



Article

Real-Time Kubernetes-Based Front-End Processor for Smart Grid

Taehun Kim ¹, Hojung Kim ^{2,*} , SeungKeun Cho ², YongSeong Kim ² , ByungKwen Song ² and Jincheol Kim ¹

¹ Security Business Department, KEPCO Knowledge Data and Network, Naju 58322, Republic of Korea; thkim_5117@kdn.com (T.K.), shine_1991@kdn.com (J.K.)

² Department of Electronics and Computer Engineering, Seokyeong University, Seoul 02713, Republic of Korea; wiw4477@skuniv.ac.kr (S.C.); wiz@skuniv.ac.kr (Y.K.); bksong@skuniv.ac.kr (B.S.)

* Correspondence: hotteok@skuniv.ac.kr; Tel.: +82-10-9559-3034

Abstract: In Supervisory Control and Data Acquisition (SCADA) systems, central to industrial automation and control systems, the Front-end Processor (FEP) facilitates seamless communication between field control devices and central management systems. As the Industrial Internet of Things (IIoT) and Industry 4.0 centered on the smart factory paradigm gain traction, conventional FEPs are increasingly showing limitations in various aspects. To address these issues, Data Distribution Service, a real-time communication middleware, and Kubernetes, a container orchestration platform, have garnered attention. However, the effective integration of conventional SCADA protocols, such as DNP3.0, IEC 61850, and Modbus with DDS, remains a key challenge. Therefore, this article proposes a Kubernetes-based real-time FEP for the modernization of SCADA systems. The proposed FEP ensures interoperability through an efficient translation mechanism between traditional SCADA protocols—DNP3.0, IEC 61850, and Modbus—and the Data Distribution Service protocol. In addition, the performance evaluation shows that the FEP achieves high throughput and sub-millisecond latency, confirming its suitability for real-time industrial control applications. This approach overcomes the limitations of conventional FEPs and enables the realization of more flexible and scalable industrial control systems. However, further research is needed to validate the system under large-scale deployment scenarios and enhance security capabilities. Future work will focus on performance evaluation in realistic conditions and the integration of quantum-resistant security mechanisms to strengthen resilience in critical infrastructure environments.



Received: 30 April 2025

Revised: 4 June 2025

Accepted: 6 June 2025

Published: 10 June 2025

Citation: Kim, T.; Kim, H.; Cho, S.; Kim, Y.; Song, B.; Kim, J. Real-Time Kubernetes-Based Front-End Processor for Smart Grid. *Electronics* **2025**, *14*, 2377. <https://doi.org/10.3390/electronics14122377>

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: common information model; data distribution service; DNP3.0; front-end processor; IEC 61850; Kubernetes; Modbus; protocol conversion; smart grid; supervisory control and data acquisition

1. Introduction

SCADA systems, a core element of industrial automation and control systems, have been pivotal in industrial settings over the past several decades. SCADA systems provide infrastructure for monitoring and controlling remote equipment and are widely used in various industries, such as power, nuclear energy, and oil [1]. Conventional SCADA systems communicate with remote facilities using industrial protocols, such as DNP3.0, IEC 61850, and Modbus, which have been widely adopted in industrial fields due to their proven stability and reliability over the decades [2–4].

In particular, with the rapid evolution of industrial environments centered around the IIoT and smart factory paradigms, existing SCADA systems are becoming increasingly complex and are required to operate in larger and more distributed environments. Industry

4.0 refers to the digital transformation in manufacturing and industrial automation, characterized by the integration of technologies such as the Internet of Things (IoT), artificial intelligence (AI), cloud computing, and edge computing to enable intelligent interconnection and optimization between production equipment and information systems [5,6]. In line with this trend, SCADA systems must evolve beyond their traditional roles of monitoring and control to support the real-time processing of large volumes of data and enable flexible interoperability with a wide range of heterogeneous devices and systems.

To address the increasing technological complexity and evolution, the Smart Grid Architecture Model (SGAM), proposed by the European standardization bodies for the electric power industry (CEN-CENELEC-ETSI), has gained considerable attention. SGAM is a three-dimensional framework that defines how various components of a smart grid system should be integrated and interoperable. It structures the system across five interoperability layers—business, function, information, communication, and component—as well as along operational zones and energy domains [7]. This model provides a systematic representation of how industrial systems, including SCADA, should be designed and scaled within future-oriented smart grid environments.

However, conventional SCADA systems show limitations in scalability, flexibility, real-time responsiveness, and fault recovery. Because conventional SCADA systems depend on fixed infrastructure, system scaling is difficult, and integrating diverse data sources and applications is complex and inefficient. Moreover, these systems are difficult to ensure rapid recovery in the event of system failures and high availability, imposing constraints on stable operations in industrial settings [8]. To overcome these limitations, the adoption of real-time middleware and cloud-native technologies has become necessary.

Kubernetes is an open-source container orchestration platform that enables automatic deployment, management, and scaling of containerized applications. By offering the flexibility to run applications consistently across diverse environments through visualization, Kubernetes can be considered an ideal platform for modernizing SCADA systems [9]. Kubernetes's automatic scaling mechanism enables horizontal scaling of SCADA systems. Leveraging containerized microservice architecture enables the modularization and flexible updates of SCADA systems. Moreover, declarative configuration and self-healing features in Kubernetes enhance the high availability and reliability of SCADA systems.

Data Distribution Service (DDS) is a data-centric communication middleware that supports real-time data processing. By enabling fine-tuned control over various parameters, such as reliability, priority, durability, and temporal constraints, through Quality of Service (QoS) policies, DDS ensures real-time data processing required in mission-critical industrial environments. In addition, DDS enables efficient data exchange between system components using a data-centric approach, allowing effective processing of massive data generated by various sensors and measuring devices [10]. Therefore, DDS-based data distribution provides a crucial technological basis for industrial automation systems.

However, the integration between conventional SCADA systems and DDS-based systems remains a key challenge. As the two systems utilize fundamentally different communication paradigms and protocols, ensuring interoperability is inherently difficult. Conventional SCADA protocols, such as DNP3.0, IEC 61850, and Modbus, primarily utilize connection-oriented communication methods based on Master–Slave or Client–Server models, whereas the DDS protocol utilizes a data-centric Publish–Subscribe communication method. Moreover, SCADA protocols primarily utilize polling-based data acquisition mechanisms, whereas DDS adopts an event-based data dissemination method. To ensure interoperability among various SCADA protocols and device vendors, data exchange must occur through a common structure. This implies that, beyond simple protocol translation, semantic-level data mapping and translation are required [11].

Therefore, this article proposes a FEP for efficient translation between conventional SCADA protocols and DDS protocols within a Kubernetes platform. The main objectives of this article are as follows: (1) to design and implement a FEP capable of translating data between conventional SCADA protocols (DNP3.0, IEC 61850, and Modbus) and the DDS protocol; (2) to ensure real-time data processing and interoperability through mechanisms embedded in the FEP; (3) to utilize Kubernetes features, such as horizontal scaling and self-healing, to provide high availability and scalability for the overall system; (4) to enable seamless integration between legacy SCADA and DDS-based systems, facilitating modernization toward a cloud-native architecture; and (5) to conduct performance evaluations of the proposed system including FEP and multiple DDS implementations in order to validate the effectiveness of the proposed system and provide performance indicators for its deployment.

The remainder of this article is structured as follows. Section 2 introduces the background knowledge necessary for understanding this article, as well as the related works relevant to this article. Section 3 details the structure and implementation of the proposed system. Section 4 presents the implementation results and the performance evaluation of the proposed system. Section 5 discusses the implications and limitations of the proposed approach. Finally, Section 6 summarizes the research findings and conclusions.

To better understand the motivations and context of this work, it is essential to first review the underlying techniques and related research in the domain.

2. Background

2.1. Kubernetes

Kubernetes, developed by Google based on its extensive container management experience, is an open-source platform for container orchestration. Released in 2014, it has evolved into a leading project under the Cloud Native Computing Foundation (CNCF), becoming the de facto standard for containerized infrastructure [12,13].

One of the key features of Kubernetes is its declarative system configuration, wherein the desired state is specified by the administrator, and the system automatically adjusts the current state to match it. Contrary to the imperative model, this approach focuses on the final system state and enables Kubernetes controllers to continuously monitor and adjust the current state, providing a self-healing mechanism [14]. YAML (YAML Ain't Markup Language, Yet Another Markup Language) or JavaScript Object Notation (JSON) configuration files used in this process facilitate not only source code version management but also version management of the platform. Combined with modern infrastructure management methodologies, such as DevOps, GitOps, and Continuous Configuration Automation, this becomes a fundamental component in realizing Infrastructure as Code (IaC) [15].

Another key feature of Kubernetes is its scalability and portability. It supports automatic scaling in response to workload increases and allows consistent application deployment and management across various computing environments, including on-premise, hybrid, and public clouds. This enables a multi-cloud strategy that mitigates vendor lock-in and supports concurrent use of various cloud platforms [16].

Figure 1 illustrates the Kubernetes architecture, which explains the roles and interactions between the control plane and worker nodes. Emphasizing how these components support cluster orchestration and security enforcement. It can also be helpful to understand how the proposed high-availability architecture is configured, a topic that will be discussed in detail later in the article. Kubernetes can operate clusters that integrate multiple physical or logical nodes. These nodes are categorized based on their roles into the control plane or master node, responsible for centralized control of the cluster, and worker nodes, which

execute application workloads. The control plane (master node) includes key components like: etcd (stores configuration data), Scheduler (assigns workloads to nodes), API Server (central communication point), Controller Manager (handles system controllers), and Cloud Controller Manager (integrates with cloud providers). The worker node includes kubelet (node agent), kube-proxy (network proxy), Container Runtime (executes containers), and Pods (groups of containers running application workloads).

In particular, the API Server of the master node serves as the core component for cluster management, offering a Representational State Transfer (REST) Application Programming Interface (API) for accessing and modifying cluster resources. It also communicates with worker nodes to monitor the state of running workloads and transmit information, such as cluster scheduling. Therefore, a failure in a worker node can directly result in application workload disruption, while a failure in the master node can result in the loss of entire cluster management capabilities. To address this, Kubernetes offers a high-availability architecture through multi-node configurations.

From a security perspective, Kubernetes enhances containerized application security through diverse security mechanisms, such as Role-based Access Control (RBAC), network policies, and secret management. Furthermore, it continues to evolve through an active open-source community and supports customization via various plugins and extensions tailored to user requirements [17,18].

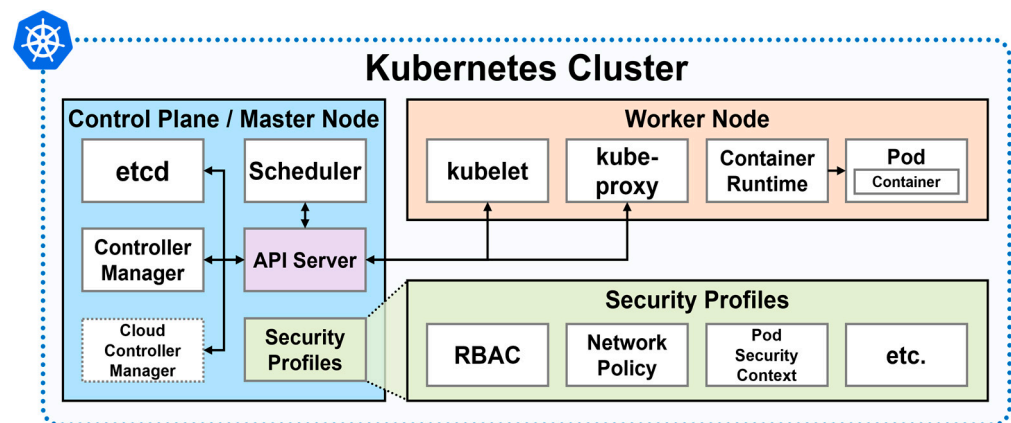


Figure 1. Kubernetes architecture (adapted from Kubernetes documentation [19]).

Continuous Integration and Continuous Deployment (CI/CD) is a methodology that automates the software development process to boost software development efficiency and minimize the time required for deployment. It is primarily employed in microservices architecture (MSA), where different components of an application are segmented into individual services, as opposed to monolithic architecture, which combines all components into a single package.

Figure 2 shows the CI/CD pipeline integrated with the Kubernetes environment. It describes how CI/CD can be used to automate the entire workflow—from code modification to container image building, testing, and deployment—thereby accelerating the development cycle. This allows development teams to perform releases more frequently and apply security patches more promptly when vulnerabilities are discovered.

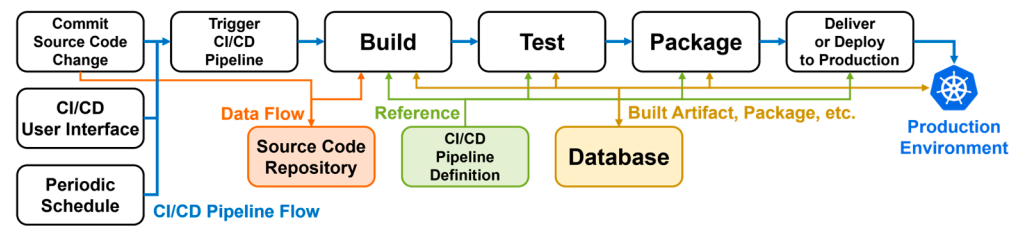


Figure 2. CI/CD Pipeline.

Due to these features, Kubernetes has become an essential tool for the development and operation of modern cloud-native applications and is recognized as a core infrastructure that supports microservices architecture and DevOps.

2.2. Data Distribution Service (DDS)

DDS is a real-time publish–subscribe communication middleware standard defined by the Object Management Group (OMG). DDS supports data-centric communication in distributed systems and is specifically designed for mission-critical real-time systems that demand high performance, reliability, and scalability—such as those used in aerospace, defense, industrial automation, and transportation systems. DDS supports a comprehensive set of QoS policies. These policies can provide precise control over communication requirements, including data transmission reliability, priority, consistency, and time constraints. QoS policies can be configured independently for each data flow, thereby enabling communication characteristics to be optimized for individual application requirements [20]. Thus, DDS has become a core technology for delivering safe and highly reliable data communication across diverse industrial sectors.

DDS’s data-centric architecture considers data as the central element of the application. In conventional message-based communications, data flow depends on the relationship between sender and receiver. In contrast, DDS is based on the concept of a Global Data Space (GDS), where data are centrally positioned within the system, thereby lowering coupling between system components and enhancing reusability. Figure 3 shows the communication architecture of DDS and Global Data Space. The data types utilized in Topics are defined using the Interface Definition Language (IDL), which enables language- and platform-independent data exchange. IDL enables automatic translation across multiple programming languages, thereby facilitating seamless integration between heterogeneous systems [21].

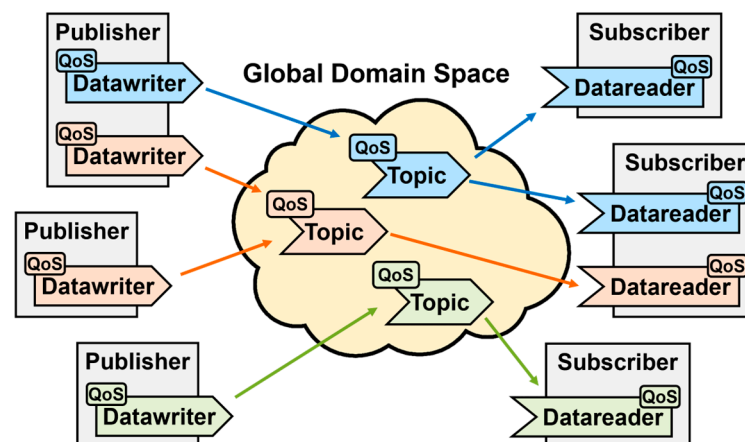


Figure 3. DDS communication architecture (adapted from DDS documentation [22]).

