NS-3 Based Implementation of MQTT protocol for Smart Warehouse Digital Twin Application

AUNG KAUNG MYAT, HRISHIKESH DUTTA, ROBERTO MINERVA, and NOEL CRESPI, SAMOVAR, Télecom SudParis,Institut Polytechnique de Paris, France

Industry 4.0 has driven the evolution of logistics into Smart Warehouses, which rely on Digital Twins (DT) and Cyber-Physical Systems (CPS) to manage complex operations with minimal human intervention. The effectiveness of these systems depends on the "communication thread" between virtual models and physical assets, where the Message Queuing Telemetry Transport (MQTT) protocol serves as the standard for data synchronization. However, the widely respected ns-3 network simulator currently lacks a native, open-source model for MQTT, limiting the ability to validate these protocols over modern wireless networks prior to deployment. This paper addresses this gap by developing a comprehensive, open-source MQTT v3.1.1 module for ns-3 that supports all Quality of Service (QoS) levels (0, 1, and 2). Using this module, a realistic Smart Warehouse scenario was designed to simulate heterogeneous traffic types, including robot control commands and sensor telemetry. The study provides a comparative analysis of MQTT performance over wired Ethernet, Wi-Fi 6 (IEEE 802.11ax), and 5G New Radio (NR). Simulation results indicate that while Wi-Fi 6 offers bandwidth efficiency, 5G NR provides superior stability for mission-critical control traffic, with latency dominated by radio scheduling rather than MQTT protocol overhead. These findings offer a validated framework for optimizing communication infrastructures in industrial DT applications.

CCS Concepts: • **Networks → Network simulations**; • **Computer systems organization → Sensor networks**.

Additional Key Words and Phrases: Digital Twin, MQTT, ns-3, IoT, 5G, Wi-Fi 6, Simulation

ACM Reference Format:
Aung Kaung MYAT, Hrishikesh Dutta, Roberto Minerva, and Noel Crespi. 2018. NS-3 Based Implementation of MQTT protocol for Smart Warehouse Digital Twin Application. *J. ACM* 37, 4, Article 111 (August 2018), 21 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Industry 4.0 has transformed traditional logistics into Smart Warehouses. In these environments, Cyber-Physical Systems (CPS) and Digital Twins (DT) manage complex tasks with very little human help. A Digital Twin is a virtual replica of a physical system. To work correctly, it must stay in sync with physical assets—such as mobile robots, and sensors—in near real-time. The performance of a Digital Twin depends entirely on the connection between the virtual model and the real world. If there is a delay or data is lost, the virtual model will not match reality. This "desynchronization" can cause failures in safety controls and maintenance systems.

To keep these systems synchronized, the Message Queuing Telemetry Transport (MQTT) protocol has become the standard for the Industrial Internet of Things (IIoT). MQTT is lightweight and uses

Authors' Contact Information: Aung Kaung MYAT, aung-kaung.myat@telecom-sudparis.eu; Hrishikesh Dutta, hrishikesh.dutta@telecom-sudparis.eu; Roberto Minerva, roberto.minerva@telecom-sudparis.eu; Noel Crespi, noel.crespi@telecom-sudparis.eu, SAMOVAR, Télécom SudParis,Institut Polytechnique de Paris, Palaiseau, Essonne, France.

ACM 1557-735X/2018/8-ART111
https://doi.org/XXXXXXX.XXXXXXX

a "publish-subscribe” model. This allows sensors to send data efficiently without needing a constant, direct link to the central controllers. However, depending on underlying network infrastructure, MQTT performance can vary widely. Modern warehouses can be implemented with different types of network such as wired Ethernet, Wi-Fi, and 5G New Radio (NR). These networks have different characteristics in terms of latency, reliability, and coverage. Therefore, it is essential to test these communication networks before building them to verify that it meet the requirement of Digital Twin applications, save money and reduce safety risks.

Network simulation is the best way to verify these systems before deployment. The ns-3 simulator is a widely respected tool in academic research because it models networks very accurately. However, ns-3 currently has a major gap: it lacks a native, open-source model for MQTT. Previous attempts to add MQTT to ns-3 often used closed code that researchers could not change or extend. This makes it hard to properly test how application protocols perform over modern wireless networks like 5G and Wi-Fi 6.

To address this research gap by providing a complete, open-source MQTT v3.1.1 module for ns-3. Using this new module, we created a realistic Smart Warehouse scenario to test the protocol’s performance. Our main contributions are:

(1) **Development of a Native ns-3 MQTT Module:** We built a transparent MQTT Client and Broker that supports all Quality of Service (QoS) levels (0, 1, and 2). This allows for deep analysis within the simulator.
(2) **Smart Warehouse Scenario Design:** We designed a realistic industrial environment with different types of data traffic, including fast robot control commands, regular sensor updates, and heavy video streams.
(3) **Comparative Network Analysis:** We systematically tested MQTT performance over Ethernet, Wi-Fi 6, and 5G NR. By measuring speed and reliability across these networks, we provide real data to help industries choose the best connection for their needs.

The rest of this paper is organized as follows: Section II reviews previous work on IIoT protocols and simulation. Section III explains our methodology, including the new ns-3 MQTT module and the warehouse scenario. Section IV presents the results of our simulations. Finally, Section V concludes the study and suggests future research.

2 Literature Review

This section review the enabling technologies for Smart Warehouse Digital Twins, focusing on the application layer protocol, the necessity of simulation for validation, and the comparative analysis of underlying wireless standards.

2.1 MQTT in Industrial IoT

In the realm of Industrial Internet of Things (IIoT), efficient data transmission is very important. MQTT has emerged as the de facto standard application-layer protocol among legacy request-response protocols like HTTP and CoAP[2]. MQTT, which is standardized by OASIS, utilizes a publish-subscribe architecture that decouples data producers (sensors) from consumers (controllers), which enable highly scalable deployments in bandwidth-constrained environments[1][3].

A distinguishing feature of MQTT for Industrial control is its support for three distinct Quality of Service (QoS) levels, which allow system architects to balance latency against reliability:

- **QoS 0 (At most once):** Minimal latency "fire-and-forget" delivery.
- **QoS 1 (At least once):** Guarantees delivery but may result in duplicate messages.
- **QoS 2 (Exactly once):** Ensures single delivery through a four-step handshake[1].

Recent performance evaluations indicate that while QoS 2 provides the reliability required for critical control but its protocol overhead can reduce throughput by up to 40% compared to QoS[4][5]. Furthermore, when security layers like TLS/SSL are added to protect integrity, computational delays increase, challenging the strict timing requirements of real-time industrial applications[6][7]. For these reasons, optimizing MQTT configurations to minimize latency while maintaining security is a critical area of active research[8].

2.2 Importance of MQTT for Digital Twins

The effectiveness of Digital Twins (DT) is heavily bound by the "communication thread" connecting the physical assets to its virtual counterpart[9]. For a DT to function as dynamic virtual replica, it requires continuous, near real-time synchronization of state data[10].

MQTT is particularly critical where multiple DTs, such as a fleet of warehouse robots, interact simultaneously. Security and reliability in these massive deployments are often enhanced through machine learning approaches, which rely on the steady stream of data MQTT provides[11]. The protocol's lightweight nature allows for high-frequency telemetry updates without saturating the network, which is esstential for maintaining the consistency of the twin[12]. Recent visions for 6G-enabled DTs emphasize that delays in this synchronization layer can lead to desynchronization between the physical and virtual worlds, reducing the effectiveness of predictive maintenance and collision avoidance algorithms[13]. Therefore, MQTT serves not just as a transport mechanism, but as foundational synchronization layer that enables modern industrial DTs.

2.3 Network Simulation with NS-3

Validating these complex interactions in physical testbeds is often cost-prohibitive and risky. Network simulation offers a viable alternative for pre-deployment verfication. The ns-3 simulator is widely recognized in academic research due to its discrete-event engine and detailed models of the full TCP/IP stack [14].

However, a significant gap exist in the current ns-3 ecosystem: the lack of a native, feature-complete MQTT model. Previous contributions, such as those by Augusto, provided complied binaries without source code, preventing meaningful modification, extension, or validation[15]. Similarly, [16] reports an MQTT implementation in ns-3 targeted at smart building IoT applications, yet neither the source code nor binaries are publicly available. A 2023 review of IoT simulators highlighted that while other tools like OMNeT++ offer MQTT modules, they lack the cross-layer simulation capabilities of ns-3, particularly regarding complex physical layer models for 5G and Wi-Fi[17]. This necessitates the development of a transparent, open-source MQTT library capable of interacting deeply with ns-3's wireless stacks.

2.4 5G NR vs WiFi for Smart Warehouse Digital Twins

The connectivity layer of a Smart Warehouse Digital Twin relies on heterogeneous wireless networks to link mobile robots and sensors. The industry currently debates the trade-offs between Wi-Fi 6 (IEEE 802.11ax) and 5G New Radio (NR)[18].

WiFi 6 offers cost-effective, high bandwidth connectivity suitable for bulk data transfer. Validation studies in ns-3 show that under low network loads, Wi-Fi 6 achieves latencies comparable to 5G (<10 ms). However, its contention-based medium access control (CSMA/CA) introduces non-deterministic jitter in dense environments where multiple robots compete for channel access[19].

5G NR: Designed with Ultra-Reliable Low-Latency Communication (URLLC), 5G NR is increasingly favored for mission-critical control. Although it may exhibit higher baseline latency due to frame structures and core network traversal, it offers deterministic scheduling suitable for time-sensitive networking (TSN)[20].

Recent literature suggests a hybrid approach for Digital Twins: utilizing 5G NR for critical, safety-related control traffic (where bounded latency is required) and Wi-Fi 6 for high-bandwidth, non-critical data such as video surveillance feeds. Validating this hybrid architecture requires the rigorous simulation environment that only ns-3 can provide.

3 Methodology

In this section, firstly we describe the MQTT protocol implementation in ns-3 simulator. Then, we explain the flow of the simulation process. Finally, we detail the Smart Warehouse Digital Twin scenario description and its implementation using MQTT protocol.

3.1 MQTT ns-3 Module

To support native publish-subscribe communiation within the ns-3 simulation environment, a complete implementation of the MQTT protocol version 3.1.1 was developed directly within ns-3 and is available at https://gitlab.com/simulations9070736/mqtt.git.. The module consists of two main components: the MQTT client and the MQTT broker, which together provide an end-to-end MQTT stack suitable for IoT and DT simulations.

The MQTT client implements the full set of MQTT control packets, including CONNECT, CONNACK, PUBLISH (QoS 0, 1, 2), SUBSCRIBE, SUBACK, UNSUBSCRIBE, UNSUBACK, and DISCONNECT. The client maintiains session state, manages retransmissions and handshake procedures for QoS level 1 and 2.

