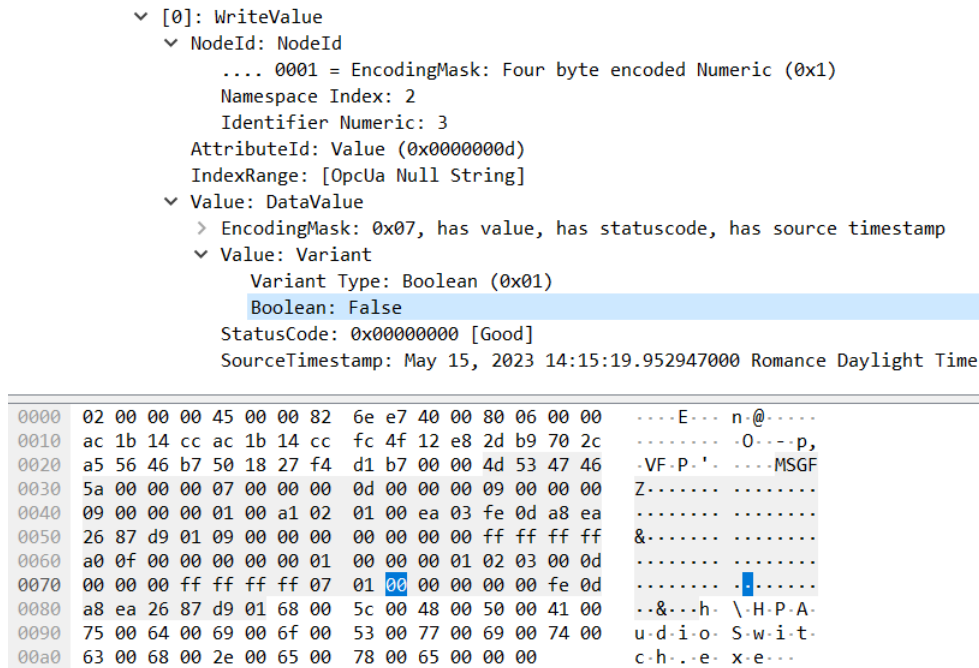


Figure C.4: WriteValue fields when Bool False value is sent.

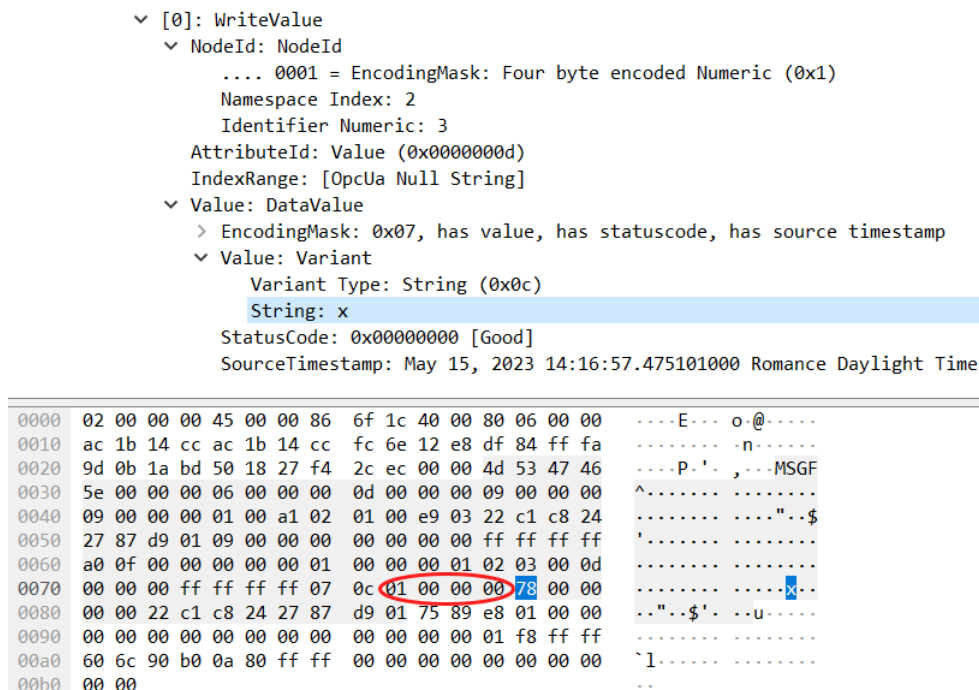


Figure C.5: WriteValue field when string character 'x' is sent.

```

    [0]: WriteValue
      NodeId: NodeId
        ... 0001 = EncodingMask: Four byte encoded Numeric (0x1)
        Namespace Index: 2
        Identifier Numeric: 3
        AttributeId: Value (0x0000000d)
        IndexRange: [OpcUa Null String]
      Value: DataValue
        > EncodingMask: 0x07, has value, has statuscode, has source timestamp
        Value: Variant
          Variant Type: String (0x0c)
          String: Claudia
          StatusCode: 0x00000000 [Good]
          SourceTimestamp: May 15, 2023 14:18:28.432746000 Romance Daylight Time
  
```

0000	02 00 00 00 45 00 00 8c	6f 51 40 00 80 06 00 00E... oQ@....
0010	ac 1b 14 cc ac 1b 14 cc	fc 8d 12 e8 81 4c e1 baL..
0020	d1 b1 b0 82 50 18 27 f4	19 5d 00 00 4d 53 47 46	...P.' .]..MSGF
0030	64 00 00 00 07 00 00 00	0d 00 00 00 09 00 00 00	d.....
0040	09 00 00 00 01 00 a1 02	01 00 ea 03 24 ca ff 5a\$.Z
0050	27 87 d9 01 09 00 00 00	00 00 00 00 ff ff ff ff	'.....
0060	a0 0f 00 00 00 00 00 01	00 00 00 01 02 03 00 0d
0070	00 00 00 ff ff ff ff 07	0c 07 00 00 00 43 6c 61Cla
0080	75 64 69 61 00 00 00 00	24 ca ff 5a 27 87 d9 01	udia....\$.Z'...
0090	08 e7 62 a5 0a 80 ff ff	08 e7 62 a5 0a 80 ff ff	..b....-b....
00a0	01 05 03 00 0a 80 ff ff	80 a0 89 a5 0a 80 ff ff
00b0	00 e7 62 a5 0a 80 ff ff		..b....

Figure C.6: WriteValue field when string value 'Claudia' is sent.

C.2.- MQTT

With MQTT we will study Int64, Float and String values since there is not an special built-in type for boolean data. Although this protocol encodes all its binary data using ASCII, there is something to do if we want to reduce the packet size when sending integer and floating values. Numbers can be encoded as HEX bytes instead of treating them as strings (which would encode them in ASCII).

We start by sending the integer value '123456789' to the topic "testn". The whole frame is 84 bytes and MQTT packet is 18 bytes. Payload is stored using 9 bytes and it is encoded in ASCII as 313233343536373839.

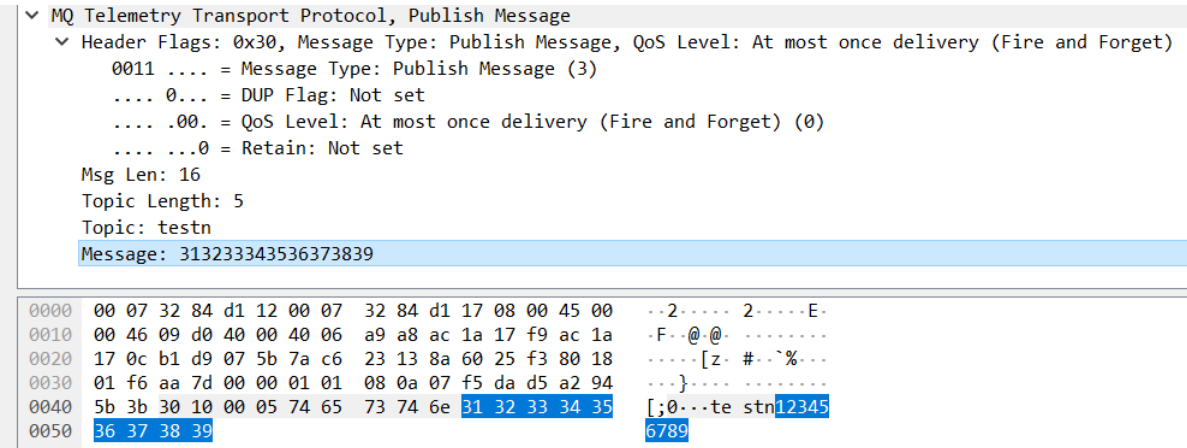


Figure C.7: Content of the MQTT packet when sending the value '123456789' as a string.

If we send the same data but encoded as HEX for integer values, we get the results shown in figure . The topic used to publish is the same size as the prior one and its called "testh". Here the full frame size is 79 bytes and the MQTT packet size is 13 bytes. Payload is encoded as 075BCD15₁₆ using 4 bytes.

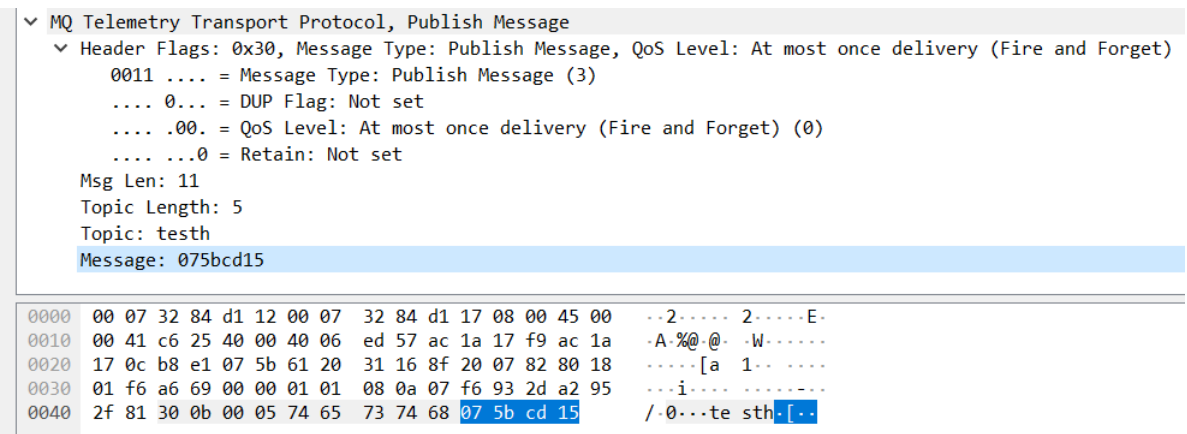


Figure C.8: Content of the MQTT packet when sending the value '123456789' as an integer.

When it comes to floating values it is also possible to reduce packet size by following a similar algorithm. A floating point number is formed by two integer values separated by a dot. If we encode each integer part using HEX and the dot using ASCII, it is possible to gather them and send it all as the packet's payload.

However, there are some details we have to take care of. It is important to know how to deal with the dot, encoded as 2E in ASCII, when sending values such in which number 46 is next to the dot since it is also encoded as 2E₁₆.

Figure C.11 displays the result of sending the value 3.14159 as text. Frame size is 83 bytes. ASCII packet is 16 bytes in which the payload is encoded using 7 bytes.

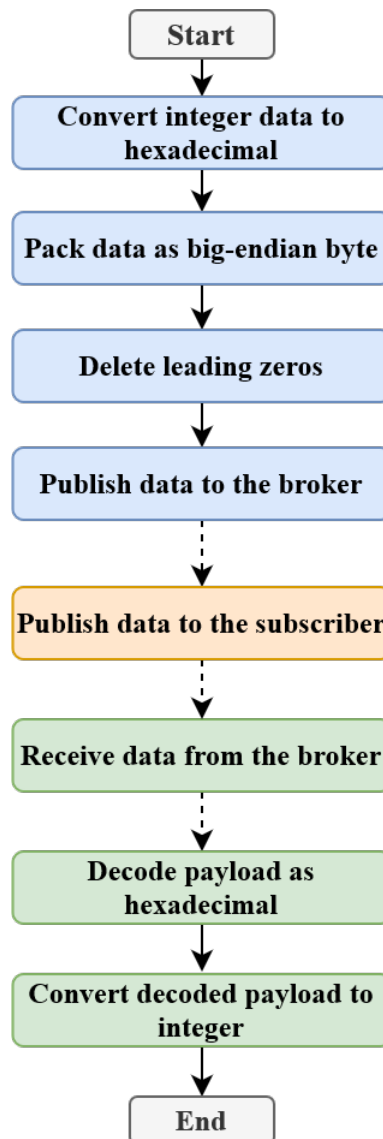


Figure C.9: Flowchart of the process of sending integer data encoded as HEX.

The flowchart of the algorithm used to reduce payload size when sending integer data is shown in figure C.9 . Blue steps are executed in Node1, orange ones in the broker and green ones in Node2.

If we now send the same value encoding it as explained above, we get the result showed in figure C.12 . This time the frame is 79 bytes and the size of the ASCII packet is 13 bytes. Payload is encoded as 032E374F using 4 bytes.

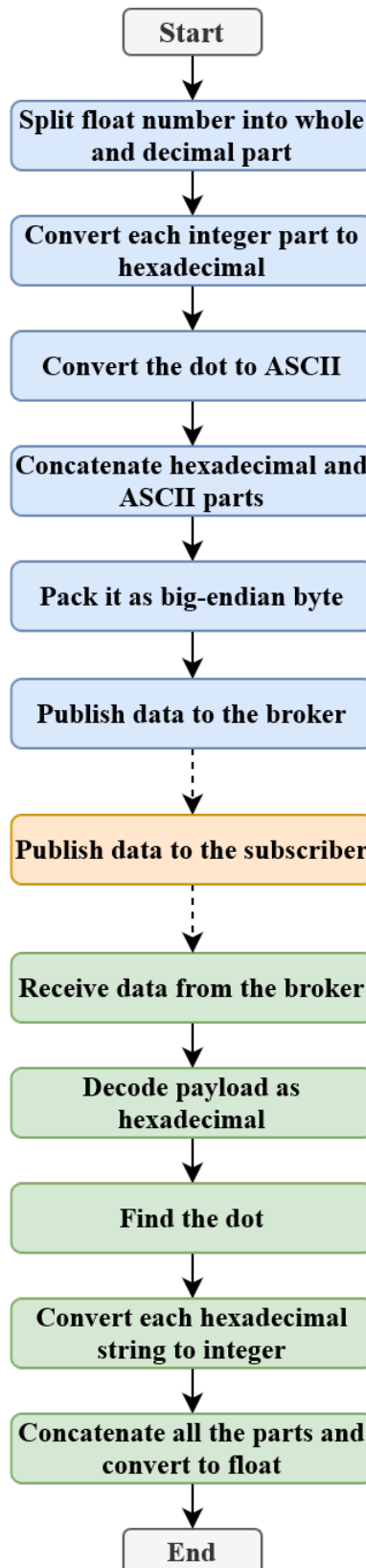


Figure C.10: Flowchart for the process of sending float data encoded as HEX.

```

MQ Telemetry Transport Protocol, Publish Message
  Header Flags: 0x30, Message Type: Publish Message, QoS Level: At most once delivery (Fire and Forget)
    0011 .... = Message Type: Publish Message (3)
    .... 0... = DUP Flag: Not set
    .... .00. = QoS Level: At most once delivery (Fire and Forget) (0)
    .... ...0 = Retain: Not set
  Msg Len: 14
  Topic Length: 5
  Topic: testn
  Message: 332e3134313539

```

```

0000 00 07 32 84 d1 12 00 07 32 84 d1 17 08 00 45 00  --2..... 2.....E-
0010 00 44 13 4a 40 00 40 06 a0 30 ac 1a 17 f9 ac 1a  -D.J@.@. -0.....
0020 17 0c 98 4b 07 5b 96 e3 c8 f2 6d 02 48 3d 80 18  ...K.[. .m.H=-.
0030 01 f6 9f fc 00 00 01 01 08 0a 07 fd 24 60 a2 9a  .....$`..
0040 ac 08 30 0e 00 05 74 65 73 74 6e 33 2e 31 34 31  ..0...te stn3.141
0050 35 39 59

```

Figure C.11: Content of the MQTT packet when sending the value '3.14159' as a string.

```

MQ Telemetry Transport Protocol, Publish Message
  Header Flags: 0x30, Message Type: Publish Message, QoS Level: At most once delivery (Fire and Forget)
    0011 .... = Message Type: Publish Message (3)
    .... 0... = DUP Flag: Not set
    .... .00. = QoS Level: At most once delivery (Fire and Forget) (0)
    .... ...0 = Retain: Not set
  Msg Len: 11
  Topic Length: 5
  Topic: testf
  Message: 032e374f

```

```

0000 00 07 32 84 d1 12 00 07 32 84 d1 17 08 00 45 00  --2..... 2.....E-
0010 00 41 d0 b5 40 00 40 06 e2 c7 ac 1a 17 f9 ac 1a  -A.-@.@. ....
0020 17 0c c2 cd 07 5b 7e 49 69 a5 bc 33 09 3f 80 18  ....[~I i..3?..
0030 01 f6 38 d2 00 00 01 01 08 0a 07 f8 00 79 a2 96  --8..... .y..
0040 96 ef 30 0b 00 05 74 65 73 74 66 03 2e 37 4f  ..0...te stf.70

```

Figure C.12: Content of the MQTT packet when sending the value '3.14159' as an integer.