DDS offers a dynamic discovery feature that automatically detects and connects publishers and subscribers over the network. This enhances the system's flexibility and scalability, allowing easy node addition or removal. In addition, DDS ensures secure communication by providing authentication, access control, and data encryption. Through selective encryption—which enables encryption or access restriction for specific DDS Topics—DDS optimizes system performance while maintaining robust data security. These security mechanisms are vital for protecting data integrity and confidentiality from cybersecurity threats, particularly in critical infrastructure systems [23].

DDS employs the real-time publish–subscribe (RTPS) protocol as the standard wire protocol at the communication layer to ensure interoperability among different DDS implementations. RTPS operates over UDP/IP (User Datagram Protocol/Internet Protocol), delivering high scalability and efficiency, and supports multicast communication to optimize network resource utilization.

2.3. Front-End Processor (FEP)

FEP is a core component that enables seamless communication between remote layers—such as Remote Terminal Units (RTUs), Programmable Logic Controllers (PLCs), and Intelligent Electronic Devices (IEDs)—and the SCADA server, which manages data aggregation and control. By acting as a mediator for bidirectional communication between field equipment and the central control system, FEP directly influences the communication efficiency and stability of the entire system.

The FEP manages multiple communication lines and can monitor line status, load, and fault responses. Furthermore, by supporting various physical communication interfaces and protocols, it facilitates data exchange among heterogeneous field devices. Accordingly, it performs data preprocessing to make protocols from diverse devices intelligible to the upper SCADA system and, when necessary, incorporates security measures, such as encryption, authentication, and firewalls [24,25].

In this article, the proposed FEP is implemented using three representative SCADA protocols: DNP3.0, IEC 61850, and Modbus.

DNP3.0 is a communication protocol predominantly used in electric power and process automation systems, developed to ensure interoperability between remote devices, such as RTUs and IEDs, and a master station. Designed with asynchronous data transmission, DNP3.0 guarantees data integrity even in cases of network latency or packet loss and is extensively applied for reliable data exchange in power systems. Furthermore, it supports event-driven data transmission, which reduces unnecessary traffic and enables efficient use of limited network resources [26].

IEC 61850 is an international standard for power automation systems that ensures communication interoperability among IEDs in Ethernet-based digital substations. IEC 61850 allows the exchange of device configuration information through Substation Configuration description Language (SCL) files and employs an object-oriented data model to facilitate more efficient information exchange among devices. In addition to the manufacturing message specification (MMS) protocol used over TCP/IP (Transmission Control Protocol/Internet Protocol) networks, IEC 61850 includes two additional protocols—Generic Object-oriented Substation Event (GOOSE) and sampled value (SV)—which are used within local substation networks (Substation LAN, non-routed networks) [27].

Modbus facilitates data exchange between PLCs and various industrial automation devices. With its relatively simple protocol structure, Modbus is easy to implement and is the most widely adopted protocol in industrial settings. Due to its simplicity and broad compatibility, Modbus can be easily integrated into existing systems and is widely adopted in industrial IoT and smart grid environments. Thus, Modbus serves as a simple yet efficient

industrial communication protocol that enables reliable data exchange between SCADA systems and automation equipment [28]. Modbus exists in two primary versions, one based on serial communication that supports RTU or ASCII (American Standard Code for Information Interchange) modes, and the other based on TCP/IP. In this article, the system was implemented and evaluated using the TCP/IP-based version of Modbus [29,30].

In addition to DNP3.0, IEC 61850, and Modbus—which are widely adopted in power system automation—numerous other protocols are commonly used across power networks, particularly as these systems become more interconnected and intelligent. Table 1 is the list of several other communication protocols used in power networks, as well as data exchange.

Table 1. Communication protocols used in power networks and data exchange.

Protocol	Type	Common Usage
OPC-UA	IP-based	Secure and scalable industrial data exchange
DLMS/COSEM	Wired/Wireless	Smart metering and energy data exchange in utility networks
ICCP	Wired (IEC60870-6/TASE.2)	Data exchange between control centers and utility operators
MQTT	IP-based, Publish/Subscribe	Lightweight messaging for IoT, DERs
AMQP	IP-based, Message-oriented	Reliable messaging middleware in distributed energy systems
PROFINET	Ethernet	Real-time industrial automation
BACnet	Wired/Wireless	Building automation and smart grids
Wi-Fi	Wireless (IEEE 802.11)	Local wireless connectivity for smart devices
Bluetooth	Wireless (Short-range)	Short-range device communication and setup
ZigBee	Wireless (IEEE 802.15.4)	Low-power sensor and metering networks
LoRaWAN	Wireless (Long-range)	Low-power wide-area networks (LPWAN) for remote assets
6LoWPAN	Wireless (IPv6 over IEEE 802.15.4)	Constrained IoT devices and networks

2.4. Common Information Model (CIM)

The CIM originated from a research project conducted by the Electric Power Research Institute (EPRI) in the late 1990s and was later standardized as the IEC 61970 series. It is a standard information model designed to ensure data consistency and interoperability within power systems. CIM is an information modeling standard defined in the Resource Description Framework (RDF) format and comprises a set of abstract information classes that represent equipment, network topology, and system data in generation, transmission, and distribution systems [31]. By employing this standardized information model, interoperability among heterogeneous applications within power systems can be significantly enhanced. CIM ensures not only structural but also semantic consistency of data, thereby enabling seamless data integration and exchange between systems [32].

CIM is constructed using an object-oriented modeling methodology based on the Unified Modeling Language (UML). All components are defined as classes, and object-oriented relationships, such as inheritance, association, aggregation, and composition, are

used to systematically represent the complex structures and relationships within the power domain [33]. This modeling approach clarifies the logical relationships among power system components and enhances both the scalability and maintainability of the model.

Figure 4 illustrates the modeling of an Alternating Current (AC) transmission line segment (ACLineSegment) using inheritance relationships. ACLineSegment models a single electrical system, comprising one or more conductors with specific electrical properties, used to transmit AC within power systems. It is one of the core information classes in the smart grid domain and is primarily used to represent major transmission components within electric power networks. Due to its rich set of attributes and inheritance structure, it effectively illustrates the hierarchical nature of the Common Information Model (CIM). Moreover, it includes attributes that directly correspond to commonly acquired parameters such as voltage and reactance in various SCADA protocols, making it a representative entity for data mapping and performance comparison across protocols. Accordingly, this article utilizes ACLineSegment as the basis for CIM mapping and for evaluating the performance of the proposed system.

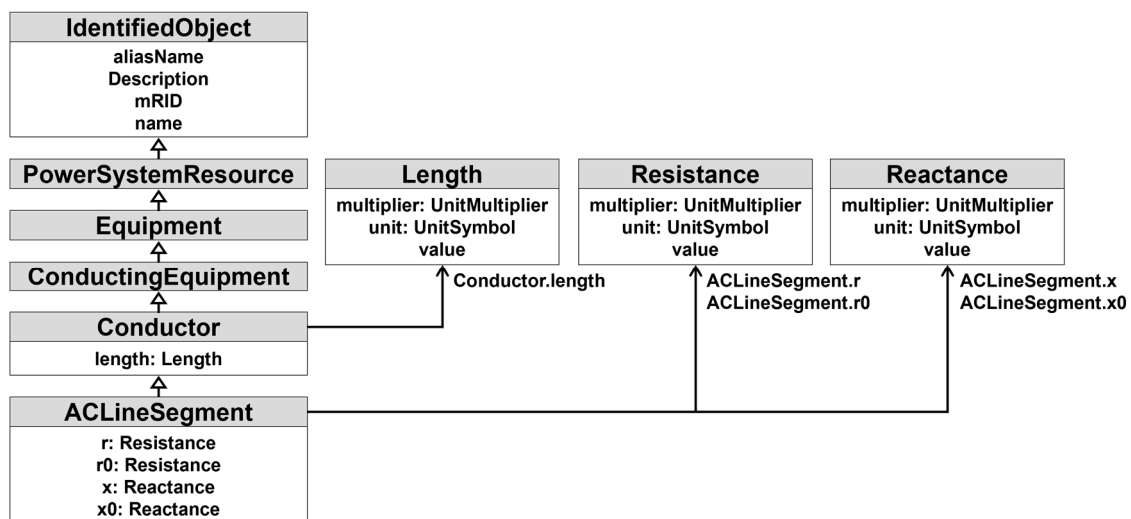


Figure 4. ACLineSegment CIM Class UML diagram.

The root class of ACLineSegment is the IdentifiedObject class, which provides a common identifier to all classes that require Identification and Naming attributes. PowerSystemResource, which inherits from IdentifiedObject, represents all physical and logical resources in the power system, while Equipment represents physical devices. ConductingEquipment refers to components conductively connected to carry current in an AC power system, while Conductor represents elements comprising conductive materials with specific electrical properties used to transmit current between distinct points. ACLineSegment, the final class in this inheritance hierarchy, is associated with the length, resistance, and reactance classes—representing length, electrical resistance, and reactance, respectively—enabling precise modeling of the physical and electrical properties of a transmission line segment. This systematic class structure of CIM allows for the clear definition of complex components in power systems and their relationships, thereby enhancing system integration and interoperability [34].

2.5. Related Work

In the smart grid environment, the communication architecture is recognized as a core element for achieving real-time data acquisition, system scalability, and interoperability among heterogeneous devices [35,36]. Accordingly, Table 2 illustrates that active research

is being conducted on DDS and cloud-native communication infrastructures, which will be further discussed in Section 5.

Table 2. Research on DDS and cloud-native communication infrastructure.

Article Title	Main Topic Addressed	Relationship with This Article	Ref.
Cloud IEC 61850: DDS Performance in Virtualized Environment with Opendedds	Evaluation of DDS-based IED 61850 automation system in virtualized environments	Demonstrates that DDS meets real-time requirements even in virtual/cloud environments	[37]
DDS-Based Interoperability Framework for Smart Grid Testbed Infrastructure	Design and implementation of a DDS-based smart grid testbed enabling communication and data synchronization between control nodes	Shows DDS enables real-time interoperability among heterogeneous devices	[38]
Implementation of DDS Cloud Platform for Real-Time Data Acquisition of Sensors	DDS-based sensor data collection and visualization using ESP32 and Django	Demonstrates a lightweight real-time data pipeline using DDS for smart grid sensing applications	[39]
Implementation of DDS Cloud Platform for Real-Time Data Acquisition of Sensors for a Legacy Machine	Cloud platform design using Real-Time Innovation (RTI) DDS and Node-RED for real-time IIoT data processing	Extends DDS architecture to a scalable cloud-native platform, validating its practicality for IIoT systems	[40]
Evaluating Performance of OMG DDS in Kubernetes Container Deployment (Industry Track)	Integration of DDS with Kubernetes for communication performance	Proves enhanced scalability and management efficiency of DDS with container orchestration	[10]
Data-Centric Publish–Subscribe Approach for Distributed Complex Event Processing Deployment in Smart Grid Internet of Things	DDS-based deployment of Distributed Complex Event Processing (DCEP) in smart grids	Presents a DDS-integrated architecture for scalable and QoS-aware real-time event processing	[41]
Intercloud Message Exchange Middleware	DDS-based Intercloud Message Exchange (ICME) architecture	Proposes scalable and interoperable cloud messaging using DDS and Web Ontology Language (OWL) ontology	[42]
High Availability Control Method in Container-Based Microservice Applications over Multiple Clusters	High Availability Control Method (HACM) with multi-Kubernetes clusters	Ensures rapid recovery and service continuity in Energy Management System (EMS) environments	[43]
Kubernetes-Container-Cluster-Based Architecture for an Energy Management System	Reliability modeling of Kubernetes-based EMS using Pod redundancy and Markov model	Achieves 99.9999504% system reliability through mathematical modeling	[44]
Microservice-Based Architecture for an Energy Management System	Microservice-based EMS using Mixed-integer Linear Programming (MILP) resource optimization	Enhances reliability via container-level isolation and hot-swapping	[9]
DDS and OPC UA Protocol Coexistence Solution in Real-Time and Industry 4.0 Context Using Non-Ideal Infrastructure	Gateway architectures between DDS and other pub-sub protocols	Highlights a research gap in real-time protocol conversion between DDS and traditional SCADA protocols	[45]

Building on the background and previous work discussed above, we now describe the architecture and implementation of the proposed system.

3. Materials and Methods

3.1. Kubernetes Cluster

Managed by CNCF, Kubernetes is inherently designed for deployment in cloud environments. This design philosophy allows Kubernetes to be flexibly deployed across a wide range of infrastructure environments—on-premises, hybrid, or public cloud—depending on user needs and requirements. In this article, the proposed architecture was implemented by constructing an on-premises cloud environment based on the Proxmox Virtual Environment (VE).

Proxmox VE is an open-source Type 1 hypervisor based on Debian, supporting both virtual machine and container virtualization through the Kernel-based Virtual Machine (KVM) hypervisor and Linux Containers (LXC). It also consolidates multiple devices into a cluster for efficient management of storage, network, and high availability. Proxmox VE allows for fine control over resources, such as Central Processing Unit (CPU) cores, memory, and network bandwidth allocated to each virtual machine (VM) and container, thereby enabling complete isolation between services. This minimizes external interference, thereby enhancing the accuracy and reliability of workload execution and analysis, and also allows for performance evaluation under resource-constrained scenarios [46].

In this article, five physical servers were individually installed with Proxmox VE and configured into a single cluster. The clustering feature of Proxmox VE allows for centralized management by grouping multiple nodes into a single administrative entity. Each node was connected to a gigabit local network. To ensure network stability and prevent address conflicts, all nodes were configured with static IP addresses and synchronized using time protocol (NTP). Virtual machines can be provisioned and operated on any node within the cluster and can be easily migrated to other nodes using the built-in migration functionality. This Proxmox cluster was used as the foundational infrastructure for deploying both a Kubernetes cluster and a Load Balancer cluster. Furthermore, the high availability (HA) features provided by Proxmox were leveraged to construct a fault-tolerant architecture, enabling automatic recovery in the event of VM failures. These VMs correspond to the nodes of the Kubernetes and Load Balancer clusters, thereby enhancing the overall resilience of the system.

The Kubernetes cluster deployed on the Proxmox cluster was constructed using Kubeadm, Kubernetes's official clustering tool, and designed with high availability in mind. To maintain a cluster state, Kubernetes internally leverages the Reliable, Replicated, Redundant, and Fault-tolerant (RAFT) consensus algorithm. As such, the control plane was configured with a minimum of three Master Nodes, ensuring quorum is preserved even in the event of a single Master Node failure. While it can be extended to more than five Master Nodes to further improve fault tolerance and stability of the cluster, this article does not encompass a quantitative evaluation of Kubernetes cluster reliability. Therefore, the cluster was constructed under the assumption that simultaneous failures of more than two Master Nodes would not occur [47].

