Contents lists available at ScienceDirect

Ad Hoc Networks

journal homepage: www.elsevier.com/locate/adhoc

On the performance of Zenoh in Industrial IoT Scenarios

Miguel Barón [a,b], Luis Diez [b,*], Mihail Zverev [a], José R. Juárez [a], Ramón Agüero [b]

[a] Ikerlan Technology Research Center, Arizmendiarrieta 2, Arrasate/Mondragón, 20500, Spain
[b] Department of Communications Engineering, Universidad de Cantabria, Plaza de la Ciencia s/n, Santander, 39005, Spain

ARTICLE INFO

Keywords:
Zenoh
MQTT
Reliability
Congestion control
IIoT
Latency
Middleware

ABSTRACT

Robust and efficient communication frameworks have become essential for the advancement of manufacturing and industrial processes in the era of Industry 4.0. This paper presents a comprehensive performance analysis of Eclipse Zenoh, a promising solution for the Industrial Internet of Things (IIoT). The analysis is conducted using a real testbed built with Raspberry Pi devices, comparing Eclipse Zenoh's performance against the widely used Message Queuing Telemetry Transport (MQTT) protocol. The study assesses Eclipse Zenoh's capabilities in terms of latency, as well as its reliability and congestion control mechanisms over various network topologies, using both Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). The results indicate that Eclipse Zenoh offers significant advantages in specific scenarios, making it a compelling choice for IIoT applications. Additionally, this paper contributes to a deeper understanding of Eclipse Zenoh's underlying principles and its communication capabilities, positioning it as a versatile and efficient solution for modern industrial environments.

1. Introduction

In the era of Industry 4.0, manufacturing and industrial processes have undergone profound transformations, revealing the importance of connectivity, data exchange, and automation [1]. Within this context, Industry 4.0 encompasses smart manufacturing systems, where machines are interconnected through the IIoT, enabling real-time data sharing and decision-making. This paradigm shift relies on robust communication frameworks that ensure reliable and efficient data transfer. In the automotive sector, Vehicle-to-everything (V2X) communication plays a crucial role in assisted and autonomous driving, allowing vehicles to communicate with each other and with the underlying infrastructure. Similarly, in robotics, Robot Operating System (ROS) [2] has emerged as a standard framework for developing robot applications [3], which require reliable middleware solutions to enable appropriate communication between various robotic components.

In the context of communication solutions, Eclipse Zenoh has recently gained recognition as a promising application-layer protocol in both the robotics and automotive sectors due to its versatile capabilities. In robotics, Eclipse Zenoh has already been adopted, particularly to support Robot-to-Anything (R2X) communications over wireless networks [4]. It has become the first non-Data Distribution Service (DDS) protocol to be natively supported in ROS 2 [5], addressing the *walled garden* problem [6], where proprietary or less compatible protocols limit the interoperability. This has led to an increase in commercially deployed solutions based on Eclipse Zenoh. In the automotive industry, its relevance is growing due to its suitability for both in-vehicle and V2X communications [4,7]. Its flexibility, which allows it to run across a broad range of platforms, from microcontrollers to datacenters [7], makes it an attractive choice for V2X applications. Furthermore, considering its performance and scalability, Eclipse Zenoh has recently been identified by the International Telecommunication Union (ITU) as the most appropriate protocol for these applications [8]. On the other hand, existing information about how Zenoh exploits the underlying communication protocols is still very limited, and this may hinder its operation in certain scenarios.

In this paper, a comprehensive performance analysis of Eclipse Zenoh over various network topologies is discussed, leveraging a real testbed implemented with Raspberry Pi devices. In order to evaluate the potential advantages offered by Zenoh, its performance is compared against that of MQTT. Considering the growing importance of mobility in robotics and vehicular applications, where Eclipse Zenoh is being proposed for adoption, tests are conducted to emulate different node velocities as they move closer to and farther from the access point, assessing how the two aforementioned protocols react to such mobility situations.

While the comparative analysis focuses primarily on the behavior of MQTT and Eclipse Zenoh over TCP [9] as the transport layer, the paper also explores Eclipse Zenoh's capability to ensure data reliability and handle potential network congestion scenarios by experimenting with

* Corresponding author.
E-mail address: diezlf@unican.es (L. Diez).

https://doi.org/10.1016/j.adhoc.2025.103784
Received 4 October 2024; Received in revised form 19 January 2025; Accepted 26 January 2025
Available online 3 February 2025

UDP [10] as the transport protocol. Additionally, the study delves into Eclipse Zenoh's communication principles, depicting its most relevant features and characteristics.

Hence, the main contributions of this paper are:

- A comprehensive description of Eclipse Zenoh's underlying principles.
- A comparative analysis of Eclipse Zenoh and MQTT in terms of latency, over various network topologies.
- An assessment of MQTT and Eclipse Zenoh's performance in mobility scenarios, tested at different node velocities.
- An evaluation of Eclipse Zenoh's reliability and congestion control mechanisms, considering both TCP and UDP transmissions.

The remainder of the paper is structured as follows. Section 2 provides background information on MQTT and Eclipse Zenoh, introducing and comparing their fundamental aspects. Then, Section 3 provides a more detailed analysis of Eclipse Zenoh, focusing on its communication level, followed by a description of its reliability and congestion control mechanisms. Then, Section 4 discusses the results of the tests conducted, which entail latency, reliability, and congestion control mechanisms. Afterwards, Section 5 positions this work, providing an overview of the related state-of-the-art. Finally, Section 6 concludes the paper, and it also provides an outlook of our future work.

2. Background

In recent years, research in the field of application-layer protocols within the IIoT domain has experienced a remarkable surge. Protocols such as DDS [11], Constrained Application Protocol (CoAP) [12] or Apache Kafka [13] have been thoroughly studied, due to their relevance in industrial settings. Their ability to provide efficient, secure, and scalable communication solutions aligns well with the stringent requirements of IIoT environments, where interoperability, reliability, and real-time performance are paramount. Among them, MQTT stands out as one of the most widely adopted solutions [14]. Its lightweight and efficient design, flexible Publish/Subscribe (Pub/Sub) model, persistent connection capabilities, and support for Quality of Service (QoS), have facilitated its adoption across several verticals [15]. Meanwhile, Eclipse Zenoh [6] has recently emerged as a prominent contender, offering unique features and promising applications well-suited to IIoT environments.

2.1. MQTT

MQTT [16,17] is a lightweight application-layer protocol specifically designed for Internet of Things (IoT) applications. Due to its small code footprint and ease of integration, it is a convenient choice for low-power sensor networks that require low bandwidth and Machine-to-Machine (M2M) environments.

MQTT uses a Point-to-Point (PtP) communication model based on the Pub/Sub pattern. In MQTT nodes can take two roles: client or broker. In turn, there are two client roles: (1) *publishers*, which publish messages on specific topics; and (2) *subscribers*, which subscribe to those topics to receive the corresponding information. Then, the broker acts as an intermediate entity between publishers and subscribers, and it is responsible for receiving, storing and forwarding published messages to subscribed clients. MQTT offers three levels of QoS to ensure reliable message delivery:

- QoS 0: At most once delivery. The message is sent without acknowledgment of receipt and may reach the destination once or not at all.
- QoS 1: At least once delivery. The message is resent until the receiver confirms its successful reception via a `PUBACK`, ensuring it arrives at least once at the destination.
- QoS 2: Exactly once delivery. A four-step exchange process (`PUBLISH`, `PUBREC`, `PUBREL`, `PUBCOMP`) is established between sender and receiver to ensure the unique delivery of the message, avoiding duplication.

The typical implementation of MQTT uses TCP at the transport layer. Additionally, security may be provided by means of the Transport Layer Security (TLS) [18] protocol. The standard port for MQTT is `1883`, although port `8883` is reserved for MQTT connections using TLS [16,17]. Nevertheless, recently alternative implementations have been proposed to leverage novel transport solutions. For instance, in [19] MQTT is implemented over QUIC [20] to better fulfill certain performance requirements, such as latency.

2.2. Zenoh

Eclipse Zenoh [6] is a protocol that supports not only real-time data transmission but also the management of data at rest through *storages*, alongside the capability to execute queries directly within the system. Thus, it operates as a Pub/Sub/Query protocol, offering a more holistic solution than traditional solutions based on the Pub/Sub or Request/Response patterns. This provides significant flexibility and scalability, simplifying the development of distributed applications and reducing infrastructure complexity.

Typically, Eclipse Zenoh is conceived as an application-layer protocol, operating over TCP. However, it is designed to function with minimal transport requirements and it can therefore operate over any transport layer that provides point-to-point, packet-based communication, even with a best-effort delivery model. This, combined with its wire efficiency, which boasts a minimal overhead of just 5 B [4], ensures that Eclipse Zenoh can seamlessly operate across multiple layers of the protocol stack, both over Internet Protocol (IP) and non-IP networks. To this end, Eclipse Zenoh incorporates a Session Protocol that allows it to operate over either the link, network or transport layers. Besides, Eclipse Zenoh offers abstractions for best-effort and reliable channels, diverse priorities, and adjustable Maximum Transmission Unit (MTU) across all layers. For instance, by extending support to serial connections, it enables the integration of devices lacking traditional networking interfaces such as Wi-Fi, Bluetooth, or Ethernet into a Zenoh network. This versatility is particularly relevant in verticals such as robotics, transportation, and maritime, agricultural, or industrial applications, where conventional network technologies may not be adequate [21].

As illustrated in Fig. 1, Eclipse Zenoh has the flexibility to operate over the transport layer using protocols like TCP (with or without TLS), UDP, or QUIC; over the network layer, using either IP or IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN); or directly over the link layer, including Wi-Fi, Ethernet, Thread, Bluetooth, or serial links.

Eclipse Zenoh nodes can play a higher number of roles. Applications can be configured to operate either as clients or peers, depending on specific requirements. Additionally, Eclipse Zenoh allows for the deployment of a Zenoh router, which can load plugins to extend its functionality. These plugins enable features such as a REST API, access to various databases, and integration with protocols like DDS or MQTT. This enhances data accessibility, while offering greater flexibility, interoperability, and ease of migration, simplifying the overall adoption process.

Exploiting the aforementioned roles, Eclipse Zenoh supports a broader range of topologies. These include configurations such as *Clique* or *Mesh* in P2P communications, as well as those involving Zenoh routers, such as *Brokered* and *Routed* communications. This versatility makes Eclipse Zenoh a suitable solution for a wide array of applications within the Industry 4.0 paradigm.

Within a *Clique* topology, all peers establish direct communication with each other, creating a tightly interconnected network (termed

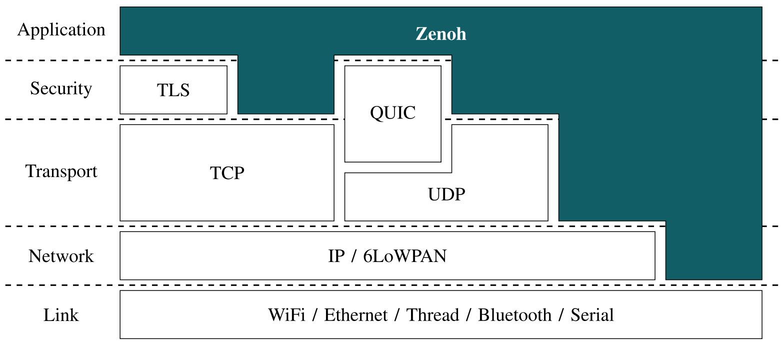


Fig. 1. Zenoh architecture. Based on [21].