Finally, sending different string sizes leads to the results presented in figures C.13 and C.14. In the first case the packet's payload is the character 'x'. It results on a 76 byte frame. MQTT packet is 10 bytes and the data is stored using one byte. For the scenario in which the payload is the word 'Claudia', a 82 bytes frame is needed in order to transmit the data. MQTT's packet size is 16 bytes and the payload is stored using 7 bytes.

```

MQ Telemetry Transport Protocol, Publish Message
  Header Flags: 0x30, Message Type: Publish Message, QoS Level: At most once delivery (Fire and Forget)
    0011 .... = Message Type: Publish Message (3)
    .... 0... = DUP Flag: Not set
    .... .00. = QoS Level: At most once delivery (Fire and Forget) (0)
    .... ...0 = Retain: Not set
  Msg Len: 8
  Topic Length: 5
  Topic: testn
  Message: 78

```

```

0000 00 07 32 84 d1 12 00 07 32 84 d1 17 08 00 45 00  --2.... 2....E-
0010 00 3e 15 4a 40 00 40 06 9e 36 ac 1a 17 f9 ac 1a  ->J@.@. -6.....
0020 17 0c a0 61 07 5b 34 fd 9c e9 be 50 d4 23 80 18  ...a-[4. ...P.#..
0030 01 f6 de c4 00 00 01 01 08 0a 07 ff 0a 59 a2 9d  ....Y..
0040 c7 49 30 08 00 05 74 65 73 74 6e 78             -I0...te stnx

```

Figure C.13: Content of the MQTT packet when sending the value 'x'.

```

MQ Telemetry Transport Protocol, Publish Message
  Header Flags: 0x30, Message Type: Publish Message, QoS Level: At most once delivery (Fire and Forget)
    0011 .... = Message Type: Publish Message (3)
    .... 0... = DUP Flag: Not set
    .... .00. = QoS Level: At most once delivery (Fire and Forget) (0)
    .... ...0 = Retain: Not set
  Msg Len: 14
  Topic Length: 5
  Topic: testn
  Message: 436c6175646961

```

```

0000 00 07 32 84 d1 12 00 07 32 84 d1 17 08 00 45 00  --2.... 2....E-
0010 00 44 3e a5 40 00 40 06 74 d5 ac 1a 17 f9 ac 1a  -D>.@.@. t.....
0020 17 0c af 7d 07 5b 54 a4 95 aa 80 53 08 7c 80 18  ...}-[T. ...S:|..
0030 01 f6 a8 63 00 00 01 01 08 0a 07 ff f8 8e a2 9e  ...c....
0040 a6 96 30 0e 00 05 74 65 73 74 6e 43 6c 61 75 64  ..0...te stnClaud
0050 69 61                                             ia

```

Figure C.14: Content of the MQTT packet when sending the value 'Claudia'.

D. Closed Control Loop Latency plots

D.1.- CCLL for MQTT over Ethernet: QoS comparison

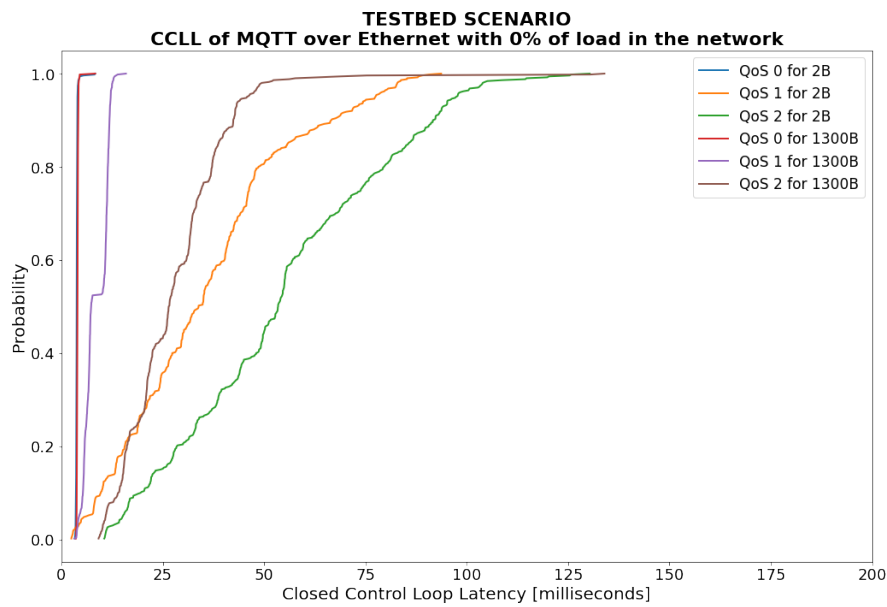


Figure D.1: CCLL for testbed scenario: MQTT over Ethernet and 0% of load in the network.

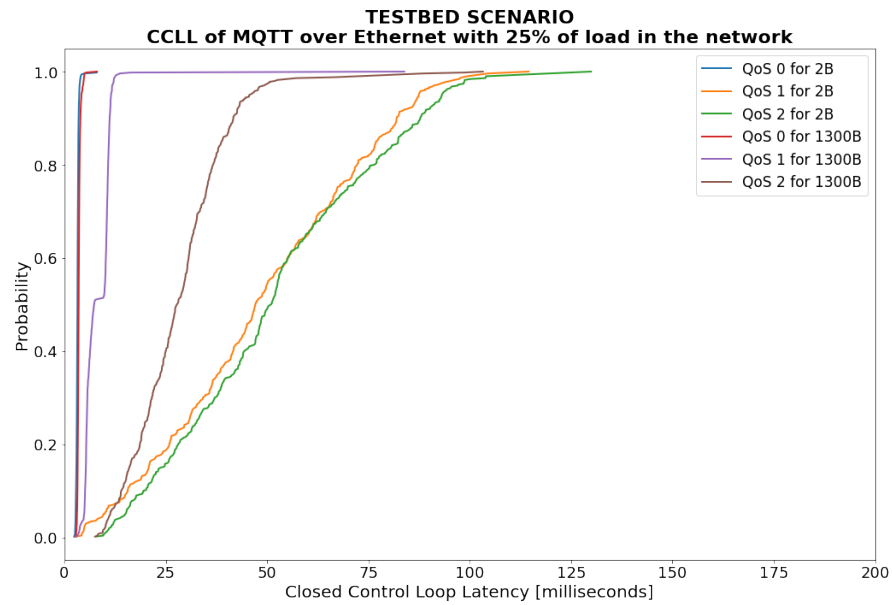


Figure D.2: Cumulative Distribution Function (CDF) for MQTT over Ethernet and 25 % load in the network.

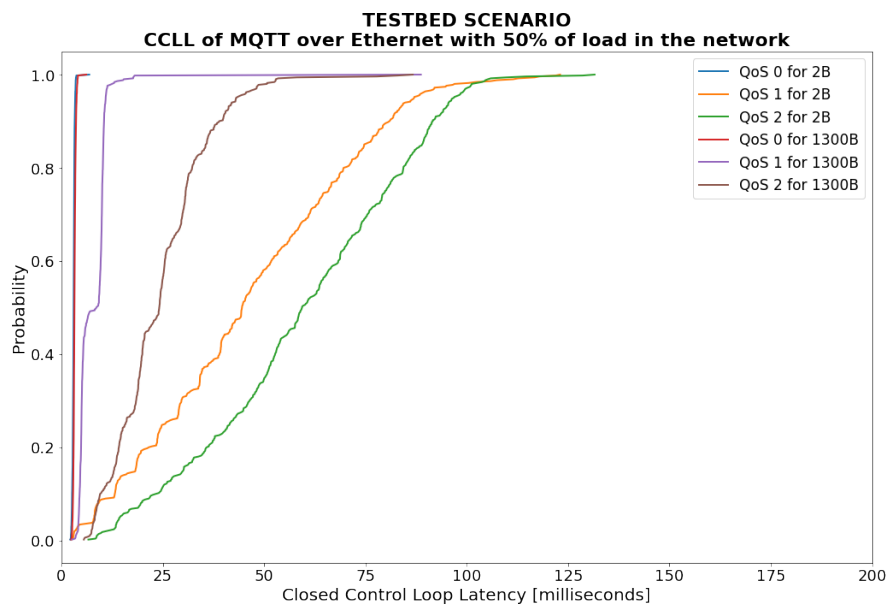


Figure D.3: CDF for MQTT over Ethernet and 50% load in the network.

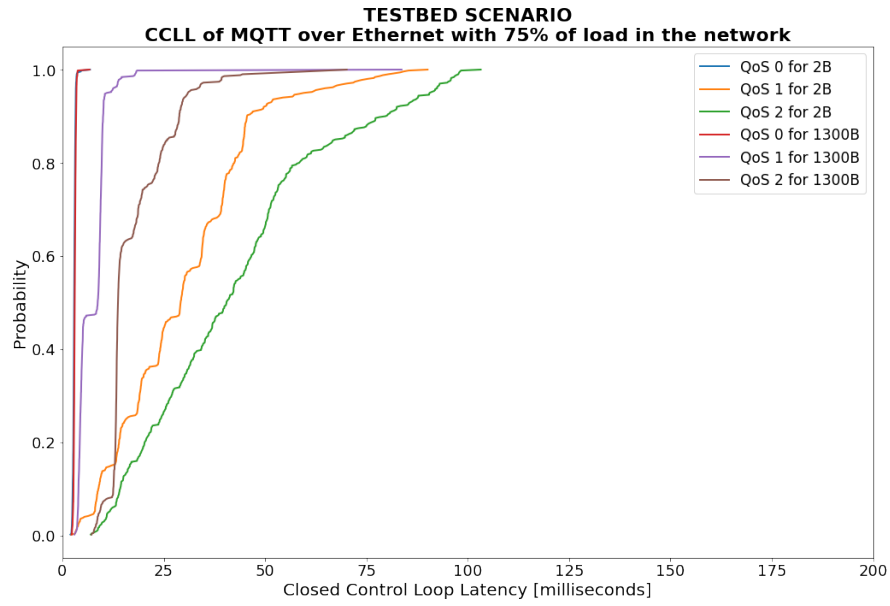


Figure D.4: CDF for MQTT over Ethernet and 75 % load in the network.

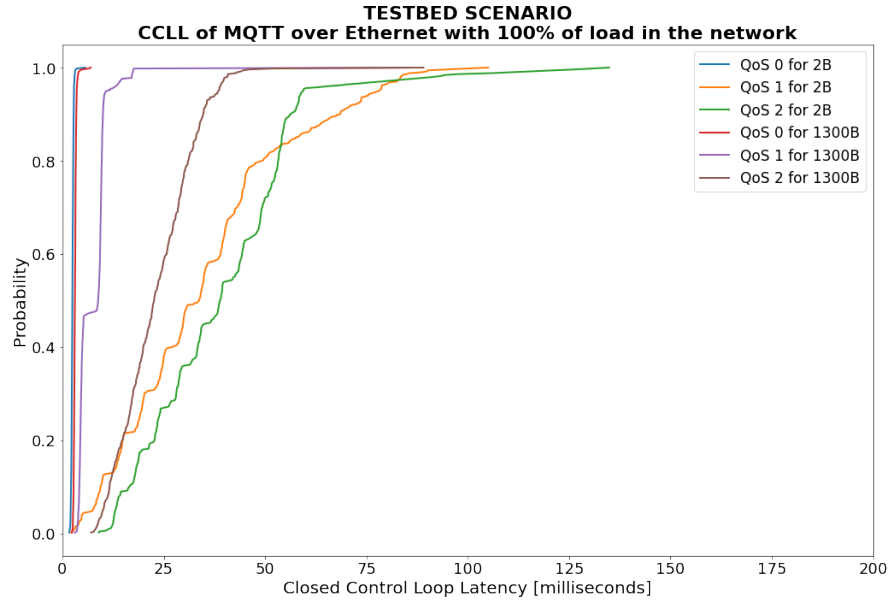


Figure D.5: CDF for MQTT over Ethernet and 100 % load in the network.

D.2.- CCLL for MQTT over Ethernet: Network load study

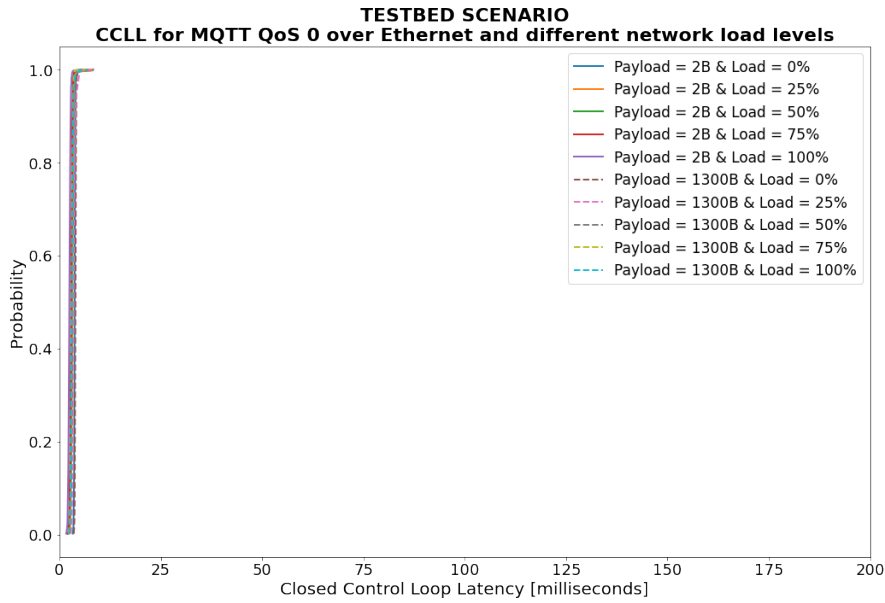


Figure D.6: CDF for MQTT QoS 0 over Ethernet and different network load levels.