The MQTT broker functions as the central coordination point, managing concurrent client sessions, decoding incoming control packets, performing topic-based routing, and executing the complete QoS handshake logic, including PUBACK, PUBREC/PUBREL, AND PUBCOMP sequences to guarantee exactly-once delivery. In addition, the broker implements mechanisms for client authentication, verifying CONNECT requests before establishing sessions and rejecting malformed or unauthorized connections. It also maintains session persistence, storing subscription information, pending QoS1 and 2 messages to ensure that message delivery is correctly resumed after client reconnection.

Together, these components provide a comprehensive and extensible MQTT implementation that can operate over any ns-3 transport or wireless technology. This integrated design enable systematic evaluation of publish-subscribe workloads in IoT and DT scenarios, leveraging ns-3's rich networking models and simulation capabilities.

3.2 Simulation Workflow for MQTT in ns-3

To evaluate publish-subscribe communication under different network conditions, the MQTT protocol was deployed as ns-3 application layer module. The following steps outline the complete simulation workflow used to test MQTT applications and measure performance indicators such as throughput and end-to-end latency.

(1) **Node Creation and Global Configuration**: First, a dedicated MQTT broker node and a configurable set of client nodes are defined. Global simulation parameters such as time, random number generator seeds, and logging components are initialized to ensure reproduciblity and deterministic results across all simulation runs.
(2) **Intenet Stack Installation and Wired Channel Configuration**: The baseline configuration is impleted upon a wired Ethernet and point-to-piont channel is installed between all nodes using *CsmaHelper*. Internet stack is deployed using the standard *InternetStackHelper*. IP addressing is then assigned using *Ipv4AddressHelper*, ensuring end-to-end reachability between all MQTT clients and the broker.

(3) **Configuration of Wireless or Cellular Access**: Depending on the scenario, nodes may communicate over Wi-Fi, LTE or 5G NR. To configure wireless access-network, access-network helpers (e.g. *YansHelper*, *LteHelper*, *NrHelper*) are used to set up the physical and MAC layers, including channel models, frequency bands, and bandwidths. Nodes are then associated with the appropriate base stations or access points. The MQTT layer remains unchanged as all MQTT applications operate over TCP/IP irrespective of underlying link technology. This flexiblity enables direction comparison of MQTT performance accross heterogeneous networks.

(4) **Configuration of MQTT Broker**: The MQTT broker is installed on the broker node as an instance of *MqttBrokerApp*. The broker's runtime attributes such as listening TCP port, maximum QoS level supported, connection-timeout interval, retransmission interval are configured using *SetAttribute* methods. Once started, the broker passively accepts incoming TCP sessions, decode control packets and perform topic-based routing to appropriate subscribers. The allowd usernames and passwords for client authentication are also set at this stage using *SetUserAuthorizations* methods.

(5) **Configuration of MQTT Clients**: Each MQTT client node is configured with an instance of *MqttClientApp*. Client attributes such as broker IP address, port number, client identifier, username, password, and other qos related parameters and session related parameters are set using *SetAttribute* methods. Subscription are configured through the client's topic list and QoS levels, which are thenn used to generate SUBSCRIBE control packets to send to the broker. Publishing is performed either periodically or via scheduled events using the *sendPUBLISHpacket()* method.

(6) **Execution of the Simulation and Data Collection**: Once all MQTT entities and network elements are configured, the simulation is executed using the ns-3 event scheduler. Measurements metrics such as throughput, loss rate, latency are obtained via the ns-3 *FlowMonitor* module or custom tracing callbacks attached to MQTT events. The collected data is then processed and analyzed to evaluate the performance of MQTT under various network conditions and configurations.

An example code snippet for configuring and installing the MQTT broker application on the broker node is shown in 1.

```
// Create broker application
Ptr<MqttBrokerApp> broker = CreateObject<MqttBrokerApp>();

// Configure broker attributes
// Default MQTT port
broker->SetAttribute("ListeningPort", UintegerValue(1883));
// QoS 0-2 supported
broker->SetAttribute("MaxQoSLevel", UintegerValue(2));
broker->SetAttribute("ConnectionTimeout", TimeValue(Seconds(30)));
broker->SetAttribute("RetransmitTimeout", TimeValue(Seconds(0.01)));

// Install on broker node
brokerNode->AddApplication(broker);

// Application start/stop times
broker->SetStartTime(Seconds(0.5));
broker->SetStopTime(Seconds(simTime));
```

Listing 1. MQTT Broker Application Configuration in ns-3

An example code snippet for configuring and installing an MQTT client application on a client node is shown in 2.

```
        Ptr<MqttClientApp> client = CreateObject<MqttClientApp>();

        // Configure broker endpoint
        client->SetAttribute("BrokerAddress",
            AddressValue(InetSocketAddress(brokerIp, 1883)));
        client->SetAttribute("BrokerPort", UintegerValue(1883));

        // Set MQTT session parameters
        client->SetAttribute("ClientId", StringValue("client-01"));
        client->SetAttribute("Username", StringValue("user"));
        client->SetAttribute("Password", StringValue("password"));
        client->SetAttribute("CleanSession", BooleanValue(true));
        client->SetAttribute("KeepAlive", UintegerValue(60));
        client->SetAttribute("RetransmitTimeout", TimeValue(Seconds(2)));

        // Add application to node
        clientNode->AddApplication(client);
        client->SetStartTime(Seconds(1.0));
        client->SetStopTime(Seconds(simTime));

        /*********************
        * SUBSCRIBING CLIENT
        *********************/
        std::vector<std::string> topics = {"sensor/temperature"};
        std::vector<uint8_t> qos = {1};   // QoS 1 delivery
        client->SetSUBSCRIBEtopics(topics, qos);

        // Send SUBSCRIBE control packet
        Simulator::Schedule(Seconds(1.2),
            &MqttClientApp::SendSubscribeRequest, client);

        /*********************
        * PUBLISHING CLIENT
        *********************/
        Simulator::Schedule(Seconds(2.0),
            &MqttClientApp::sendPUBLISHpacket,
            client,
            "sensor/temperature",          // topic
            "23.5C",                       // payload
            1,                             // QoS 1
            false,                         // DUP flag
            false);                        // Retain flag
```

Listing 2. MQTT Client Application Configuration in ns-3

3.3 Smart Warehouse Digital Twin Scenario Description

In the smart warehouse scenario illustrated in Figure 1 dicpicts a cyber-physical system where various interconnected components work together to manage warehouse operations efficiently. The *controller* serve as the central hub for coordinating activities, sending commands, storing data, and maintaining overall situational awareness. *Package sensors* are placed at the conveyor belts that receive incoming packages, detecting their arrival and send notifications to the controller when new package arrives. When new package is arrived, the controller check is there any available storage space in the racks by sending request to it and receiving the status update. It also verify that there are available *mobile robots* to pickup and transport the packages to the designated storage locations. Once the controller confirms the availability of both storage space and mobile robots, it dispatches commands to the mobile robots to pick up the packages from the conveyor belt and deliver them to the assigned racks. Whenever the controller received request to retrieve a package, it check availability of mobile robots and send command to the robot to fetch the package from the specified rack and deliver it to the *drop zone* and update the status of the racks accordingly. The environmental sensors such as temperature and humidity sensors continuously monitor the warehouse conditions and send periodic updates to the controller to ensure optimal storage environment. The video cameras positioned throughout the warehouse stream real-time footage to the controller for safety monitoring and surveillance.

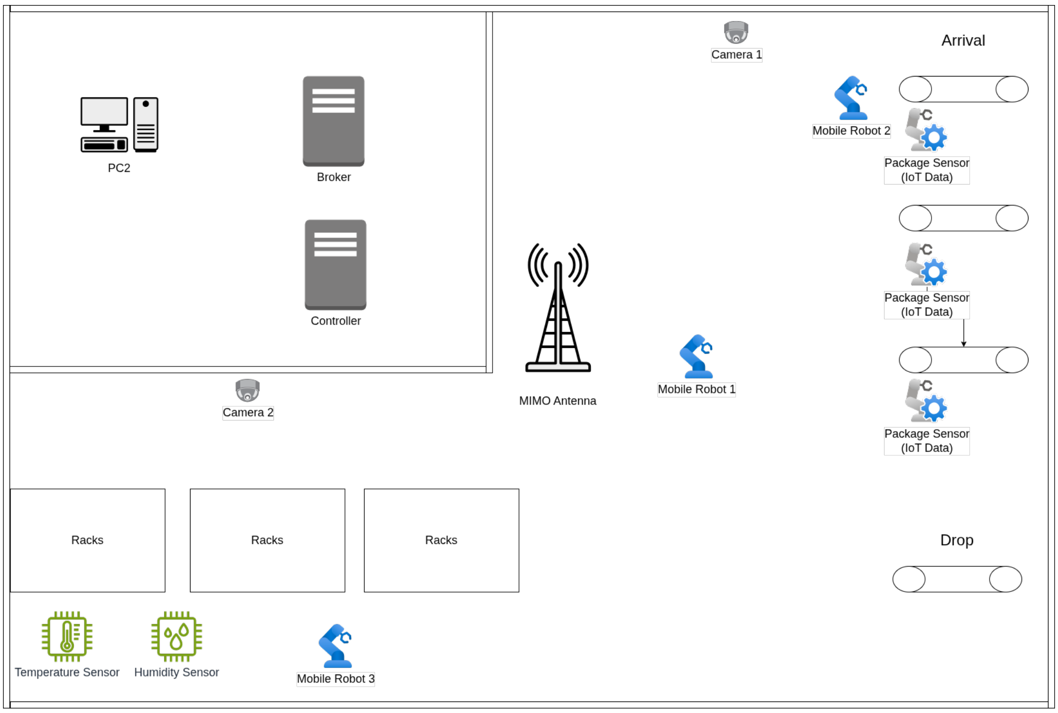

Fig. 1. Smart Warehouse Digital Twin Scenario

The components in the smart warehouse Digital Twin system and their interactions are illustrated in the class diagram in Figure 2.

These components generate diverse traffic types:

- Robot control commands (e.g. destination update, pick and place instructions), which require ultra-low latency ($<$ 10–20 ms) and high relaiblity, as they influence mechanical actuatio and safety.
- Status updates from environmental sensors, package sensors, and rack availability sensors, which produce periodic traffic with moderate latency requirements ($<$ 100–200 ms).
- Video streams from fixed cameras, which generate high-bandwidth data flow with tighter jitter constraints ($<$ 150–300 ms) to support real-time monitoring.