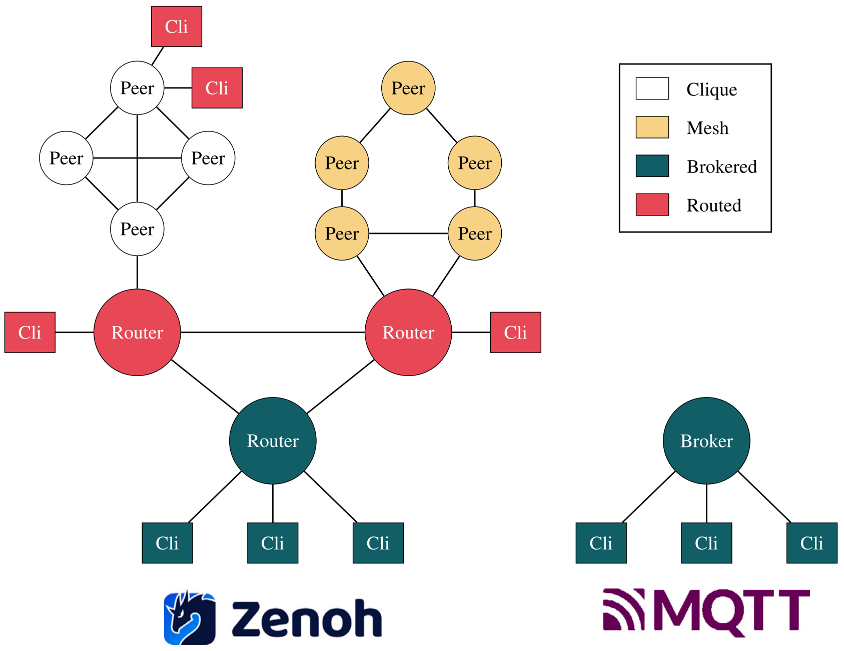


Fig. 2. Eclipse Zenoh and MQTT topologies. Based on [6].

`peer_to_peer` mode in the Eclipse Zenoh configuration file). Conversely, *Mesh* topologies assume a more selective approach to P2P communication, allowing nodes to selectively engage with specific peers, based on a defined criteria (referred to as `linkstate` mode). In *Brokered* communications, the Zenoh router assumes a broker-like role, enabling client communication across the network alike MQTT. This vision simplifies the communication process, enhancing efficiency [22, 23]. In contrast, *Routed* communications introduce a more dynamic approach, where every network node has the capability to communicate with others, by means of interconnected routers. Additionally, clients can establish connections with peers, allowing information to be routed. This decentralized mechanism ensures scalability and robust connectivity [22].

Fig. 2 illustrates the various topologies available in Eclipse Zenoh, which are compared to the brokered model in MQTT.

A more detailed and technical description of Eclipse Zenoh will be provided in Section 3.

3. Zenoh in detail

Eclipse Zenoh, as a protocol designed to enable efficient and scalable communications in distributed environments, incorporates several advanced features that allow it to operate flexibly and reliably. This section explores some of its core components, which enable Eclipse Zenoh to effectively manage data transmission across complex networks.

First, we delve into the scouting process, which allows nodes to efficiently discover and connect with other nodes. Next, we describe how Eclipse Zenoh handles port opening, facilitating the communication between nodes across different transport layers.

The Zenoh session, a key component of the protocol, is explained in terms of its complete cycle: from the initial handshake between nodes, through data transmission and reception, to the closure of the session.

Finally, we discuss the reliability and congestion control mechanisms that Zenoh implements to ensure stable communication in dynamic and high-performance networks.

3.1. Multicast scouting

Eclipse Zenoh enables direct configuration of the network nodes through a configuration file, allowing users to specify both the endpoints to connect to and those to listen on. This feature is particularly advantageous in scenarios that require precise and controlled network setups, as well as in static applications, where minimizing network overhead is a priority.

In addition, Eclipse Zenoh offers an automatic node discovery feature (scouting), which is useful in dynamic environments where the quick integration of new devices is essential. An illustrative use case for automatic discovery is V2X communications, where vehicles can connect to various devices as they move through different locations.

In Eclipse Zenoh, network node discovery typically relies on UDP scout messages. Any node seeking to join the network must subscribe to the multicast group `224.0.0.224` on UDP port `7446` (both address and port are configurable), using a frame that identifies its role within the network.

The specifics of the scouting mechanism may vary, depending on the type of node, as detailed in the following sections.

3.1.1. *Peers and routers*

Upon subscribing to the network, any peer keeps sending scout messages, indicating its role, to the multicast group at specific time intervals, which follow a geometric progression: 1, 2, 4, 8 s, remaining constant from the fourth round onward. On the other hand Zenoh routers behave similarly to peers in multicast scouting, and they also announce their role in the scout message.

Each time a not-client node joins the network, existing peers and Zenoh routers transmit their IP address and port (hereinafter, "@IP:Port") where they listen for incoming Zenoh communications (TCP, UDP, QUIC, etc.); we will refer to this as "Server Port". This transmission also follows the geometric progression mentioned earlier. Afterwards the connected not-client node sends its @IP:Port every 8 s to each node it had previously connected to. Thus, every peer and router in the network becomes acquainted with the Server Port of all other participants.

3.1.2. *Clients*

Clients send a single scout message to the multicast group. A Zenoh router will respond sending its @IP:Port (usually, `7447`) to establish the Zenoh connection. No more UDP frames are sent or received by clients to establish connections.

3.2. *Gossip scouting*

Particularly designed for P2P scenarios where multicast communications are not available, Eclipse Zenoh introduces gossip scouting. It allows peers to perform application-level gossiping to propagate the @IP:Port pairs of all discovered peers throughout the network. However, for a peer to start gossiping, it must have already established a connection with another node in the network. Therefore, it is essential for nodes to be connected to an initial entry point, typically a Zenoh router, to discover the rest of the network.

Since this mechanism is part of Eclipse Zenoh's session protocol, it involves specific Zenoh packet types, such as `OAM`, which will be explained later in Section 3.4. `OAM` is also included in Table 1, where Zenoh control packets are listed.

Unlike multicast scouting, the propagation of @IP:Port pairs does not occur periodically. Each time a new node enters the network, the node it connects to sends its table of IP address and Server Port assignments of the known nodes within a `Frame` (`OAM`) packet, see Table 1, to the newly discovered node. It then propagates the @IP:Port pair of the latter to the next hop, also encapsulated within a `Frame` (`OAM`) packet. Subsequently, the new node establishes connections with the other nodes and exchange their tables among themselves.

When not all nodes have direct connectivity with each other, it is also possible to enable *Mesh* communications in the network. In this situation peers are configured to operate in linkstate routing mode and multihop gossip scouting can be set up. This entails increased scouting traffic and reduced scalability, while gossip scouting information can be propagated across multiple hops to all nodes in the local network, ensuring complete knowledge of each other.

3.3. *Opening ports*

As depicted in Fig. 3, in Eclipse Zenoh the opening of ports for nodes in a network depends on their roles. The involved procedures under different topologies are described below.

3.3.1. *P2P communications*

To initiate a Zenoh communication with another peer, a peer only needs to open a port (hereinafter referred to as "Client Port") to establish a communication with the Server Port of each of the other peers in the network. In a P2P communication scenario, the first node only opens its Server Port. When the second node joins, it opens a Client Port to communicate with the existing node, and a Server Port. The third node would then open two Client Ports (one for each existing node) and a Server Port. This process continues with each new node joining the Zenoh network.

As can be observed, the number of open ports increases with the size of the network. Let P_{S_i} and P_{C_i} be the number of Server Ports and Client Ports, respectively, in the pair that is incorporated in the ith position of P2P communications. For all nodes, we have:

$$P_{S_i} = 1, P_{C_i} = i - 1$$

3.3.2. *Brokered communications*

Regarding *Brokered* communications, it has been observed that clients exclusively open a Client Port and do not require a Server Port. This Client Port is used to communicate through the Zenoh router, which, in turn, opens a Server Port, but does not need to open any Client Ports to interact with the clients.

Using the same notation as in P2P communications, and if the Zenoh router corresponds to node 0, the number of Server and Client Ports that the router and the other nodes open can be expressed as:

$$P_{S_0} = 1, \quad P_{C_0} = 0; \quad P_{S_i} = 0, P_{C_i} = 1 \quad \forall i \neq 0$$

3.3.3. *Routed communications*

In a *Routed* communication scenario, the configuration incorporates elements of the two aforementioned modes. The Zenoh router is capable of opening Client Ports to communicate through the Server Ports of other peers or Zenoh routers. Meanwhile, client nodes only open a Client Port, which they use to connect either to the router or directly to another peer.

If we have a set of client nodes $\mathcal{N}_C$ and non-client nodes $\mathcal{N}_N$ (either routers or peers), the number of open port for both node types can be defined as:

$$P_{S_i} = 1, P_{C_i} = i - 1 \quad \forall i \in \mathcal{N}_C;$$

$$P_{S_j} = 0, P_{C_j} = 1 \quad \forall j \in \mathcal{N}_N$$

3.4. *Zenoh session*

Upon discovering a new node, an application-layer exchange of frames is initiated to establish the communication, opening the corresponding channel: (1) `InitSyn`, (2) `InitAck`, (3) `OpenSyn`, and (4) `OpenAck`, as illustrated in Fig. 4 (see Table 1).

Zenoh communication relies on the transmission and reception of values associated with different *keys*. Throughout a connection or session, values can be transmitted or removed using `Push` frames (`Put`/`Del`). Entities that frequently send values to the same *key expression* can be declared as *publishers*. It is important to note that this declaration serves as an optimization technique to enhance system efficiency, especially in scenarios involving frequent data transmission. However, it is not mandatory, as any application can perform write operations (`Put`) without having explicitly been designated as a *publisher*.

To receive all updates related to one or more *keys*, an entity within a node can be declared as a *subscriber* to a specific *key expression*, or as a *pull subscriber*, so that it decides when to retrieve new data samples from the nearest node as they become available. Alternatively, the entity can request specific data through *queries*. These requests are handled by processes declared as *queryables* for such *key expressions*. As shown in Table 1, information queries are dispatched within `Request` frames, and replies are delivered via `Response` frames.