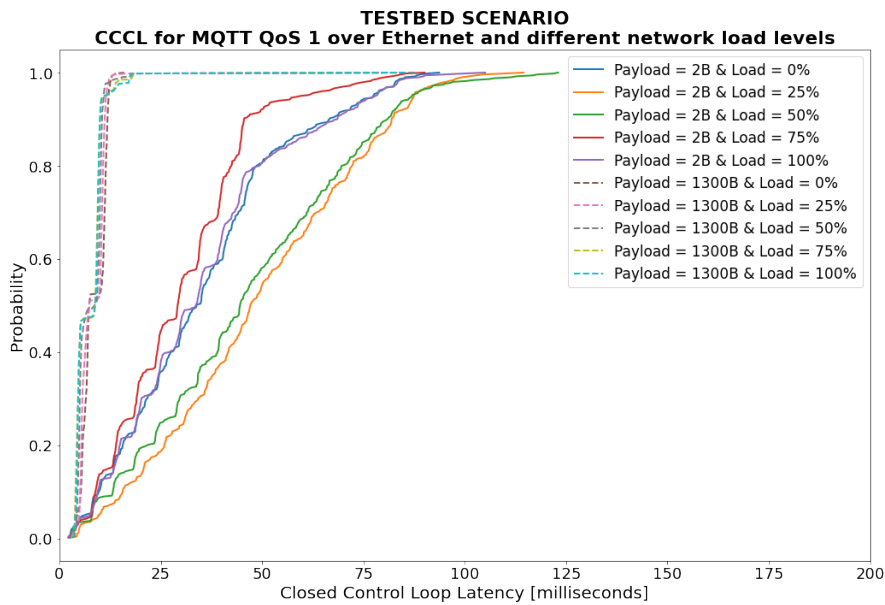


Figure D.7: CDF for MQTT QoS 1 over Ethernet and different network load levels.

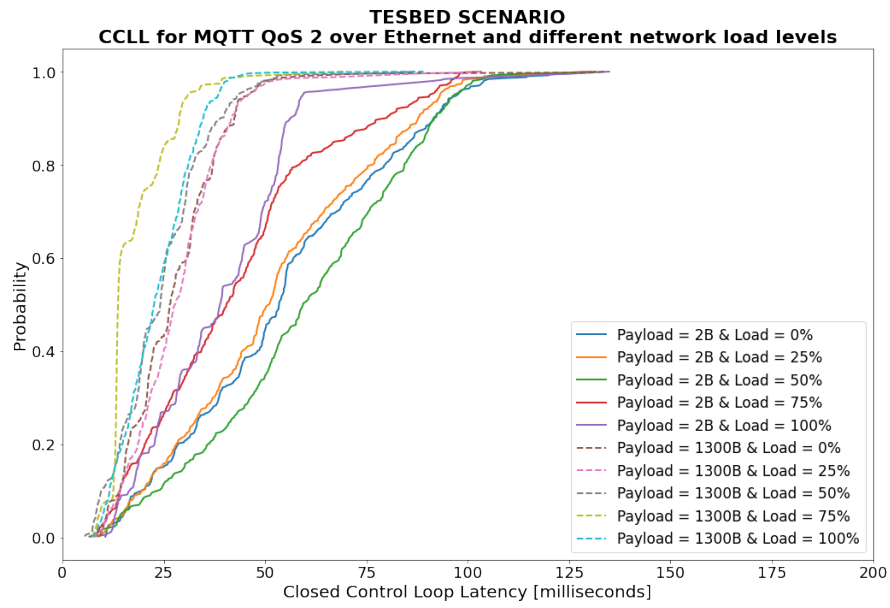


Figure D.8: CDF for MQTT QoS 2 over Ethernet and different network load levels.

D.3.- CCLL for MQTT-SN over Ethernet

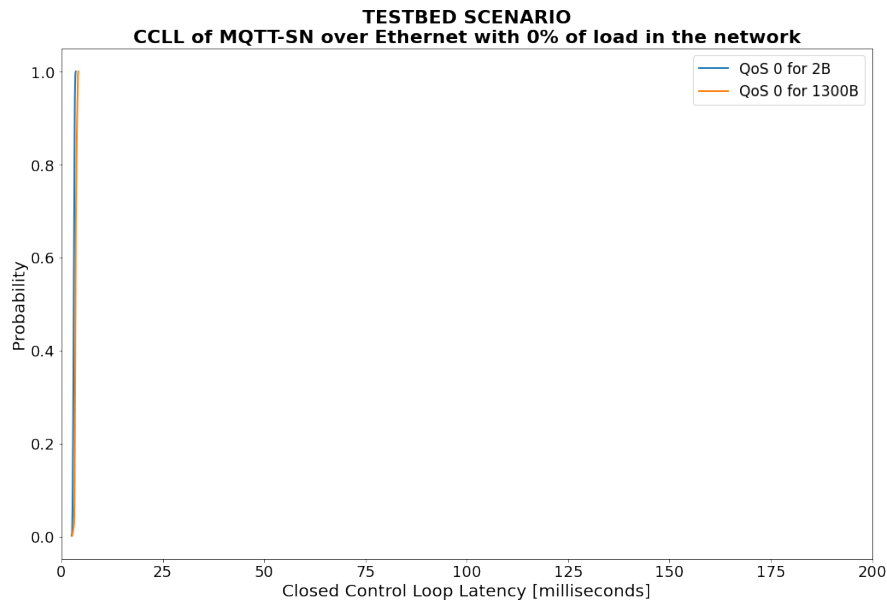


Figure D.9: CDF for MQTT-SN over Ethernet and 0% load in the network.

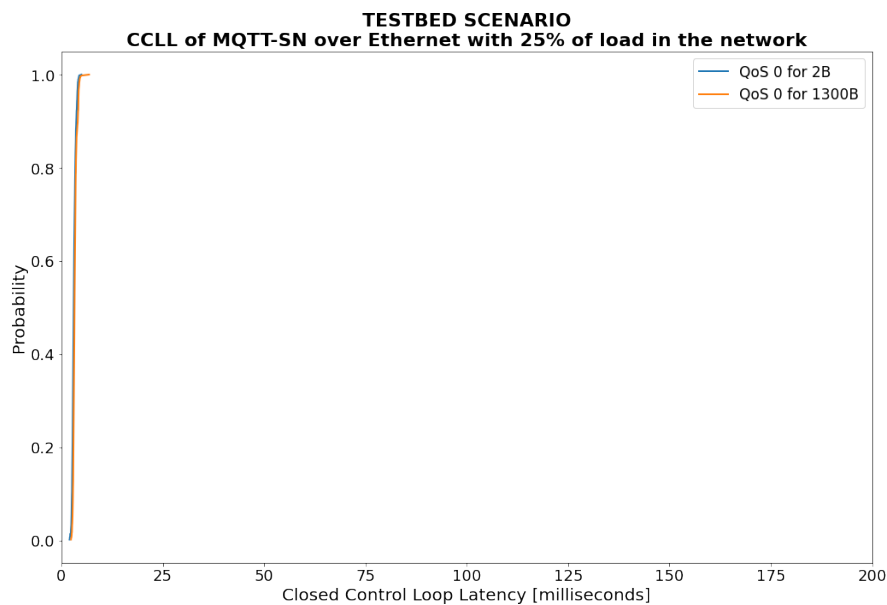


Figure D.10: CDF for MQTT-SN over Ethernet and 25% load in the network.

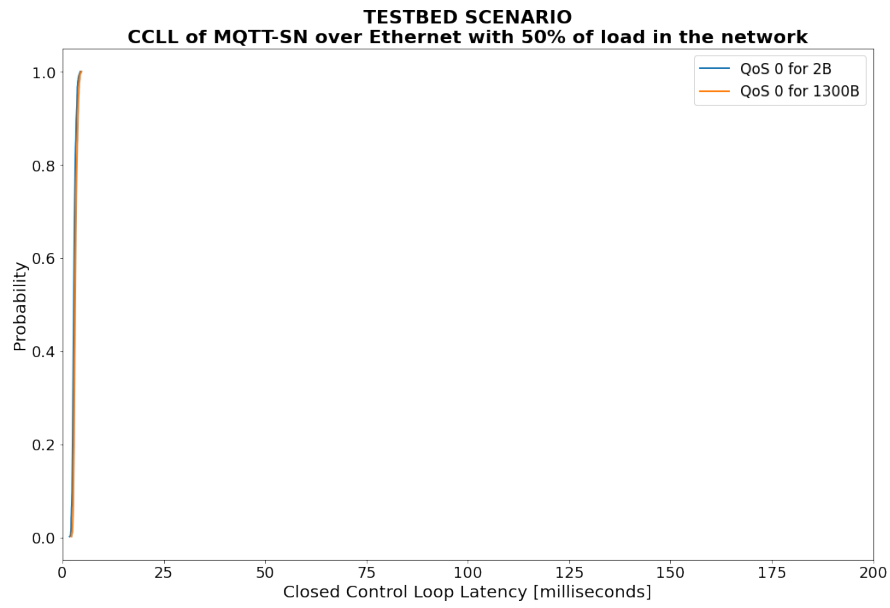


Figure D.11: CDF for MQTT-SN over Ethernet and 50% load in the network.

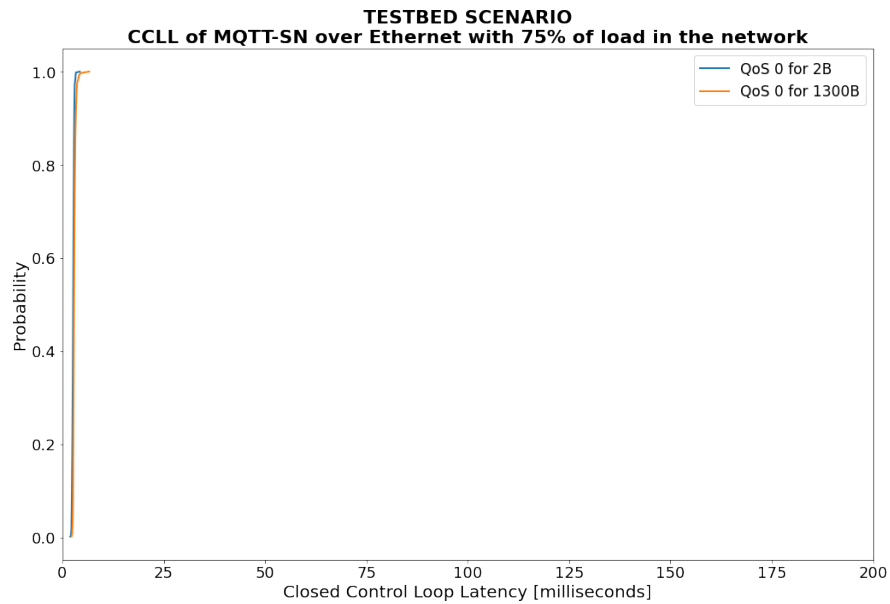


Figure D.12: CDF for MQTT-SN over Ethernet and 75% load in the network.

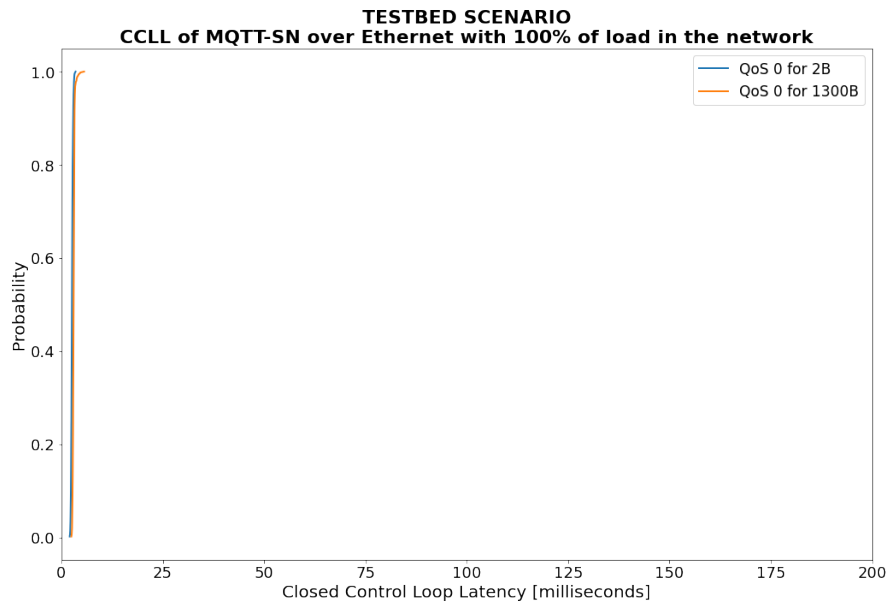


Figure D.13: CDF for MQTT-SN over Ethernet and 100 % load in the network.

D.4.- CCLL for MQTT-SN over Ethernet: Network load study

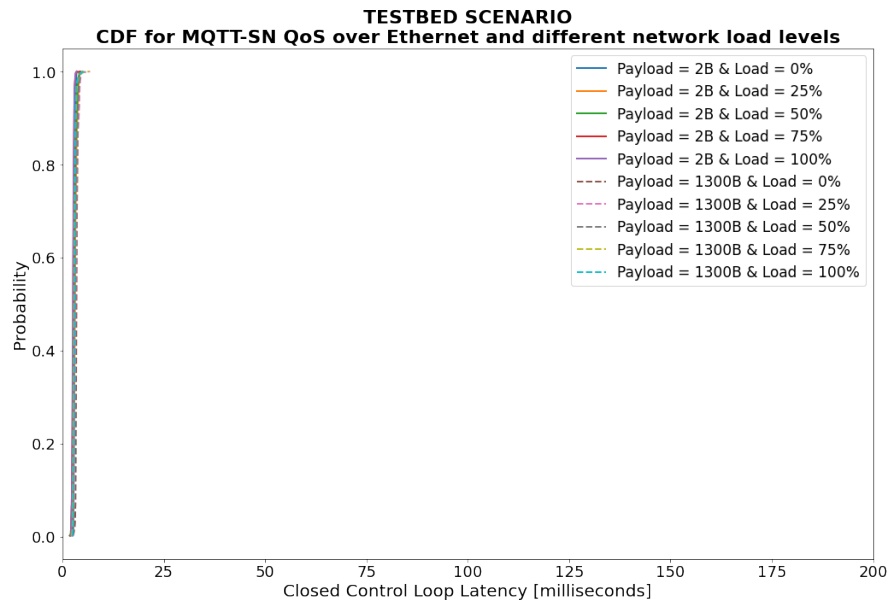


Figure D.14: CDF for MQTT-SN QoS 0 over Ethernet and different network load levels.

