electronics

MDPI

Article

Data Distribution Service Converter Based on the Open Platform Communications Unified Architecture Publish–Subscribe Protocol

Woongbin Sim [1], ByungKwen Song [1], Junho Shin [2] and Taehun Kim [3,*]

[1] Department of Electronics and Computer Engineering, SeoKyeong University, Seoul 02713, Korea; swb1015@skuniv.ac.kr (W.S.); bksong@skuniv.ac.kr (B.S.)
[2] Smart Manufacturing Research Center, Korea Electronics Technology Institute, Seongnam-si 13509, Korea; jhshin@keti.re.kr
[3] School of Cybersecurity, Korea University, Seoul 02841, Korea
* Correspondence: kurie@korea.ac.kr; Tel.: +82-10-9029-5117

Abstract: The open platform communications unified architecture (OPC UA) is a major industry-standard middleware based on the request–reply pattern, and the data distribution service (DDS) is an industry standard in the publish–subscribe form. The OPC UA cannot replace fieldbuses at the control and field levels. To facilitate real-time connectionless operation, the OPC Foundation added the publish–subscribe model—a new specification that supports broker functions, such as message queuing telemetry transport (MQTT), and advanced message queuing protocol (AMQP)—to the OPC UA Part 14 standard. This paper proposes a protocol converter for incorporation into the application layer of the DDS subscriber to facilitate interoperability among publisher–subscriber pairs. The proposed converter comprises a DDS gateway and bridge. The former exists inside the MQTT and AMQP brokers, which convert OPC UA publisher data into DDS messages prior to passing them on to the DDS subscriber. The DDS bridge passes the messages received from the DDS gateway to the OPC UA subscriber in the corresponding DDS application layer. The results reported in existing studies, and those obtained using the proposed converter, allow all devices supporting the OPC UA and OPC UA PubSub standards to realize DDS publish–subscribe interoperability.

Keywords: data distribution service; middleware; open platform communications unified architecture (OPC UA); protocol converter; publish–subscribe protocol

Citation: Sim, W.; Song, B.; Shin, J.; Kim, T. Data Distribution Service Converter Based on the Open Platform Communications Unified Architecture Publish–Subscribe Protocol. *Electronics* **2021**, *10*, 2524. https://doi.org/10.3390/electronics10202524

Academic Editor: In Lee

Received: 21 August 2021
Accepted: 11 October 2021
Published: 16 October 2021

1. Introduction

The data exchange process between modern industrial systems has attracted increased attention in recent years. Industrial automation and control systems include intelligent devices operating in conjunction with the Industrial Internet of Things (IIoT) while utilizing a variety of middleware for data exchange, such as message queuing telemetry transport (MQTT), advanced message queuing protocol (AMQP), HTTP, and data distribution service (DDS). These middleware have different information models and communication protocols, which makes it difficult for them to communicate directly with each other. For example, the open platform communications unified architecture (OPC UA) is a client–server (request–reply) architecture while MQTT, AMQP, and DDS are based on the publish–subscribe model. Devices use the MQTT or AMQP links and publish topic-related data through brokers, and data consumers or subscribers connect to brokers to subscribe to the desired topic data. However, DDS does not use brokers.

The OPC UA is a de facto standard communication middleware technology used in the upper layer of the automation pyramid and is one of the most promising industry automation communication protocols for Industry 4.0 [1]. It is used for data exchange between control systems and enterprise-level automation devices through connection-oriented communication based on a service-oriented architecture [2].

At the current field level, Ethernet-based fieldbus protocols (e.g., SERCOS, ProfiNET, CAN) are used to achieve high performance. However, because these protocols have different operating characteristics, interoperability problems abound. If different types of protocols are used at the field level, a man–machine interface is required for each protocol owing to their different characteristics. In the overall system integration, it is also necessary to integrate multiple types of gateways, which provide the protocol interconversion function, and all automation levels into the same protocol. To satisfy these requirements, the OPC Foundation pursues real-time connectionless mechanisms and has added the OPC UA publish–subscribe model (hereinafter PubSub), which is a new specification that supports the broker function used by MQTT and AMQP, to the OPC UA Part 14 standard [3]. This paper proposes a protocol converter that allows interoperability between OPC UA publishers and subscribers present in the application layer of DDS subscribers. The proposed protocol converter consists of a DDS gateway and a DDS bridge. The DDS gateway exists inside the MQTT and AMQP brokers. It converts the publisher data into DDS messages and sends them to the DDS subscriber. The DDS bridge sends the DDS messages transmitted to the DDS subscriber to the OPC UA subscriber located in the application layer. To verify the implementation accuracy and usability of the proposed protocol converter, a testbed is built and the scenario-based throughput and latency are measured for analysis.