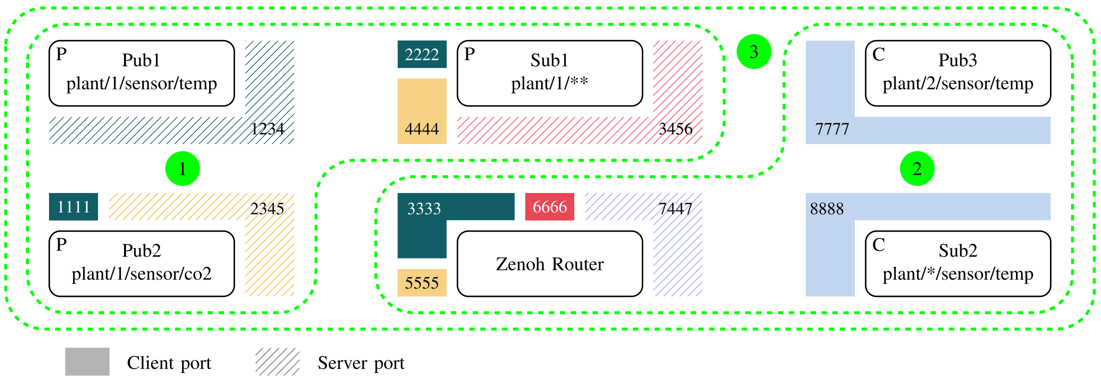


Fig. 3. Port configuration in Eclipse Zenoh. This figure presents six different Zenoh applications grouped into three scenarios: (1) Scenario 1: Three pairs (two publishers (Pubs) and one subscriber (Sub)) forming a *Clique* topology; (2) Scenario 2: Two clients (one Pub and one Sub) and a Zenoh router, forming a *Brokered* topology; (3) Scenario 3: All six nodes combined, forming a *Routed* topology. The figure illustrates the ports opened by each node, distinguishing between Server Ports and Client Ports, and highlights the paths of direct communication between nodes. The color of the ports indicates direct communication: nodes with ports of the same color communicate directly with each other. The figure also specifies the key expressions to which nodes publish or subscribe.

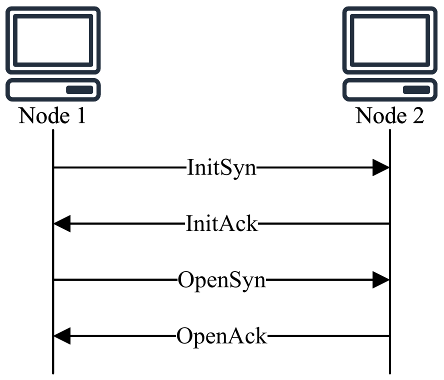


Fig. 4. Eclipse Zenoh Four-Way Handshake.

Once the Zenoh session is established, nodes can start exchanging `Declare` frames, indicating subscription interest or availability for querying specific *keys*. Nodes retain the ability to undeclare at their discretion.

To maintain the connection, nodes transmit `KeepAlive` frames if no data has been exchanged during a specified interval set to one-fourth of the session lease time, which is, by default, 10 seconds. Therefore, in the absence of data exchange, `KeepAlive` frames are sent every 2.5 s. Session closure is conducted through a `Close` frame.

All control packet types and their corresponding meaning are summarized in Table 1, along with their hexadecimal values, which were obtained from Wireshark captures.

3.5. Reliability and congestion control mechanisms

In distributed systems, ensuring both reliability and efficient congestion control is crucial to ensure robust communications. Eclipse Zenoh offers a comprehensive framework to appropriately manage these two aspects.

It provides various reliability strategies to accommodate different application or service requirements while maintaining scalability as follows:

- Hop to hop reliability. The default strategy in Eclipse Zenoh ensures reliability at each hop in the network. While offering good scalability, it may result in data loss during topology changes.
- End to end reliability. This strategy establishes a reliability channel between each data producer–consumer pair, minimizing data loss even during topology changes. However, it is less scalable and may induce higher resource consumption.
- First router to last router reliability. By establishing a reliability channel between the first and last routers of each data route, Eclipse Zenoh offloads pressure from data producers and consumers, enhancing scalability.

The reliability strategies in Eclipse Zenoh are facilitated by two core protocols: the session and the routing protocols. The first one establishes and manages bidirectional connections between the different nodes in the Zenoh network. It handles the setup and maintenance of communication channels, including both best-effort and reliable channels, and it optimizes network usage through techniques like automatic batching and fragmentation.

- Best-Effort Channel. This channel allows the transmission of data without any delivery guarantees nor ordered reception. It is suitable for non-critical data where occasional losses are acceptable.
- Reliable Channel. This channel ensures the delivery of all data in the correct order. It is suitable for services having critical data, where integrity and reliability are essential.

As mentioned in Section 3.4, producers are applications that perform write operations (`put()`), regardless of whether they have been explicitly declared as *publishers*. Consumers, on the other hand, are applications that subscribe to *key expressions* or request data via *queries*.

The routing protocol allows the session protocol to propagate interests, sending data from producers to consumers in the Zenoh network. It establishes the most efficient path for data transmission, ensuring timely and reliable delivery. This protocol is essential for directing data flows effectively across the network, regardless of the chosen reliability strategy.

In Eclipse Zenoh, reliability is managed by data consumers, which specify whether they require the retransmission of lost messages (`RELIABLE()`) or not (`BEST_EFFORT()`). Conversely, producers fix the congestion control to be applied during the session in the event of significant network congestion. They decide, for each message, whether it can be discarded (`DROP()`) or if the publisher should stop sending it (`BLOCK()`) upon a congestion situation, prioritizing message delivery over other operations. This strategy is propagated to all involved nodes and implemented across the entire route. Both congestion control and reliability mechanisms are summarized in Table 2.

4. Performance evaluation

In this section we discuss the performance of Eclipse Zenoh and MQTT. All tests aim to evaluate two key aspects of IIoT scenarios: latency and reliability.

Table 1
Zenoh control packets.

Packet types			Description	Hex code
InitSyn			Node requests to initialize a connection to another node.	81
InitAck			Initialization acknowledgment.	a1
OpenSyn			Node initiates a session for communication with another node.	42
OpenAck			Opening acknowledgment.	62
Close			Node closes the connection with another node.	03
KeepAlive			Connection maintenance.	04
Frame	Declare	DeclareKeyExpr	Declaration of the key expression utilized.	25
		DeclareSubscriber	Declaration of a subscriber.	
		DeclareQueryable	Declaration of a queryable.	
		UndeclareSubscriber	Undeclaration of a subscriber.	
		UndeclareQueryable	Undeclaration of a queryable.	
	OAM		Transmission of nodes network information.	
	Push	Put	Sending of a value.	
		Del	Deletion of a value.	
	Request	Query	Sending of a query.	
	Response	Reply	Reply to a query.	

Table 2
Congestion control and reliability mechanisms.

Mechanism	Value	Definition	Decision
Congestion control	`BLOCK()`	Ensures messages not being dropped under any circumstances.	Producer
	`DROP()`	Allows the message to be discarded if all buffers are full.	
Reliability	`RELIABLE()`	Informs the network that it is required all publications to be reliably delivered.	Consumer
	`BEST_EFFORT()`	Informs the network that it is acceptable to discard some messages.	

First, latencies of both protocols are compared under different reliability configurations across various topologies, including P2P, centralized, and distributed architectures, common setups in industrial environments. Additionally, the impact of node movement within the network is also studied. This includes scenarios where a node is placed at the cloud, or moved away from the access point, emulating mobility conditions.

For Eclipse Zenoh, the analysis also explores the differences in packet loss and delay when using both UDP and TCP at the transport layer. These tests cover all available reliability and congestion control configurations (see Table 2).

The experiments were conducted over a physical testbed, which comprised *Raspberry Pi 3 Model B V1.2* and *B+*, operating with *Linux raspberrypi 6.1.21-v7+* and *Raspbian GNU/Linux 11 (bullseye)* on different hosts. Wireless connections were established through an *Asus RT-N18U* 2.4 GHz Wi-Fi Router, capable of delivering speeds of up to 600 Mbps (*802.11n* standard in infrastructure mode). Additionally, a *NETGEAR GS108 Gigabit Ethernet* Switch was employed for wired connections. Notably, mobility scenarios incorporated two STE2300 shielded test enclosures,[1] alongside a 90 dB Programmable Attenuator RCDAT-8000-90.[2]

As for protocol implementation, we used the Eclipse Zenoh Python API,[3,4] which is built upon Eclipse Zenoh's primary Rust implementation.[5] The Zenoh router (*zenohd*[6]) has been configured to listen on TCP port 7447. To analyze Zenoh traffic, we used the Eclipse Zenoh Dissector[7] for Wireshark 4.0.10.

Regarding MQTT clients and brokers, we relied on the Eclipse Paho,[8] MQTT client library and Eclipse Mosquitto[9] respectively. The MQTT server was configured to accommodate remote clients on port 1883.

4.1. Comparative latency analysis

In the first set of experiments, we compare Eclipse Zenoh and MQTT in terms of latency, considering scenarios typically seen in industrial environments, deploying various applications that emulate sensors or actuators, and routing communication through routers or brokers. In all tests, a wireless connection was established between the two nodes of the network, mimicking the growing presence of wireless technologies in M2M and IIoT communications [24]. Notably, one Zenoh application operates in peer mode across all scenarios, and TCP is consistently used as the transport protocol in all tests.

As depicted in Fig. 5, three distinct scenarios (α, β, γ) were considered, comprising three (α, β) or four hosts (γ):

- Scenario α. Two brokered applications within the same network, with full wireless connectivity.
- Scenario β. Two brokered applications in two different networks, one of them using Ethernet.
- Scenario γ. Two routed applications connected through two routers across three different networks, with the two edge networks using Ethernet.

For Eclipse Zenoh, an additional scenario δ was considered. Its performance in P2P topologies with wireless connectivity is analyzed to assess the advantages this approach offers compared to the other network configurations.

The latency for the different configurations was obtained by measuring the application level RTT, based on the communication between publishers and subscribers, through a request/response mechanism.

[1] https://ctscorp-usa.com/wp-content/uploads/2018/03/STE2300_Data_Sheet-CTS.pdf.
[2] https://www.minicircuits.com/pdfs/RCDAT-8000-90.pdf.
[3] https://github.com/eclipse-zenoh/zenoh-python (accessed on 24 September 2024), version 0.10.0-rc.
[4] https://zenoh-python.readthedocs.io/en/0.10.0-rc/ (accessed on 24 September 2024).
[5] https://github.com/eclipse-zenoh/zenoh (accessed on 24 September 2024).
[6] https://github.com/eclipse-zenoh/zenoh/tree/main/zenohd (accessed on 24 September 2024), version v0.10.1-rc-1-g15b36a0f.
[7] https://github.com/ZettaScaleLabs/zenoh-dissector (accessed on 24 September 2024).
[8] https://github.com/eclipse/paho.mqtt.python (accessed on 24 September 2024), version 1.6.1.
[9] https://github.com/eclipse/mosquitto (accessed on 24 September 2024), version 2.0.11.