D.5.- CCLL for OPC-UA over Ethernet

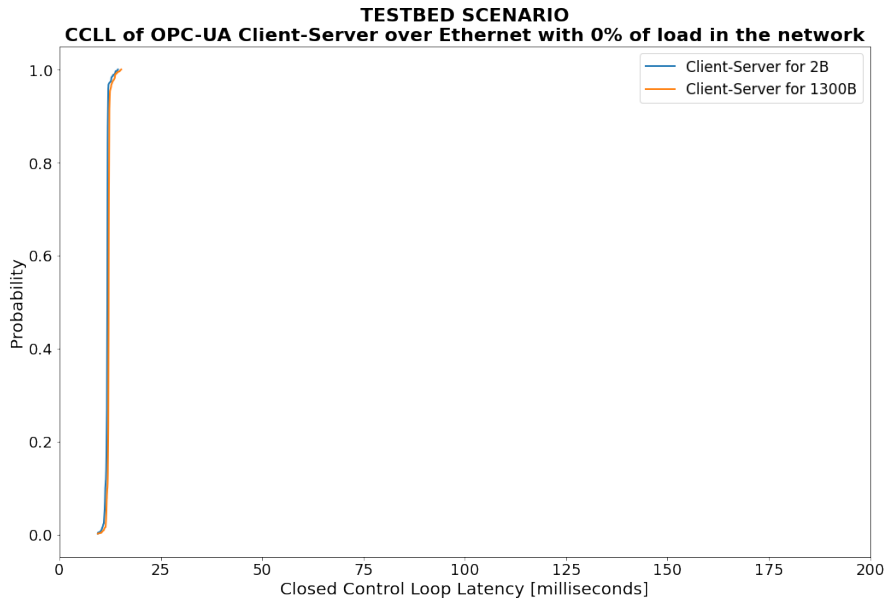


Figure D.15: CDF for OPC-UA Client-Server over Ethernet and 0 % load in the network.

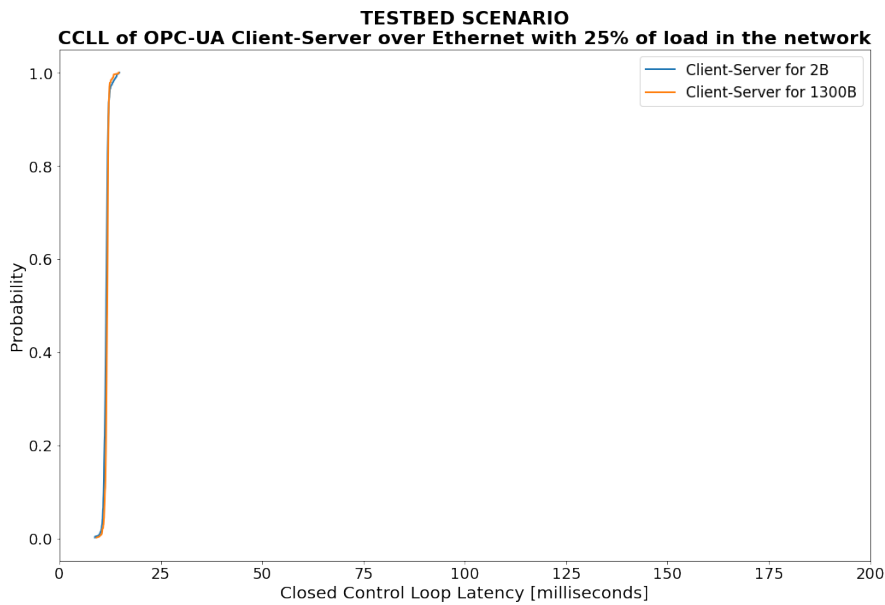


Figure D.16: CDF for OPC-UA Client-Server over Ethernet and 25 % load in the network.

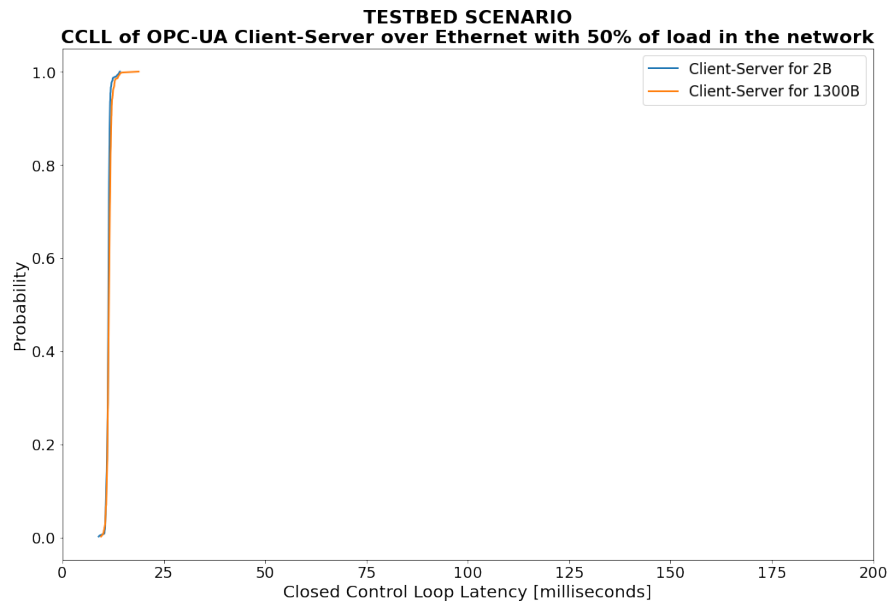


Figure D.17: CDF for OPC-UA Client-Server over Ethernet and 50% load in the network.

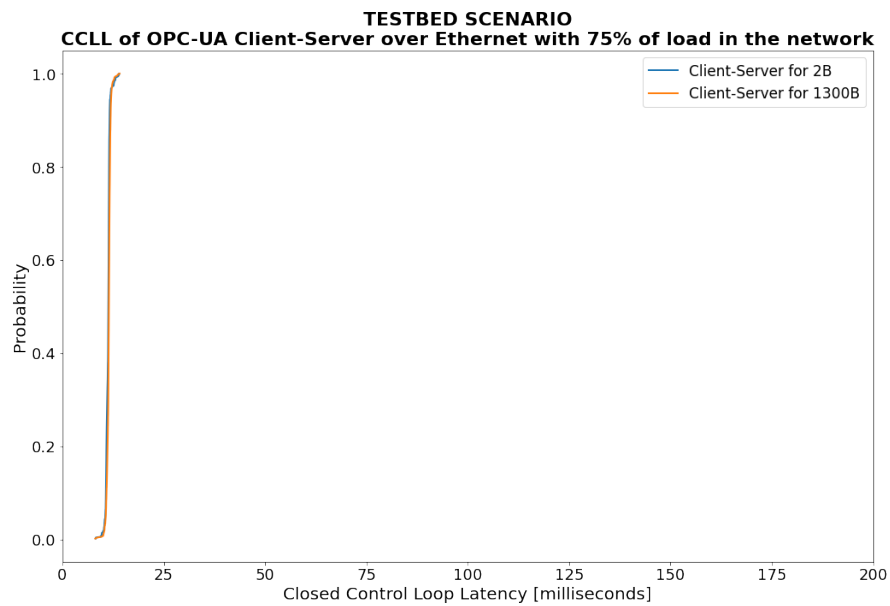


Figure D.18: CDF for OPC-UA Client-Server over Ethernet and 75% load in the network.

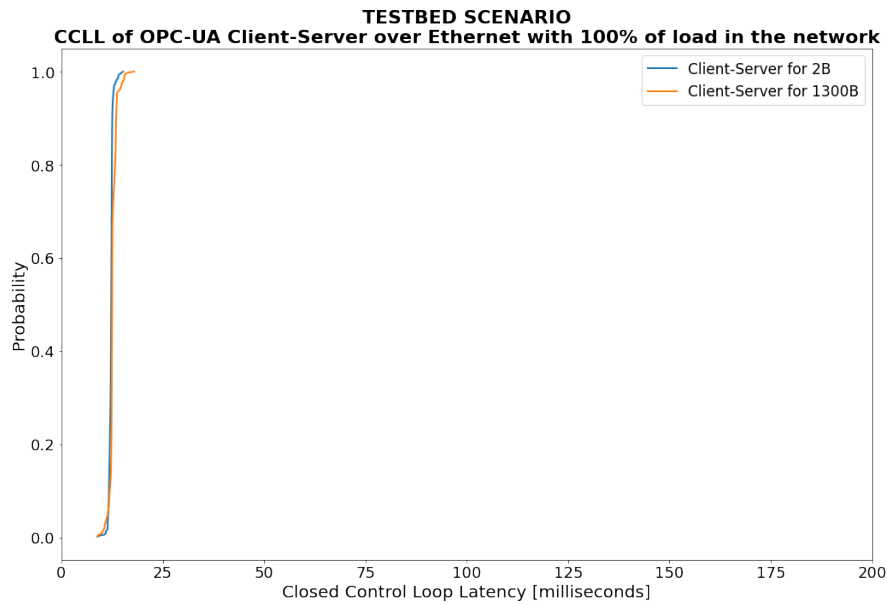


Figure D.19: CDF for OPC-UA Client-Server over Ethernet and 100 % load in the network.

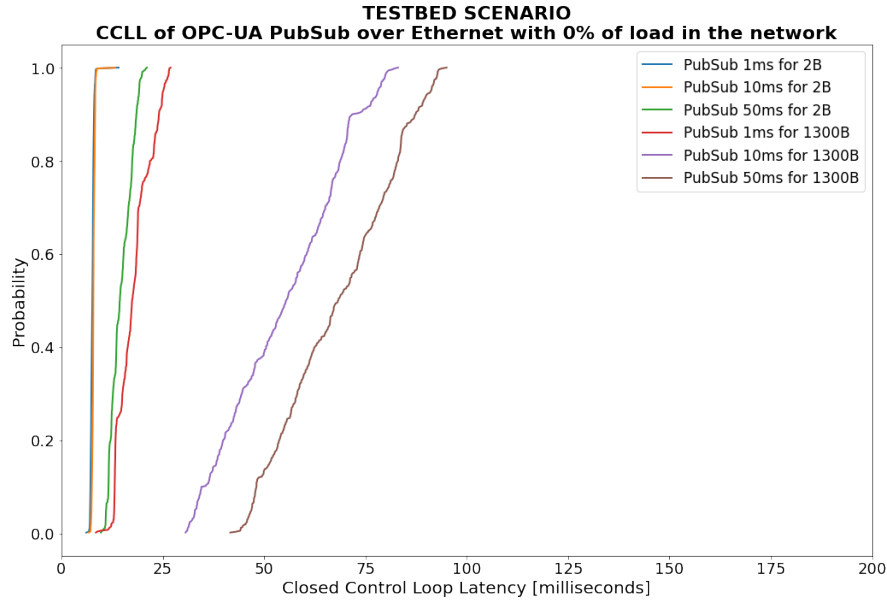


Figure D.20: CDF for OPC-UA PubSub over Ethernet and 0 % load in the network.

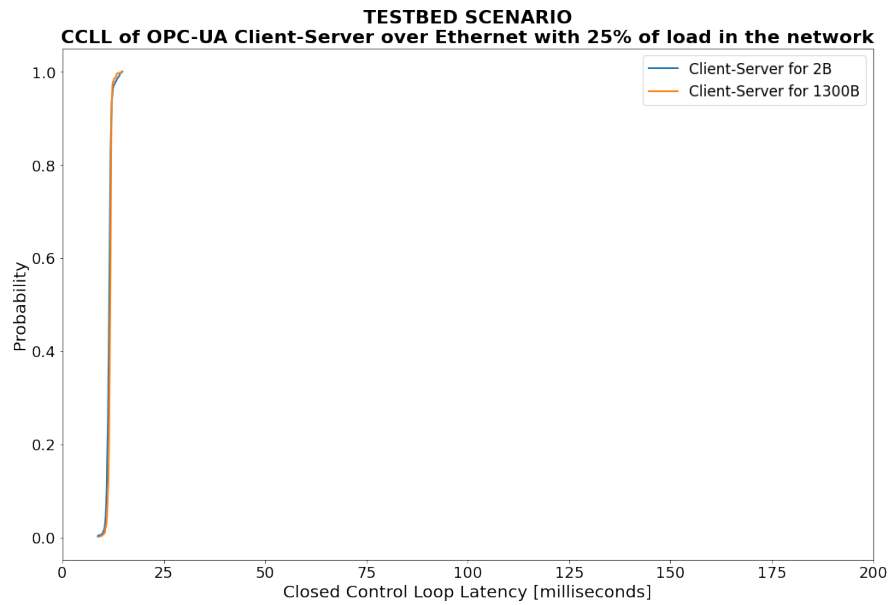


Figure D.21: CDF for OPC-UA PubSub over Ethernet and 25% load in the network.

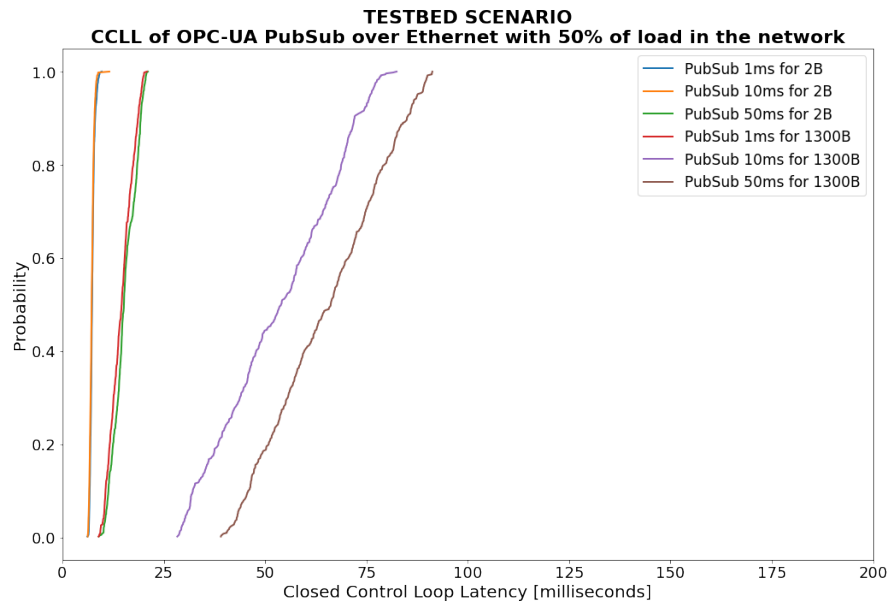


Figure D.22: CDF for OPC-UA PubSub over Ethernet and 50% load in the network.