Together, these heterogeneous traffic profiles reflect realistic operational conditions in a smart warehouse DT system.

3.4 Smart Warehouse Digital Twin Scenario Implementation using MQTT

The Smart Warehouse Digital Twin scenario is implemented using the MQTT protocol to facilitate communication between various components of the warehouse system. The implementation consists of three main subsystems: Inventory Management System, Environmental Monitoring System, and Robot Management System. Each subsystem utilizes MQTT topics to publish and subscribe to relevant data.

Environmental Monitoring System: This subsystem illustrated in Figure 3, environmental parameters such as temperature and humidity within the warehouse. All the environmental sensors register to controller using the topic */warehouse/sensor/[sensor_name]/register* where the controller

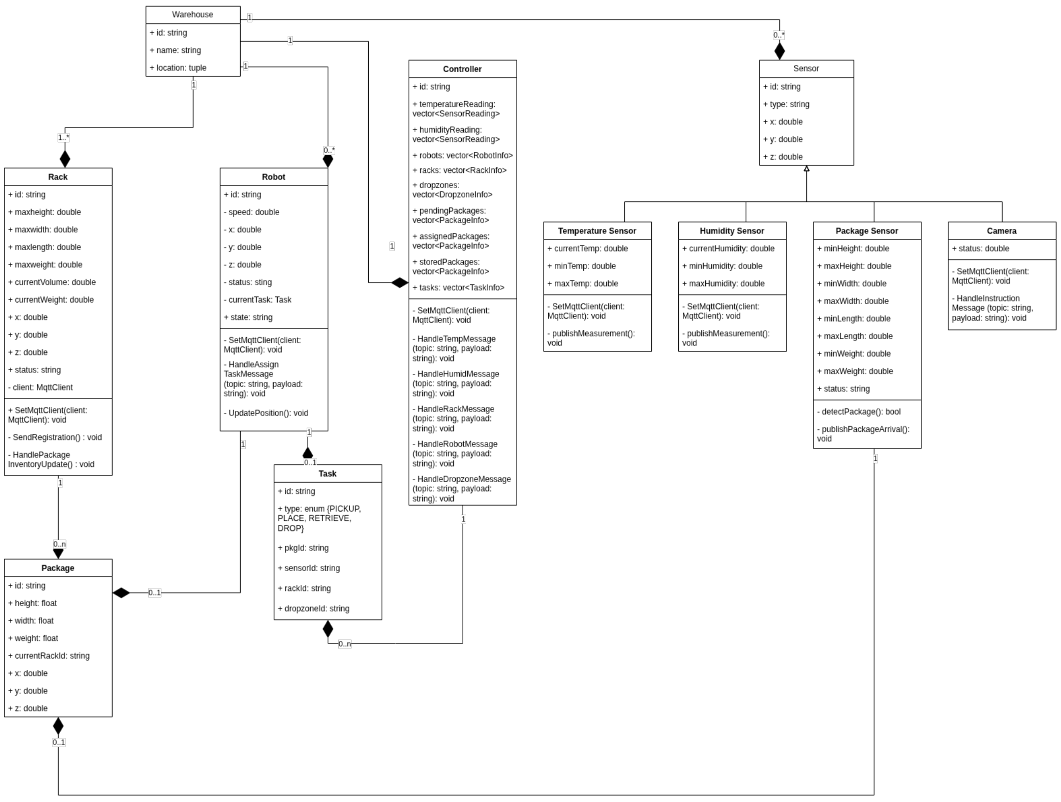
Warehouse
+ id: string
+ name: string
+ location: tuple
Controller
+ id: string
+ temperatureReading: vector<SensorReading>
+ humidityReading: vector<SensorReading>
+ robots: vector<RobotInfo>
+ racks: vector<RackInfo>
+ dropzones: vector<DropzoneInfo>
+ pendingPackages: vector<PackageInfo>
+ assignedPackages: vector<PackageInfo>
+ storedPackages: vector<PackageInfo>
+ tasks: vector<TaskInfo>
- SetMqttClient(client: MqttClient): void
- HandleTempMessage (topic: string, payload: string): void
- HandleHumidMessage (topic: string, payload: string): void
- HandleRackMessage (topic: string, payload: string): void
- HandleRobotMessage (topic: string, payload: string): void
- HandleDropzoneMessage (topic: string, payload: string): void
Sensor
+ id: string
+ type: string
+ x: double
+ y: double
+ z: double
Rack
+ id: string
+ maxheight: double
+ maxwidth: double
+ maxlength: double
+ maxweight: double
+ currentVolume: double
+ currentWeight: double
+ x: double
+ y: double
+ z: double
+ status: string
- client: MqttClient
+ SetMqttClient(client: MqttClient): void
- SendRegistration() : void
- HandlePackage InventoryUpdate() : void
Robot
+ id: string
- speed: double
- x: double
- y: double
- z: double
- status: sting
- currentTask: Task
+ state: string
- SetMqttClient(client: MqttClient): void
- HandleAssign TaskMessage (topic: string, payload: string): void
- UpdatePosition(): void
Task
+ id: string
+ type: enum {PICKUP, PLACE, RETRIEVE, DROP}
+ pkgId: string
+ sensorId: string
+ rackId: string
+ dropzoneId: string
Package
+ id: string
+ height: float
+ width: float
+ weight: float
+ currentRackId: string
+ x: double
+ y: double
+ z: double
Temperature Sensor
+ currentTemp: double
+ minTemp: double
+ maxTemp: double
- SetMqttClient(client: MqttClient): void
- publishMeasurement(): void
Humidity Sensor
+ currentHumidity: double
+ minHumidity: double
+ maxHumidity: double
- SetMqttClient(client: MqttClient): void
- publishMeasurement(): void
Package Sensor
+ minHeight: double
+ maxHeight: double
+ minWidth: double
+ maxWidth: double
+ minLength: double
+ maxLength: double
+ minWeight: double
+ maxWeight: double
+ status: string
- detectPackage(): bool
- publishPackageArrival(): void
Camera
+ status: double
- SetMqttClient(client: MqttClient): void
- HandleInstruction Message (topic: string, payload: string): void
1
0..*
1..*
0..n
0..1
0..n

Fig. 2. Class Diagram of Smart Warehouse Digital Twin System

subscribe and listening on the topic */warehouse/sensor/[sensor_name]/register*. Temperature and humidity sensor periodically publish their reading to the topic */warehouse/sensor/[sensor_name]/temperature* and */warehouse/sensor/[sensor_name]/humidity* respectively. The controller subscribes to these topics to receive real-time updates on environmental conditions. However, the camera sensor, only listen to the controller command on the topic */warehouse/sensor/[sensor_name]/video/commands* to start or stop the video streaming. The controller control the video stream of camera by publishing command message to the topic */warehouse/sensor/[sensor_name]/video/commands/startStream* and */warehouse/sensor/[sensor_name]/video/commands/stopStream*.

Inventory Management System: This subsystem illustrated in Figure 4, is responsible for tracking and managing the inventory within the warehouse. Same as in the environmental monitoring system, all the inventory sensors register to controller using the topic */warehouse/sensor/[sensor_name]/register* where the controller subscribe and listening on the topic */warehouse/sensor/[sensor_name]/register*. The package sensor publish the notification message to the topic */warehouse/sensor/[sensor_name]/package* whenever a package is arrived to warehouse. The controller subscribes to this topic and assign to the available robot to pick up the package and place on the rack. Both the rack sensor and dropzone sensor waits for the controller command on the topic */warehouse/rack/[sensor_name]/commands* and */warehouse/dropzone/[sensor_name]/commands* respectively. When the robot reaches to the rack, the controller publish the command to the topic */warehouse/rack/[sensor_name]/commands/addPackage* to update the capacity of the rack. As soon as the dropzone receive request to retrieve the package,

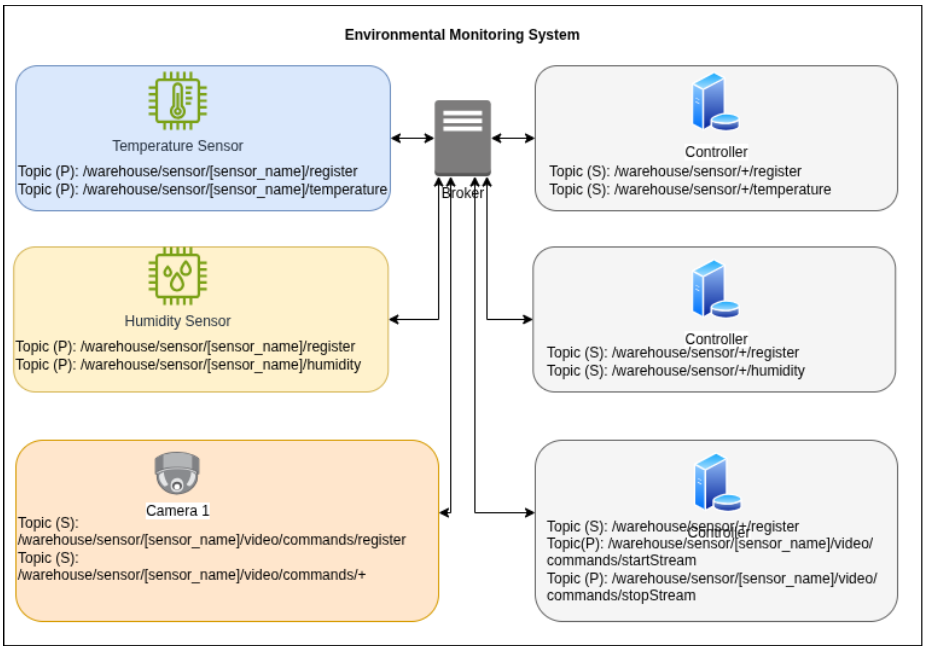
Environmental Monitoring System
Temperature Sensor
Topic (P): /warehouse/sensor/[sensor_name]/register
Topic (P): /warehouse/sensor/[sensor_name]/temperature
Broker
Controller
Topic (S): /warehouse/sensor/+/register
Topic (S): /warehouse/sensor/+/temperature
Humidity Sensor
Topic (P): /warehouse/sensor/[sensor_name]/register
Topic (P): /warehouse/sensor/[sensor_name]/humidity
Controller
Topic (S): /warehouse/sensor/+/register
Topic (S): /warehouse/sensor/+/humidity
Camera 1
Topic (S): /warehouse/sensor/[sensor_name]/video/commands/register
Topic (S): /warehouse/sensor/[sensor_name]/video/commands/+
Controller
Topic (S): /warehouse/sensor/+/register
Topic(P): /warehouse/sensor/[sensor_name]/video/commands/startStream
Topic (P): /warehouse/sensor/[sensor_name]/video/commands/stopStream

Fig. 3. Environmental Monitoring System MQTT Implementation

it notify the controller on the topic */warehouse/dropzone/[sensor_name]/commands/retrievePackage* and controller assign robot to retrieve the package from the rack. Similar to place package operation, when the robot reaches to the rack, the controller publish the command to the topic */warehouse/rack/[sensor_name]/commands/removePackage* to update the capacity of the rack. As soon as the robot drop the package to the dropzone, the controller notify the customer or truck using the topic */warehouse/dropzone/[sensor_name]/commands/dropPackage.*

Robot Management System: This subsystem illustrated in Figure 5, manages the fleet of robots operating within the warehouse. All the robots register to controller using the topic */warehouse/robot/[robot_name]/register* where the controller subscribe and listening on the topic */warehouse/robot/[robot_name]/register.* All the PICKUP, PLACE, RETRIEVE, and DROP operation are assigned to the robot as task by the controller through the topic */warehouse/robot/[robot_name]/command/assignTas* When the robot complete the assigned task, it notify the controller by publishing message to the topic */warehouse/robot/[robot_name]/status/taskCompleted.*

4 Results

In this section, we present the results obtained from our experiments. First, we verify that our implementaiton working as expected by comparing QoS metrics across different Ethernet speeds and payload sizes. Then, we analyze the performance of our proposed architecture in the context of a Smart Warehouse Digital Twin application.