Additionally, the cluster was configured with three Worker Nodes to ensure efficient load balancing and optimal resource utilization during workload execution. This 3-Master and 3-Worker configuration constitutes a medium-scale cluster suitable for research purposes, offering balanced stability and performance, and can be seamlessly scaled to a larger cluster without structural modifications.

To ensure uninterrupted access to the Kubernetes API Server even during Master Node failures, reliable connectivity to a healthy Master Node is essential. To achieve this, this article implemented a load-balancing mechanism by combining HAProxy and KeepAlive, enabling real-time monitoring of Master Node status and effective redirection of traffic to healthy Master Nodes [48].

HAProxy is a highly reliable reverse-proxy solution that provides high availability and load balancing for TCP- and HyperText Transfer Protocol (HTTP)-based applications. By performing periodic checks to identify healthy Master Nodes, HAProxy ensures automatic traffic redirection, enabling consistent access to a functioning API Server. To mitigate potential failures of the HAProxy, multiple HAProxy instances were deployed in conjunction with a KeepAlived service based on the Virtual Router Redundancy Protocol (VRRP). High availability is ensured by dynamically managing priority among nodes and assigning the Virtual IP (VIP) address to the node with the highest priority.

Figure 5 illustrates the process by which traffic to the API Server is routed to the Master Node through the high-availability architecture. When attempting to access the API Server via a Worker Node or a developer REST API, the request is initially directed to the VIP address assigned to the Load Balancer cluster. Because the VIP is assigned to the Load Balancer node with the highest priority, its HAProxy instance distributes traffic to the available Master Nodes. This approach eliminates the single point of failure in accessing the API Server, thereby significantly enhancing the availability and reliability of the entire Kubernetes cluster.

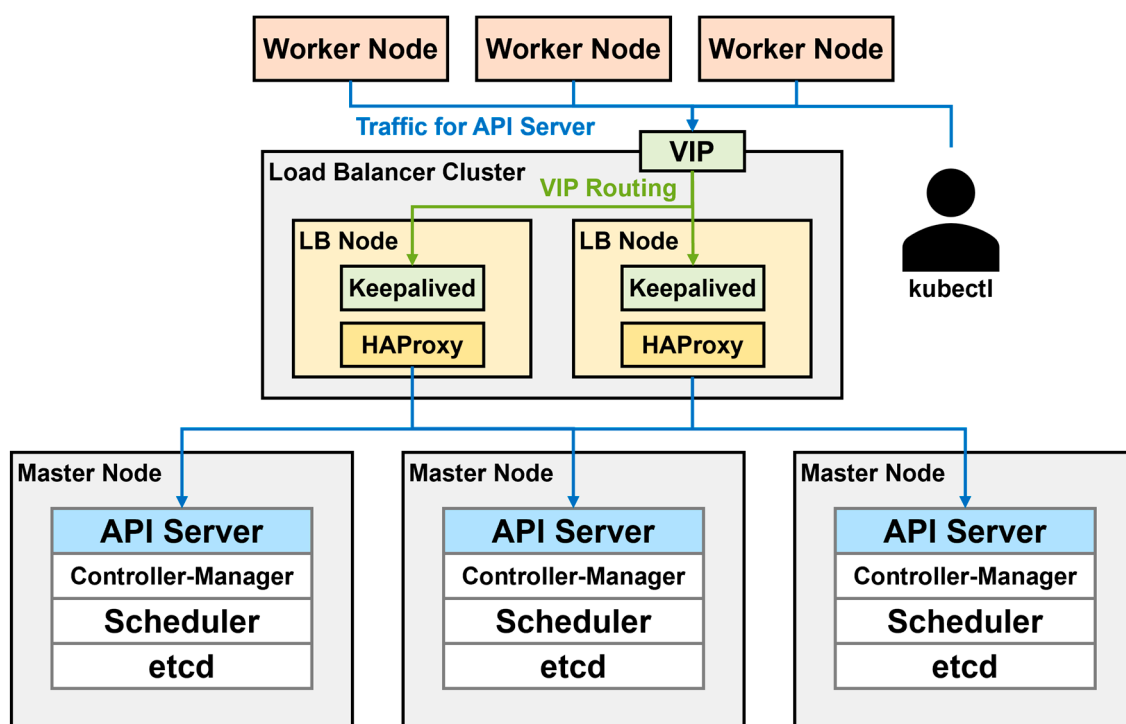


Figure 5. Kubernetes cluster high availability setup (adapted from Kubernetes documentation [49]).

3.2. Kubernetes-Based FEP System

Figure 6 presents the overall architecture of the Kubernetes-based FEP system. The Kubernetes-based FEP system consists primarily of two core components: the FEP Protocol Converter and DDS Integration Service. The FEP Protocol Converter, the core module proposed in this article, acquires data from the Outstation Simulator via the SCADA protocol and performs conversion into the CIM-based DDS protocol. This protocol conversion process enables consistent handling of data collected from various Outstation devices using an integrated protocol. Furthermore, by establishing a clear separation between the data acquisition and data application layers, inter-component dependencies are reduced, thereby improving overall system stability and maintainability.

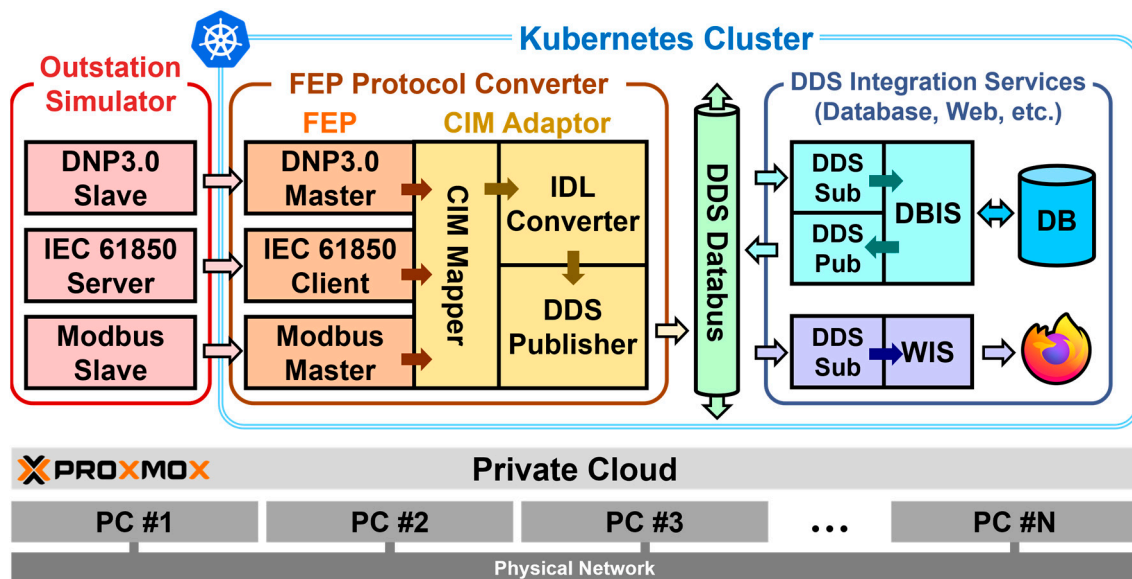


Figure 6. Kubernetes-based FEP system architecture.

The DDS Integration Service subscribes to the standardized data published by the FEP Protocol Converter and offers various services, such as storing the data in a database and enabling real-time monitoring via a web interface. This architecture is designed based on microservices, allowing independent scaling and deployment of each service. Leveraging Kubernetes orchestration, the system achieves high availability and self-healing capabilities, enabling a stable SCADA system.

The Outstation Simulator is implemented to simulate measurement devices of remote facilities and deployed on a standalone virtual machine, separate from the Kubernetes cluster. Each remote device—namely the DNP3.0 Slave, IEC 61850 Server, and Modbus Slave—simulates data corresponding to the entity serving the same role as the ACLine-Segment described in Figure 4, using a protocol-specific format. For example, in DNP3.0, the wire length is represented as a point in Group 40 Variation 2 using a 16-bit signed integer; in IEC 61850, every piece of information of the wire is encoded as a ZLIN element representing an overhead line; in Modbus, wire resistance is expressed as a 16-bit read-only word in the Input Register.

3.3. Front-End Processor

The FEP Protocol Converter is a component responsible for acquiring data from remote devices using SCADA protocols, such as DNP3.0, IEC 61850, and Modbus, and converting them into CIM-based DDS Topics for use by the DDS Integration Service. The FEP Protocol Converter comprises two major components: the Front-end Processor and CIM Adaptor.

The FEP acquires data from remote equipment using SCADA protocols and transmits the data to the CIM Adaptor via a Unix Domain Socket. The Unix Domain Socket is an efficient mechanism for inter-process communication on the same host, offering lower overhead and higher bandwidth compared to TCP/IP sockets [50]. This architecture enables the simultaneous execution of multiple FEP processes that support various SCADA protocols. Moreover, due to the asynchronous processing method, even if performance bottlenecks occur in the CIM Adaptor, they do not affect the operation of FEP processes, thereby ensuring their independent and stable execution.

The FEP was implemented using open-source libraries corresponding to each protocol. As shown in Table 3, the selected libraries were chosen based on popularity indicators, such as the programming language used, the number of stars and forks, and the count of

open issues and pull requests on GitHub. As of September 2022, the opendnp3 library was designated end of life and archived, as development efforts shifted toward stepfunc/dnp3, a DNP3.0 implementation in Rust. Nonetheless, opendnp3 remains the best C/C++-based library for this protocol and was thus adopted in this article [51–53].

Table 3. SCADA Protocol Libraries Used.

	Opendnp3 [51]	Libiec61850 [52]	Libmodbus [53]
Protocol	DNP3.0	IEC 61850	Modbus
Language	C++	C	C
Stars	305	944	3.6k
Forks	233	488	1.8k
Issues	archived	152	79
Pull requests	archived	44	84
Library Version	3.1.2	1.6.0	3.1.11

3.4. CIM Adaptor

The power data collected by the FEP do not directly conform to the RDF-based CIM data structure defined by IEC 61970. Therefore, to enable utilization in applications, it is necessary to analyze the differences in representation between the original data and CIM data and apply mapping rules accordingly via a CIM-based Adaptor that transforms the data into the CIM format.

Figure 7 illustrates the architecture of the CIM Adaptor, which consists of three primary components: the CIM Mapper, IDL Converter, and DDS Publisher.

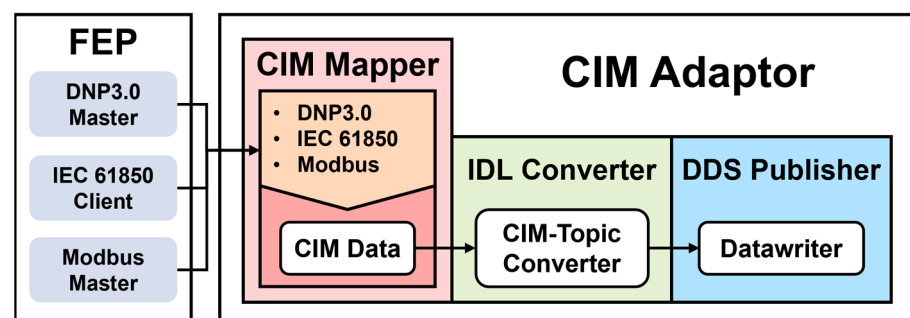


Figure 7. CIM Adaptor.

3.4.1. CIM Mapper

It analyzes the data received from the FEP and converts protocol-specific data—such as from DNP3.0, IEC 61850, and Modbus—into the CIM format. During this process, mapping rules appropriate to each protocol are applied.

In the case of IEC 61850, the data are self-descriptive and thus can be mapped into CIM format while preserving its logical structure. That is, by converting structural elements, such as Logical Nodes (LN) and Data Attributes defined in IEC 61850 into corresponding components in the CIM Object model, the semantic integrity of the original data is preserved.

In contrast, DNP3.0 and Modbus fundamentally provide address-based register values that do not include semantic data. Therefore, to convert such data into CIM data, it must be mapped based on predefined tag information. The data collected from DNP3.0 and Modbus are associated with specific CIM Classes and attributes based on a mapping table, thereby enabling the conversion of each data point into meaningful CIM data.

Algorithm 1 shows the logical flow of the data-to-CIM mapping process for each protocol as previously described, represented in pseudocode.

Algorithm 1 CIM Mapper

Require: Input Data from FEP (IEC 61850, DNP3.0, Modbus)
Ensure: Mapped CIM Data

```

1:  Initialize CIM Data Structure
2:  for each Data Packet received from FEP do
3:      Extract protocol type, source ID, and raw values
4:      if Protocol == IEC 61850 then
5:          Load Mapping Table for IEC 61850
6:          Split IEC 61850 Object into hierarchical structure
7:          Identify Logical Node (LN) and Data Attribute (DA)
8:          Map LN and DA to corresponding CIM Class
9:      else if Protocol == DNP3.0 then
10:         Load Mapping Table for DNP3.0
11:         Extract Object Type and Index Number
12:         Identify Data Value associated with Index Number
13:         Map Point Number to CIM if rule exists
14:      else if Protocol == Modbus then
15:         Load Mapping Table for Modbus
16:         Extract Register Address and associated Data Value
17:         Validate Register Type (Holding, Input, Coil, Discrete)
18:         Map Register Address to CIM if within valid range
19:      end if
20:  Store Mapped CIM Data
21: end for

```

3.4.2. IDL Converter

It converts the transformed CIM data into a Topic format for use within DDS. IDL, a language for defining data structure and types, is used to convert CIM data into IDL-based Topic data, enabling data exchange between heterogeneous systems in a DDS environment.

3.4.3. DDS Publisher

It publishes the converted DDS Topics, making them available for use by other applications. This allows data collected from various remote devices (RTU, PLC, IED, etc.) to be converted into IEC 61970-compliant CIM-based DDS data, which can be utilized in diverse applications (databases, web, etc.).

Algorithm 2 represents the logical flow of publishing the previously converted CIM-based DDS data in pseudocode.

The DNP3.0 protocol represents data using the concepts of Group and Variation. An Object Group is a higher-level concept that defines the data type (Binary, Analog, Counter, Time), while Variation is a subordinate concept specifying the detailed format (integer, float16/32). Object data with the same Group and Variation are distinguished by their Index, which is a simple value without any semantic meaning in the protocol standard. Accordingly, the Master and Outstation must mutually agree on defining the semantics of each Index, allowing for the determination of which device value is represented by the respective Object.

To map between DNP3.0 Objects to IEC 61970 CIM Classes, the Binary Output Status (Object Group 10, Variation 2) and Analog Output Status (Object Group 40, Variation 2) are primarily used, as these types correspond to most values used in CIM Class definitions. Table 4 shows part of the mapping table used to associate the ACLineSegment Class of CIM

with DNP3.0. The label g40v2 refers to Group 40, Variation 2, and the Object Indexes are defined sequentially starting from 0x00.