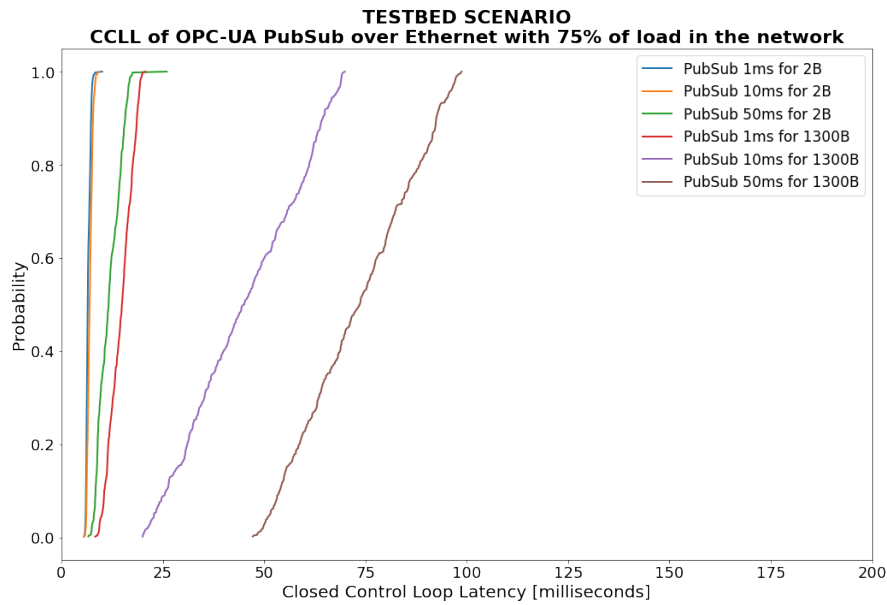


Figure D.23: CDF for OPC-UA PubSub over Ethernet and 75 % load in the network.

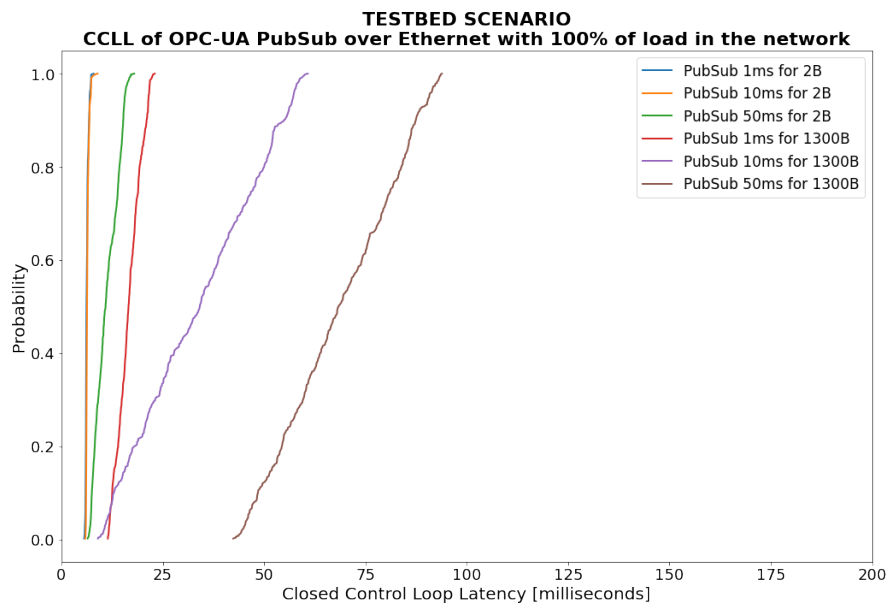


Figure D.24: CDF for OPC-UA PubSub over Ethernet and 100 % load in the network.

D.6.- CCLL for OPC-UA over Ethernet: Network load study

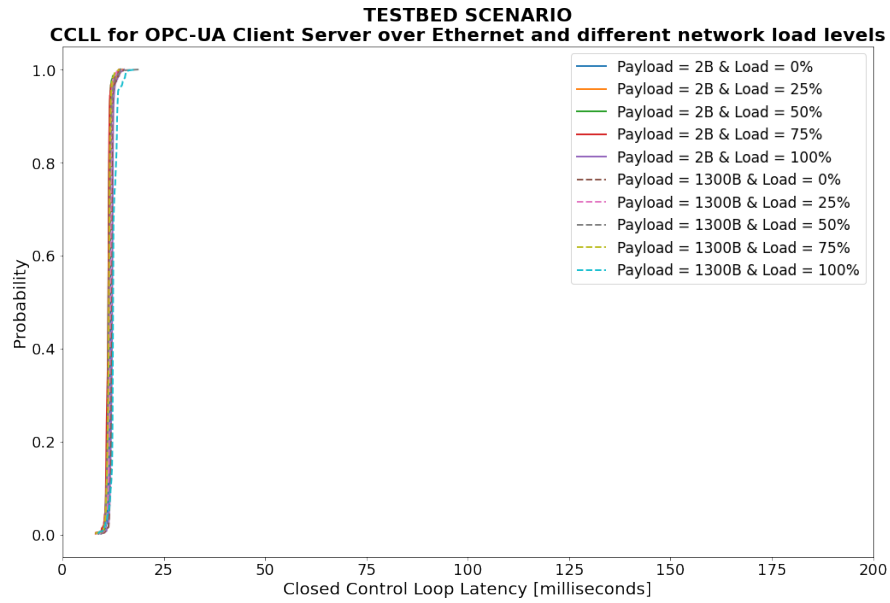


Figure D.25: CDF for OPC-UA Client-Server over Ethernet and different load levels.

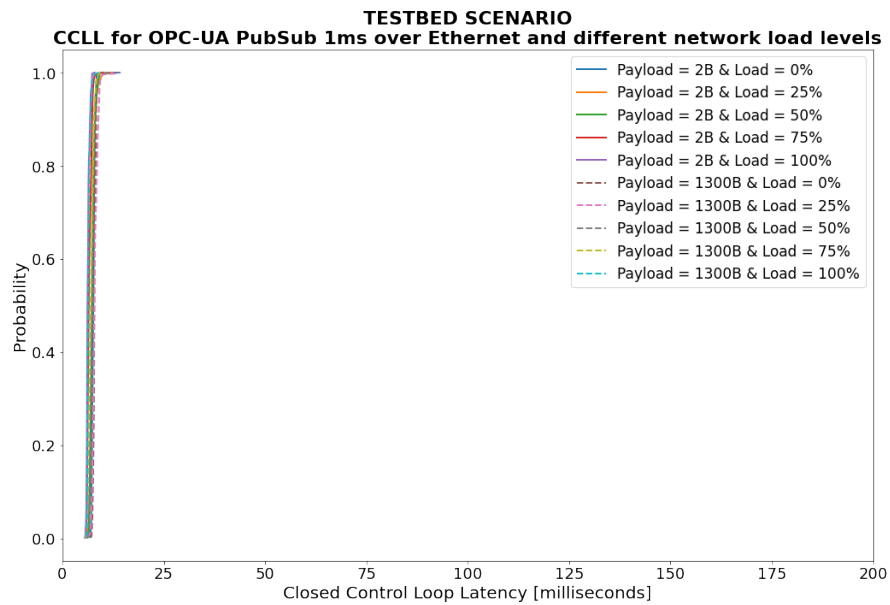


Figure D.26: CDF for OPC-UA PubSub 1ms over Ethernet and different load levels.

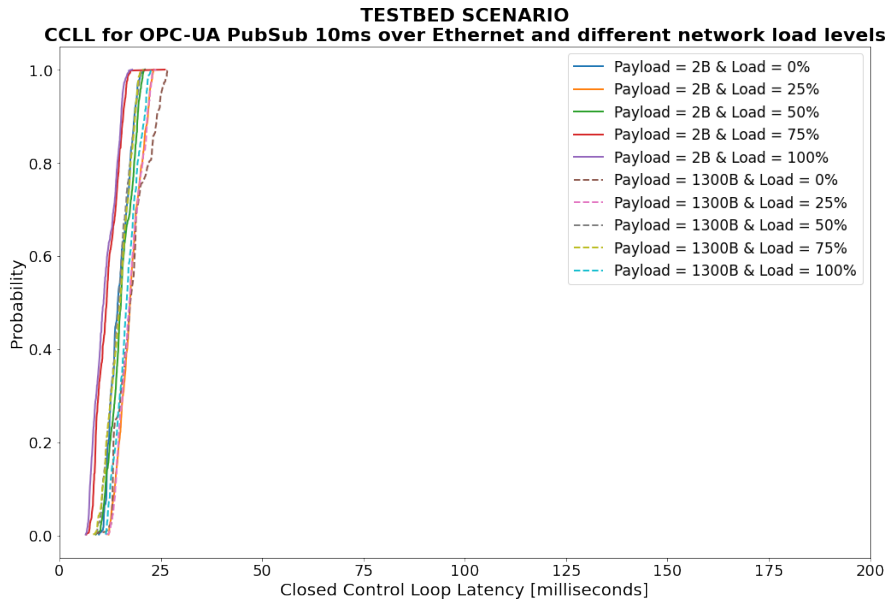


Figure D.27: CDF for OPC-UA PubSub 10ms over Ethernet and different load levels.

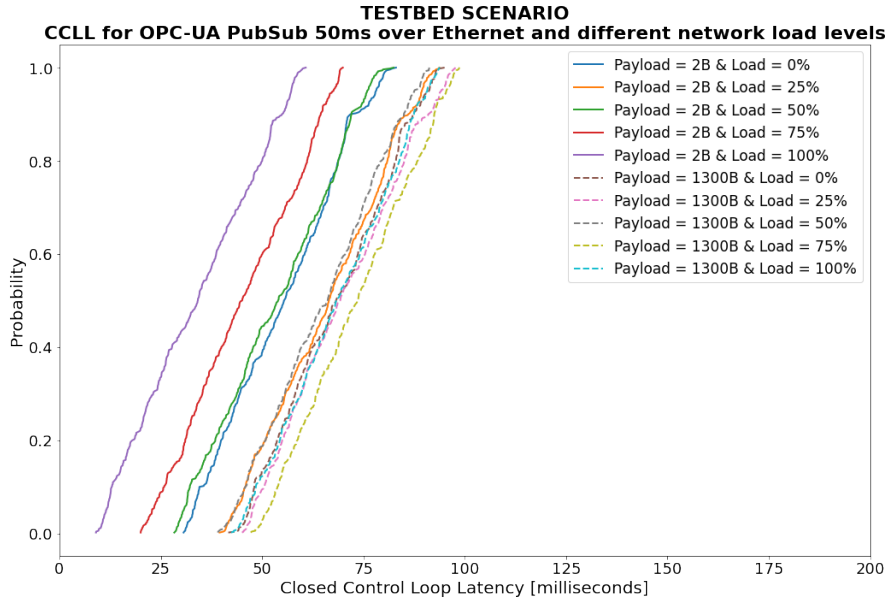


Figure D.28: CDF for OPC-UA PubSub 50ms over Ethernet and different load levels.

D.7.- CCLL for MQTT over 5G: Testbed scenario

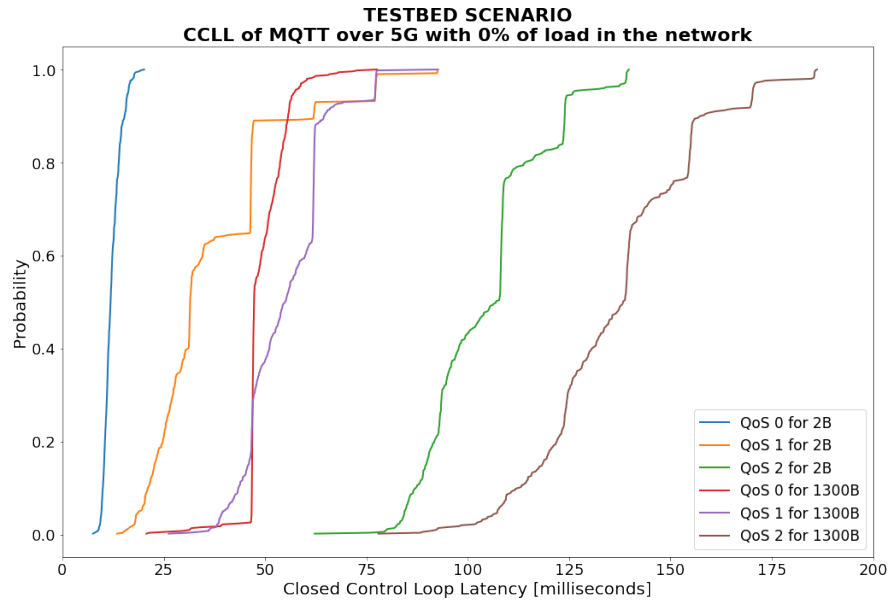


Figure D.29: CDF for MQTT over 5G and 0% load in the network.

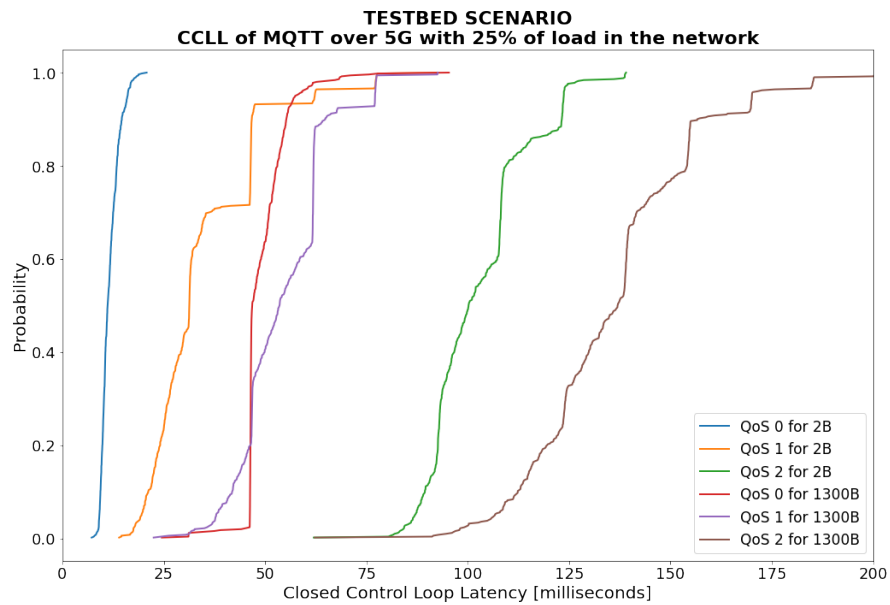


Figure D.30: CDF for MQTT over 5G and 25% load in the network.

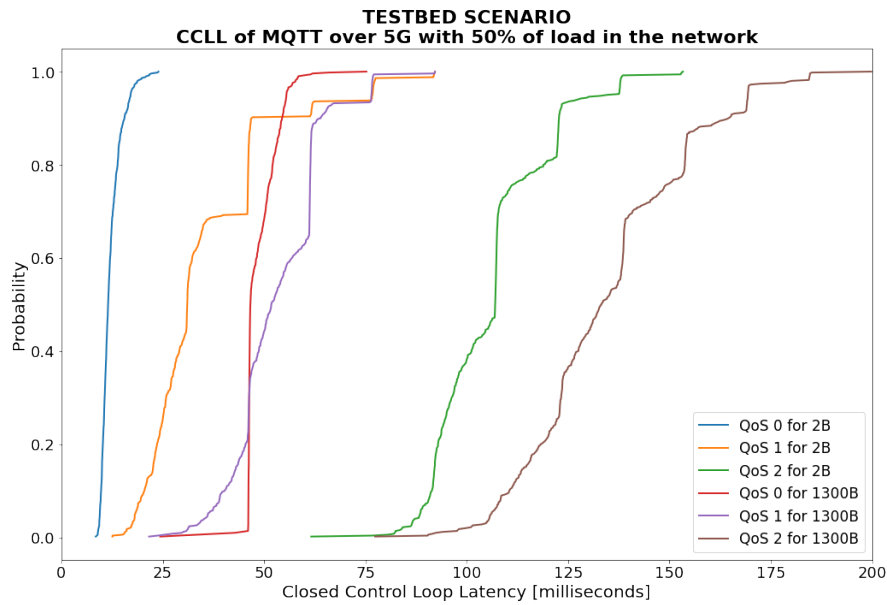


Figure D.31: CDF for MQTT over 5G and 50% load in the network.