4.1 Comparison of QoS Metrics upon Varying Ethernet Speeds and Payload Sizes

As for the first step, the implementation of MQTT protocol in NS-3 will be verified. In order to verify that our Implementation of MQTT protocol in NS-3 is functioning correctly, we analyze the QoS metrics under different Ethernet speeds and payload sizes. We conduct simulations with

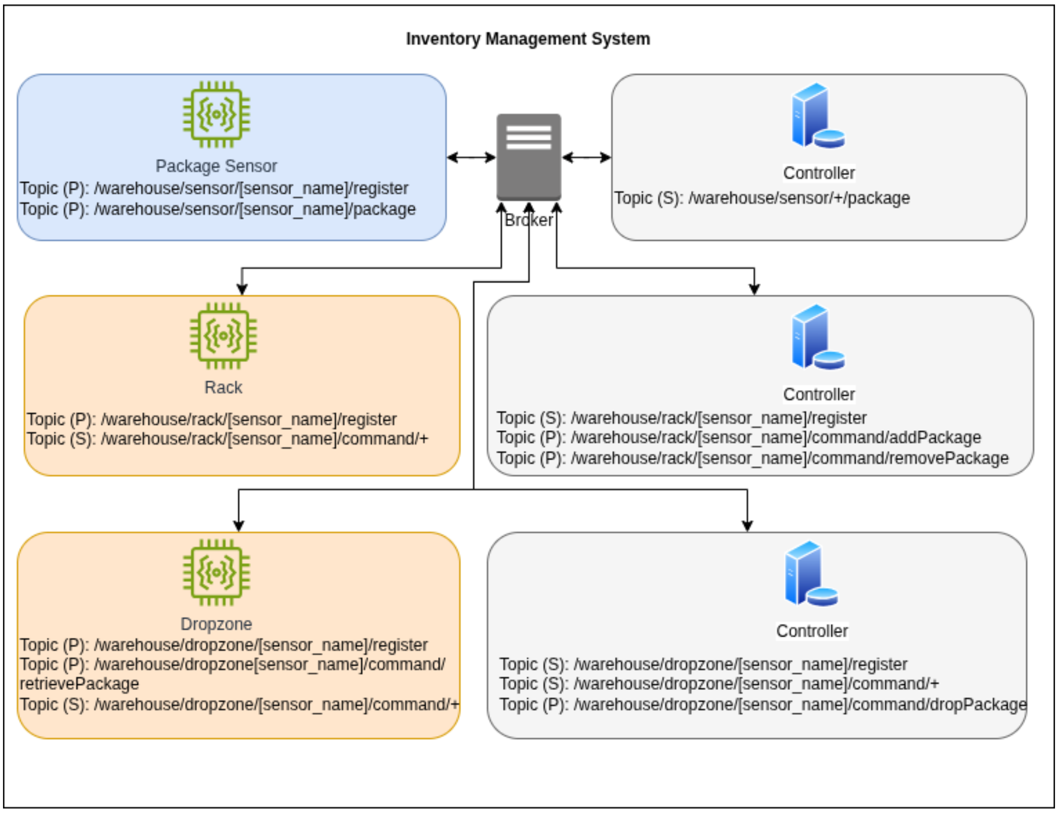
Inventory Management System
Package Sensor
Topic (P): /warehouse/sensor/[sensor_name]/register
Topic (P): /warehouse/sensor/[sensor_name]/package
Broker
Controller
Topic (S): /warehouse/sensor/+/package
Rack
Topic (P): /warehouse/rack/[sensor_name]/register
Topic (S): /warehouse/rack/[sensor_name]/command/+
Controller
Topic (S): /warehouse/rack/[sensor_name]/register
Topic (P): /warehouse/rack/[sensor_name]/command/addPackage
Topic (P): /warehouse/rack/[sensor_name]/command/removePackage
Dropzone
Topic (P): /warehouse/dropzone/[sensor_name]/register
Topic (P): /warehouse/dropzone[sensor_name]/command/retrievePackage
Topic (S): /warehouse/dropzone/[sensor_name]/command/+
Controller
Topic (S): /warehouse/dropzone/[sensor_name]/register
Topic (S): /warehouse/dropzone/[sensor_name]/command/+
Topic (P): /warehouse/dropzone/[sensor_name]/command/dropPackage

Fig. 4. Inventory Management System MQTT Implementation

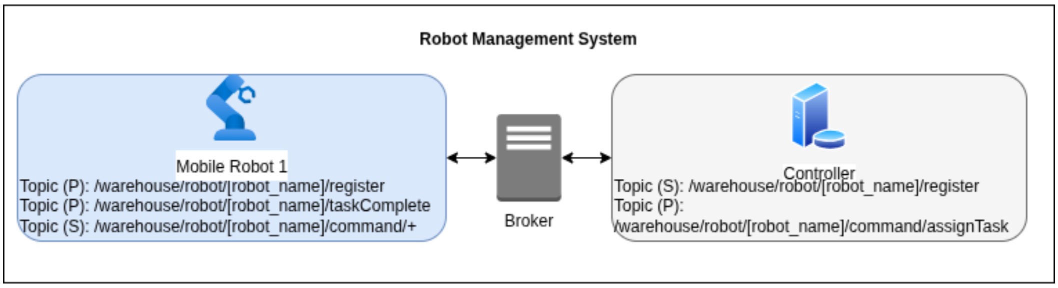
Robot Management System
Mobile Robot 1
Topic (P): /warehouse/robot/[robot_name]/register
Topic (P): /warehouse/robot/[robot_name]/taskComplete
Topic (S): /warehouse/robot/[robot_name]/command/+
Broker
Controller
Topic (S): /warehouse/robot/[robot_name]/register
Topic (P): /warehouse/robot/[robot_name]/command/assignTask

Fig. 5. Robot Management System MQTT Implementation

Ethernet speeds of 10 Mbps, 100 Mbps, and 1 Gbps, while varying the payload sizes from 100 bytes to 10,000 bytes. All of the three different level of QoS (0, 1, and 2) are tested in the simulations.

In MQTT protocol, base on the QoS levels, different ackowledgement messages are exchanged between the subscriber and broker to ensure the delivery of the messages. These different acknowledgement messages introduce additional overhead in the communication, which can impact the latency and throughput of the system. For the QoS level 0, there is no acknowledgement message exchanged between the subscriber and broker, resulting in the lowest latency and highest throughput. The QoS level 1 introduces a PUBACK message from the broker to the subscriber, which adds some overhead and increases the latency while slightly reducing the throughput while ensuring message delivery at least once. The QoS level 2 involves a four-step handshake process (PUBLISH, PUBREC, PUBREL, PUBCOMP) to guarantee that the message is delivered exactly once. This process introduces the highest overhead, resulting in the highest latency and lowest throughput among the three QoS levels. The results of the simulations can be found in Figure 6 and simulation results show that it reflects the expected behavior of MQTT protocol under different QoS levels. As the Ethernet speed increases, the latency decreases and throughput increases for all QoS levels. Similarly, as the payload size increases, the latency increases and throughput decreases for all QoS levels

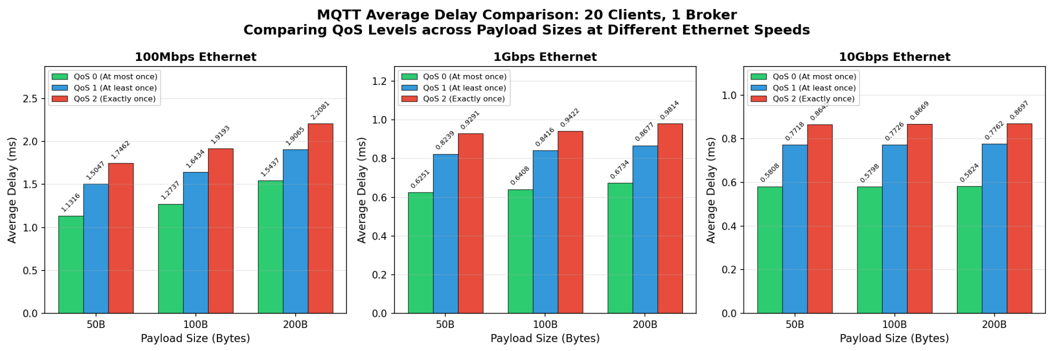

Fig. 6. Comparison of QoS Metrics upon Varying Ethernet Speeds and Payload Sizes

4.2 Analysis for Smart Warehouse Digital Twin Application

As for the next step, we want to analyze the performance of traffic types that are generated in Smart Warehouse Digital Twin Application. All the communication between the digital twin components are carried through MQTT protocol over Ethernet. The MQTT messages exchanged between the components can be categorized into three types: telemetry, operations, and control messages. Telemetry messages include periodic updates from temperature and humidity sensors. Operations messages consist of commands sent between controllers and package detection sensors, racks and dropzone. Control messages are commands sent from the controller to moving robots to manage their navigation and tasks within the warehouse. The average payload sizes for all types of messages can be seen in Table 1.

The performance of these three types of traffic over Ethernet, WiFi, and 5G networks are analyzed in the following subsections in order to figure out the delay and packet loss experienced by each type of message in different network setups.