Algorithm 2 CIM to DDS Topic Publisher (With Source Order)

Require: Initialized DDS Domain Participant, List of CIM Data

Ensure: DDS Message Published Successfully

```

1:  Assume DDS Domain Participant and Publisher are initialized
2:  Load IDL Type Definition and Target Topic Name
3:  if DataWriter already exists for Topic then
4:    Reuse existing DataWriter
5:  else
6:    Create New DataWriter for Topic
7:    Set QoS Parameters: ▷ Example: Reliable, DestinationOrder, KeepLast(10)
8:      • Reliability ← Reliable
9:      • DestinationOrder ← BySourceTimestamp
10:     • History ← KeepLast(10)
11:  end if
12:  for each CIM_Object in CIM_Data do
13:    Extract {ID, Value}
14:    Try to BuildMessage(CIM_Object)
15:    if Message is Invalid then
16:      Log Error and Continue
17:    end if
18:    Write(DataWriter, DDS_Message)
19:  end for
  
```

Table 4. DNP3.0 CIM mapping.

DNP3.0		CIM	
Group, Variation	Index	ACLineSegment Class	
g40v2	0x00	length	UnitSymbol
g40v2	0x01		UnitMultiplier
g40v2	0x02		Value
g40v2	0x03	r	UnitSymbol
g40v2	0x04		UnitMultiplier
g40v2	0x05		Value

Both IEC 61850 and IEC 61970 standards provide foundational system information necessary to construct equipment models for power systems. When representing the interconnections and configurations of system equipment in a data model, IEC 61850 employs a hierarchical structure, while IEC 61970 utilizes an RDF-based reference structure, resulting in representational differences between the two standards. Furthermore, while both standards define core system components, they differ in model names, attributes, and structural composition, resulting in the existence of mismatched data between the two data models. Figure 8 illustrates the mapping process of IEC 61850 Object data into the CIM.

The IEC 61850 to CIM Mapper module applies mapping rules to perform the transformation between IEC 61850 and CIM data. IEC 61850 data comprises Objects and their corresponding Values and are mapped while preserving their logical structure. Accordingly, IEC 61850 data are categorized into hierarchical elements, such as Logical Node (LN), Function Class (FC), Data Object (DO), and Data Attribute (DA), and are subsequently mapped to CIM Classes based on the IEC 61850-CIM mapping rules.

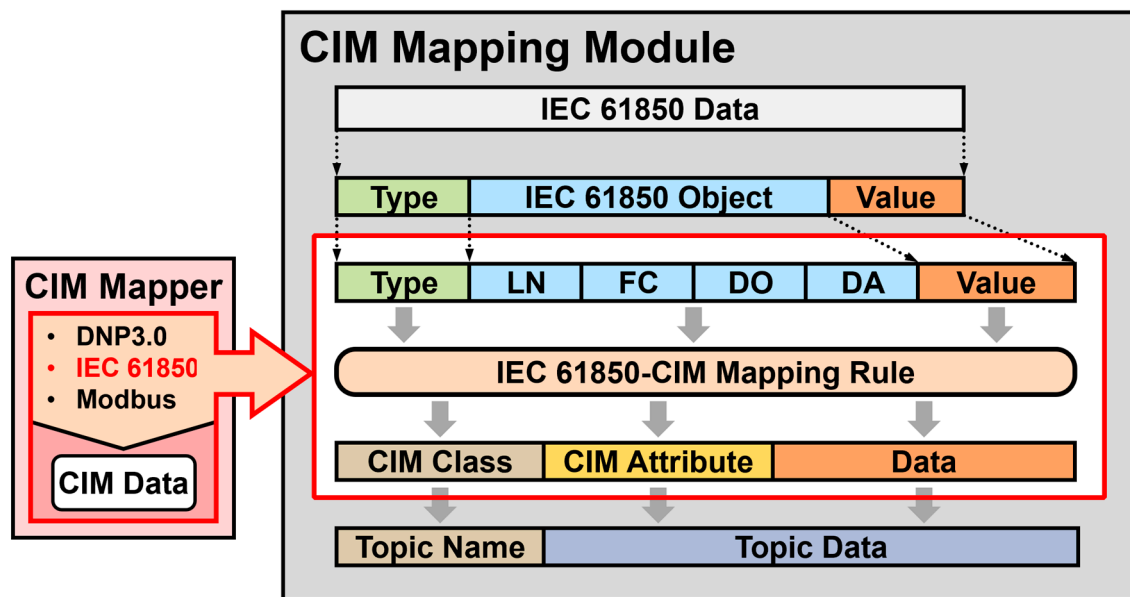


Figure 8. CIM Adaptor—IEC 61850 CIM mapping.

Table 5 presents an example of a matching analysis between LNs in IEC 61850 and power system model information in IEC 61970 CIM Classes. A minimal system configuration model consists of components, such as generators, loads, transmission lines, transformers, circuit breakers, and disconnectors.

Table 5. IEC 61850 LN, CIM Class matching.

IEC 61850 Logical Node	IEC 61970
Power Overhead line (ZLIN)	AcLineSegment
Circuit Breaker (XCBR)	Breaker
Disconnector/Switch (XSWI)	Switch
PowerTransformer (YPTR)	PowerTransformer
Generator (ZGEN)	GeneratingUnit

Using ZLIN (LN), which models information about electrical lines, and AcLineSegment (CIM) as examples, a part of the mapping table derived through comparative analysis of the two data models is shown in Table 6.

The Modbus protocol represents data using the concept of Offset Addresses based on the data type. Each data type has a predefined format and size, and values stored in memory can be read or written based on an address that is offset from a reference point. Therefore, similar to DNP3.0, Modbus also transmits data based on data types and addresses. Because each address is a simple value without inherent semantics, the Master and Slave must mutually agree on assigning meaning to it.

Table 6. IEC 61850 CIM mapping.

IEC 61850		CIM	
ZLIN LN		AcLineSegment Class	
LinLenkm	ZLIN\$CF\$LinLenkm\$units\$SIUnit	length	UnitSymbol
	ZLIN\$CF\$LinLenkm\$units\$multiplier		UnitMultiplier
	ZLIN\$SP\$LinLenkm\$setMag\$f		value
RPs	ZLIN\$CF\$RPs\$units\$SIUnit	r	UnitSymbol
	ZLIN\$CF\$RPs\$units\$multiplier		UnitMultiplier
	ZLIN\$SP\$RPs\$setMag\$f		value

To map Modbus data to IEC 61970 CIM Classes, the Data Type corresponding to Modbus Input Registers is utilized, enabling compatibility with most attribute values used in CIM Classes. Table 7 presents a part of the mapping table used to align the CIM ACLineSegment Class with Modbus. The mapping uses the Input Register as the data type, with Offset Addresses assigned sequentially beginning from 0x00.

Table 7. Modbus CIM mapping.

Modbus		CIM	
Data Type	Address	ACLineSegment Class	
Input Register	0x00	length	UnitSymbol
Input Register	0x01		UnitMultiplier
Input Register	0x02		Value
Input Register	0x03	r	UnitSymbol
Input Register	0x04		UnitMultiplier
Input Register	0x05		Value

3.5. Performance Evaluation Testbed and Scenario

In real-time systems such as smart grids and industrial automation platforms, communication performance is directly linked to the overall safety and reliability of the system. Therefore, in this section, performance evaluations of the proposed system—including the FEP and multiple DDS implementations—were conducted to validate its effectiveness and to provide key performance indicators for deployment. Among various metrics, throughput and latency were considered the primary indicators for evaluating system performance. Table 8 presents the definitions and measurement methodologies for each metric. Throughput refers to the amount of data a system can process per unit of time and serves as a key indicator for evaluating system scalability and load-handling capability. In smart grid environments, where numerous sensors and devices continuously transmit high-frequency data, failure to maintain sufficient throughput can lead to data loss or system bottlenecks.

Latency refers to the time it takes for a specific piece of data to travel from the transmission point to the reception point, and is particularly critical in real-time systems where control commands and acquired data must be delivered without delay. Excessive latency can introduce timing errors in control signals, potentially leading to equipment malfunctions or reduced operational efficiency. Therefore, minimizing latency is essential to ensure the real-time responsiveness of the system.

Table 8. Evaluation metrics used in performance evaluation.

Metrics	Definition	Calculation
Throughput (Mbit/s)	The amount of data successfully processed per unit of time	$\text{Throughput} = \frac{N_{\text{messages}}}{T_{\text{total}}}$ <p>N_{messages}: total number of successfully processed messages T_{total}: the total time duration of the measurement</p>
Latency (ms)	The time delay between when a data message is sent from the source and when it is received at the destination	$\text{Latency} = T_{\text{receive}} - T_{\text{send}}$ <p>T_{send}: the timestamp at which the message is sent T_{receive}: the timestamp at which the message is received</p>

3.5.1. FEP Protocol Converter

In this section, the performance of one of the core functionalities of the proposed system—protocol conversion for SCADA protocols—is evaluated. The FEP Protocol Converter receives data from protocols such as DNP3.0, IEC 61850, and Modbus, and transforms them into CIM-based DDS Topic messages. As illustrated in Figure 6, throughput and latency were measured during the transformation process, specifically from the FEP to the CIM Mapper. This segment includes the mapping of data received from each protocol's Master or Client into the Common Information Model format. The subsequent stages, including the IDL Converter and DDS Publisher, were not evaluated in this section because their implementations differ depending on the specific DDS vendor. Instead, their performance is separately assessed in the following section for each DDS implementation.

As previously mentioned, the FEP Protocol Converter employs Unix Domain Sockets—a form of inter-process communication (IPC)—to maximize communication performance, given that all components reside on the same host. Accordingly, a multithreaded program was developed in which the FEP and CIM Mapper were implemented as separate threads, and this program was used to conduct the performance evaluation.

In the detailed evaluation scenario, the FEP receives data via protocol-specific Master and Client and maps the data to the ACLineSegment class of the CIM. In this setup, data were initially acquired once from the Outstation Simulator and then stored in memory, where it was repeatedly used for conversion operations. This configuration was intended to minimize variability introduced by the data acquisition process specific to each protocol and implementation library, thereby enabling a more accurate measurement of the pure processing performance within the transformation phase. Figures 9 and 10 illustrate the procedures for measuring throughput and latency, respectively, based on this scenario.

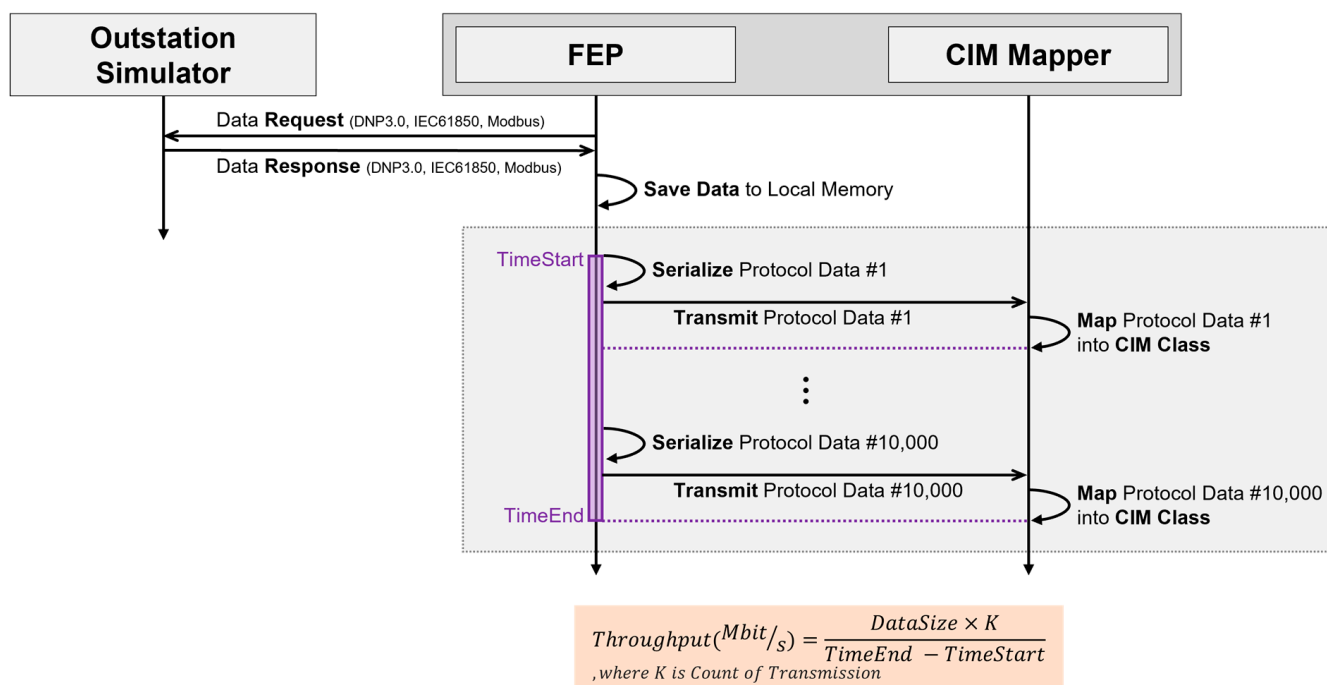


Figure 9. Throughput test sequence diagram for FEP Protocol Converter.

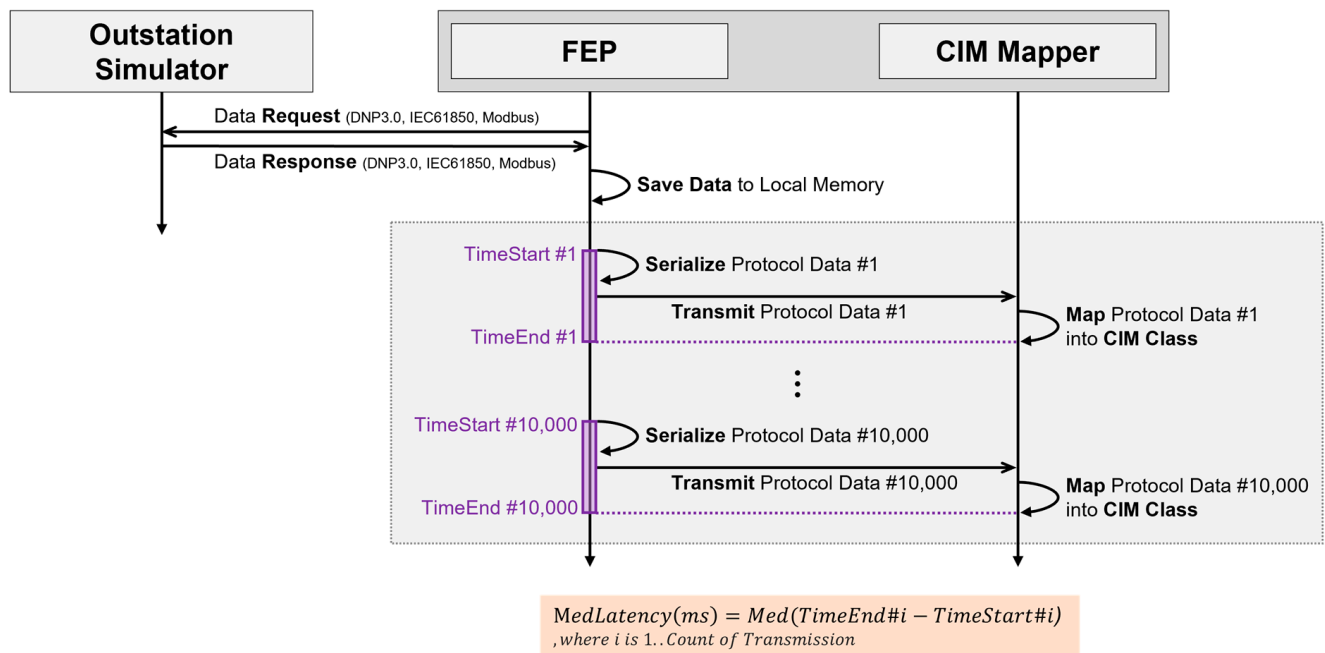


Figure 10. Latency test sequence diagram for FEP Protocol Converter.