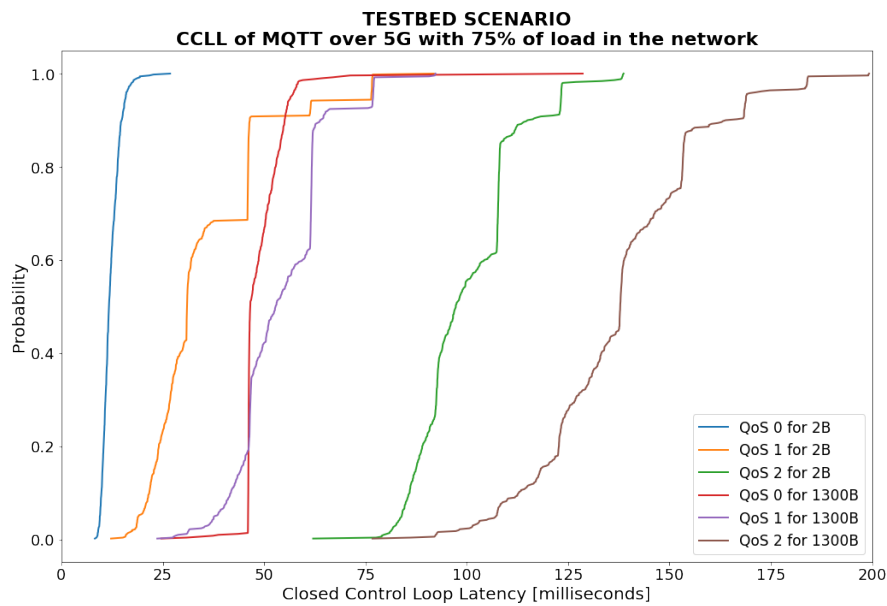


Figure D.32: CDF for MQTT over 5G and 75% load in the network.

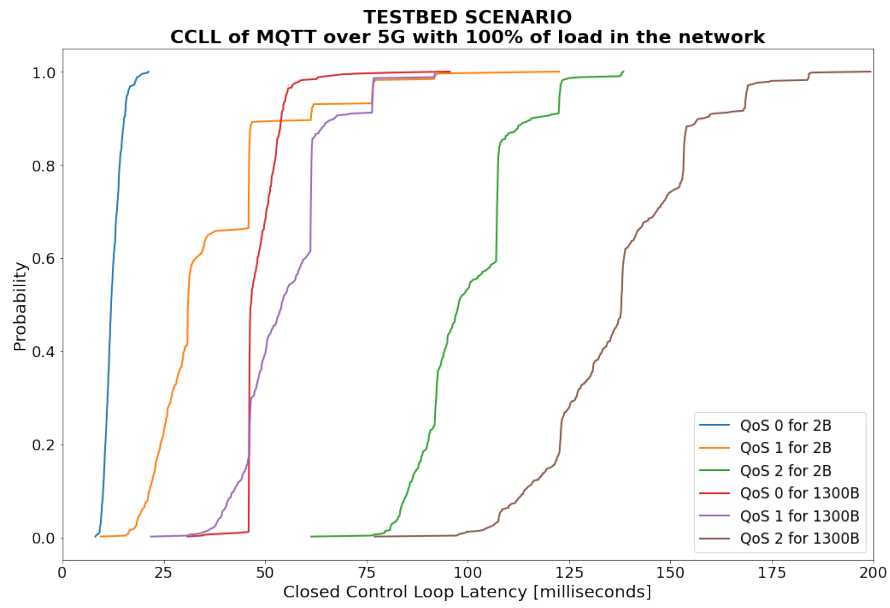


Figure D.33: CDF for MQTT over 5G and 100% load in the network.

D.8.- CCLL for MQTT over 5G: Network load study (testbed)

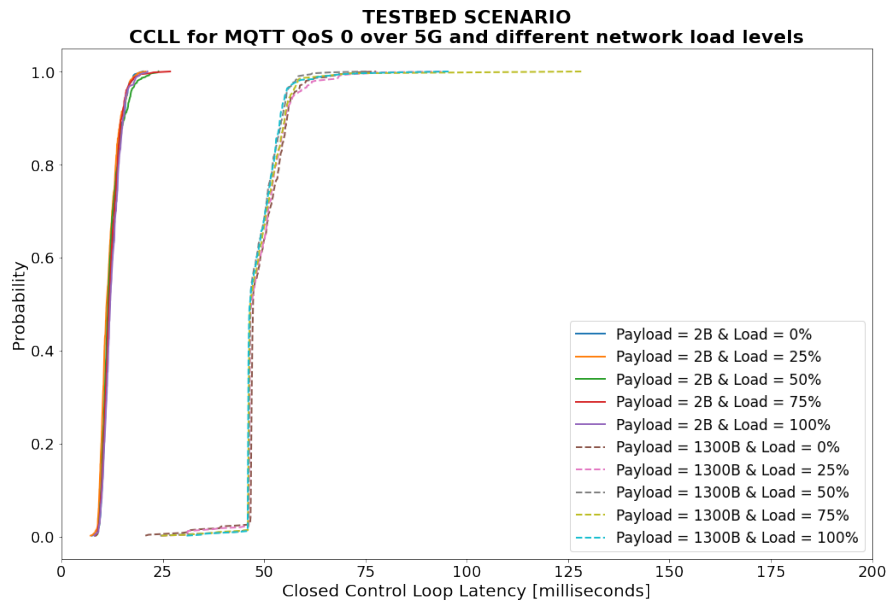


Figure D.34: CDF for MQTT QoS 0 over 5G and different load levels (testbed).

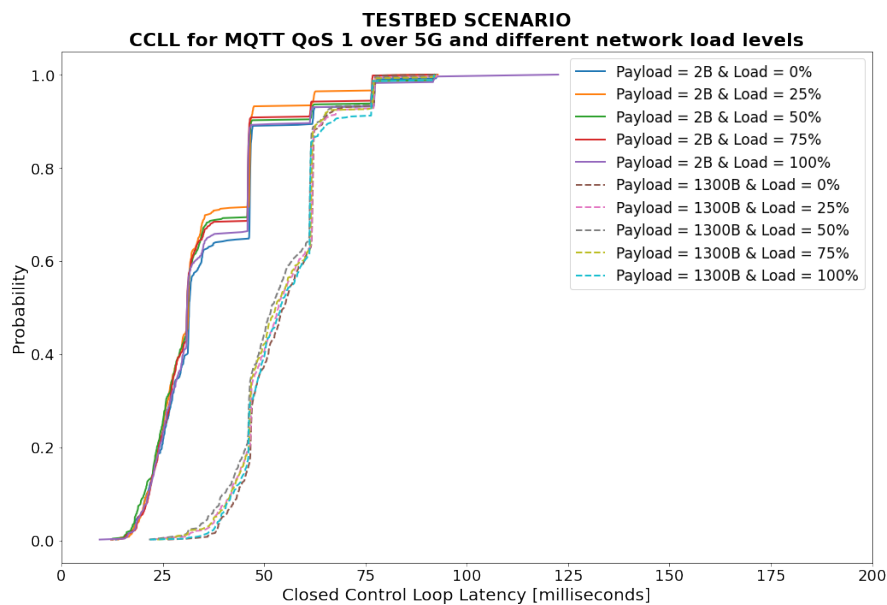


Figure D.35: CDF for MQTT QoS 1 over 5G and different load levels (testbed).

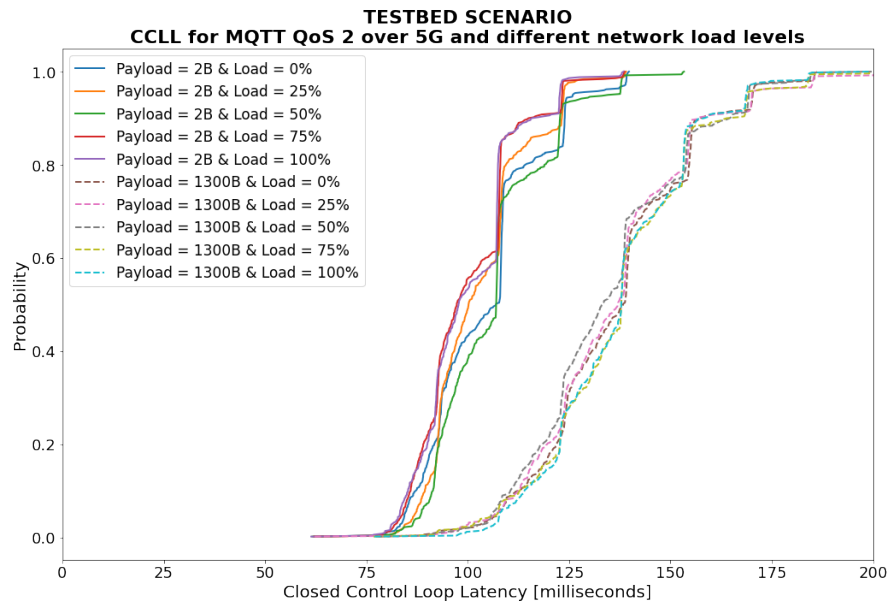


Figure D.36: CDF for MQTT QoS 2 over 5G and different load levels (testbed).

D.9.- CCLL for MQTT-SN over 5G: Testbed scenario

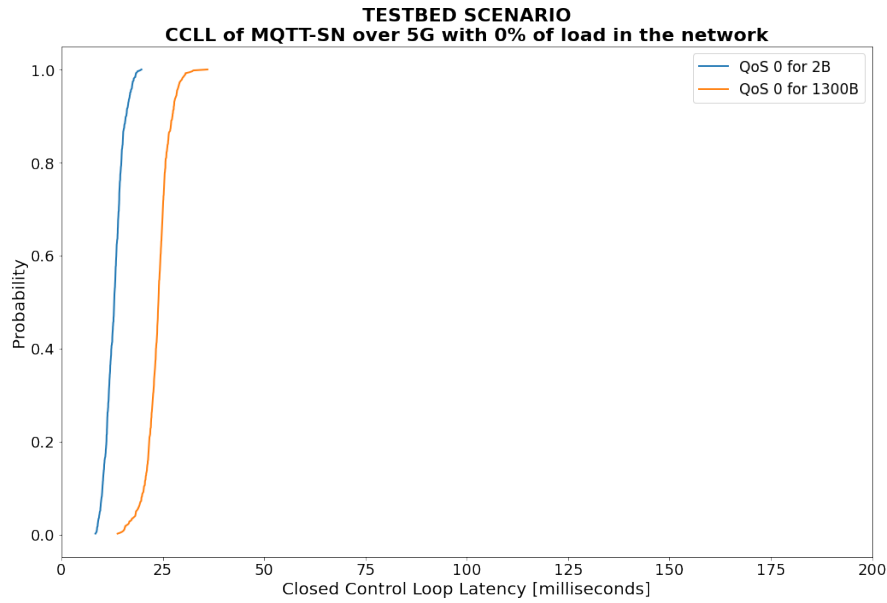


Figure D.37: CDF for MQTT-SN over 5G and 0% load in the network.

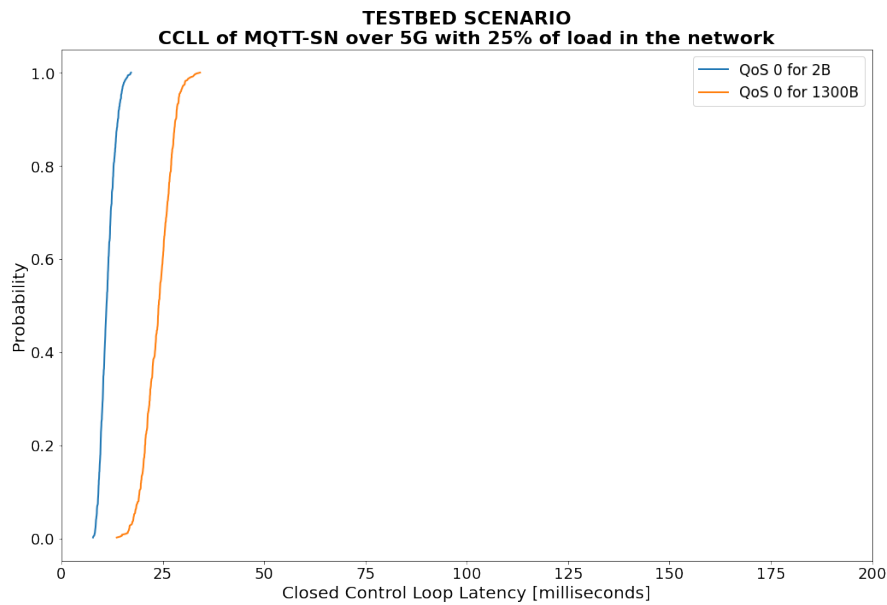


Figure D.38: CDF for MQTT-SN over 5G and 25% load in the network.

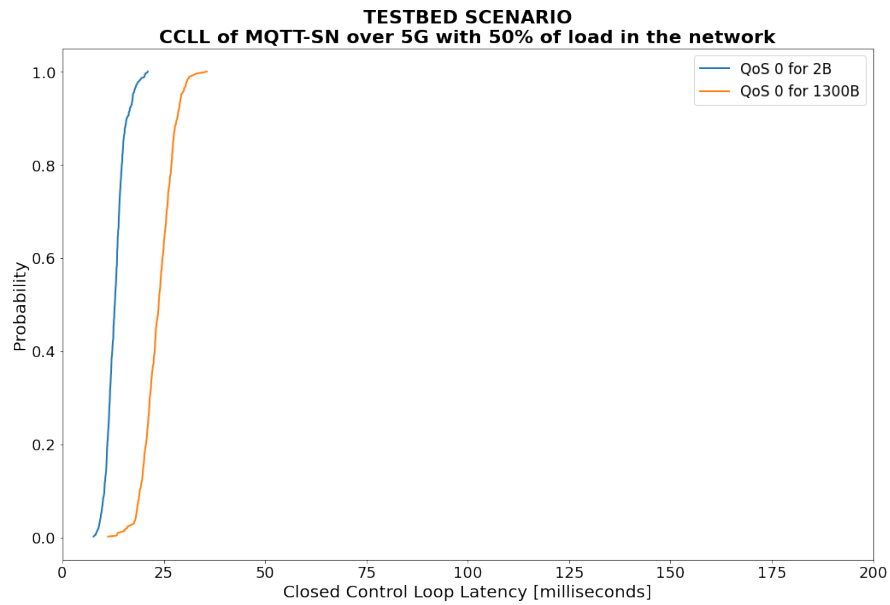


Figure D.39: CDF for MQTT-SN over 5G and 50% load in the network.