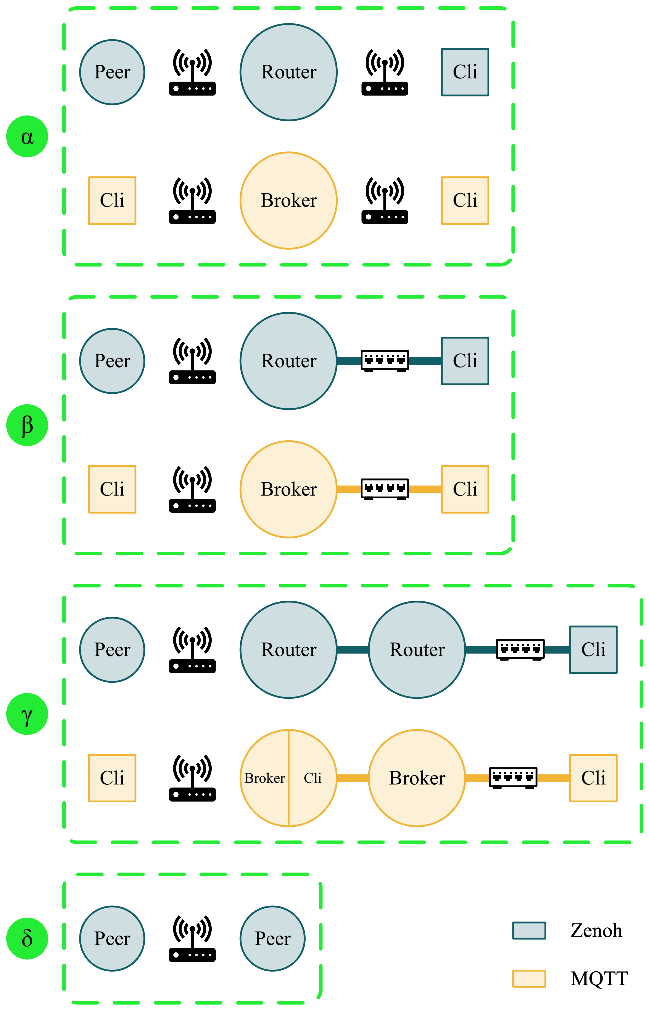


Fig. 5. Topologies of the four proposed scenarios to evaluate the performance of applications in different network configurations: (1) scenario α: two brokered applications within the same network, using wireless connectivity; (2) scenario β: two brokered applications in two different networks, with the second network connected via Ethernet; (3) scenario γ: two routed applications through two routers across three different networks, with the last two networks wired via Ethernet; (4) scenario δ: two P2P applications within the same network, using wireless connectivity; For scenarios α, β, and γ, Eclipse Zenoh is depicted on the left and MQTT on the right. The technology used in each network (Wi-Fi or Ethernet) is illustrated accordingly. The term "Cli" refers to Zenoh applications operating in client mode.

The RTT is calculated by subtracting the initial time t_0 from the final time t_1. In general, each scenario comprises a sequence of nodes, see Fig. 5, which allow the communication between the edge ones. The value of t_0 is recorded when the first node in the network sends data to a key the last node is subscribed to. This value is then delivered to the last node, possibly through intermediate nodes, if they exist. When the last node receives the message, it publishes a confirmation message (acknowledgment (ACK)) with a payload of 2 B to another key the first node in the network is subscribed to. Time t_1 is recorded when the first node receives such ACK. This process is repeated 1000 times without adding any waiting time between reception and transmission, to obtain a tighter latency average value.

For both protocols, a session was established at the first and last nodes with a client that publishes on and subscribes to distinct keys/topics. On the first node, the client publishes on the key/topic *test/data*, and it subscribes to *test/ack*. Conversely, the client at the last node subscribes to *test/data* and publishes on *test/ack*.

Regarding reliability and congestion control, two scenarios were considered for Eclipse Zenoh: (1) `RELIABLE` and `BLOCK`, and (2) `BEST_EFFORT` and `DROP`. For MQTT, analysis were conducted for its three levels of QoS (see Section 2.1). It is worth noting that QoS level 0 would align more closely to scenario (2) [25].

Zenoh applications were configured via configuration files, using the structure outlined in the schema[10] provided by ZettaScale Technology. Configuration parameters include the node type (peer or client) and the endpoint address for connection establishment. Multicast scouting was enabled, with the default settings, due to its potential use in IIoT environments. Additionally, `Peer_to_peer` was employed as the routing strategy for peers, while transport parameters were kept at their default values. For the Zenoh router, the default configuration of *zenohd* was used.

4.1.1. Scenario α: Brokered topology – Single network

The first scenario comprises a network that exclusively operates with Wi-Fi connectivity, with three nodes: a publisher, a Zenoh router/MQTT broker, and a subscriber.

Fig. 6 shows the box plot of the measured latency upon the different reliability setups. For each configuration, the box upper and lower limits represent the first and third quartiles (25th and 75th percentiles), respectively. The middle line within each box indicates the median (50th percentile), while the circle represents the mean. The whiskers extend to show the range that contains approximately 99% of the samples.

The results depicted in Fig. 6 evince that Eclipse Zenoh, using both *Best Effort* (BE) and *Reliable* (Rel.) mechanisms, showcases a performance similar to that observed for MQTT with QoS levels 0 and 1, yielding slightly lower latencies in both cases. Eclipse Zenoh significantly outperforms MQTT QoS 2, which might not be a sensible solution when employing TCP as the underlying transport protocol. This is due to TCP's implementation of flow control, congestion control, and packet retransmission mechanisms, which aim at ensuring minimal packet loss. Nevertheless, the adoption of MQTT with QoS 2 might be a sensible choice in applications requiring an additional level of reliability and message delivery assurance.

4.1.2. Scenario β: Brokered topology – Multiple networks

In this second scenario, one of the wireless connections (Zenoh router/MQTT Broker to client) is replaced with an Ethernet one, emulating a cloud-based setup. Over this scenario two different configurations are considered, as shown in Fig. 7. The first one mimics different distances to the cloud, which yield different delays, while the wireless connection remains the same. Then, we assume mobility in the wireless node.

For this first modification of scenario β, latency is varied using the Traffic Control (`tc`) Linux command. In particular, the performance of both protocols is compared when there is no delay and when the delay to the cloud instance is 200 ms.

We can observe that this setup yields, as was expected, lower latencies compared to scenario α, as shown in Fig. 8(a). The results also evince slight improvements when using Eclipse Zenoh, compared to MQTT. Fig. 8(b) depicts the obtained results with additional delay. As can be seen, there exists a much larger difference between Eclipse Zenoh and MQTT than before, reaching more than 0.5 ms difference in the RTT in the best-case scenario.

Additionally, mobility has been emulated by further modifying scenario β. As depicted in Fig. 9, two shielded test enclosures (STE2300) were used to house the devices, creating a controlled wireless environment. The first enclosure contained the Raspberry Pi implementing the first node (Zenoh peer or MQTT client), while the second housed the Wi-Fi router, the Zenoh router/MQTT broker and the third node

[10] https://github.com/eclipse-zenoh/zenoh/blob/master/commons/zenoh-config/src/lib.rs (accessed on 24 September 2024).

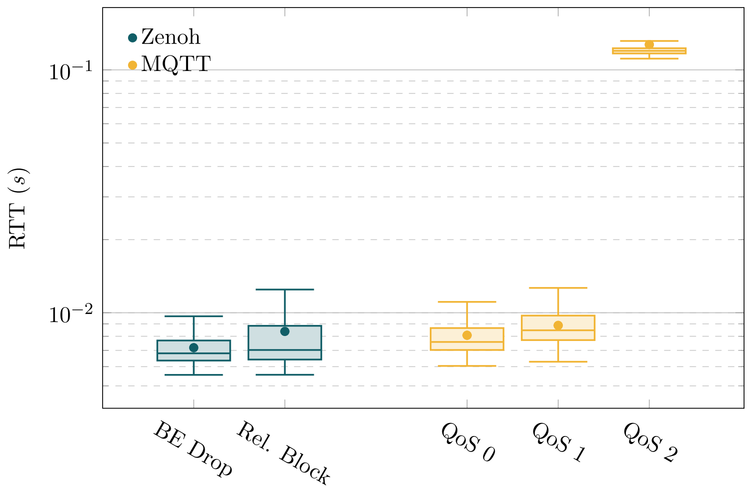


Fig. 6. RTT obtained in scenario α.

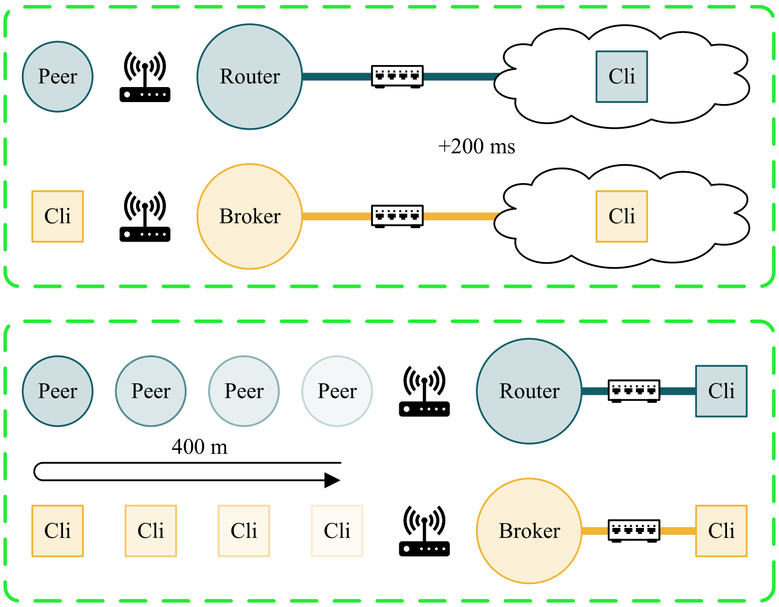


Fig. 7. Cloud-based and mobility approaches using scenario β.

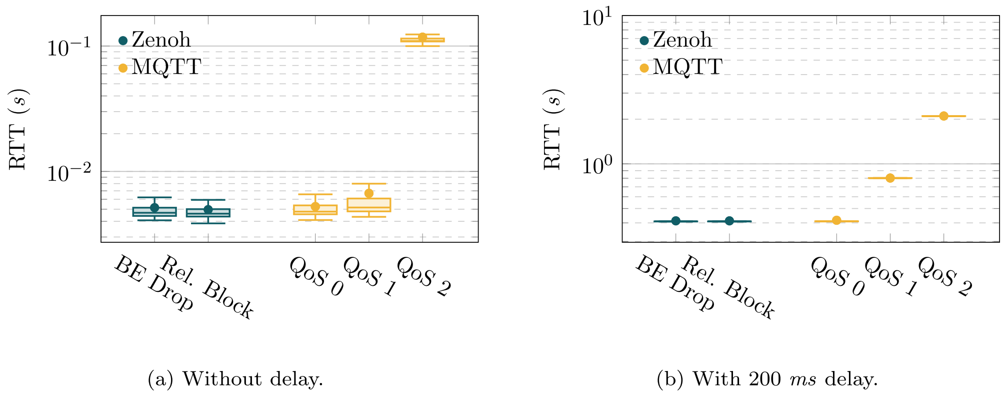


Fig. 8. RTT obtained in scenario β without motion.

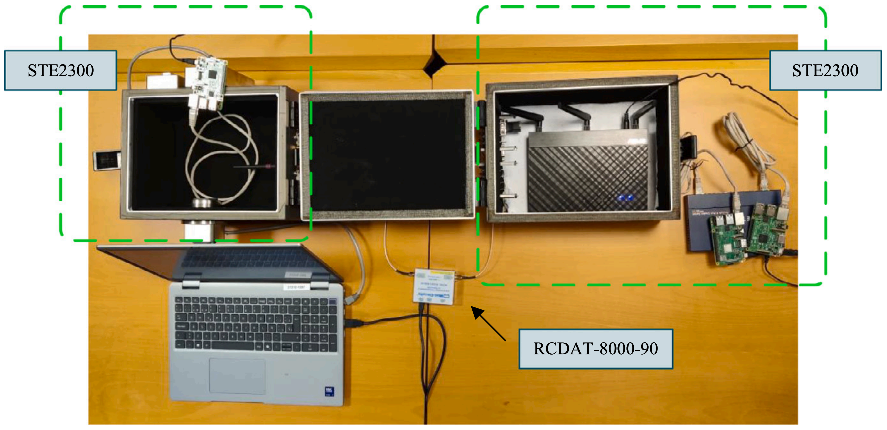


Fig. 9. Setup to emulate mobility.

connected via Ethernet to the previous one. Antennas inside each enclosure were connected by a cable with a programmable attenuator (RCDAT-8000-90), allowing us to precisely control wireless channel conditions.