3.5.2. DDS Implementations

Because the DDS standard defined by OMG is open, multiple DDS implementations based on this standard are available. This article selects four DDS implementations (i.e., RTIDDS, CycloneDDS, FastDDS, and OpenDDS) for comparative performance analysis. Each DDS implementation provides its performance evaluation library tailored to its product, but due to vendor-specific configurations and communication methods, it is difficult to objectively compare their performance. Therefore, a platform named DDS-PerfTester was developed and used to compare and analyze the performance of multiple DDS implementations under a uniform structure and configuration [54–57].

Although standard documentation exists for DDS architecture and APIs, implementation differences—such as in QoS configuration methods—exist across vendors. Furthermore, due to the presence of additional QoS parameters and configurations aimed at enhancing the performance of each DDS implementation, creating a single performance evaluation code applicable to all implementations may be impractical. Accordingly, the performance evaluation code was unified across all implementations, while DDS-specific configuration and communication codes were separately developed to ensure compatibility, thereby maintaining consistent architecture and enhancing maintainability.

DDS-PerfTester is a performance analysis tool designed to provide quantitative metrics for evaluating, comparing, and analyzing the performance of various DDS implementations. The main goal of the DDS-PerfTester is to manage and run performance analysis scenarios and compute the corresponding results to enable effective comparison across DDS implementations. As shown in Figure 11, DDS-PerfTester comprises two modules: DDS-PerfTest Master (hereinafter referred to as DP-Master) and DDS-PerfTest Slave (hereinafter referred to as DP-Slave). The DP-Master module is responsible for managing performance evaluation scenarios and computing results, while the DP-Slave module handles data transmission control and event delivery via interfaces with specific DDS implementations.

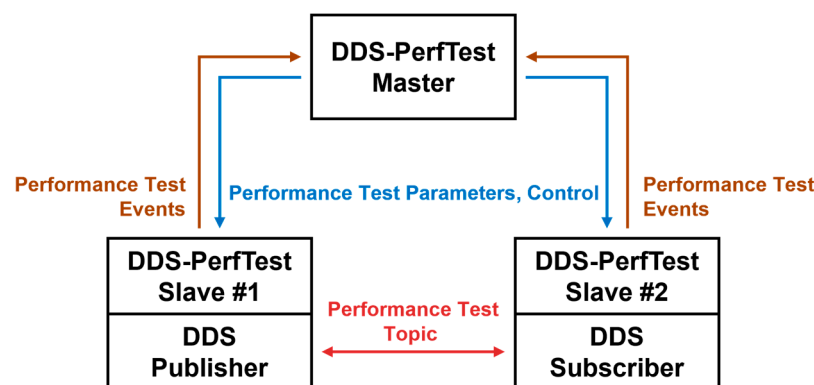


Figure 11. DDS-PerfTester architecture.

DDS-PerfTester primarily focuses on evaluating and analyzing two key performance metrics: throughput and latency. Throughput is the number or size of topics that can be processed per unit of time, whereas latency is the elapsed time from the moment a Publisher publishes a topic to the moment a Subscriber receives it. The DP-Slave detects the publish and subscribe timings of topics and transmits them as event data to the Master. The DP-Master calculates the time differences between the events reported by DP-Slave to derive the throughput and latency values. This method enables objective measurement and comparison of performance characteristics across various DDS implementations, thereby providing quantitative indicators for selecting a DDS solution optimized for specific application environments.

The DDS-PerfTester allows for scenario management and automation via configuration files in YAML format. Although DDS-PerfTester was developed with the primary purpose of evaluating DDS implementation performance, it is designed to allow for the substitution of DDS Publishers and Subscribers with other applications, enabling performance evaluation of non-DDS targets as well. Figure 12 presents an example configuration file for DDS performance testing. The scenario involves one Publisher and four Subscribers, with a topic size of 32,768 bytes and a multicast DDS communication model. The DP-Master distributes the performance evaluation scenario specified in the configuration file to each DP-Slave, ensuring that all DDS Publishers and Subscribers conduct a performance evaluation for the same target.

```
# Configuration File Example for DDS Performance Test
mode: throughput                                # Performance Test Mode (Throughput / Latency)
logFile: ddsThroughput.log                      # Performance Test Log File Name
resultFile: ddsThroughput.res                   # Performance Test Result File Name
count: 10,000                                   # Total Data(Topic) Count
protocol:
  type: DDS                                     # Performance Test Protocol – Data Distribution Service
  model: multicast                             # Communication Model (Unicast / Multicast)
  datsize: 32,768                              # Topic Size
  pubCount: 1                                 # Count of Publisher
  subCount: 4                                 # Count of Subscriber
```

Figure 12. DDS-PerfTester configuration example.

Figure 13 illustrates the method used by DDS-PerfTester to measure throughput. Once the DP-Master loads the configuration file and distributes it to each DP-Slave, each DP-Slave applies the received scenario and initiates performance evaluation. Upon receiving the test initiation event from the DP-Master, the DP-Slave sends a publish-start event back to the DP-Master. It then publishes a predefined number of topics according to

the performance evaluation scenario. Each time a DDS Subscriber subscribes to a topic, it reports the corresponding event to the DP-Master. Once the DP-Master receives the publish-start event and subscription-completion events for all topics, it determines that all topics have been published and subscribed, and then terminates the test. The time interval between the publish-start event and the final topic subscription event is then calculated to determine the total delivery time for all topics, which is used to compute the overall throughput.

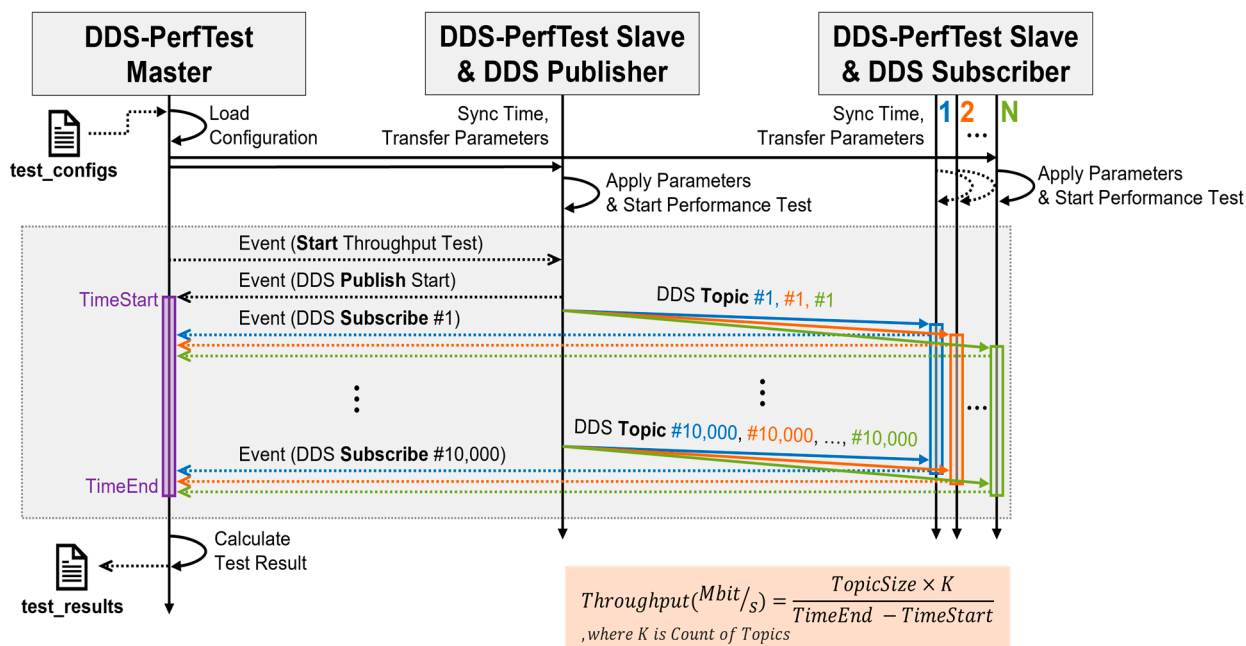


Figure 13. Throughput test sequence diagram for 1Publisher-NSubscriber case.

Figure 14 illustrates the method used by DDS-PerfTester to measure latency. Similarly to throughput measurement, the DP-Master loads the configuration file and distributes it to each DP-Slave, but instead of triggering a throughput test, it sends a latency test initiation event. Upon receiving this event, the DP-Slave reports a topic-publish event to the DP-Master before publishing the topic via the DDS Publisher. The DDS Subscriber that subscribes to the corresponding topic subsequently reports a topic-subscription event to the DP-Master. The DP-Master calculates the time interval between the topic-publish event and the subscription event to measure the latency of a single topic transmission. This process is repeated multiple times, and statistical values, such as the average, median, and 95th percentile (p95), are calculated from the collected data to derive the final latency measurement.

To evaluate the performance of each DDS implementation, a testbed was constructed using a Proxmox VE cluster environment, and individual VMs with the same performance were allocated for each instance of DDS-PerfTester, Publisher, or Subscriber. The configuration of each VM and the version of the DDS implementation used are detailed in Table 9. Each VM was allocated four cores from an Intel® Core™ i7-12700 processor (Intel Corporation, Santa Clara, CA, USA), and 4 GB of memory was assigned to each VM. The network environment was constructed using Proxmox VE's Software Defined Network (SDN) to implement a VLAN environment, with a bandwidth cap of 300 Mbit/s applied to inter-VM communication to maintain a consistent network setting. The Ubuntu 22.04 LTS Server was used as the operating system. All DDS implementations were compiled and executed uniformly under the Linux Kernel version 5.15.0-134 and GNU Compiler Collection (GCC) version 11.4.0.

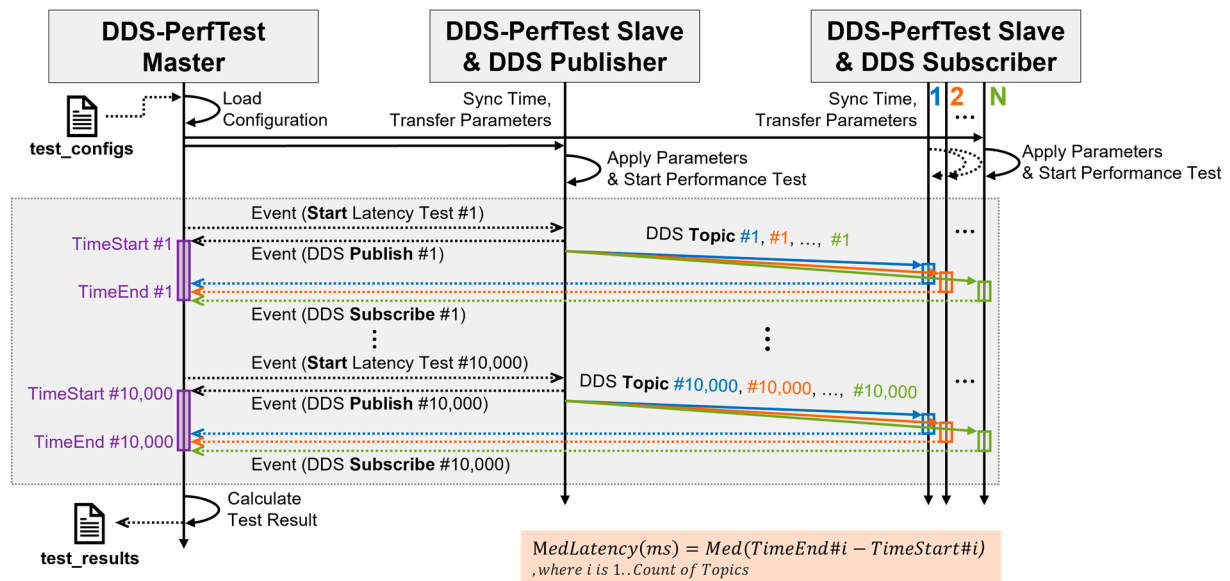


Figure 14. Latency test sequence diagram for 1Publisher–NSubscriber case.

Table 9. Testbed used for DDS performance test.

Hardware	
CPU	Intel® Core™ i7-12700 4Cores
Memory	4 GB
Software	
Network Bandwidth	Proxmox VE SDN (VLAN) 300 Mbit/s
OS	Ubuntu 22.04 LTS Server
Kernel	Linux 5.15.0-134
GCC	v11.4.0
DDS	
RTIDDS	v6.1.1
CycloneDDS	v0.11.0
FastDDS	v3.1.1
OpenDDS	v3.28.1

Each DDS implementation provides differing characteristics in its QoS configurations. Even for QoS policies defined in the OMG standard, default values may differ across implementations. Additionally, vendors may introduce proprietary QoS policies not defined in the standard to optimize performance. Accordingly, this article adopted a uniform QoS configuration for all implementations—focusing on core OMG-defined QoS policies (e.g., reliability, durability, history) that directly affect communication performance—as shown in Table 10, to maintain fairness in evaluation. In addition, for vendor-specific QoS policies not based on the standard, artificial tuning was excluded, and the default values provided by each vendor were used unchanged. This approach prevents biased results that may arise from configurations optimized for specific implementations.

Table 10. QoS used in DDS-PerfTester.

QoS	Value	QoS	Value
Reliability	Reliable	Destination Order	By Reception Timestamp
History	Keep All	Latency Budget	0
Durability	Volatile	Liveliness	Automatic
Deadline	Infinite	Ownership	Shared

To derive performance evaluation results, not only were the numerical metrics—throughput and latency—provided by DDS-PerfTester utilized, but visual analysis using Wireshark’s I/O graph was also conducted. Wireshark is an open-source network packet capture and analysis tool that allows the inspection of packet contents or protocol-level analysis by capturing packets transmitted through a specific network interface. The I/O graph feature in Wireshark allows packet or protocol data to be visualized as a time-based graph, assisting with temporal analysis. This enables an intuitive comparison of packet transmission volume over time and is useful in directly or indirectly analyzing and comparing the performance of each DDS implementation [58].

Accordingly, performance evaluations for throughput and latency were conducted for each DDS implementation. Furthermore, performance was measured under various scenarios: the 1Publisher–1Subscriber and 1Publisher–NSubscribers scenarios. In the 1Publisher–1Subscriber scenario, the number of Subscribers was fixed, while the Topic size was varied from 128 bytes to 32,768 bytes to observe changes. In the 1Publisher–NSubscribers scenario, the Topic size was fixed at 32,768 bytes, and the number of Subscribers was increased from two to eight to examine variations. In the 1Publisher–1Subscriber scenario, unicast was employed as the DDS communication mechanism, whereas multicast was used in the 1Publisher–NSubscribers scenario. This allowed for the evaluation of throughput and latency variations exhibited by each DDS implementation under different environments.

Based on the proposed system design, we conducted a series of experiments to evaluate its performance, as described in the following section.

4. Results

4.1. Demonstration of the Proposed System

This section presents a practical example of the implemented system to help readers better understand the operational structure of the proposed architecture. In the detailed scenario, data are acquired from an Outstation Simulator and transformed into CIM-based DDS Topic messages through the FEP Protocol Converter. The generated messages are then published and subsequently subscribed to by other DDS-based applications for use. To enhance clarity, IEC 61850—a SCADA protocol with relatively well-defined semantic structures—was employed in this example.