4.2.1 Deployment over Ethernet Connectivity. The performance of the three types of MQTT messages over Ethernet connectivity is analyzed in this subsection. Three different Ethernet speeds (100 Mbps, 1 Gbps, and 10 Gbps) are considered for the analysis. Different level of MQTT QoS levels (0, 1, and 2) are also taken into account while analyzing the performance.

The results for telemetry messages over Ethernet connectivity are shown in Figure 7. As the level of QoS increases, the average delay increases from 47μs, 75μs, to 207μs for 100 Mbps Ethernet speed. This is due to the additional overhead for ensuring message delivery. It can be observed that as the Ethernet speed increases, the average delay experienced by telemetry messages decreases from 47μs, 45μs, to 37μs for QoS level 0. But for QoS levels 1 and 2, the reduction of average delay is significantly small when the Ethernet speed increases from 1 Gbps to 10 Gbps.

Table 1. Average Payload Sizes of MQTT Messages in Smart Warehouse Digital Twin Application

MQTT Message Type	**Average Payload Size (bytes)**
Temperature telemetry	81
Humidity telemetry	81
Package detection operation	158
Rack Add Package operation	119
Rack Remove Package operation	119
Dropzone Retrieve Package operation	100
Dropzone Drop Package operation	94
Robot Assign Task control	112
Robot Task Complete control	37

The results for operations messages over Ethernet connectivity are shown in Figure 8. The trend of average delay decreases with increasing Ethernet speed and increases with higher QoS levels is similar to the telemetry messages. Among the messages Rack Remove Package messages experiences packet loss for QoS level 0 due to the non-reliable delivery nature of this QoS level but using higher QoS levels (1 and 2) ensures reliable delivery of these messages without any packet loss. Under QoS 0, most operations fall below $30 - 90\mu s$ delay for all three Ethernet speeds. Under QoS 1 and 2, the delays increase significantly due to the acknowledgment and retransmission mechanisms yet benefit from higher speeds, notablely for package detection messages. QoS 2 exhibits the highest delays, reach $> 500\mu s$ at 100 Mbps, decreasing below $200\mu s$ by 10 Gbps, showing that although reliability mechanisms add overhead, sufficient link capcity can mitigate queuing effects for operational traffics.

The results for control messages over Ethernet connectivity are shown in Figure 9. Among the control messages, the assign task message have higher payload size (112 bytes) compared to the complete task message (37 bytes). Under QoS 0, the difference of average delay between these two messages is not significant across all three Ethernet speeds. However, under QoS levels 1 and 2, the assign task message experiences higher average delay compared to the complete task message due to the larger payload size. Only QoS 0 ensures that average delay remain below $20ms$ for both control messages across all three Ethernet speeds while QoS levels 1 and 2 were not able to meet this requirement as average delays exceed $70ms$ for assign task messages. Higher delays are observed because the robots are moving around the warehouse and connectivity degradations may occur, leading to retransmissions and increased latency. As QoS level 0 does not guarantee reliable delivery of messages, deployment over ethernet connectivity does not meet the requirements for control messages in smart warehouse digital twin application.

4.2.2 Deployment over WiFi Connectivity. The performance of the three types of MQTT messages over WiFi connectivity is analyzed in this subsection. Three different WiFi standards (802.11n, 802.11ac, and 802.11ax) are considered for the analysis. Different level of MQTT QoS levels (0, 1, and 2) are also taken into account while analyzing the performance.

The Figure 10 depicts the results for telemetry messages over WiFi connectivity. Compared to the deployment over Ethernet connectivity, the average delay experienced by telemetry messages over WiFi connectivity is significantly higher due to the shared medium access nature of WiFi. Similart to the Ethernet connectivity, as the QoS level increases, the average delay also increases and decreases as the WiFi standard improves from 802.11n to 802.11ax. Under QoS 0, delays decline from 1.28–1.30 ms with 802.11n to 1.11–1.12 ms under 802.11ax, reflecting efficiency gains for small

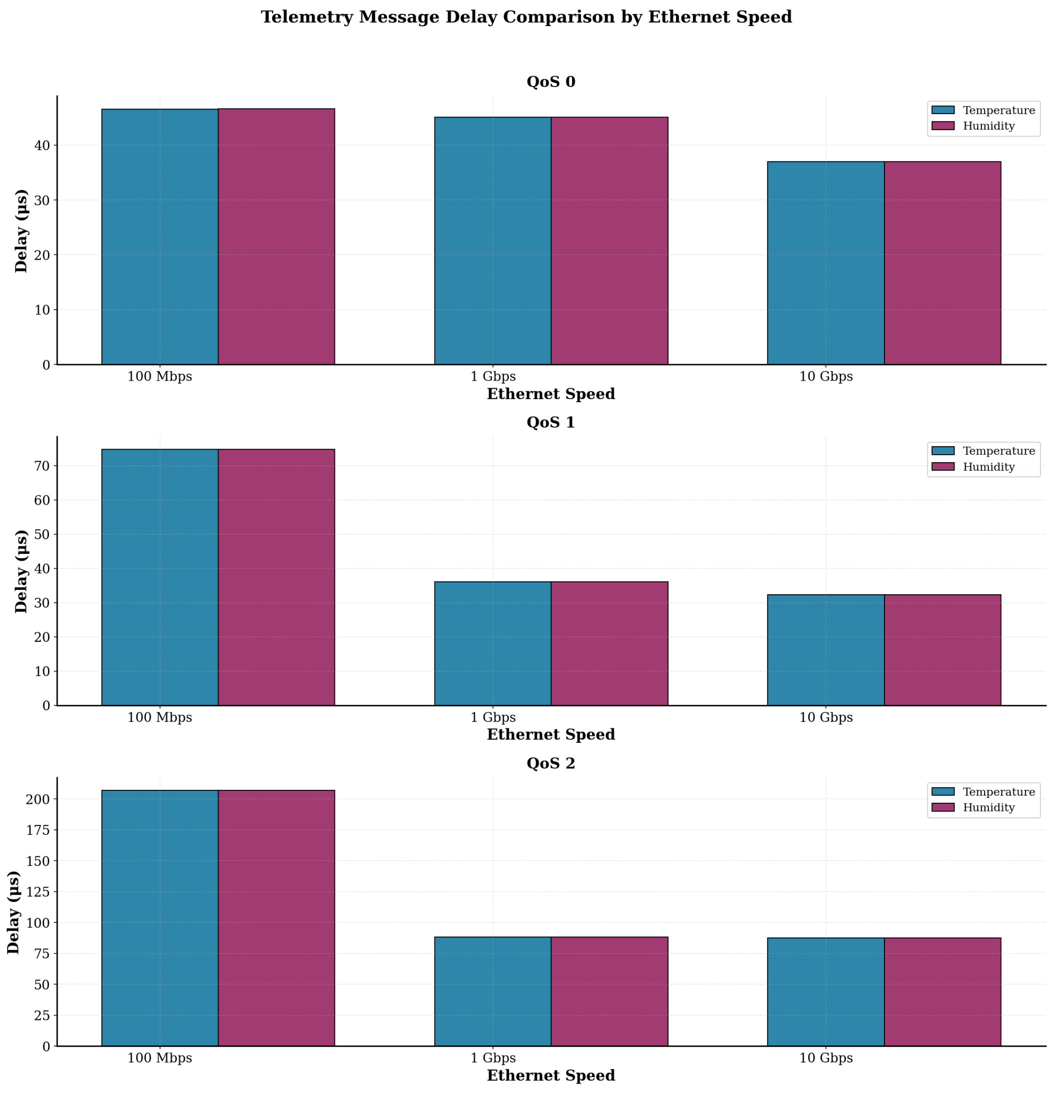

Fig. 7. Ethernet Telemetry Results for Smart Warehouse Digital Twin Application

payload messages. With QoS 1, telemetry delays increase to approximately 1.57−1.78 ms, with only minor improvements across Wi-Fi generations, while QoS 2 further elevates latency to 1.82−1.97 ms, effectively saturating performance gains from PHY upgrades. Under QoS level 2, the average delay experience by temperature messages is slightly more than humidity messages even though both messages have the same payload size (81 bytes).

The Figure 11 depicts the results for operations messages over WiFi connectivity. For QoS 0, delays significantly drop from 2.5 ms on 802.11n to 2.05-2.07 ms on 802.11ac/ax for pakcage detection messages while maintaining sub-milisecond values ($< 0.8ms$) for other operation messages. At QoS 2, the delay overhead becomes more significant, with package detection delays rising to 3.5 ms on 802.11n and stabilizing around 3.1 ms on 802.11ax while other operational messages remain