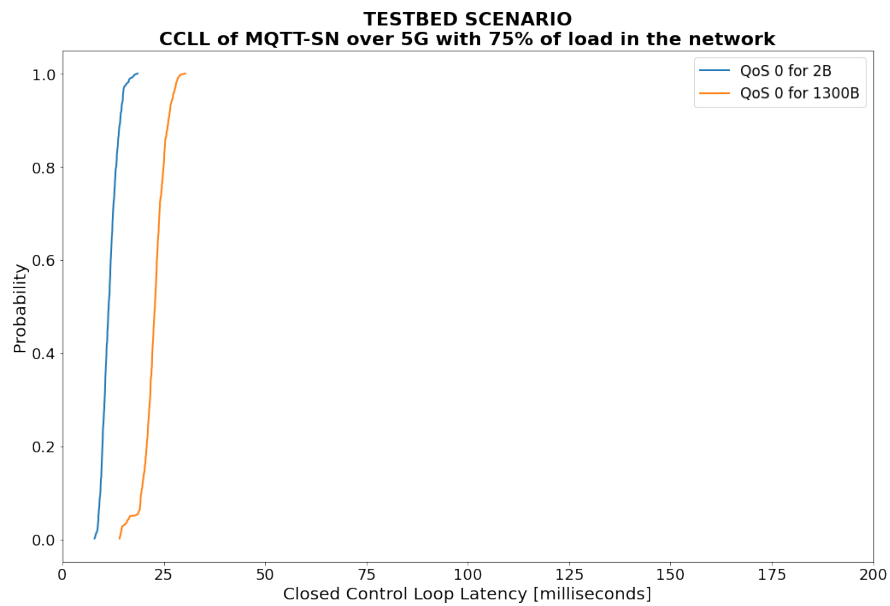


Figure D.40: CDF for MQTT-SN over 5G and 75% load in the network.

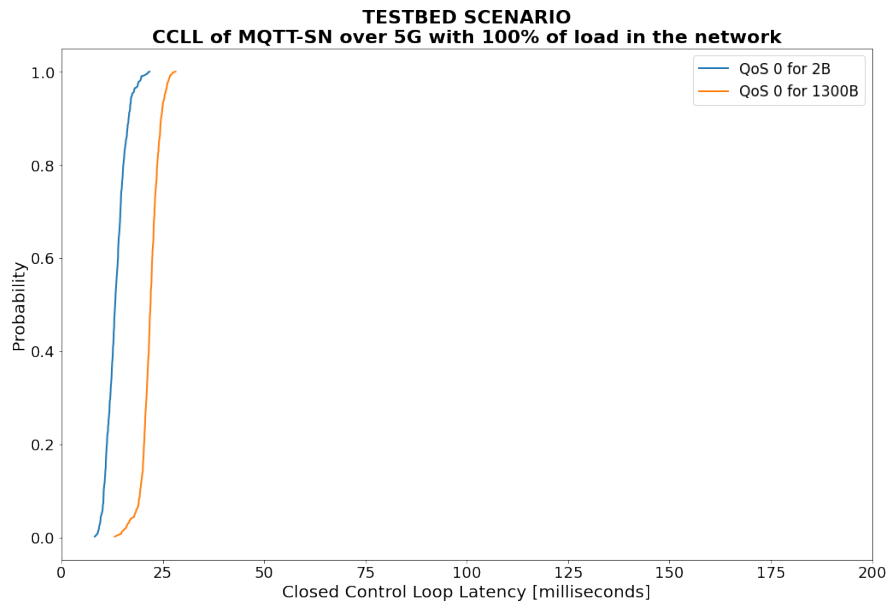


Figure D.41: CDF for MQTT-SN over 5G and 100% load in the network.

D.10.- CCLL for MQTT-SN over 5G: Network load study (testbed)

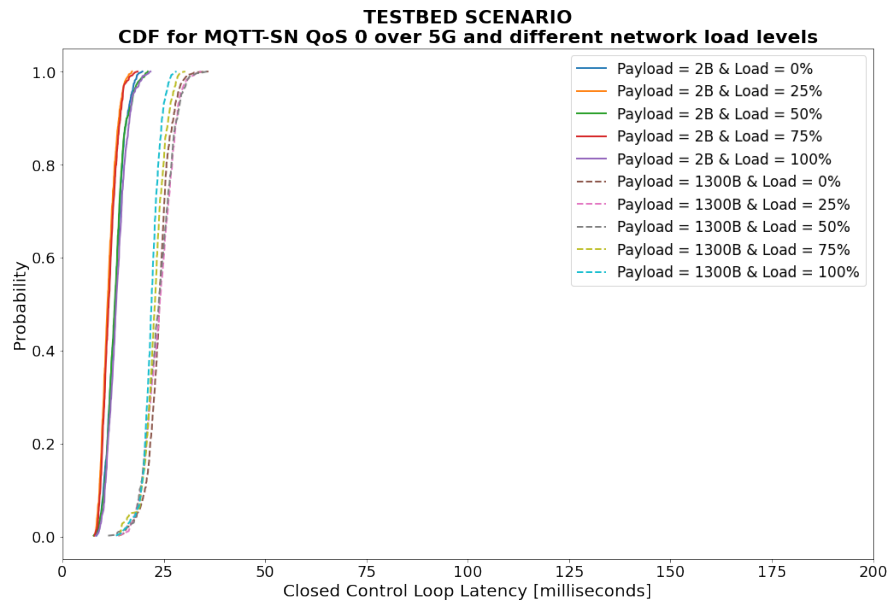


Figure D.42: CDF for MQTT-SN QoS 0 over 5G and different network load levels (testbed).

D.11.- CCLL for OPC-UA over 5G: Testbed scenario

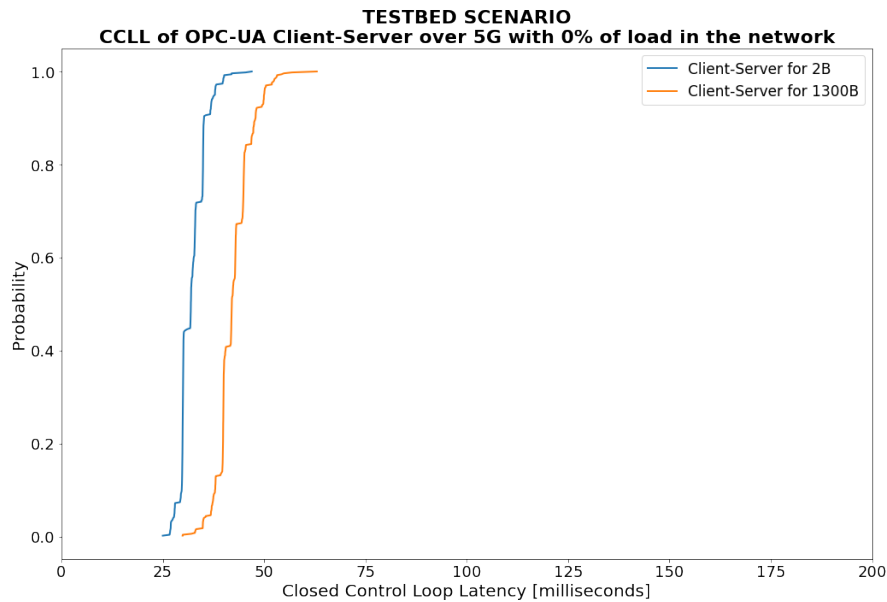


Figure D.43: CDF for OPC-UA Client-Server over 5G and 0% load in the network.

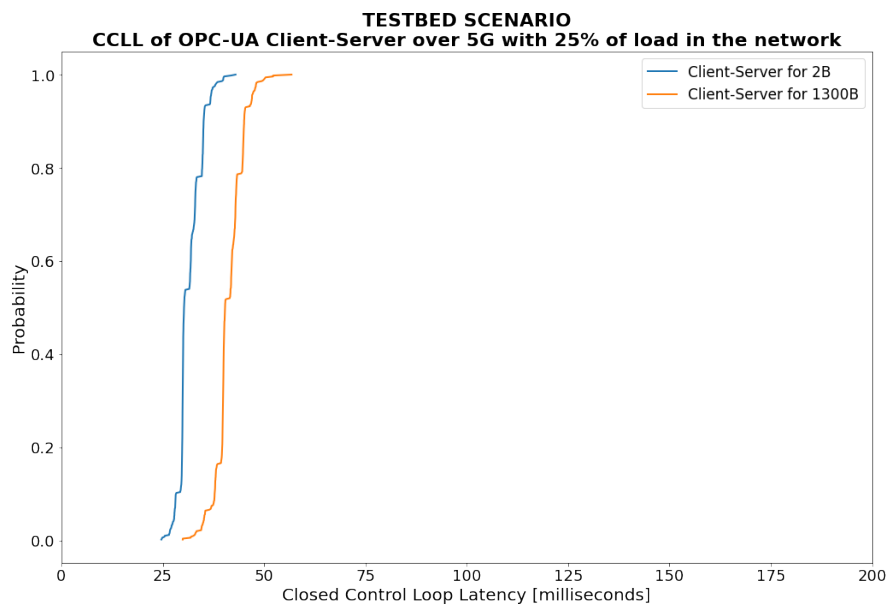


Figure D.44: CDF for OPC-UA Client-Server over 5G and 25% load in the network.

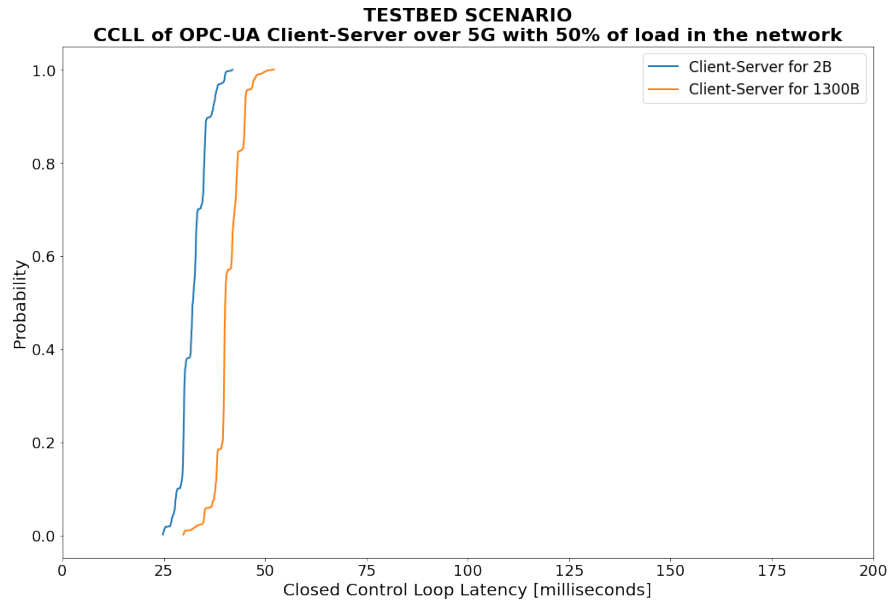


Figure D.45: CDF for OPC-UA Client-Server over 5G and 50 % load in the network.

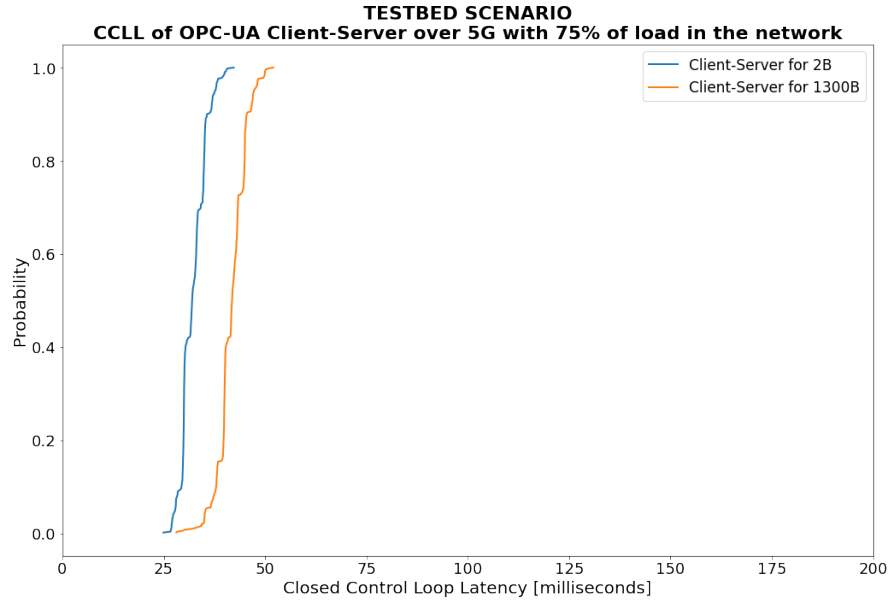


Figure D.46: CDF for OPC-UA Client-Server over 5G and 75 % load in the network.

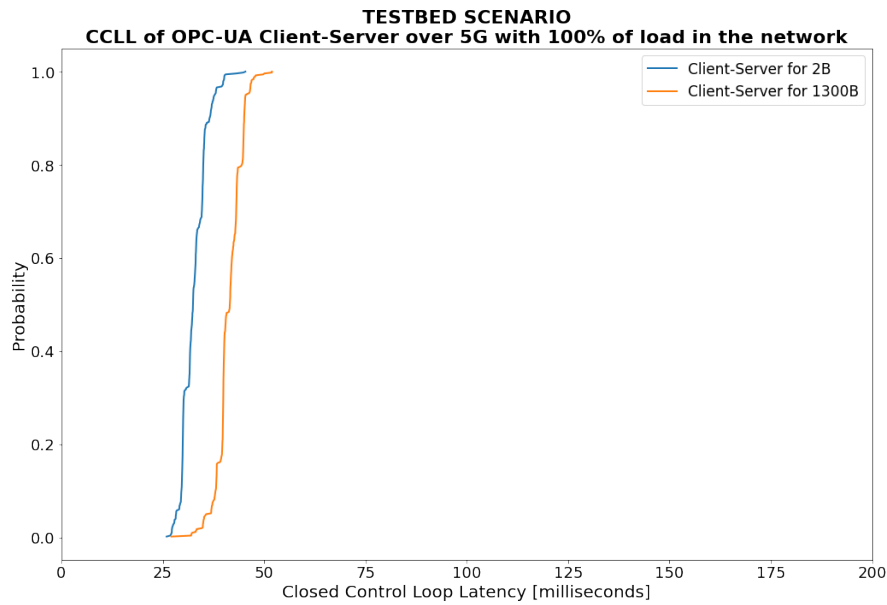


Figure D.47: CDF for OPC-UA Client-Server over 5G and 100 % load in the network.

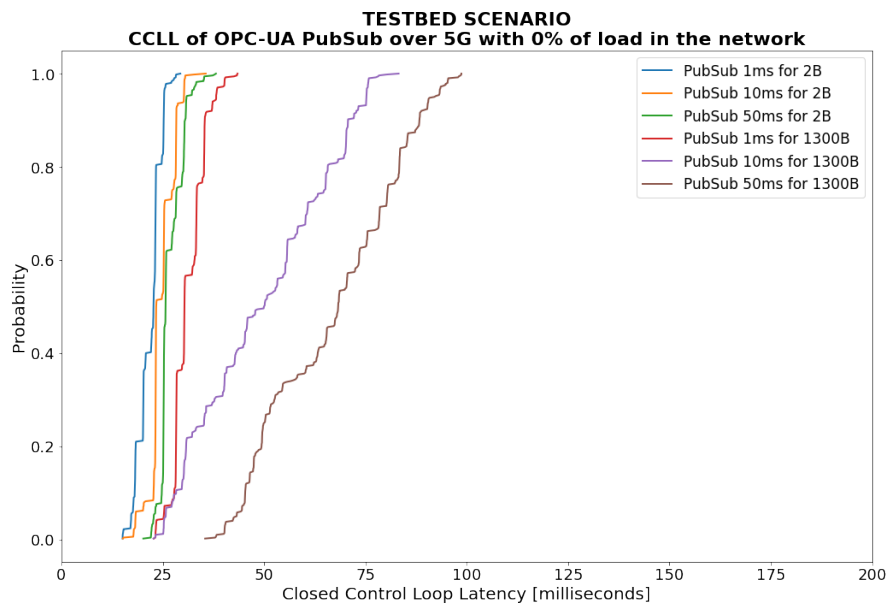


Figure D.48: CDF for OPC-UA PubSub over 5G and 0% load in the network.