Figure 15 shows an example of the IEC 61850 CIM mapping process as executed during the operation of the CIM Adaptor. The IEC 61850 Server is configured with a ZLIN LN, where the data point ZLIN\$SP\$LinLenkm\$setMag\$f has a value of 10.25. The following steps are then performed:

1. The IEC 61850 Client within the FEP retrieves the value from the Server via MMS communication and forwards it to the CIM Adaptor.
2. The CIM Adaptor converts the data into a DDS Topic format based on CIM mapping and publishes it via a DDS Publisher.
3. Among the DDS Integration Services, the Database Integration Service (DBIS) subscribes to the Topic and stores the Topic information in a database.

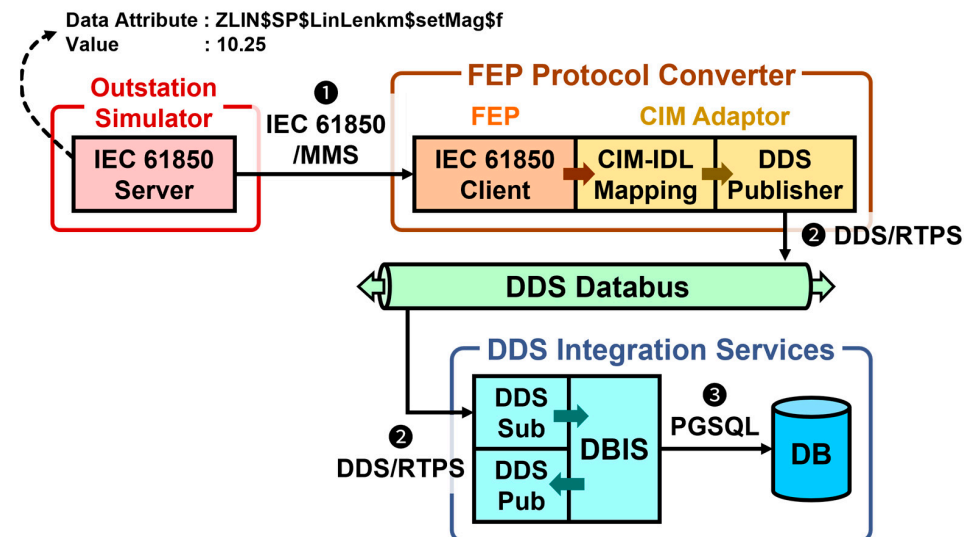


Figure 15. Protocol conversion scenario.

Figure 16 illustrates the packet flow involved when the FEP acquires the ZLIN sub-element values via IEC 61850 MMS communication. At this stage, the value 10.25 is encoded according to the Abstract Syntax Notation One (ASN.1) representation used by MMS and converted to the IEEE754 floating-point standard format, resulting in the value 0x41240000, which is then delivered to the FEP. The FEP collects not only this value but also values of the common attributes shared with the mapped CIM Class, ACLineSegment. It then transmits all this information to the CIM Adaptor via a Unix Domain Socket.

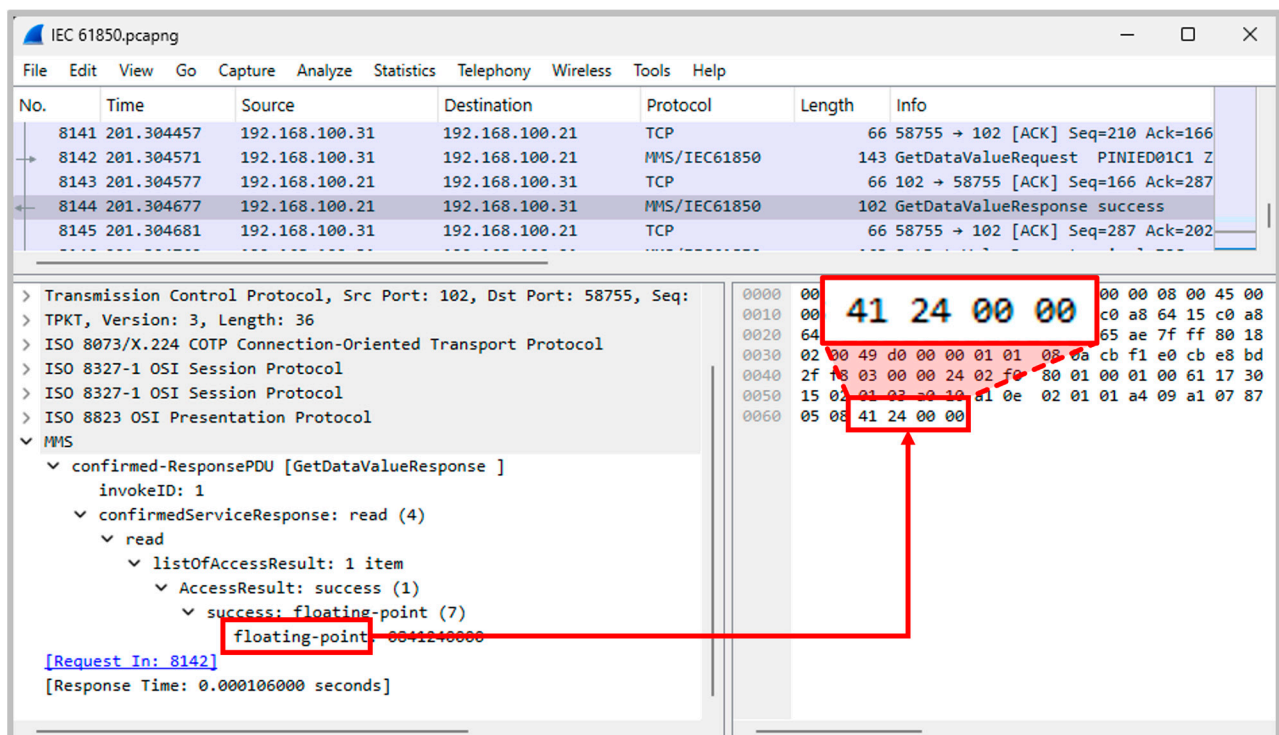


Figure 16. Protocol conversion scenario—① IEC 61850/MMS.

Upon receiving the data, the CIM Adaptor performs IEC 61850 CIM mapping to convert it into a CIM-based DDS IDL structure, which is then published through a DDS Publisher. Figure 17 shows the DDS Publisher publishing the cim-aclinesegment Topic

using the RTPS protocol. While DDS defaults to a big-endian format during message serialization, a little-endian format can be adopted to accommodate specific system requirements and enhance flexibility. Figure 17 illustrates a case where communication is conducted using little-endian formatting; hence, the value is transmitted as 0x00002441, not 0x41240000.

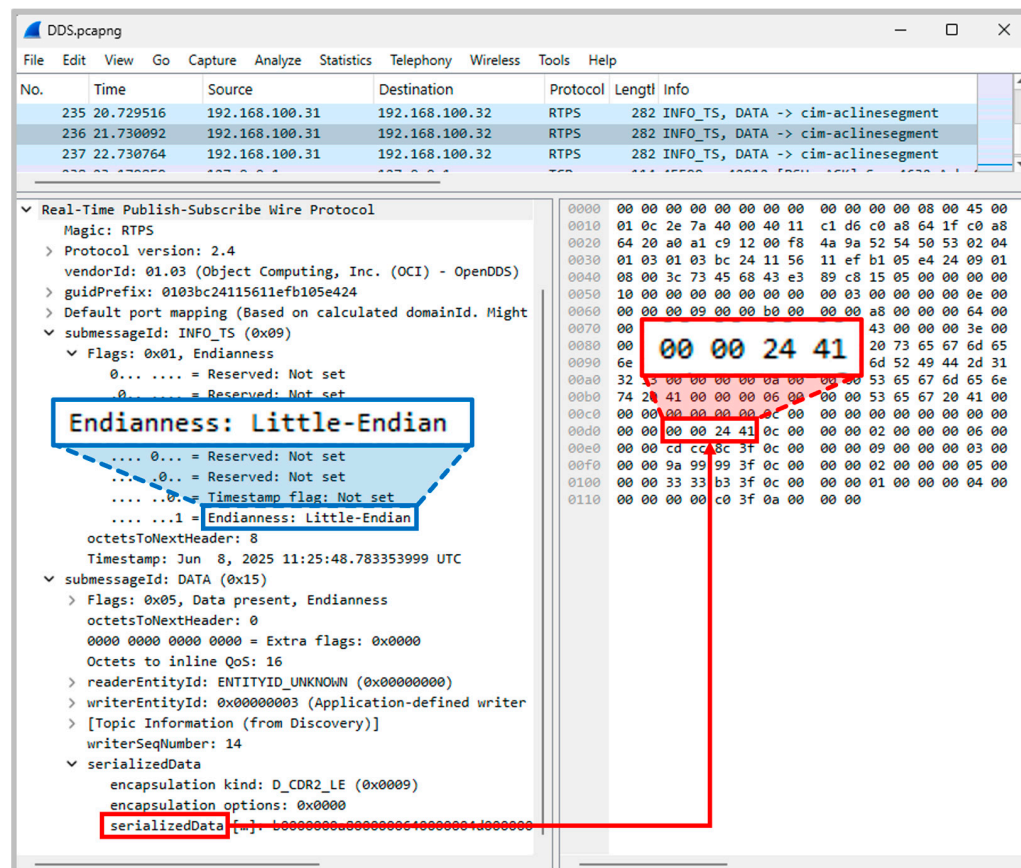


Figure 17. Protocol conversion scenario—**2** DDS/RTPS.

The DBIS serves as a service integrating DDS and databases. It subscribes to designated DDS Topics to store received data in the database, or maps data from the database to a DDS Topic for publication. The DBIS in Figure 15 is designed to store the ACLineSegment Topic data in a PostgreSQL database using Java Database Connectivity (JDBC). PostgreSQL is an open-source object-relational database system that features scalability, support for standard Structured Query Language (SQL), and compatibility with Vector DB. It operates by default on TCP Port 5432 and manages databases via a TCP/IP-based protocol designed by PostgreSQL. Figure 18 illustrates the process of storing ACLineSegment Topic data in PostgreSQL using JDBC. Because the INSERT SQL used to store values in a database is transmitted in text form, the value “10.25” is converted into ASCII codes and delivered as 0x31 (“1”), 0x30 (“0”), 0x2E (“.”), 0x32 (“2”), and 0x35 (“5”).

Figure 19 represents the data stored in the PostgreSQL database via DBIS, where the value 10.25 from the IEC 61850 server's ZLIN\$SP\$LinLenkm\$setMag\$f is mapped and stored as the length.value of the ACLLineSegment.

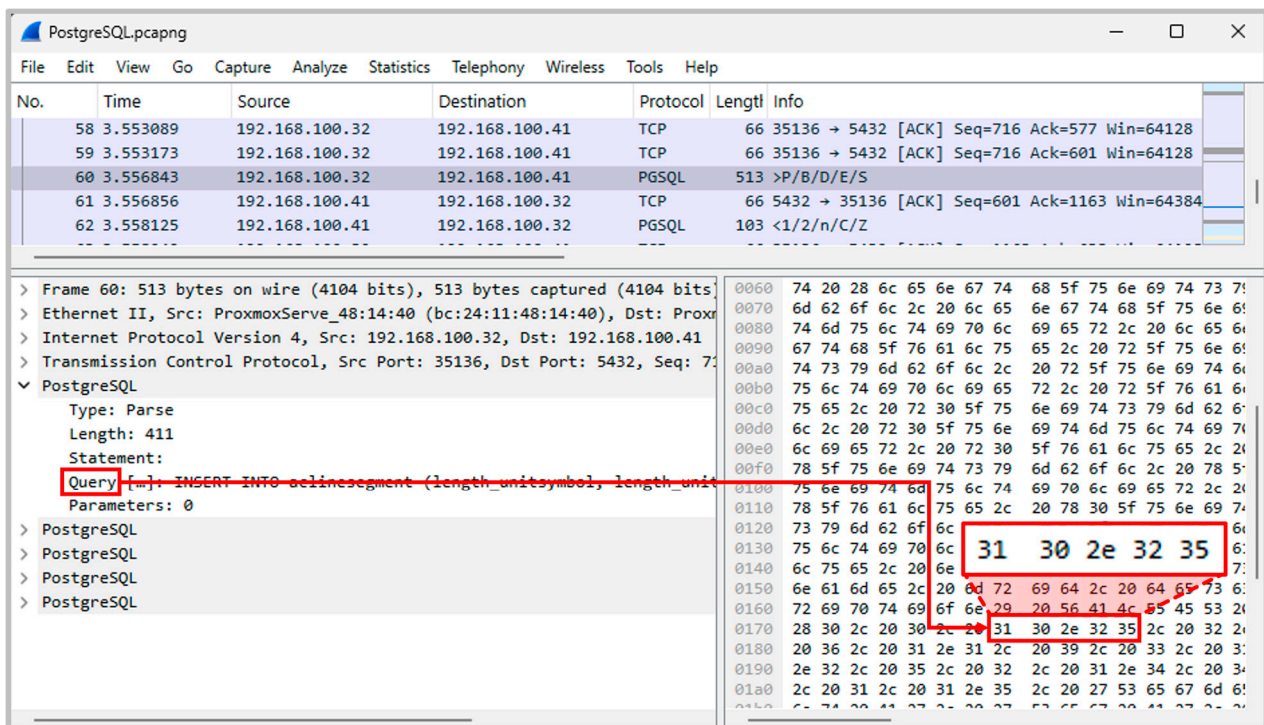


Figure 18. Protocol conversion scenario—③ PGSQL (PostgreSQL).

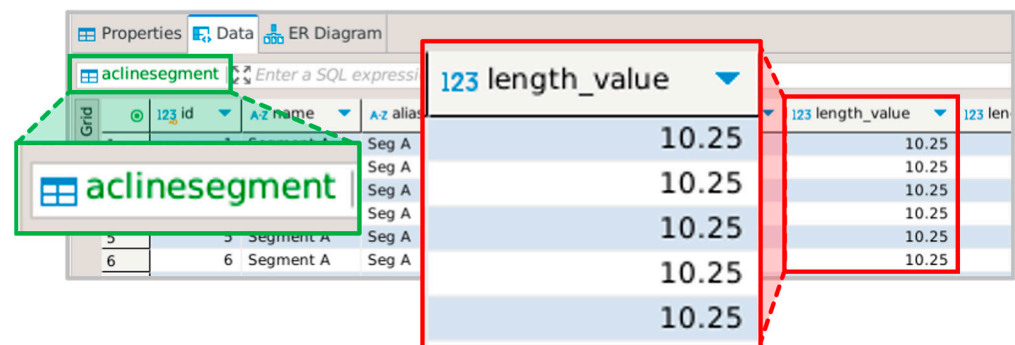


Figure 19. Protocol conversion scenario—database.

4.2. FEP Performance Evaluation

Table 11 presents the performance measurement results obtained by the FEP Protocol Converter for each SCADA protocol. For each protocol, data were collected and mapped to the ACLineSegment class of the CIM. In the experiment, a data packet of 176 bytes was constructed and transmitted 10,000 times using the same structure to evaluate the transmission and conversion processes.

Table 11. Performance test results—throughput and latency for various SCADA protocols.