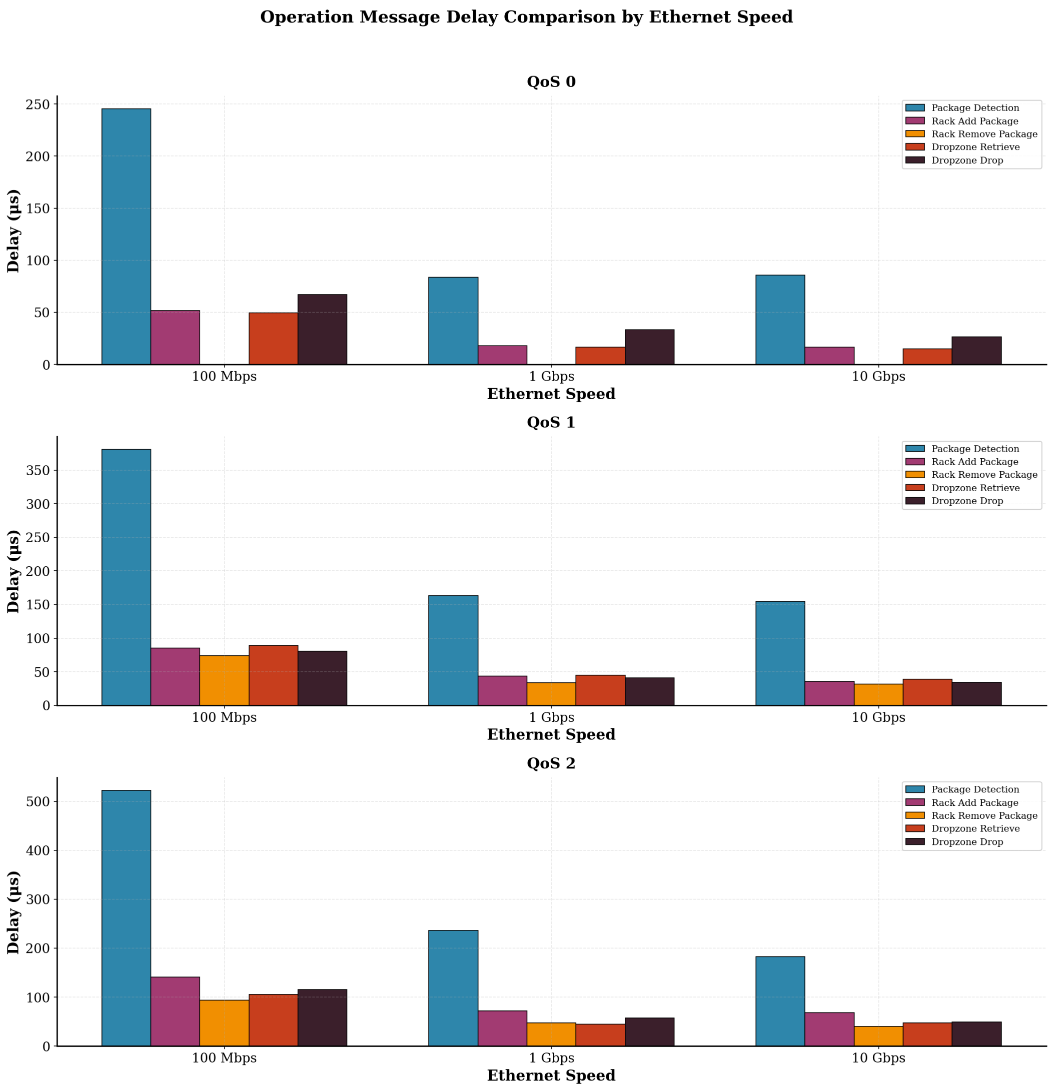

Fig. 8. Ethernet Operations Results for Smart Warehouse Digital Twin Application

between 0.85-1.3 ms. These results demonstrate that WiFi improvements modestly reduce queuing and transmission delays, particularly for heavy operational workloads, but cannot fully compensate for protocol-level acknowledgement overhead at higher MQTT QoS levels.

The Figure 12 depicts the results for control messages over WiFi connectivity. Under QoS 0, delay decreases from approximately 1.0 ms on 802.11n to 0.83 ms on 802.11 ac/ax, indicating that newer WiFi standards help moderately reduce delay through higher data rates and better channel efficiency. For QoS 1 and QoS 2, however, improvement remain limited: delays stays around 1.1-1.3 ms for assign task messages and 0.7-1.0 ms for task completion messages, with only marginal reductions under 802.11ax. Unlinke what was obeserved in ethernet connectivity, the delays are below 2 ms even at QoS 2 for assign task messages, likely due to lower baseline delays in WiFi.

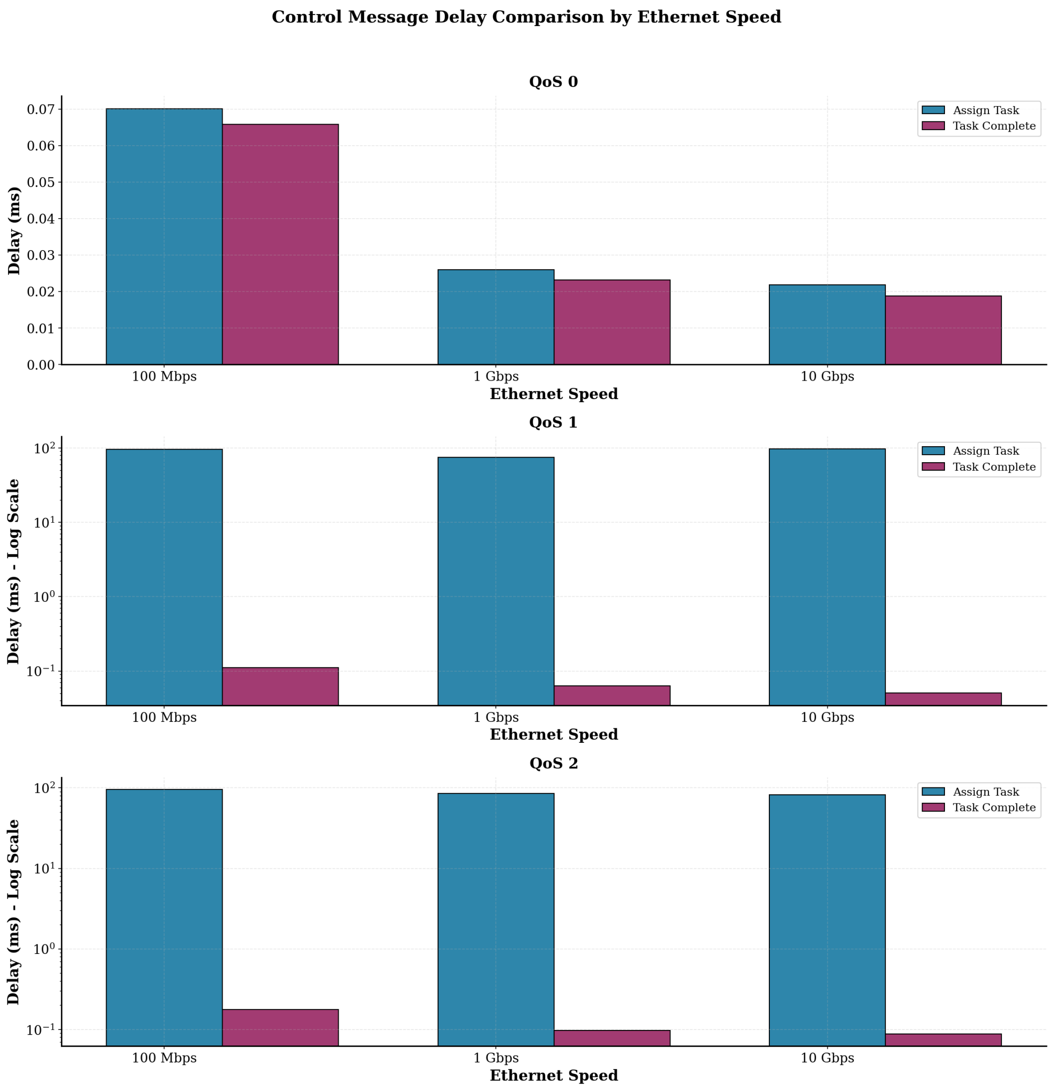

Fig. 9. Ethernet Control Results for Smart Warehouse Digital Twin Application

4.2.3 Deployment over 5G NR Connectivity. The performance of the three types of MQTT messages over 5G NR connectivity is analyzed in this subsection. The 5G deployment utilizes a standalone (SA) architecture with a 3.5 GHz frequency band. Different level of MQTT QoS levels (0, 1, and 2) are also taken into account while analyzing the performance.

The Figure 13 depicts the results for telemetry messages over 5G NR connectivity. Across all QoS levels, measured delays remain nearly identical at approximately 23.46 m, indicating negligible sensitivity to QoS-induced overhead for small telemetry payloads in this 5G setup. This unusually flat performance contrasts with Ethernet and Wi-Fi results, where acknowledgment overhead clearly increased latency. In the 5G case, the dominant contributors to delay are radio scheduling latency, frame structure timing (TTI granularity), and core network traversal, which overshadow

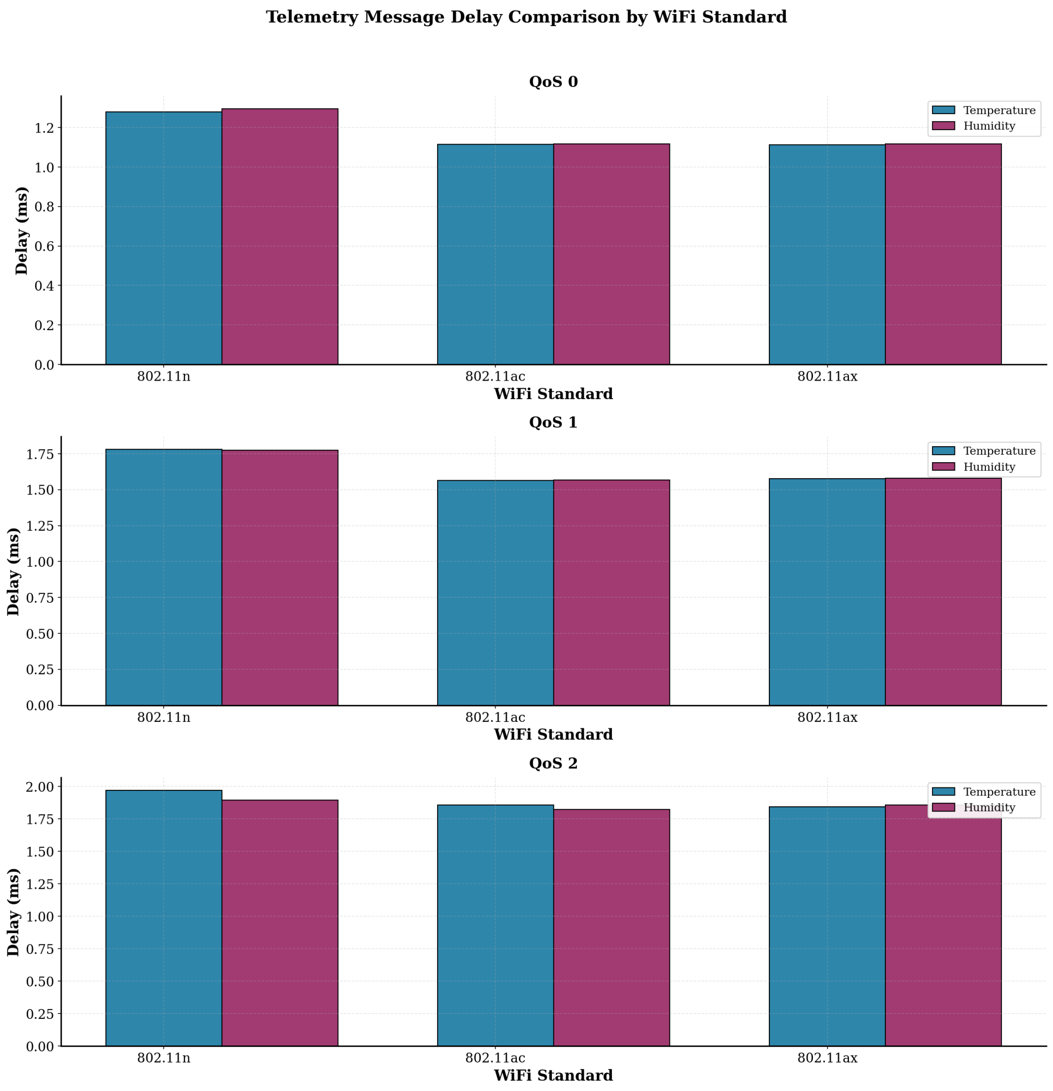

Fig. 10. WiFi Telemetry Results for Smart Warehouse Digital Twin Application

the incremental overhead introduced by MQTT acknowledgments for small payloads. As a result, improvements or changes in MQTT QoS levels do not significantly alter telemetry performance under 5G NR.