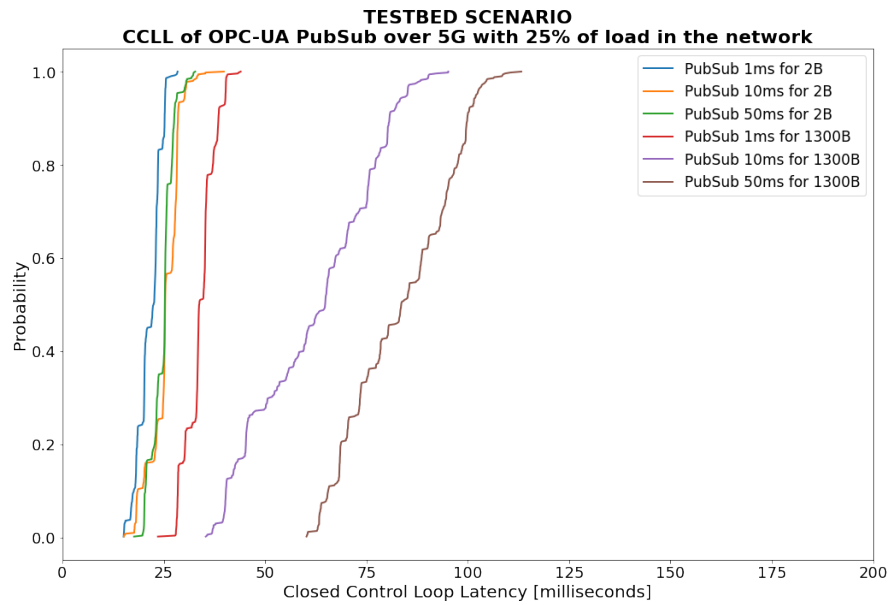


Figure D.49: CDF for OPC-UA PubSub over 5G and 25 % load in the network.

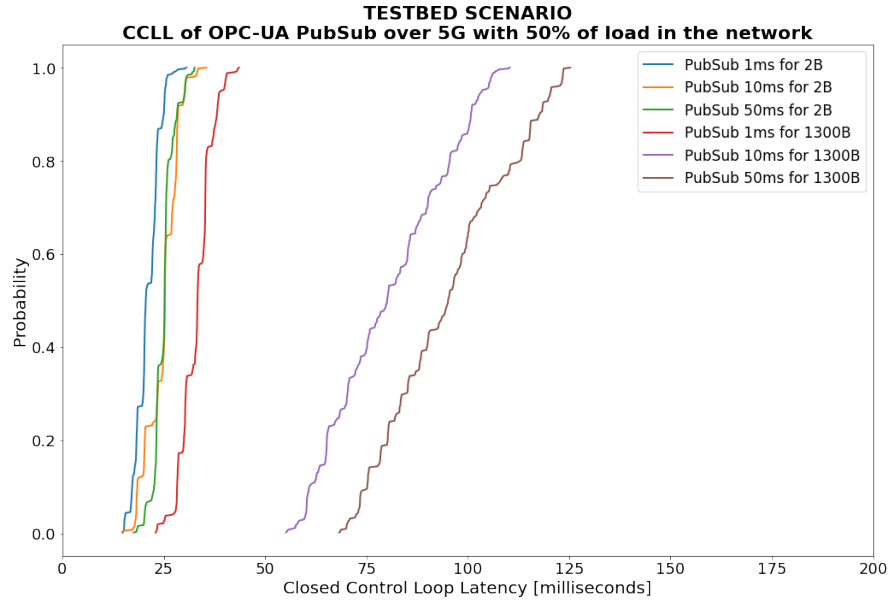


Figure D.50: CDF for OPC-UA PubSub over 5G and 50 % load in the network.

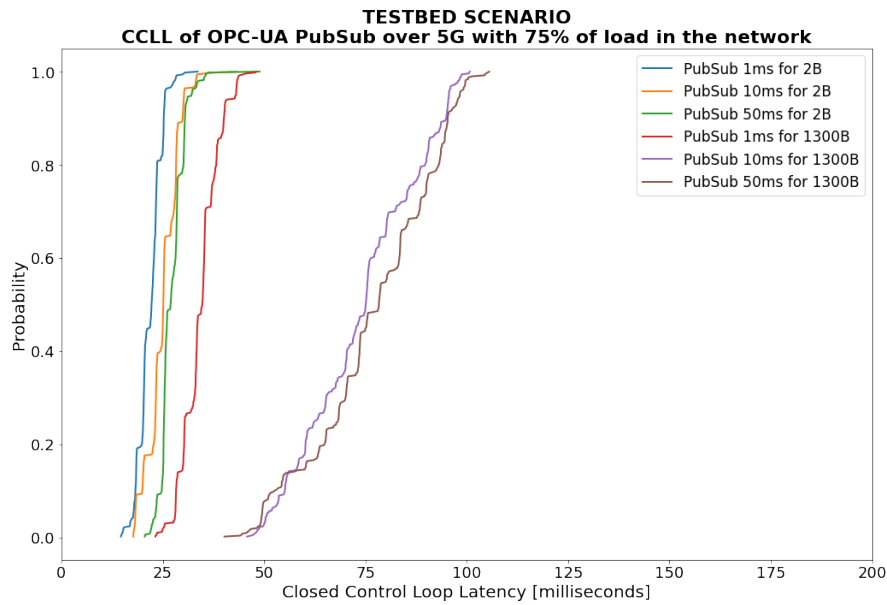


Figure D.51: CDF for OPC-UA PubSub over 5G and 75 % load in the network.

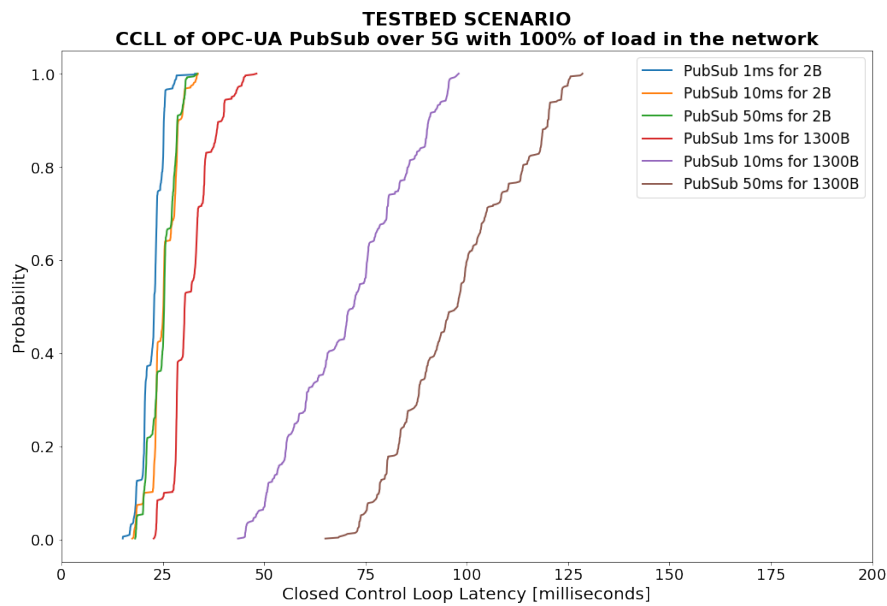


Figure D.52: CDF for OPC-UA PubSub over 5G and 100 % load in the network.

D.12.- CCLL for OPC-UA over 5G: Network load study (testbed)

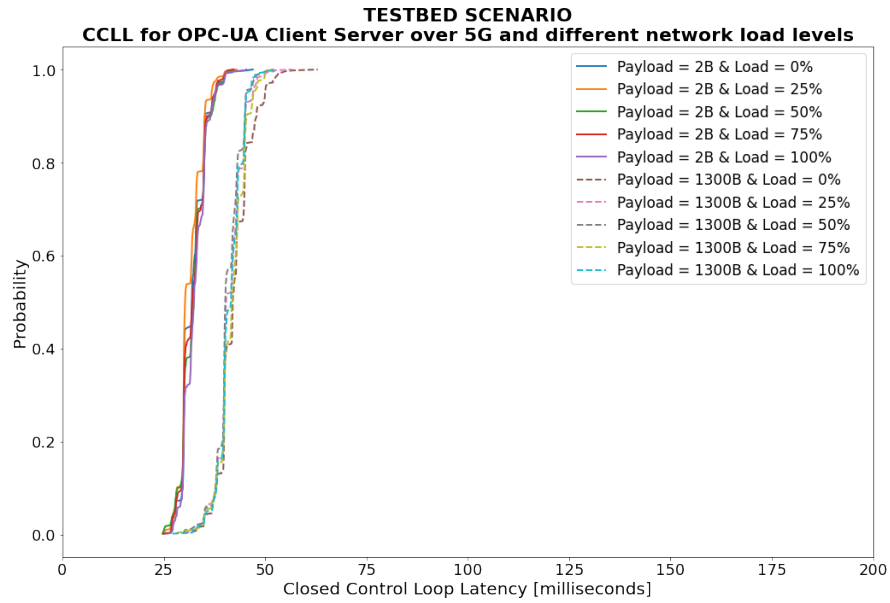


Figure D.53: CDF for OPC-UA Client-Server over 5G and different load levels (testbed).

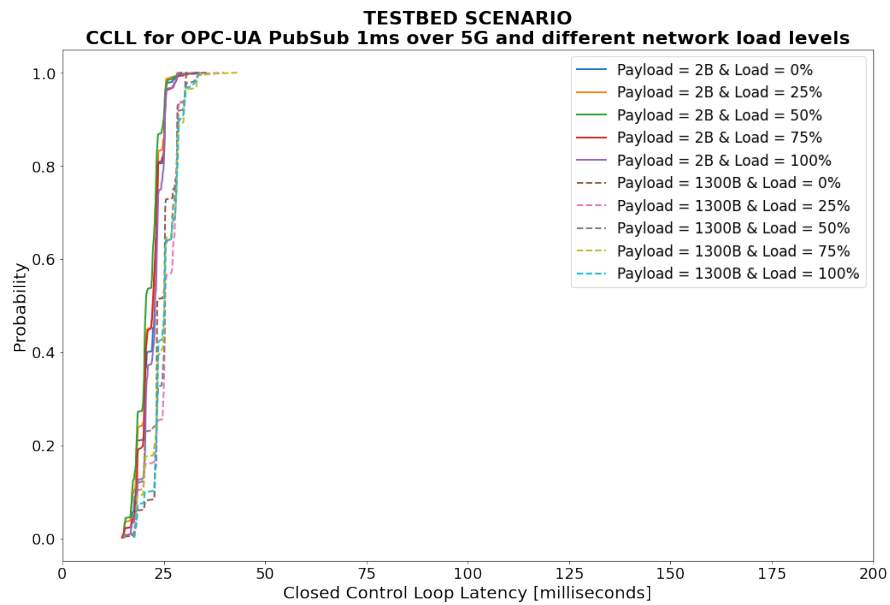


Figure D.54: CDF for OPC-UA PubSub 1ms over 5G and different load levels (testbed).

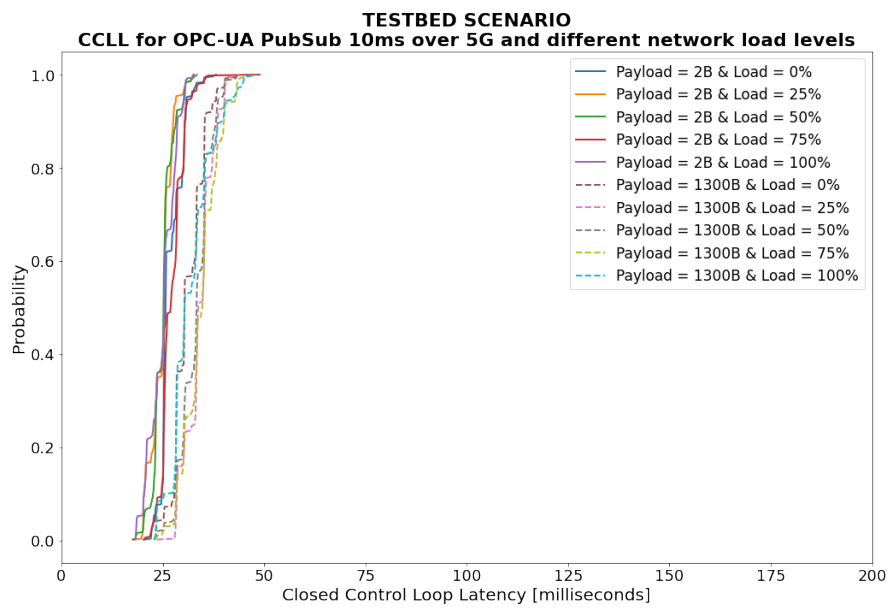


Figure D.55: CDF for OPC-UA PubSub 10ms over 5G and different load levels (testbed).

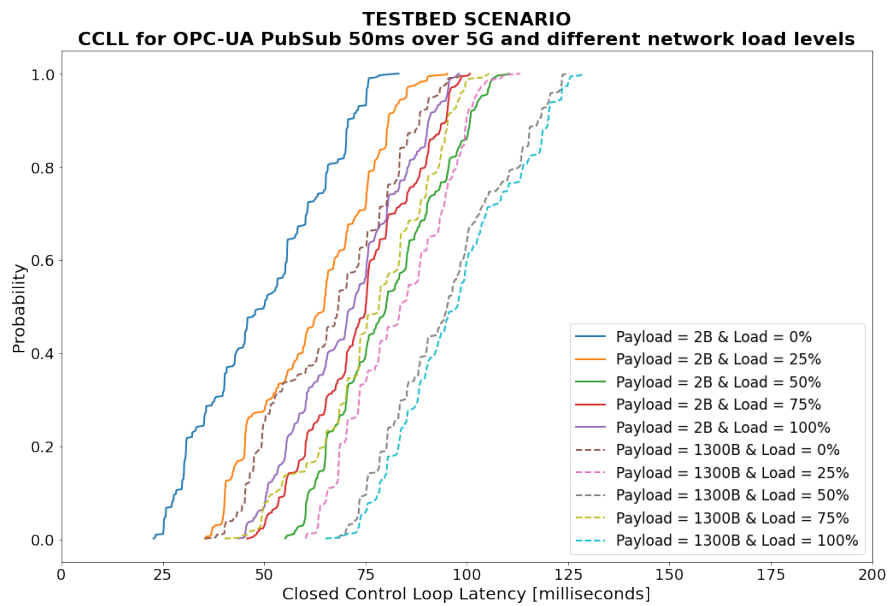


Figure D.56: CDF for OPC-UA PubSub 50ms over 5G and different load levels (testbed).