A vehicular channel model proposed in [26] was used to simulate mobility. The attenuation was programmed to reflect the distance-dependent path loss and so emulate movement of the first node, away from the access point. To correctly apply the path loss model, the signal level in free space was first compared with that obtained using the boxes, to establish the implicit attenuation added by the setup (antennas, cables, etc.). In this sense, Received Signal Strength Indicator (RSSI) values (recorded with the `iwconfig` Linux command) obtained in free space and with the setup were −30 and −42 dBm respectively. This difference (−12 dB) was taken into account to tune the attenuator. In particular, the adopted model [26] mentions propagation losses of around 92 dB at a distance of 400 m, so that the dynamic range in the attenuator was configured between 0 and 80 dB.

Mobility at different speeds (1 m/s, 10 m/s, and 15 m/s) was emulated by adjusting the attenuation ramp and dwell time at each attenuation value, enabling the simulation of a node first moving away and then towards the access point, twice during the experiment. These speeds were chosen to represent a range of scenarios, from pedestrian and robotic movement to drones and vehicles, reflecting diverse mobility conditions found in manufacturing environments and vehicular networks.

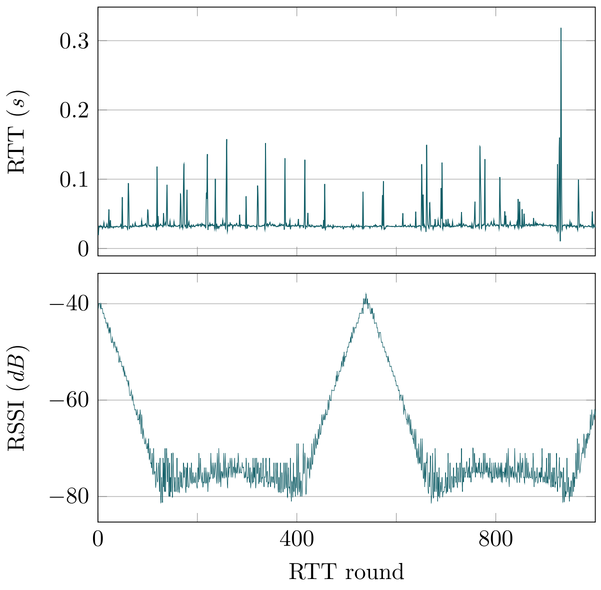


Fig. 10. Emulation example of measurements outcome.

Fig. 10 provides a sample of the RTT variation as a function of the RSSI throughout a mobility experiment. It shows the impact of distance and attenuation on communication performance. It is worth noting that the underlying Wi-Fi implementation (*802.11n*) adapts its modulation and coding scheme (MCS) to channel quality variations.

Fig. 11 illustrates the results for the three speeds. As observed, Eclipse Zenoh consistently outperforms MQTT, achieving much lower latencies and less variability across all configurations. These results underscore the robustness of Eclipse Zenoh in scenarios involving mobility.

4.1.3. *Scenario γ: Routed topology – Multiple networks*

In scenario γ with MQTT, a single host has been configured to act as both broker and client bridging between the two brokers. This setup ensures that the results obtained are appropriate to be compared against an Eclipse Zenoh Routed topology.

Fig. 12 shows that the performance (latencies) achieved with Eclipse Zenoh are significantly better (lower) compared to MQTT. It is noteworthy that, although Eclipse Zenoh exhibits slightly worse results than in scenario β (there is an additional hop in the communication), it still outperforms the results from scenario α. Conversely, MQTT exhibits latencies that are even higher than those observed in α. This highlights the potentially low performance of MQTT in distributed systems with multiple hops between producers and consumers. In contrast, Eclipse Zenoh provides an appropriate performance, and it also allows a more straightforward deployment in such scenarios.

4.1.4. *Scenario δ: P2P topology – Single network*

The aim of this last scenario was to characterize the latency that Eclipse Zenoh would cause in its most favorable conditions, which embrace a direct communication between two peers on the same network, with Wi-Fi coverage. As shown in Fig. 13, the delays obtained with the two reliable mechanisms are much lower than those observed in the rest of the analyzed scenarios.

To sum up, Table 3 presents the median RTT values for each scenario, to ease the comparison of the behaviors observed in all of them. As can be observed, Eclipse Zenoh steadily outperforms MQTT in all cases when reliability is not added (QoS 0 in MQTT and BE in Eclipse Zenoh). In addition, it can be seen that, in terms of delay, Eclipse Zenoh with Reliable configuration yields similar performance to MQTT without QoS.

4.2. *TCP vs. UDP Connections with Zenoh*

This section studies the interplay of Eclipse Zenoh's reliability and congestion control mechanisms (see Table 2) with the underlying transport protocols, specifically TCP and UDP. In this case, the scenario was

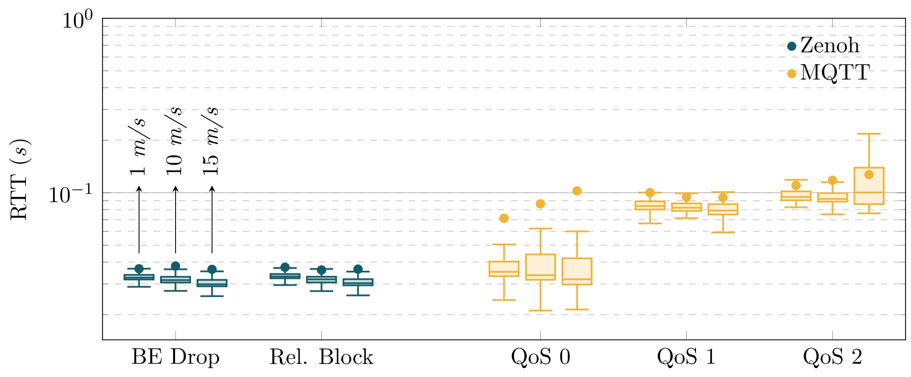


Fig. 11. RTT obtained in scenario β with motion and no delay to the cloud. For each configuration results with motion speed of 1, 10 and 15 m/s are obtained.

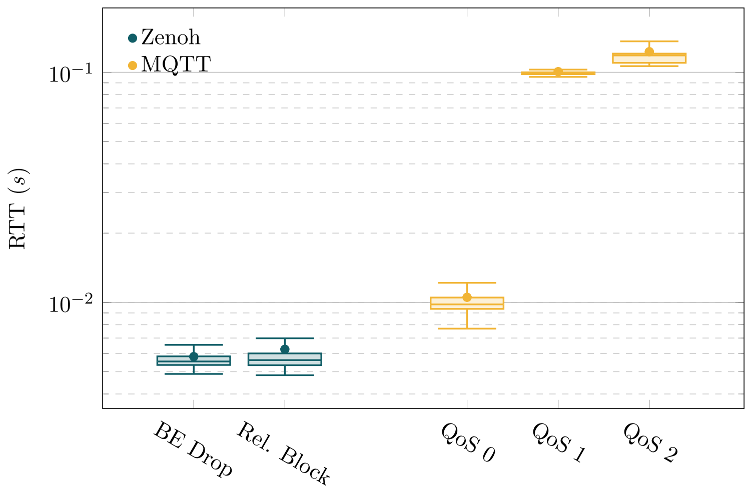


Fig. 12. RTT obtained in scenario γ.

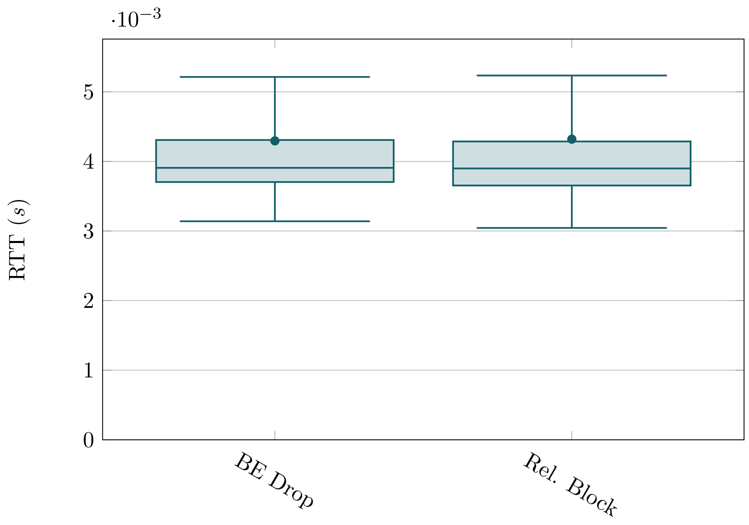


Fig. 13. RTT obtained in scenario δ.

simplified to exclusively focus on Eclipse Zenoh's interaction with these mechanisms. As depicted in Fig. 14, two applications, operated in `peer` mode on separate hosts, are connected via Ethernet to avoid channel-induced losses. Over this setup packet losses and latencies are analyzed for different packet sizes.

The tests consisted on the first peer publishing a 5 MB file, segmented into: (1) 5000 packets of 1000 B, (2) 10 000 packets of 500 B, and (3) 50 000 packets of 100 B. No waiting time was, deliberately, applied between transmissions to induce congestion. The second peer published a 2 B ACK message to prevent affecting buffer congestion. This process is illustrated in Fig. 15.

Configuration files were adjusted for peers to connect and listen on specific endpoints, switching between TCP and UDP protocols. Multicast and gossip scouting were disabled, and P2P discovery was thus directly managed, minimizing additional scouting traffic. Timestamping was also disabled to prevent message dropping. Peers are

Table 3
Mean and median RTT (ms) in all scenarios.

Scenario	Zenoh				MQTT					
	Best effort		Reliable		QoS 0		QoS 1		QoS 2	
	Mean	Median	Mean	Median	Mean	Median	Mean	Median	Mean	Median
α	7.18	6.81	8.38	7.03	8.08	7.58	8.87	8.47	127.07	119.73
β	5.15	4.68	4.96	4.60	5.25	4.78	6.69	5.16	117.72	109.66
β (a)	413.01	409.00	412.96	409.08	416.62	409.66	802.55	802.57	2102.80	2099.00
β (1 m/s)	36.66	32.60	37.36	33.13	71.29	35.20	100.18	83.89	110.10	94.59
β (10 m/s)	37.98	31.56	36.04	31.86	86.38	33.66	94.35	82.01	117.83	92.19
β (15 m/s)	36.36	29.86	36.46	30.31	102.45	31.91	93.80	79.01	127.08	100.31
γ	5.81	5.54	6.26	5.61	10.53	9.80	100.64	99.00	122.90	118.39
δ	4.30	3.91	4.32	3.90	–					

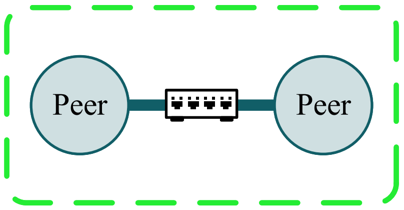


Fig. 14. Eclipse Zenoh's reliability and congestion control mechanisms test setup: two applications running in `peer` mode on separate hosts connected via wired Ethernet.

configured to operate in peer-to-peer mode, simplifying routing and reducing overhead. The transmission settings were selected to optimize data throughput, with a maximum batch size of 65 535 B. Reception settings included a 65 535-byte buffer, and supported message sizes up to 1 GiB, ensuring efficient handling of large data loads.