The Figure 14 depicts the results for operations messages over 5G NR connectivity. Package detection messages exhibits the highest latency, ranging from approximately 59 ms (QoS 0) to 55 ms (QoS 2). Other operation messages remain clustered around 22–34 ms, with only minor variations across QoS levels. Notably, increasing QoS does not consistently increase latency; in some cases QoS 2 achieves slightly lower delays than QoS 0, likely due to improved packet delivery reliability, fewer retransmission timeouts at the RLC/MAC layer, and more stable traffic scheduling. Overall,

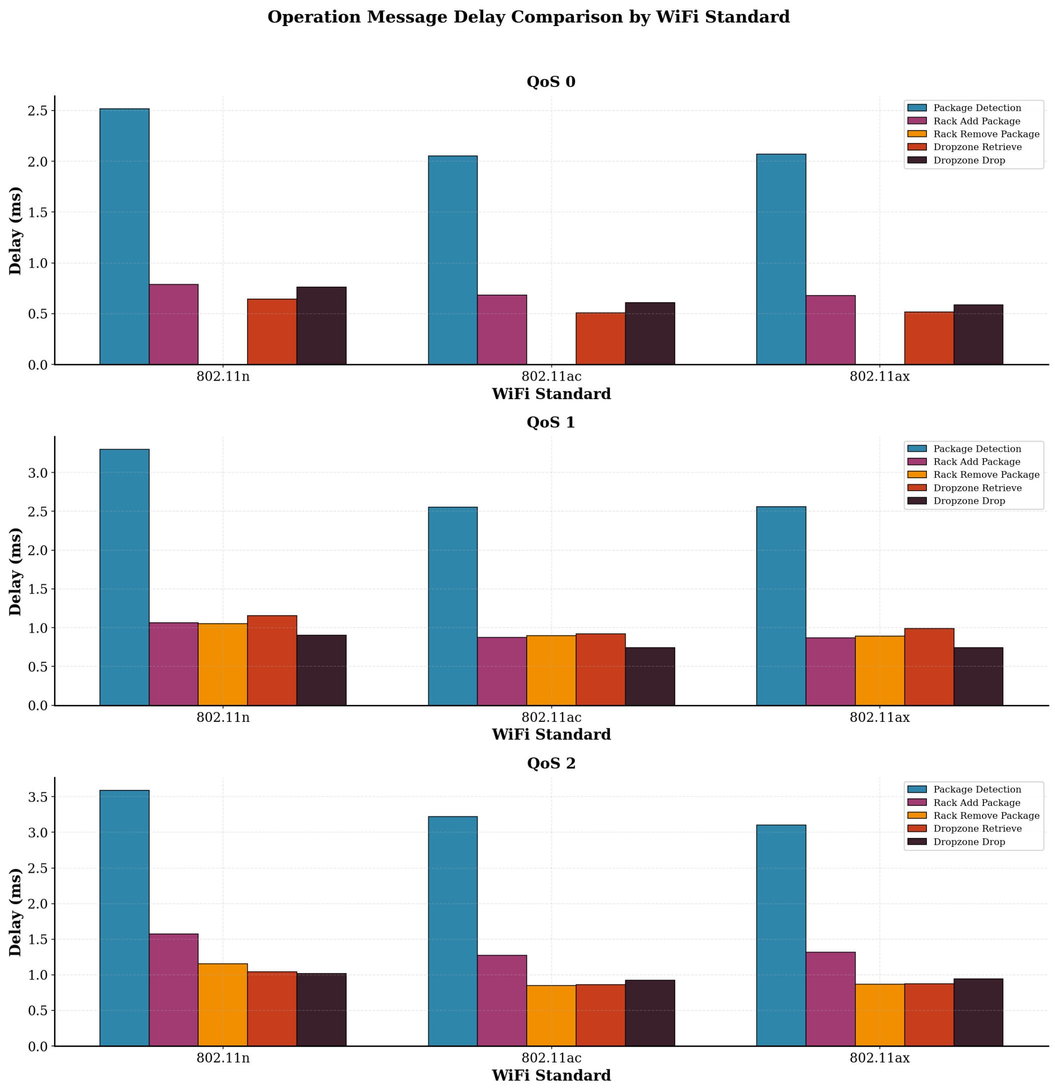

Fig. 11. WiFi Operations Results for Smart Warehouse Digital Twin Application

the results indicate that radio access and scheduling processes dominate latency performance for operation traffic, while MQTT acknowledgment overhead remains secondary.

The Figure 15 depicts the results for control messages over 5G NR connectivity. Task assignment delays ranges between 31.7 ms (QoS 0) and 39.2 ms (QoS 1/2), while task completion messages consistently remain lower at approximately 25–26 ms across all QoS levels. The modest increase for assignment commands under higher QoS reflects the additional acknowledgment exchanges but remains comparatively small relative to the baseline radio access and scheduling delays inherent to the 5G system. Importantly, the differences between QoS levels are limited to single-digit milliseconds, confirming that control message latency is primarily constrained by wireless air-interface and core-network delays rather than MQTT protocol semantics. For real-time robotic

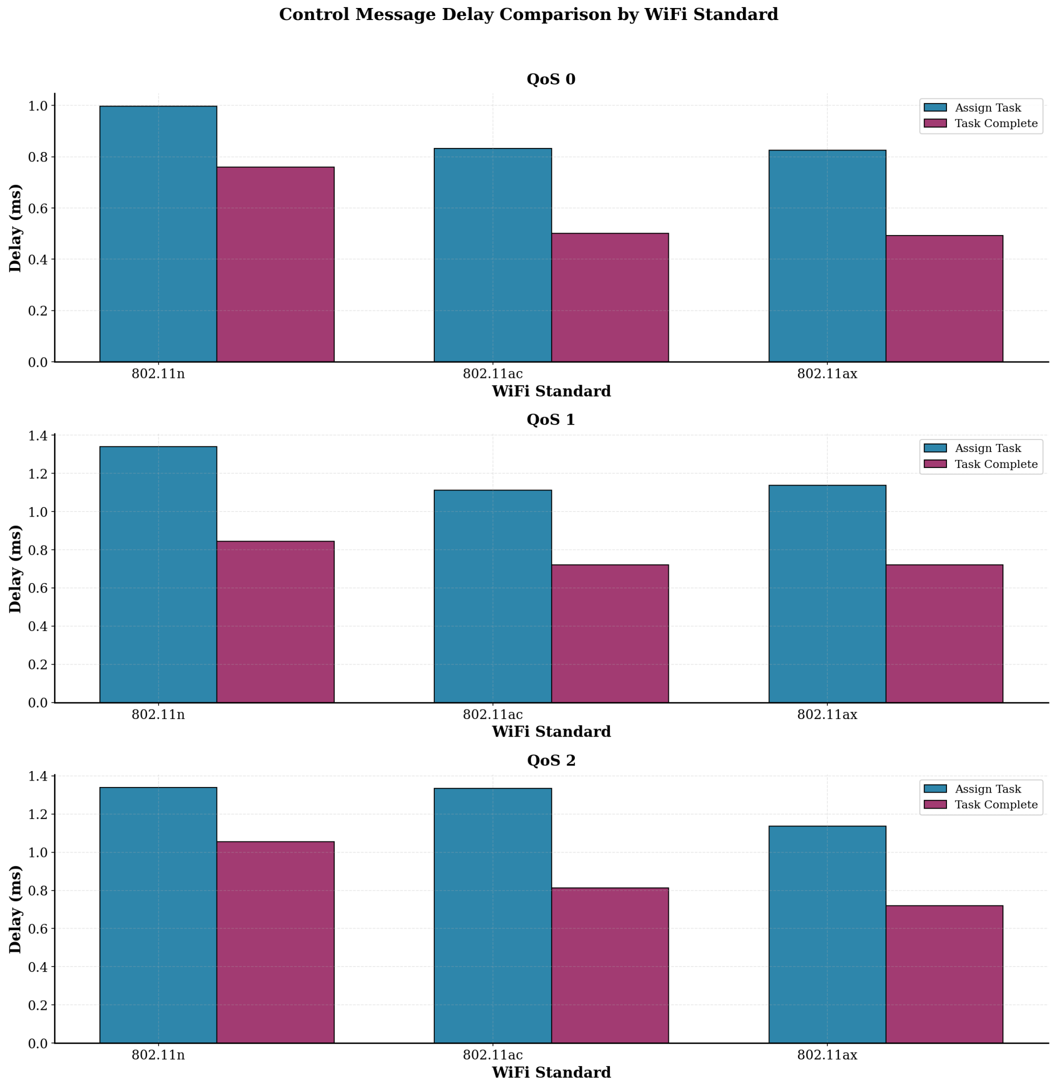

Fig. 12. WiFi Control Results for Smart Warehouse Digital Twin Application

coordination, this suggests that reliability-enhancing QoS levels can be safely employed over 5G NR without severely degrading control responsiveness.