SCADA Protocols and Corresponding Performance Test Results		
	Throughput (Mb/s)	Latency (μs)
DNP3.0	1064.81	11.42
IEC 61850	1058.12	11.59
Modbus	1071.05	11.38

The measurement results showed that the performance differences among the protocols were within 5%, and the variations observed in repeated experiments were inconsistent.

Therefore, it is difficult to conclude that the conversion process for any specific protocol demonstrated superior performance. Consequently, the performance differences among protocols within the proposed FEP architecture are considered statistically insignificant.

It should be noted that this experiment excluded the actual data acquisition procedures via SCADA protocols. Therefore, in real-world operational environments, performance differences may arise, depending on factors such as the level of optimization in protocol implementation libraries and the frequency of data acquisition.

4.3. DDS Performance Evaluation

Figure 20 shows the throughput when the Topic size is 128 bytes in the 1Publisher–1Subscriber scenario. CycloneDDS achieved 14.5468 bps and RTIDDS 14.1771 bps, both significantly outperforming OpenDDS at 8.7475 bps and FastDDS at 7.3389 bps.

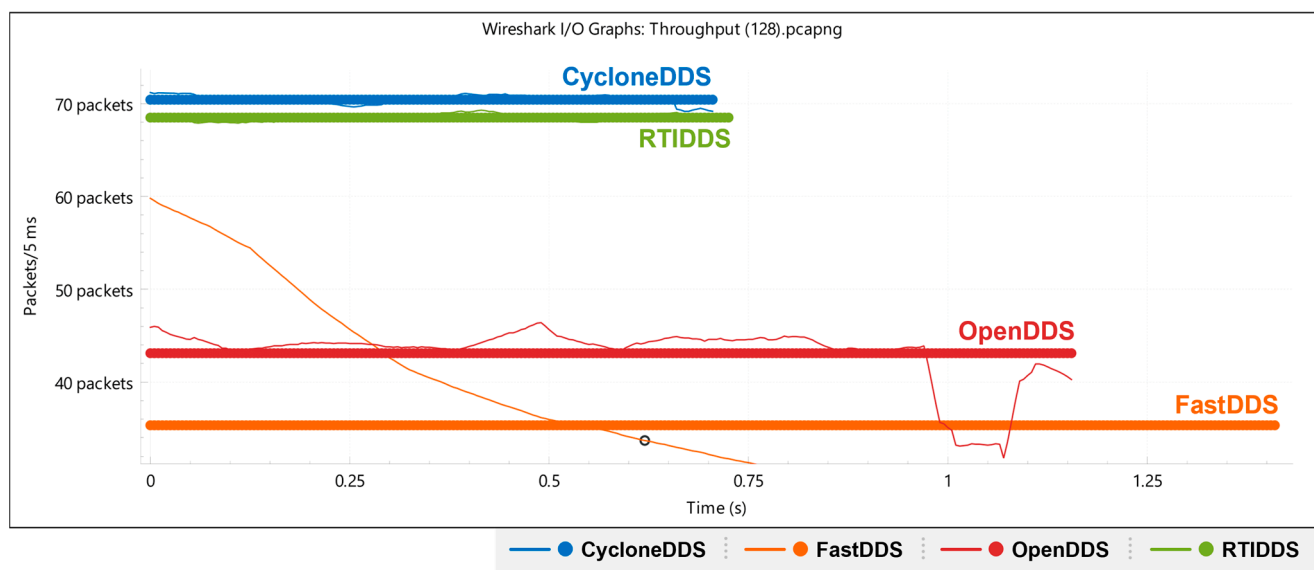


Figure 20. Throughput for 1Publisher–1Subscriber case.

Table 12 shows the throughput in the 1Publisher–1Subscriber scenario across varying Topic sizes. Specifically, when the Topic size was set to 32,768 bytes, RTIDDS showed the highest throughput at 297.8868 bps, with CycloneDDS close behind at 290.8084 bps. FastDDS exhibited the lowest throughput at 146.4058 bps. Compared to the 128-byte Topic case, OpenDDS demonstrated improved performance at 283.9379 bps, becoming more comparable to the top performers, RTIDDS and CycloneDDS. Additionally, all four DDS implementations showed increased throughput as the Topic size grew. This can be attributed to the higher relative overhead of metadata—such as RTPS headers—when the Topic size is small, resulting in greater time spent on encoding and decoding processes and, thus, lower effective throughput. Conversely, for larger Topic sizes, the proportion of payload data increases relative to metadata, causing transmission time over the network interface to dominate, resulting in throughput values approaching the available bandwidth.

Table 13 shows the throughput for the 1Publisher–NSubscriber scenario. In all test cases, the Topic size was fixed at 32,768 bytes. Because multicast was employed as the DDS communication mechanism, one packet can be simultaneously delivered to all Subscribers, eliminating the need for individual transmissions. Consequently, increases in the number of Subscribers had minimal impact on throughput. However, in some cases, throughput declined due to the increased number of Heartbeat and Ack/Nack messages to be processed. For example, FastDDS dropped from 200.8935 bps to 175.4527 bps, and OpenDDS dropped from 281.2498 bps to 278.6184 bps. The throughput rankings among DDS implementations

remained consistent with the 1Publisher–1Subscriber scenario when the Topic size was 32,768 bytes.

Table 12. Performance test results—throughput for various topic sizes.

	Topic Sizes and Corresponding Throughputs				
	128 B (Mb/s)	512 B (Mb/s)	2 KB (Mb/s)	8 KB (Mb/s)	32 KB (Mb/s)
RTIDDS	14.02	56.67	221.59	288.23	297.88
CycloneDDS	14.45	56.49	219.31	282.82	290.80
FastDDS	7.25	28.72	47.57	77.01	146.40
OpenDDS	8.84	33.82	142.58	241.99	283.93

Table 13. Performance test results—throughput for various subscriber counts.

	Subscriber Counts and Corresponding Throughputs		
	1:2 (Mb/s)	1:4 (Mb/s)	1:8 (Mb/s)
RTIDDS	295.24	296.32	297.76
CycloneDDS	290.56	291.18	291.32
FastDDS	200.89	191.15	175.45
OpenDDS	281.24	279.05	278.61

Figure 21 shows the latency for the 1Publisher–1Subscriber scenario when the Topic size is 128 bytes. FastDDS recorded 0.1662 ms, CycloneDDS 0.1689 ms, and RTIDDS 0.1880 ms, while OpenDDS showed the highest latency at 0.3053 ms, indicating relatively lower performance.

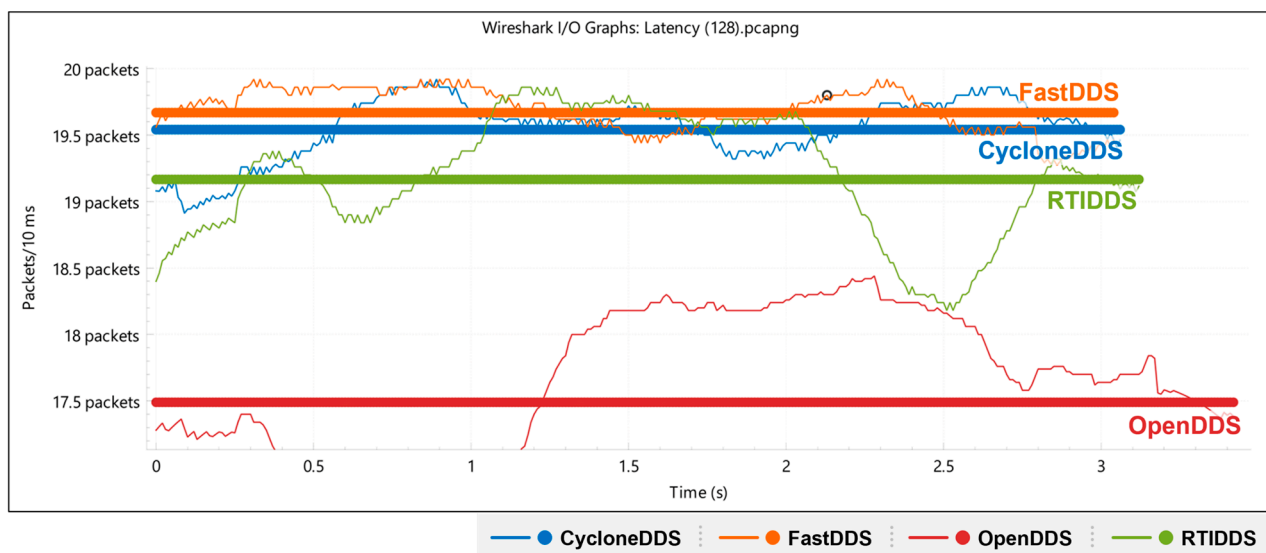


Figure 21. Latency for 1Publisher–1Subscriber case.

Table 14 expands on the above analysis by presenting latency measurements based on varying topic sizes in the 1Publisher–1Subscriber scenario. Notably, when the topic size is 32,768 bytes, FastDDS recorded a latency of 1.0448 ms, CycloneDDS 1.0542 ms, and RTIDDS 1.0861 ms, while OpenDDS remained to show low performance at 1.2240 ms. Compared to the case where the topic size was 128 bytes, as the topic size increased, the size of metadata per topic remained constant; however, the overall packet size increased. Consequently, as the Maximum Transmission Unit (MTU)—which denotes the maximum

frame size that can be transmitted over the network—is smaller than the topic size, the data must be fragmented into multiple packets. This fragmentation increases the time required to traverse the network interface, resulting in higher latency.

Table 14. Performance test results—latency for various topic sizes.

	Topic Sizes and Corresponding Latencies				
	128 B (ms)	512 B (ms)	2 KB (ms)	8 KB (ms)	32 KB (ms)
RTIDDS	0.1880	0.1802	0.2204	0.3845	1.0861
CycloneDDS	0.1689	0.1779	0.1953	0.3680	1.0542
FastDDS	0.1662	0.1777	0.1852	0.3661	1.0448
OpenDDS	0.3053	0.3138	0.3054	0.4725	1.2240

Table 15 presents the latency in the 1Publisher–NSubscribers scenario. As with the throughput measurements, the topic size was set to 32,768 bytes across all cases. Due to the utilization of multicast as the DDS communication mechanism, latency performance aligns with that observed in the 1Publisher–1Subscriber scenario. However, as the number of subscribers increased from two to eight, FastDDS and OpenDDS, which had shown reduced throughput performance, also exhibited degraded latency performance, while FastDDS showed low latency performance compared to CycloneDDS and RTIDDS.

Table 15. Performance test results—latency for various subscriber counts.

	Subscriber Counts and Corresponding Latencies		
	1:2 (ms)	1:4 (ms)	1:8 (ms)
RTIDDS	1.1218	1.1348	1.1685
CycloneDDS	1.1198	1.1183	1.1411
FastDDS	1.1035	1.1187	1.2135
OpenDDS	1.1919	1.2715	1.3157

The results obtained from the performance evaluation offer several insights, which are discussed in the next section to interpret their implications and limitations.

5. Discussion

Recent studies have actively explored DDS-based communication architecture with Kubernetes to improve scalability, flexibility, and manageability in smart grid systems [37,38]. These efforts have shown that container orchestration platforms can effectively deploy and manage DDS-based applications under various network configurations, offering enhanced service continuity and resource efficiency [41,42]. In parallel, research on cloud-native architecture for core power grid systems—such as EMS—has demonstrated the feasibility of achieving high availability and fault tolerance through mechanisms like multi-cluster redundancy, service-level failover, and dynamic resource optimization [9,43,44].

However, most of these prior works have focused on either the individual benefits of DDS and Kubernetes or proposed high-availability architecture using microservices. And, while some studies, such as [45], have examined gateway mechanisms between DDS and other publish–subscribe messaging protocols, few have addressed real-time protocol conversion between legacy SCADA protocols and DDS. Moreover, the integration of such gateways into Kubernetes-based microservice frameworks remains underexplored.

Therefore, this article proposes and implements a Kubernetes-based FEP that enables real-time data integration between conventional SCADA systems and DDS-based platforms. The proposed system collects data from legacy industrial protocols, such as DNP3.0,

IEC 61850, and Modbus, transforms it according to the CIM, and publishes the result through DDS, thereby enabling interoperability across multiple protocols required in smart grid environments.

In addition, virtualization based on Proxmox VE and Kubernetes enabled the system to be decoupled from physical platforms. This approach provides the flexibility to run the system consistently across diverse environments and offers the scalability required for integration with various systems and services.

The performance evaluation of the FEP demonstrated that, despite each message being only 176 bytes—smaller than the data size used in the DDS performance tests—the FEP consistently achieved higher throughput than all tested DDS implementations. This confirms that the FEP itself does not act as a performance bottleneck. In terms of latency, delays were measured in the microsecond range, indicating that the overhead introduced by protocol conversion and data transmission does not compromise real-time performance. These results suggest that the overall real-time capability of the system may depend on the performance of the selected DDS implementation, highlighting the importance of choosing a high-performance DDS solution.

The DDS performance evaluation showed that RTI Connex DDS and Cyclone DDS exhibited the highest throughput, while Fast DDS, Cyclone DDS, and RTI DDS demonstrated the lowest latency in that order. Notably, Cyclone DDS, an open-source implementation provided by the Eclipse Foundation, delivered performance comparable to that of the commercial RTI Connex DDS, demonstrating its viability as a practical open-source alternative. Furthermore, the results confirmed that the proposed system architecture enables efficient data exchange between system components.

Nevertheless, several limitations remain. Since the objective of this experiment was to measure the theoretical maximum performance of the proposed system, it did not account for factors commonly encountered in actual smart grid environments, such as complex data acquisition cycles, network jitter, and concurrent connections from multiple devices. Although the use of a Kubernetes architecture provides scalability and stability, this article did not include a quantitative evaluation of system behavior under failure conditions or the resulting performance degradation.

Future research will focus on establishing mapping rules for a broader range of CIM Classes to enhance semantic interoperability, as well as validating the proposed system under large-scale power grid scenarios that simulate real-world operating environments. In addition, a comparative analysis with alternative messaging middleware solutions such as MQTT and AMQP will be conducted to gain insights into the most suitable communication models for different deployment contexts.

Based on these results and implications, the contributions of this article are summarized, and directions for future research are outlined in the conclusion.

6. Conclusions

With the growing complexity of modern industrial environments and the advancement of IIoT technologies, existing FEPs are increasingly exhibiting limitations in various aspects. These challenges highlight the need for novel approaches to data communication and system architecture in smart grid environments.

Traditional FEPs are not well suited for the increasingly large-scale and distributed nature of modern industrial environments and smart grid architectures. Moreover, existing solutions are generally difficult to adapt to cloud computing environments and struggle to ensure high availability or rapid recovery in the event of system failures.

To overcome these limitations, this article proposes a real-time FEP based on the Kubernetes platform and designs and implements an efficient conversion mechanism

between existing SCADA protocols (DNP3.0, IEC 61850, Modbus) and the DDS protocol. The proposed FEP enables semantic-level data mapping and conversion between two systems operating under different communication paradigms, while also ensuring high availability and scalability by leveraging Kubernetes' auto-scaling and self-healing features.

Furthermore, the performance evaluation demonstrated that the proposed FEP achieves high throughput and sub-millisecond latency, indicating that it does not introduce any bottlenecks in the overall protocol conversion process. Among the evaluated DDS implementations, RTI Connex DDS and Cyclone DDS exhibited superior performance characteristics. The findings of this article are expected to promote the transformation of industrial automation systems into cloud-based architectures and contribute to the modernization of SCADA systems to meet the demands of future industrial environments.