Tests used default buffer settings on Raspberry Pi devices. That is, with maximum and default lengths of 180 224 B for both receive and transmit buffers in network sockets.

A total of 100 samples were collected for each of the four configurations analyzed: (1) `BEST_EFFORT()` with `DROP()`, (2) `BEST_EFFORT()` with `BLOCK()`, (3) `RELIABLE()` with `DROP()`, and (4) `RELIABLE()` with `BLOCK()`. The impact of each mechanism on packet loss and RTT, due to network congestion, was studied.

In this case, the box plots in Figs. 16 and 17 display, with circles, outlier values that are below or above the 1 and 99 percentiles, respectively.

Based on the results shown in Fig. 16, we can conclude that the use of UDP as the transport layer leads to a significant number of packet losses for all configurations, with minor differences between them. As the packet count gets higher, increasing network congestion, the loss rate becomes significantly more relevant, particularly with configuration (1).

While Eclipse Zenoh's congestion control mechanisms demonstrate limited effectiveness with UDP, the results obtained with TCP evince that there were no packet losses when employing the `BLOCK()` option to halt transmissions. Notably, as discussed in [27], the reliability requirement at the receiver may not be consistently met if the producer uses `DROP()` for the congestion control. This is clearly seen by the packet losses that occur, even with TCP, which is designed with flow and congestion control: Eclipse Zenoh discards packets before reaching the transport layer, which does not cause any loss control mechanism, nor packet retransmissions.

Finally, Fig. 17 illustrates the distribution of the RTT for each packet. As expected, UDP transmissions exhibit considerably lower latency compared to TCP, reflecting a trade-off between packet loss and delay. While RTT values are similar when the connection entails a smaller number of packets, significant differences can be seen when having a larger number of packets. In extreme cases, Eclipse Zenoh over TCP leads to RTT times close to 15 s for individual packets.

5. Related work

Given its novelty, related works focused on Eclipse Zenoh are quite scarce to date. Nevertheless, there exist some works, mainly conducted by Eclipse Zenoh's developers and maintainers, such as ZettaScale [4, 25], and ADLINK [28] that provide the foundation of this work.

The wide array of topologies offered by Eclipse Zenoh enables comparisons in P2P communications with distributed protocols like Cyclone DDS or in brokered communications with centralized ones like MQTT or Kafka. In this context, Liang et al. provide in [25] a comparative analysis of these protocols, showing rather favorable results for Eclipse Zenoh in terms of latency and throughput. Regarding latency, both single and multi-host scenarios were encompassed. In the first case, DDS exhibited lower performance compared to Eclipse Zenoh when using the P2P configuration. The authors concluded that the observed behavior was due to DDS utilizing UDP multicast mechanisms. Since Eclipse Zenoh was not yet implemented over UDP, Eclipse Zenoh's implementation for constrained devices, i.e. Zenoh-Pico,[11] implemented over UDP, was also tested, providing the lowest latencies. Additionally, Zenoh brokered obtained better results in comparison to MQTT. In the multi-host scenario, comprising three hosts (publisher, subscriber, and router/broker) and using a 100 Gb Ethernet, Eclipse Zenoh consistently outperformed MQTT and Cyclone DDS, demonstrating significantly lower values across all scenarios.

Corsaro et al. refer to the aforementioned results in [4], where they introduce the principles and key features of Eclipse Zenoh, such as resources, entities and primitives, and showcase the supported topologies. The paper also reflects the wire efficiency Eclipse Zenoh offers compared to DDS and MQTT due to its significantly lower minimal overhead.

Shih et al. [28] focus their study on evaluating Eclipse Zenoh's capability over scalable, cross-network, and real-time systems. They demonstrate that Eclipse Zenoh and Cyclone DDS exhibit different behaviors depending on the payload size. The results evince that Eclipse Zenoh shows better performance with smaller payloads, but it is affected as the payload size increases. The analysis was conducted with the peers on a single machine simulating the network conditions that edge devices might encounter on a 802.11g network [29], using the Linux kernel to throttle traffic.

More recently, some other researches have been conducted in order to explore the potential that Eclipse Zenoh could have in different use cases. Zhang et al. [30] study the performance of Cyclone DDS, MQTT, and Eclipse Zenoh in distributed ROS 2 systems. Three network environments are set: an Ethernet and a Wi-Fi local network, simulating Edge-to-Edge communication, and a 4G setup with a ZeroTier[12] virtualized network to emulate Edge-to-Cloud communication. They achieve good results for Eclipse Zenoh when operating under 4G and Wi-Fi conditions, in comparison to MQTT and DDS. However, under Ethernet,

[11] https://github.com/eclipse-zenoh/zenoh-pico (accessed on 24 September 2024).

[12] https://www.zerotier.com/ (accessed on 30 September 2024).

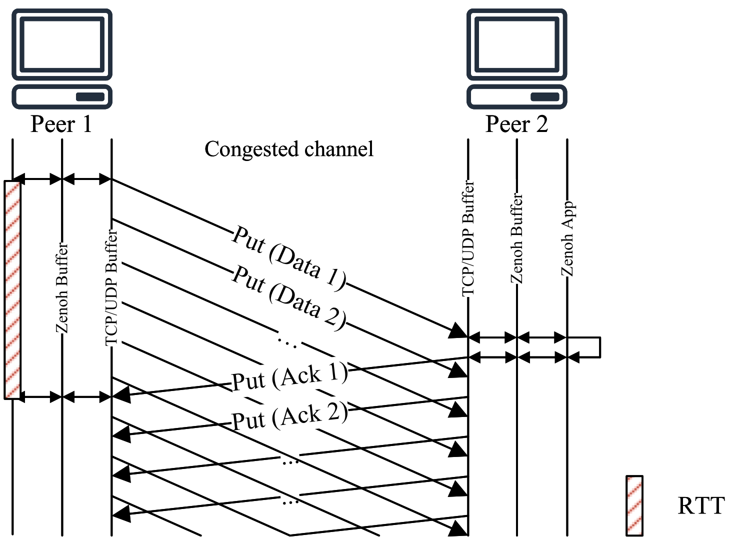


Fig. 15. Sequence diagram illustrating RTT measurement for analyzing reliability and congestion control mechanisms in Eclipse Zenoh.

Cyclone DDS exhibits minimal latency and maximum throughput. The authors, as in [4,25], conclude that it is due to the unfair comparison with DDS, since it works over UDP.

Furthermore, López Escobar et al. [31] have observed significant potential in utilizing Eclipse Zenoh to implement lightweight services within the Cloud-to-Edge infrastructure, fostering various use cases such as emergency rescue and mobile e-health monitoring applications. Their focus is to assess the performance of centralized and decentralized communication approaches, comparing MQTT as a centralized strategy and Eclipse Zenoh P2P as a decentralized one. Three architecture testbeds are defined to depict various communication scenarios in the Cloud-to-Edge Continuum: Mist, Fog-Mist, and Cloud-Fog-Mist. These involve distributed computing across different layers depending on processing and networking requirements. The evaluation is conducted over Wi-Fi (802.11ac) and 5G (Stand Alone (SA) core with an indoor n78-band gNodeB (3.3–3.8 GHz)) access infrastructure. Both MQTT and Eclipse Zenoh communicate over TCP, with MQTT set to QoS level 0 and Eclipse Zenoh to *Best-Effort* mode.

Recently, Teixeira et al. [32] explored the use of Eclipse Zenoh as an alternative to the Message Passing Interface (MPI) for distributed learning tasks in the context of federated learning (FL), using realistic 6G simulated environments. They show that Eclipse Zenoh exhibits slightly higher execution time, but their work highlights its reduced communication overhead and increased flexibility, making it a suitable solution for resource-constrained and dynamically connected devices in FL applications.

Especially noteworthy is the different scope given in [33], where Zenoh-Flow is introduced. This platform is designed to streamline the development of applications for autonomous robots, vehicles, and other use cases requiring data transfer between the cloud and devices. It provides tools and abstractions to enhance performance and efficiency in control applications while also supporting higher-level data flows for artificial intelligence and machine learning. Zenoh-Flow is developed based on the positive results obtained in [34]. Additionally, other researchers have explored this platform, as Gramaglia et al. [35], who exploit it to implement heterogeneous Network Intelligence (NI) algorithms.

This work complements existing research by further exploring the comparative performance of MQTT and Eclipse Zenoh across network topologies not yet addressed. It is worth highlighting our focus on routed communication, where two routers can efficiently forward data directly between them, eliminating the need for a client to serve as a bridge, an aspect not covered in previous studies like [25,30] or [31]. By analyzing its performance in multi-router environments, we extend the knowledge about Eclipse Zenoh by providing insights into network scalability and efficiency, which is especially relevant for IIoT settings where multi-hop routing is common. Besides, the impact of a node's mobility relative to a Wi-Fi access point, tested at varying speeds and its impact on latency, is another novel aspect not addressed in earlier research.

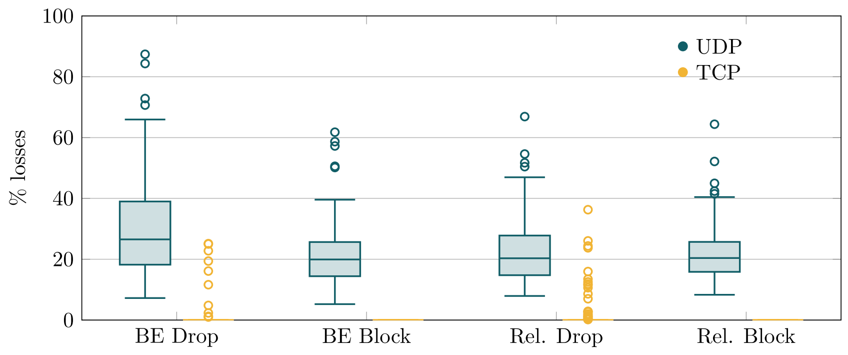
Additionally, this work provides an in-depth analysis of the congestion control and reliability mechanisms described in [4], providing a deeper analysis of their behavior under different configurations. Specifically, it examines Eclipse Zenoh's operation over both TCP and UDP transport protocols, focusing on scenarios involving packet loss in a fully wired connection. This has been largely overlooked in previous studies. We consider that the evaluation of Zenoh's `RELIABLE` and `BLOCK` configurations, in contrast to the `BEST EFFORT` and `DROP` settings, is particularly valuable in revealing how these mechanisms affect packet delivery and RTT under high-traffic conditions. By simulating real-world scenarios involving the transmission of large volumes of segmented data (e.g., 5 MB), this analysis offers new insights into the protocol's reliability and congestion management strategies. This understanding is essential for optimizing the balance between reliability and efficiency in industrial networks that handle large amounts of data.