5 Conclusion

This study successfully bridges a significant gap in network simulation tools by introducing a native, open-source MQTT v3.1.1 module for ns-3, enabling rigorous validation of Industrial IoT environments which can be found in https://gitlab.com/simulations9070736/mqtt.git. Through the implementation of a comprehensive Smart Warehouse Digital Twin scenario, we demonstrated that while MQTT is inherently lightweight, its reliability for mission-critical control is heavily dependent on the underlying network infrastructure. Our comparative analysis reveals that 5G

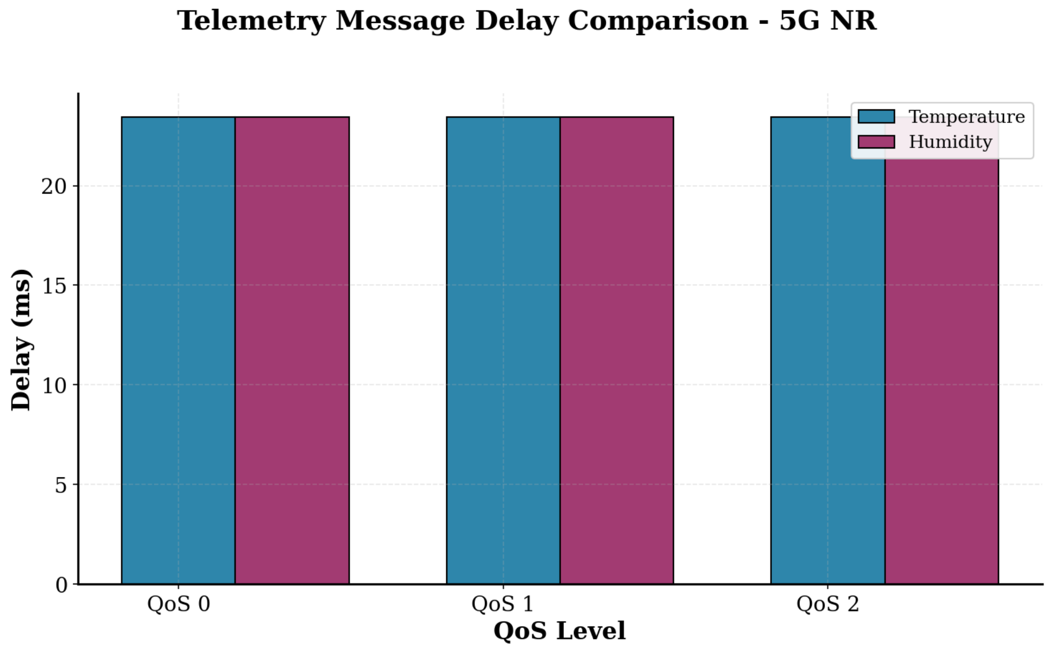

Fig. 13. 5G Telemetry Results for Smart Warehouse Digital Twin Application

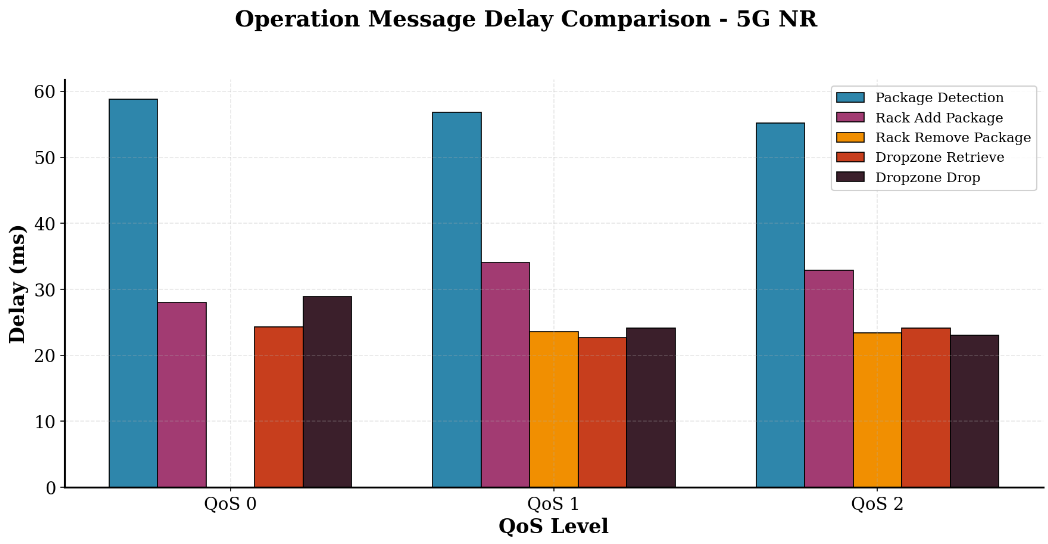

Fig. 14. 5G Operations Results for Smart Warehouse Digital Twin Application

New Radio (NR) provides the deterministic latency required for robot synchronization, effectively mitigating the protocol overhead observed in higher Quality of Service (QoS) levels, whereas Wi-Fi 6 remains a viable, cost-efficient option for high-bandwidth telemetry where jitter is less critical.

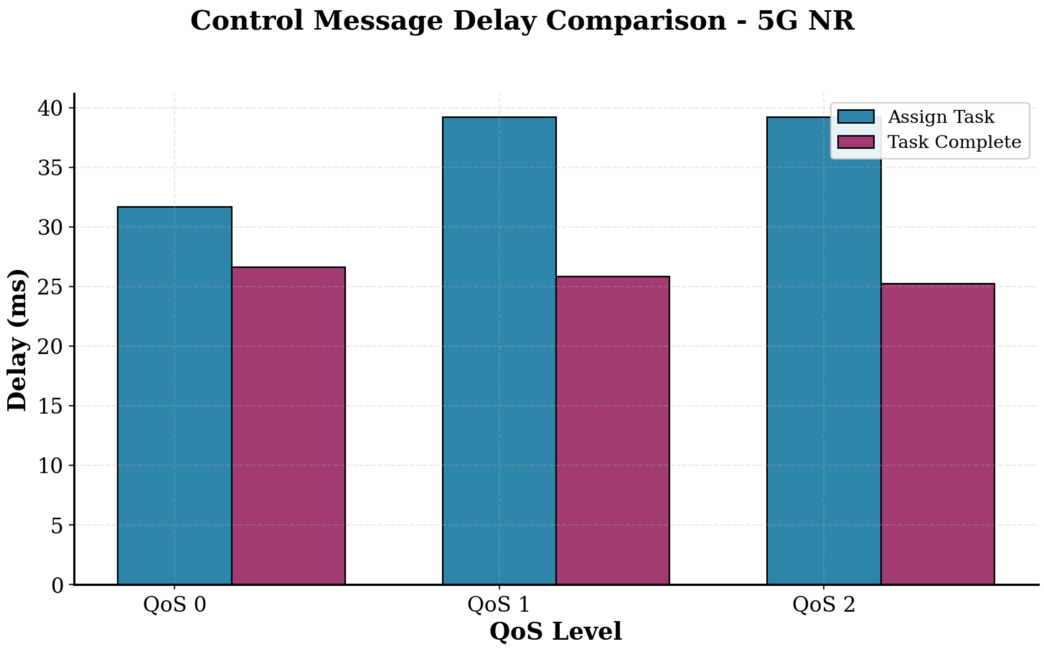

Fig. 15. 5G Control Results for Smart Warehouse Digital Twin Application

References

[1] A. Banks and R. Gupta *MQTT Version 3.1.1 Plus Errata 01*, OASIS Standard Incorporating Approved Errata 01, 10 December 2015. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.html. Latest version: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html.

[2] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[3] R. A. Light, "Mosquitto: server and client implementation of the MQTT protocol," *Journal of Open Source Software*, vol. 2, no. 13, p. 265, 2017.

[4] M. B. Yassein, W. Q. Al-Sarayrah, and A. Al-Zoubi, "Performance Analysis of MQTT Protocol in IoT," in *Proc. 2023 IEEE International Conference on Engineering & MIS (ICEMIS)*, Istanbul, Turkey, 2023, pp. 1–6.

[5] S. K. Prasad, J. Rachna, and A. Khalaf, "Performance Evaluation of MQTT Protocol in Internet of Things," *Telecommunications and Radio Engineering*, vol. 82, no. 8, pp. 45–58, 2023.

[6] T. T. D. Nguyen, G. Armitage, and P. Branch, "Performance Impact Analysis of Securing MQTT Using TLS," in *Proc. 2021 ACM/SPEC International Conference on Performance Engineering*, 2021, pp. 1–12.

[7] A. F. Gentile, D. Macrì, D. L. Carnì, E. Greco, and F. Lamonaca, "A Network Performance Analysis of MQTT Security Protocols with Constrained Hardware in the Dark Net for DMS," *Applied Sciences*, vol. 14, no. 18, p. 8501, 2024.

[8] J. Silva, P. Pedreiras, and L. Almeida, "A Scalable Real-Time SDN-Based MQTT Framework for Industrial Applications," *IEEE Open Journal of the Industrial Electronics Society*, vol. 5, pp. 102–115, 2024.

[9] F. Tao, H. Zhang, A. Liu, and A. Y. C. Nee, "Digital Twin in Industry: State-of-the-Art," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 4, pp. 2405–2415, Apr. 2019.

[10] T. H. Nguyen, Z. J. Wang, and V. C. M. Leung, "Digital Twin Empowered Industrial IoT Based on Credibility-Weighted Swarm Learning," *IEEE Transactions on Industrial Informatics*, vol. 20, no. 1, pp. 567–578, Jan. 2024.

[11] M. A. Al-Garadi, A. Mohamed, A. Al-Ali, X. Du, and M. Guizani, "A Survey of Machine and Deep Learning Methods for Internet of Things (IoT) Security, " *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1646–1685, 2020.

[12] Y. Han, W. Zhang, Z. Niu, and S. Zhou, "Machine and Deep Learning for Digital Twin Networks: A Survey," *IEEE Internet of Things Journal*, vol. 11, no. 2, pp. 1234–1256, Jan. 2024.

[13] L. U. Khan, W. Saad, D. Niyato, Z. Han, and C. S. Hong, "Digital-Twin-Enabled 6G: Vision, Architectural Trends, and Future Directions," *IEEE Communications Surveys & Tutorials*, vol. 24, no. 4, pp. 2234–2275, 2022.

[14] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network simulations with the ns-3 simulator," *SIGCOMM Demonstration*, vol. 14, no. 14, p. 527, 2008.

[15] M. Augusto, "MQTT for NS-3," Technical Report, Jan. 2023. Last accessed on Nov. 20, 2025. [Online]. Available: https://sourceforge.net/projects/mqttforns3/.

[16] M. A. Salimee, M. A. Pasha, and S. Masud, "NS-3 Based Open-Source Implementation of MQTT Protocol for Smart Building IoT Applications," in *Proc. 2023 Int. Conf. Communication, Computing and Digital Systems (C-CODE)*, Islamabad, Pakistan, May 2023, pp. 1-6. doi: 10.1109/C-CODE58145.2023.10139859.

[17] F. A. Alenizi and F. Al-Turjman, "A Survey on IoT Simulators: A Performance Comparison," *IEEE Access*, vol. 11, pp. 12345–12360, 2023.

[18] R. Maldonado, A. Karstensen, G. Pocovi, and A. A. Esswein, "Comparing Wi-Fi 6 and 5G Downlink Performance for Industrial IoT," *IEEE Access*, vol. 9, pp. 123456–123467, 2021.

[19] S. Deronne, T. R. Henderson, and S. Roy, "Validation of Wi-Fi Models in ns-3," in *Proc. of the Workshop on ns-3 (WNS3)*, 2019, pp. 1–8.

[20] 5G-ACIA, "Integration of 5G with Time-Sensitive Networking for Industrial Communications," 5G Alliance for Connected Industries and Automation, White Paper, 2021.