Despite these results, this article does not provide a quantitative evaluation of the proposed system's performance under large-scale communication scenarios typical of real-world smart grid environments. Additionally, the advantages of the Kubernetes architecture in terms of fault tolerance and system stability were not demonstrated through quantitative results.

A notable research gap exists in the integration of future-proof security mechanisms, especially emerging threats such as quantum computing, and Advanced Persistent Threats. There needs to be a consideration of critical security factors within industrial control systems.

To enhance the proposed FEP for real-world deployment, future research will focus on the following:

1. **Performance and Stability Evaluation Under Large-Scale Scenarios**
In real-world smart grid environments, numerous devices operate with varying communication cycles, and adverse conditions such as system failures and network jitter frequently occur. Future research will therefore aim to comprehensively evaluate the performance and stability of the cloud-based FEP under such large-scale and realistic deployment scenarios.
2. **Enhancement of Security Functions Using Post-Quantum Cryptography (PQC)**
Most of the cryptographic algorithms currently in use are vulnerable to the advances of quantum computing. As such, future work will focus on integrating quantum-resistant security mechanisms, such as PQC, to enhance the security of communications between SCADA systems and DDS-based platforms.
3. **Development of Security Evaluation Scenarios Based on the MITRE ATT&CK Framework**
To proactively identify and mitigate security vulnerabilities, future research will develop diverse attack scenarios based on the MITRE ATT&CK framework. These scenarios will be used to conduct security assessments and evaluate the defensive capabilities of the proposed FEP. This approach is expected to contribute to the systematic strengthening of the platform's resilience against cyber threats.

Through these future research directions, the proposed FEP system is expected to evolve into an advanced cloud-based smart grid solution that not only enhances performance but also strengthens security and reliability.

Author Contributions: Conceptualization, T.K. and H.K.; methodology, T.K. and S.C.; software, S.C. and H.K.; validation, T.K. and J.K.; formal analysis, S.C.; investigation, T.K. and H.K.; resources, J.K.; data curation, T.K.; writing—original draft preparation, T.K.; writing—review and editing, H.K. and B.S.; visualization, Y.K.; supervision, B.S.; project administration, H.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this article are available on request from the corresponding author. The data are not publicly available due to internal institutional policy restrictions.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Lee, J.-K.; Lee, S.-Y.; Kim, T.-H.; Ham, K.-S. Development of Unified SCADA System Based on IEC61850 in Wave-Offshore Wind Hybrid Power Generation System. *Trans. Korean Inst. Electr. Eng.* **2016**, *65*, 811–818. [CrossRef]
2. Horalek, J.; Matyska, J.; Sobeslav, V. Communication Protocols in Substation Automation and IEC 61850 Based Proposal. In Proceedings of the 2013 IEEE 14th International Symposium on Computational Intelligence and Informatics (CINTI), Budapest, Hungary, 19–21 November 2013; pp. 321–326.
3. Mughaid, A.; Alzu'bi, S.; Alkhatib, A.A.; AlZioud, A.; Al Ghazo, A.; AL-Aiash, I. Simulation-Based Framework for Authenticating SCADA Systems and Cyber Threat Security in Edge-Based Autonomous Environments. *Simul. Model. Pract. Theory* **2025**, *140*, 103078. [CrossRef]
4. Pliatsios, D.; Sarigiannidis, P.; Lagkas, T.; Sarigiannidis, A.G. A Survey on SCADA Systems: Secure Protocols, Incidents, Threats and Tactics. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 1942–1976. [CrossRef]
5. Folgado, F.J.; Calderón, D.; González, I.; Calderón, A.J. Review of Industry 4.0 from the Perspective of Automation and Supervision Systems: Definitions, Architectures and Recent Trends. *Electronics* **2024**, *13*, 782. [CrossRef]
6. Sverko, M.; Grbac, T.G.; Mikuc, M. Scada Systems with Focus on Continuous Manufacturing and Steel Industry: A Survey on Architectures, Standards, Challenges and Industry 5.0. *IEEE Access* **2022**, *10*, 109395–109430. [CrossRef]
7. Santodomingo, R.; Uslar, M.; Göring, A.; Gottschalk, M.; Nordström, L.; Saleem, A.; Chenine, M. SGAM-Based Methodology to Analyse Smart Grid Solutions in DISCERN European Research Project. In Proceedings of the 2014 IEEE International Energy Conference (ENERGYCON), Cavtat, Croatia, 13–16 May 2014; pp. 751–758.
8. Yang, C.-T.; Chen, W.-S.; Huang, K.-L.; Liu, J.-C.; Hsu, W.-H.; Hsu, C.-H. Implementation of Smart Power Management and Service System on Cloud Computing. In Proceedings of the 2012 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing, Fukuoka, Japan, 4–7 September 2012; pp. 924–929.
9. Lyu, Z.; Wei, H.; Bai, X.; Lian, C. Microservice-Based Architecture for an Energy Management System. *IEEE Syst. J.* **2020**, *14*, 5061–5072. [CrossRef]
10. Kang, Z.; An, K.; Gokhale, A.; Pazandak, P. Evaluating Performance of OMG DDS in Kubernetes Container Deployment (Industry Track). In Proceedings of the 21st ACM/IFIP International Middleware Conference (Middleware'20), Delft, The Netherlands, 7–11 December 2020.
11. Pu, C.; Ding, X.; Wang, P.; Xie, S.; Chen, J. Semantic Interconnection Scheme for Industrial Wireless Sensor Networks and Industrial Internet with Opc Ua Pub/Sub. *Sensors* **2022**, *22*, 7762. [CrossRef] [PubMed]
12. Sekigawa, S.; Sasaki, C.; Tagami, A. Toward a Cloud-Native Telecom Infrastructure: Analysis and Evaluations of Kubernetes Networking. In Proceedings of the 2022 IEEE Globecom Workshops (GC Wkshps), Rio de Janeiro, Brazil, 4–8 December 2022; pp. 838–843.
13. Kapočius, N. Performance Studies of Kubernetes Network Solutions. In Proceedings of the 2020 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream), Vilnius, Lithuania, 30 April 2020; pp. 1–6.
14. Church, P.; Mueller, H.; Ryan, C.; Gogouvitis, S.V.; Goscinski, A.; Haitof, H.; Tari, Z. SCADA Systems in the Cloud. In *Handbook of Big Data Technologies*; Zomaya, A.Y., Sakr, S., Eds.; Springer International Publishing: Cham, Switzerland, 2017; pp. 691–718. ISBN 978-3-319-49339-8.
15. Paavola, E. Managing Multiple Applications on Kubernetes Using GitOps Principles. Bachelor's Thesis, Metropolia University of Applied Sciences, Helsinki, Finland, 2021.
16. Kubernetes. Available online: <https://kubernetes.io/> (accessed on 7 April 2025).
17. Rostami, G. Role-Based Access Control (Rbac) Authorization in Kubernetes. *J. ICT Stand.* **2023**, *11*, 237–260. [CrossRef]
18. Ali, A.; Imran, M.; Kuznetsov, V.; Trigazis, S.; Pervaiz, A.; Pfeiffer, A.; Mascheroni, M. Implementation of New Security Features in CMSWEB Kubernetes Cluster at CERN. *EPJ Web Conf.* **2024**, *295*, 07026. [CrossRef]
19. Kubernetes Components. Available online: <https://kubernetes.io/docs/concepts/overview/components/> (accessed on 27 May 2025).
20. OMG Data Distribution Service Version 1.4. Available online: <https://www.omg.org/spec/DDS/1.4/PDF> (accessed on 27 May 2025).
21. Interface Definition Language Version 4.2. Available online: <https://www.omg.org/spec/IDL/4.2/PDF> (accessed on 27 May 2025).
22. What Is DDS? Available online: <https://www.dds-foundation.org/what-is-dds-3/> (accessed on 27 May 2025).

23. DDS Security Version 1.2. Available online: <https://www.omg.org/spec/DDS-SECURITY/1.2/PDF> (accessed on 27 May 2025).
24. I/A Series® Intelligent SCADA SCADA Platform. Available online: <https://paresource.schneider-electric.com/iaseries/pss/21s2/21s2m1b3.pdf> (accessed on 27 May 2025).
25. Maria, A. GGSN Front End Processor (GFEP) System for SCADA Inter-Domain Communications. US8527653B2, 3 September 2013.
26. *IEEE Std 1815-2012*; IEEE Standard for Electric Power Systems Communications: Distributed Network Protocol (DNP3). IEEE Standards Association: Piscataway, NJ, USA, 2012.
27. *IEC 61850*; Communication Networks and Systems for Power Utility Automation. International Electrotechnical Commission: Geneva, Switzerland, 2013.
28. MODBUS Application Protocol Specification V1.1b3. Available online: https://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf (accessed on 27 May 2025).
29. MODBUS over Serial Line Specification and Implementation Guide v1.02. Available online: https://modbus.org/docs/Modbus_over_serial_line_V1_02.pdf (accessed on 27 May 2025).
30. MODBUS Messaging on TCP/IP Implementation Guide V1.0b. Available online: https://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf (accessed on 27 May 2025).
31. Uslar, M.; Specht, M.; Rohjans, S.; Trefke, J.; González, J.M. *The Common Information Model CIM: IEC 61968/61970 and 62325-A Practical Introduction to the CIM*; Springer Science & Business Media: Berlin, Germany, 2012.
32. Schumilin, A.; Duepmeier, C.; Stucky, K.-U.; Hagenmeyer, V. A Consistent View of the Smart Grid: Bridging the Gap between IEC CIM and IEC 61850. In Proceedings of the 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Prague, Czech, 29–31 August 2018; pp. 321–325.
33. Specht, M.; Rohjans, S. ICT and Energy Supply: IEC 61970/61968 Common Information Model. In *Standardization in Smart Grids; Power Systems*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 99–114. ISBN 978-3-642-34915-7.
34. Cim: ACLineSegment. Available online: https://ontology.tno.nl/IEC_CIM/cim_ACLineSegment.html (accessed on 27 May 2025).
35. Mohammadabadi, S.M.S.; Entezami, M.; Moghaddam, A.K.; Orangian, M.; Nejadshamsi, S. Generative Artificial Intelligence for Distributed Learning to Enhance Smart Grid Communication. *Int. J. Intell. Netw.* **2024**, *5*, 267–274. [\[CrossRef\]](#)
36. Sauter, T.; Lobashov, M. End-to-End Communication Architecture for Smart Grids. *IEEE Trans. Ind. Electron.* **2010**, *58*, 1218–1228. [\[CrossRef\]](#)
37. Ferreira, R.D.F.; De Oliveira, R.S. Cloud IEC 61850: DDS Performance in Virtualized Environment with Opendds. In Proceedings of the 2017 IEEE International Conference on Computer and Information Technology (CIT), Helsinki, Finland, 21–23 August 2017; pp. 231–236.
38. Youssef, T.A.; Elsayed, A.T.; Mohammed, O.A. DDS Based Interoperability Framework for Smart Grid Testbed Infrastructure. In Proceedings of the 2015 IEEE 15th International Conference on Environment and Electrical Engineering (EEEIC), Rome, Italy, 10–13 June 2015; pp. 219–224.
39. Ho, M.-H.; Yen, H.-C.; Lai, M.-Y.; Liu, Y.-T. Implementation of Dds Cloud Platform for Real-Time Data Acquisition of Sensors. In Proceedings of the 2021 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS), Hualien City, Taiwan, 16–19 November 2021; pp. 1–2.
40. Ho, M.-H.; Lai, M.-Y.; Liu, Y.-T. Implementation of DDS Cloud Platform for Real-Time Data Acquisition of Sensors for a Legacy Machine. *Electronics* **2022**, *11*, 2096. [\[CrossRef\]](#)
41. Zu, X.; Bai, Y.; Yao, X. Data-Centric Publish-Subscribe Approach for Distributed Complex Event Processing Deployment in Smart Grid Internet of Things. In Proceedings of the 2016 7th IEEE International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 26–28 August 2016; pp. 710–713.
42. Amin, M.B.; Khan, W.A.; Awan, A.A.; Lee, S. Intercloud Message Exchange Middleware. In Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication, Kuala Lumpur Malaysia, 20 February 2012; pp. 1–7.
43. Ramasamy, B.; Na, Y.; Kim, W.; Chea, K.; Kim, J. Hacm: High Availability Control Method in Container-Based Microservice Applications over Multiple Clusters. *IEEE Access* **2022**, *11*, 3461–3471. [\[CrossRef\]](#)
44. Li, Z.; Wei, H.; Lyu, Z.; Lian, C. Kubernetes-Container-Cluster-Based Architecture for an Energy Management System. *IEEE Access* **2021**, *9*, 84596–84604. [\[CrossRef\]](#)
45. Ioana, A.; Korodi, A. DDS and OPC UA Protocol Coexistence Solution in Real-Time and Industry 4.0 Context Using Non-Ideal Infrastructure. *Sensors* **2021**, *21*, 7760. [\[CrossRef\]](#) [\[PubMed\]](#)
46. Proxmox Virtual Environment. Available online: <https://www.proxmox.com/en/products/proxmox-virtual-environment/overview> (accessed on 27 May 2025).
47. Netto, H.; Pereira Oliveira, C.; Rech, L.D.O.; Alchieri, E. Incorporating the Raft Consensus Protocol in Containers Managed by Kubernetes: An Evaluation. *Int. J. Parallel Emergent Distrib. Syst.* **2020**, *35*, 433–453. [\[CrossRef\]](#)

48. Han, M.; Yao, D.G.; Yu, X.L. A Solution for Instant Response of Cloud Platform Based on Nginx+ Keepalived. In Proceedings of the International Conference on Computer Science, Communications and Multimedia Engineering, Beijing, China, 22–23 September 2019; pp. 24–25.
49. Options for Highly Available Topology. Available online: <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/ha-topology/> (accessed on 27 May 2025).
50. Wright, K.; Gopalan, K.; Kang, H. *Performance Analysis of Various Mechanisms for Inter-Process Communication*; Operating Systems and Networks Lab, Department of Computer Science, Binghamton University: Binghamton, NY, USA, 2007.
51. Dnp3/Opendnp3. Available online: <https://github.com/dnp3/opendnp3> (accessed on 27 May 2025).
52. Mz-Automation/Libiec61850. Available online: <https://github.com/mz-automation/libiec61850> (accessed on 27 May 2025).
53. Raimbault, S. Stephane/Libmodbus. Available online: <https://github.com/stephane/libmodbus> (accessed on 27 May 2025).
54. RTI Products. Available online: <https://www.rti.com/products> (accessed on 27 May 2025).
55. Eclipse-Cyclonedds/Cyclonedds. Available online: <https://github.com/eclipse-cyclonedds/cyclonedds> (accessed on 27 May 2025).
56. eProsima/Fast-DDS. Available online: <https://github.com/eProsima/Fast-DDS> (accessed on 27 May 2025).
57. OpenDDS/OpenDDS. Available online: <https://github.com/OpenDDS/OpenDDS> (accessed on 27 May 2025).
58. Tuli, R. Analyzing Network Performance Parameters Using Wireshark. *Int. J. Netw. Secur. Appl.* **2023**, *15*, 1–13. [CrossRef]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.