Moreover, this paper provides a detailed communication-level analysis of Eclipse Zenoh, encompassing processes such as node discovery, port usage, handshake mechanisms, and data transmission/reception. These insights are relevant for understanding how the protocol operates at a fundamental level and complement the performance evaluation by detailing the protocol's inner mechanisms. Such a detailed analysis has not been previously explored in related works, making it a distinctive contribution to the field.

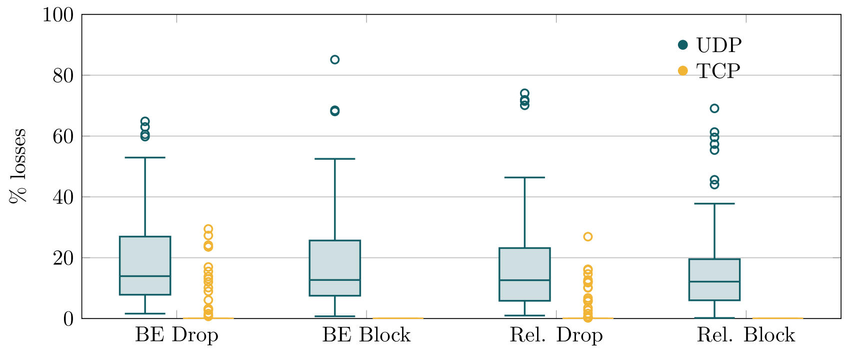
Table 4 summarizes the comparative features of this study and related works. While existing research provides important groundwork, this paper extends it by addressing critical gaps, including routed communication, mobility, in-depth analysis of reliability and congestion mechanisms, and detailed communication-level insights. These contributions not only cover existing gaps in the available literature, but also offer a more comprehensive understanding of Eclipse Zenoh's potential in real-world deployments.

6. Conclusion

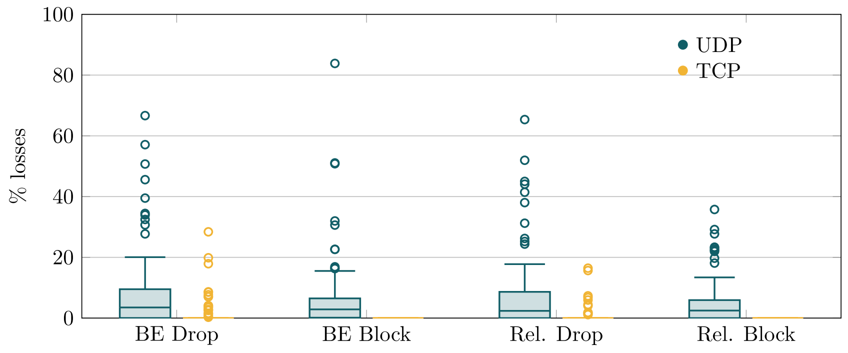
Given that Eclipse Zenoh is a relatively new protocol and does not yet have a formal specification or standard, the lack of detailed documentation poses a significant challenge. This work has analyzed



(a) 100 B packets.



(b) 500 B packets.



(c) 1000 B packets.

Fig. 16. Distribution of the percentage of losses for UDP and TCP using different configurations and for various packet sizes.

Table 4
Features covered by related works. The terms "BE" and "Rel." are used to indicate whether *Best Effort* and *Reliable* configurations, respectively, are analyzed.

Works	Rel., CC/QoS	Multiple hosts	Wireless connectivity	Zenoh topologies	Impact of mobility in latency	Zenoh over UDP
This work	BE/Rel., Block/Drop/QoS 0, 1, 2	✓	✓	P2P, Brokered, Routed	✓	✓
[4]	BE, Block/QoS 0	✓	✗	P2P, Brokered	✗	✗
[25]	BE, Block/QoS 0	✓	✗	P2P, Brokered	✗	✗
[28]	BE, Drop/Volatile	✗	Simulated	P2P	✗	✗
[30]	Unknown	✓	✓	P2P, Brokered	✗	✗
[31]	BE/QoS 0	✓	✓	P2P	✗	✗
[32]	–	✗	Simulated	–	✗	✗
[33–35]	–	–	–	–	✗	–

its network-level operation to better understand its capabilities and communication management. Additionally, a thorough evaluation of Eclipse Zenoh's performance compared to MQTT across various network topologies has been presented. The results show that Eclipse Zenoh delivers lower latency under specific conditions, particularly in distributed (*routed*) architectures, but also in centralized (*brokered*) scenarios. This makes Eclipse Zenoh particularly well-suited for robotics and automotive applications, where real-time data communication and flexible topologies are essential.

The study has also thoroughly examined Eclipse Zenoh's reliability and congestion control mechanisms by measuring packet loss and latency for UDP and TCP transmissions. These mechanisms are vital for ensuring communication integrity and stability, especially in industrial settings where continuous operation with minimal loss is required.

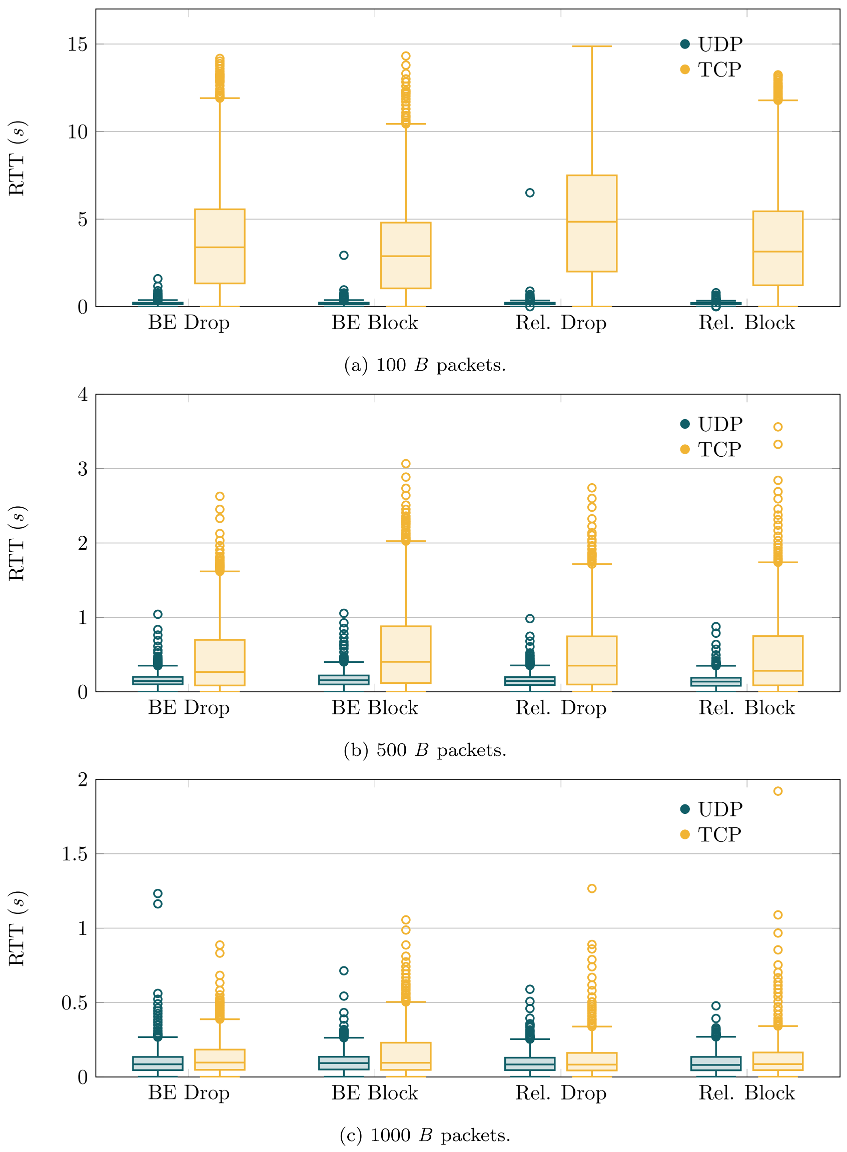


Fig. 17. Distribution of packet latency for UDP and TCP using different configurations and various packet sizes.

Eclipse Zenoh's versatile architecture, which spans multiple layers of the protocol stack and accommodates various communication topologies, positions it as a highly adaptable solution for the IIoT landscape. This adaptability suggests that industries could leverage it to enhance their communication infrastructures, supporting more advanced and interconnected industrial systems.

In our future work we will further explore Eclipse Zenoh's performance in larger and more complex network environments, addressing its integration with emerging technologies such as 5G or Non-Terrestrial Network (NTN) communications. Additionally, we will study the performance impact of the improvements introduced in the latest version of Eclipse Zenoh and compare its behavior over QUIC with that of TCP combined with TLS. Other research activities will include testing of Eclipse Zenoh and Zenoh-Pico with serial communication on constrained devices, as well as exploring the capabilities of Zenoh-Flow, its native data flow programming framework.

Moreover, we plan to extend our research by testing Eclipse Zenoh in real-world scenarios involving mobile robots, particularly in robotics and automotive applications. We will thus focus on dynamic environments, where actual robot mobility might affect communication and routing, such as robotic scouting or autonomous vehicle networking. By using physical robots, we will assess Eclipse Zenoh's capability to handle challenges that might appear in real-world situations, like mobility-induced disconnections, re-connections, and routing adjustments.

Finally, we will broaden our analysis to include other application-layer protocols such as DDS and CoAP, to compare their performance with Eclipse Zenoh in terms of latency, scalability, and efficiency, especially in constrained environments with strict delay requirements. This will provide further insights into the suitability of Eclipse Zenoh for various IoT and industrial applications.

CRediT authorship contribution statement

Miguel Barón: Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Investigation. **Luis Diez:** Writing – review & editing, Validation, Supervision, Methodology, Investigation, Formal analysis, Conceptualization. **Mihail Zverev:** Validation, Methodology, Formal analysis. **José R. Juárez:** Supervision, Methodology, Funding acquisition, Conceptualization. **Ramón Agüero:** Writing – review & editing, Investigation, Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work has been partially supported by SONETO project from ELKARTEK research programme of the Basque Government, Spain (Ref. KK-2023/00038). It has been also supported by the Spanish Ministry of Economic Affairs and Digital Transformation with the project SITED (PID2021-125725OB-I00), and by the regional Cantabria Government, Fondo Europeo de Desarrollo Regional (FEDER): "Grants for research projects with high industrial potential of technological agents of excellence for industrial competitiveness (TCNIC)" program (2023/TCN/002).

Data availability

Data will be made available on request.

References

[1] M. Soori, F. Karimi Ghaleh Jough, R. Dastres, B. Arezoo, Connectivity, automation, and data exchange in advanced manufacturing of industry 4.0, 2024, http://dx.doi.org/10.13140/RG.2.2.18189.55522.

[2] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A.Y. Ng, et al., ROS: an open-source robot operating system, in: ICRA Workshop on Open Source Software, Vol. 3, Kobe, Japan, 2009, p. 5.

[3] S. Chitta, E. Marder-Eppstein, W. Meeussen, V. Pradeep, A.R. Tsouroukdissian, J. Bohren, D. Coleman, B. Magyar, G. Raiola, M. Lüdtke, E.F. Perdomo, Ros_control: A generic and simple control framework for ROS, J. Open Source Softw. 2 (20) (2017) 456, http://dx.doi.org/10.21105/joss.00456.

[4] A. Corsaro, L. Cominardi, O. Hecart, G. Baldoni, J. Avital, J. Loudet, C. Guimares, M. Ilyin, D. Bannov, Zenoh: Unifying communication, storage and computation from the cloud to the microcontroller, in: 2023 26th Euromicro Conference on Digital System Design, DSD, IEEE Computer Society, Los Alamitos, CA, USA, 2023, pp. 422–428, http://dx.doi.org/10.1109/DSD60849.2023.00065, URL https://doi.ieeecomputersociety.org/10.1109/DSD60849.2023.00065.

[5] ROS 2 Core Team, ROS 2 Middleware Interface (RMW) Alternate, Study Item, Open Source Robotics Alliance (OSRA), 2023.

[6] Eclipse Foundation, Zenoh, 2022, https://zenoh.io/. (Accessed 01 October 2024).

[7] ZettaScale, V2X communication in the automotive industry with Zenoh, 2024, URL https://www.zettascale.tech/news/v2x-communication-in-the-automotive-industry-with-zenoh/. Blog. (Accessed 01 October 2024).

[8] Focus Group on AI for autonomous and assisted driving (FG-AI4AD), ITU-T Automated Driving Safety Data Protocol – Specification, Tech. Rep., TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU, 2022.

[9] W. Eddy, Transmission control protocol (TCP), 2022, http://dx.doi.org/10.17487/RFC9293, URL https://www.rfc-editor.org/info/rfc9293. RFC 9293.

[10] User datagram protocol, 1980, http://dx.doi.org/10.17487/RFC0768, URL https://www.rfc-editor.org/info/rfc768. RFC 768.

[11] Object Management Group (OMG), Data distribution service (DDS), 2015, https://www.omg.org/spec/DDS/. Version 1.4.

[12] Z. Shelby, K. Hartke, C. Bormann, The constrained application protocol (CoAP), 2014, http://dx.doi.org/10.17487/RFC7252, URL https://www.rfc-editor.org/info/rfc7252. RFC 7252.

[13] Apache Software Foundation, Apache kafka, 2024, https://kafka.apache.org/. (Accessed 01 October 2024).

[14] Eclipse Foundation, 2023 IoT & edge developer survey report, 2023, URL https://outreach.eclipse.foundation/iot-edge-developer-survey-2023. (Accessed 01 October 2024).

[15] MQTT.org, MQTT: The standard for IoT messaging, 2022, https://mqtt.org/. (Accessed 01 October 2024).

[16] A. Banks, R. Gupta, MQTT Version 3.1.1, Tech. Rep., OASIS Standard, 2014, URL http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html. Latest version: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html.

[17] A. Banks, E. Briggs, K. Borgendale, R. Gupta, MQTT Version 5.0, Tech. Rep., OASIS Standard, 2019, URL https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html. Latest version: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

[18] E. Rescorla, The transport layer security (TLS) protocol version 1.3, 2018, http://dx.doi.org/10.17487/RFC8446, URL https://www.rfc-editor.org/info/rfc8446. RFC 8446.

[19] F. Fernández, M. Zverev, P. Garrido, J.R. Juárez, J. Bilbao, R. Agüero, And QUIC meets IoT: performance assessment of MQTT over QUIC, in: 2020 16th International Conference on Wireless and Mobile Computing, Networking and Communications, WiMob, 2020, pp. 1–6, http://dx.doi.org/10.1109/WiMob50308.2020.9253384.

[20] J. Iyengar, M. Thomson, QUIC: A UDP-based multiplexed and secure transport, 2021, http://dx.doi.org/10.17487/RFC9000, URL https://www.rfc-editor.org/info/rfc9000. RFC 9000.

[21] C. Guimarães, G. Baldoni, There is land besides IP: How to cross it with Zenoh, 2022, https://zenoh.io/blog/2022-08-12-zenoh-serial/. (Accessed 01 October 2024).

[22] O. Salman, I. Elhajj, A. Kayssi, A. Chehab, An architecture for the internet of things with decentralized data and centralized control, in: 2015 IEEE/ACS 12th International Conference of Computer Systems and Applications, AICCSA, 2015, pp. 1–8, http://dx.doi.org/10.1109/AICCSA.2015.7507265.

[23] Raymond Andrè Hagen, The future of IT: A balanced perspective on centralized and decentralized systems, 2023, URL https://www.linkedin.com/pulse/future-balanced-perspective-centralized-decentralized-hagen/. (Accessed 01 October 2024).

[24] Y. Liu, M. Kashef, K. Lee, L. Benmohamed, R. Candell, Wireless network design for emerging IIoT applications: Reference framework and use cases, Proc. IEEE 107 (2019) http://dx.doi.org/10.1109/JPROC.2019.2905423.

[25] W.-Y. Liang, Y. Yuan, H.-J. Lin, A performance study on the throughput and latency of Zenoh, MQTT, Kafka, and DDS, 2023, arXiv:2303.09419.

[26] D. Wang, R.R. Sattiraju, H.D. Schotten, Performances of C-V2X communication on highway under varying channel propagation models, in: 2018 10th International Conference on Communications, Circuits and Systems, ICCCAS, 2018, pp. 305–309, http://dx.doi.org/10.1109/ICCCAS.2018.8768912.

[27] ZettaScale Zenoh team, Zenoh API reference, 2023, https://zenoh-python.readthedocs.io/en/0.10.0-rc/. (Accessed 01 October 2024).

[28] C.-S. Shih, H.-J. Lin, Y. Yuan, Y.-H. Kuo, W.-Y. Liang, Scalable and bounded-time decisions on edge device network using eclipse Zenoh, in: 2022 IEEE 28th International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA, 2022, pp. 170–179, http://dx.doi.org/10.1109/RTCSA55878.2022.00024.

[29] J. Jun, P. Peddabachagari, M. Sichitiu, Theoretical maximum throughput of IEEE 802.11 and its applications, in: Proceedings of the Second IEEE International Symposium on Network Computing and Applications, NCA '03, IEEE Computer Society, USA, 2003, p. 249.

[30] J. Zhang, X. Yu, S. Ha, J. Peña Queralta, T. Westerlund, Comparison of middlewares in edge-to-edge and edge-to-cloud communication for distributed ROS 2 systems, J. Intell. Robot. Syst. 110 (4) (2024) 162, http://dx.doi.org/10.1007/s10846-024-02187-z.

[31] J.J. López Escobar, R.P. Díaz-Redondo, F. Gil-Castiñeira, Unleashing the power of decentralized serverless IoT dataflow architecture for the cloud-to-edge continuum: a performance comparison, Ann. Telecommun. 79 (3) (2024) 135–148, http://dx.doi.org/10.1007/s12243-023-01009-x.

[32] R. Teixeira, G. Baldoni, M. Antunes, D. Gomes, R.L. Aguiar, Leveraging decentralized communication for privacy-preserving federated learning in 6G networks, SSRN Electron. J. (2024) http://dx.doi.org/10.2139/ssrn.4817067, URL https://ssrn.com/abstract=4817067.

[33] G. Baldoni, J. Loudet, L. Cominardi, A. Corsaro, Y. He, Zenoh-based dataflow framework for autonomous vehicles, in: 2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion, QRS-C, 2021, pp. 555–560, http://dx.doi.org/10.1109/QRS-C55045.2021.00085.

[34] G. Baldoni, J. Loudet, L. Cominardi, A. Corsaro, Y. He, Facilitating distributed data-flow programming with eclipse Zenoh: the ERDOS case, in: Proceedings of the 1st Workshop on Serverless Mobile Networking for 6G Communications, MobileServerless '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 13–18, http://dx.doi.org/10.1145/3469263.3469858.

[35] M. Gramaglia, M. Camelo, L. Fuentes, J. Ballesteros, G. Baldoni, L. Cominardi, A. Garcia-Saavedra, M. Fiore, Network intelligence for virtualized RAN orchestration: The DAEMON approach, in: 2022 Joint European Conference on Networks and Communications & 6G Summit, EuCNC/6G Summit, 2022, pp. 482–487, http://dx.doi.org/10.1109/EuCNC/6GSummit54941.2022.9815816.

Miguel Barón received his Bachelor's degree in Telecommunication Technologies Engineering from the University of Cantabria in 2022 and completed his M.Sc. at the same institution in 2024. He is now pursuing a joint Ph.D. degree through a collaboration between the University of Cantabria and the Ikerlan Technology Research Centre. His research interests include performance analysis in Industrial Internet of Things (IIoT) and Tactile Internet environments, with a particular focus on the MQTT and Eclipse Zenoh protocols.

Luis Diez received his M.Sc. and Ph.D. from University of Cantabria in 2013 and 2018 respectively. He is currently Associate Professor at the Communications Engineering Department in that University. He has been involved in different international and industrial research projects. His research focuses on future network architectures, resource management in wireless heterogeneous networks, and IoT solutions and services. He has published more than 70 scientific and technical papers in those areas, and he has served as TPC member and reviewer in a number of international conferences and journals. As for teaching, Dr. Diez has supervised 40 BSc and MSc Thesis, and he teaches in courses related to cellular networks, network dimensioning and service management.

Mihail Zverev was born in Dzhambul, Kazakhstan, in 1991. He received the degree in telecommunications engineering from University of Cantabria, Santander, Spain, in 2014, and the M.Sc. degree in project management from La Salle, University of Ramónn Llull, Barcelona, in 2017. In 2023 he defended the joint Ph.D. degree with Ikerlan Technology Research Centre in collaboration with Communications Engineering department at the University of Cantabria. From 2015 to 2018, he was a Writing Systems Engineer, hired through different companies to develop large format industrial printers. His area of work is mainly centered around efficient IoT communications.

José Ramón Juárez, Telecommunications Engineer in 2004 by the University of the Basque Country, obtained a PhD in the program “Technologies for Distributed Information Management” in 2011 by the Public University of Navarra. During the development of the PhD, he has been Assistant Professor in the Department of Computer Languages and Systems and member of the Distributed Systems Group at the Public University of Navarra, and conducted research stays at the Universidade do Minho (Braga, Portugal) and at the Università della Svizzera italiana (Lugano, Switzerland). He is currently principal investigator of the Smart Connectivity research team at the IKERLAN technology center. His main interests range from high availability systems, dynamic and adaptive distributed systems, machine learning, wireless communications, 5G, lightweight IoT protocols and fog-edge-cloud architectures.

Ramón Agüero received his MSc in Telecommunications Engineering (1st class honors) from the University of Cantabria in 2001 and the PhD (Hons) in 2008. He is currently a Professor at the Communications Engineering Department in that university. His research focuses on future network architectures, especially regarding the (wireless) access part of the network and its management. He has published more than 240 scientific papers in such areas and he is a regular TPC member and reviewer on various related conferences and journals. Ramon Aguero serves in the Editorial Board of IEEE Communication Letters (Senior Editor since 2019), IEEE Open Access Journal of the Communications Society, Wireless Networks (Springer), IEEE Systems Journal. Dr. Aguero has supervised 7 PhDs and more than 70 BSc and MSc thesis. He is the main instructor in courses dealing with Networks, and Traffic Modeling, both at BSc and MSc levels. From 2016 to 2024 he was the Head of the IT Area (deputy CIO) at the University of Cantabria.