The OPC UA PubSub module used here is based on the Open62541 PubSub [4] open source and uses A-Open62541 PubSub [5], which implements the OPC UA Part 14 specification without the security key service. The MQTT supported in A-Open62541 PubSub does not use MQTT-C [6] applied in Open62541 PubSub but instead uses Mosquitto [7], which is commonly used as an industry standard. While Open62541 PubSub does not support AMQP, A-Open62541 PubSub supports two models, RabbitMQ [8] and Qpid-Proton [9], provided by Apache. OpenDDS open source is used for the DDS [10]. The DDS gateway module and broker run on Raspberry Pi 4, whereas A-Open62541 PubSub and DDS subscribers run on Ubuntu-based virtual machines.

The remainder of this paper is organized as follows. Section 2 reviews the background and studies related to the proposed technique. Section 3 presents the proposed OPC UA publish–subscribe protocol converter model. Section 4 summarizes the implementation and performance analysis results of the proposed model. Finally, Section 5 presents our conclusions and future research directions.

2. Background and Related Work

2.1. Background

2.1.1. OPC UA PubSub

OPC UA PubSub is a standard added as OPC UA Part 14 [3]. Figure 1 illustrates the overall OPC UA protocol stack. TCP/IP, UDP/IP, and the time-sensitive network (TSN) are at the bottom and MQTT, AMQP, and the unified architecture datagram packet (UADP) are configured on it; based on this, the PubSub (OPC UA Part 14) model exists.

The current OPC UA standard has been extended to include OPC UA PubSub, which was added as Part 14 on top of the OPC UA based on the existing client–server communication model. The client–server communication model forms a new session whenever additional clients are connected to a server, resulting in performance problems if the server is linked to numerous industrial devices.

Figure 2 shows three OPC UA clients connected to one OPC UA server in a state where each client has made a subscription request to the nodes in "AddressSpace" of the server via the "MonitoredItem" storage space for collecting information. Although the clients no longer request other services and only receive notifications on the nodes they have already subscribed to, the server must maintain a session with each client to perform the "Subscription" service. As communication parties, the server and clients are forced to perform transmission-related functions individually during each session—such as transmission buffering, acknowledgment of reception, and data resending—which consumes a

significant amount of communication resources in proportion to the number of sessions. Therefore, the number of clients simultaneously connected or the number of "MonitoredItem/Subscription" needs to be capped for servers with limited communications resources, such as hardware specifications.

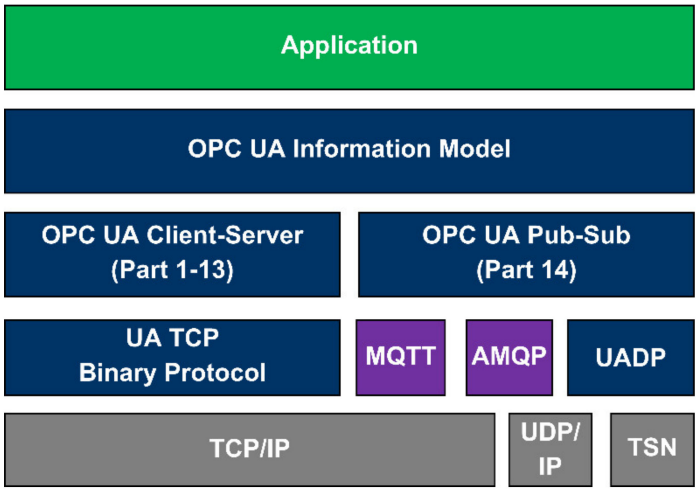

Figure 1. The open platform communications unified architecture (OPC UA) protocol stack. MQTT, message queuing telemetry transport; AMQP, advanced message queuing protocol; TSN, time-sensitive network.

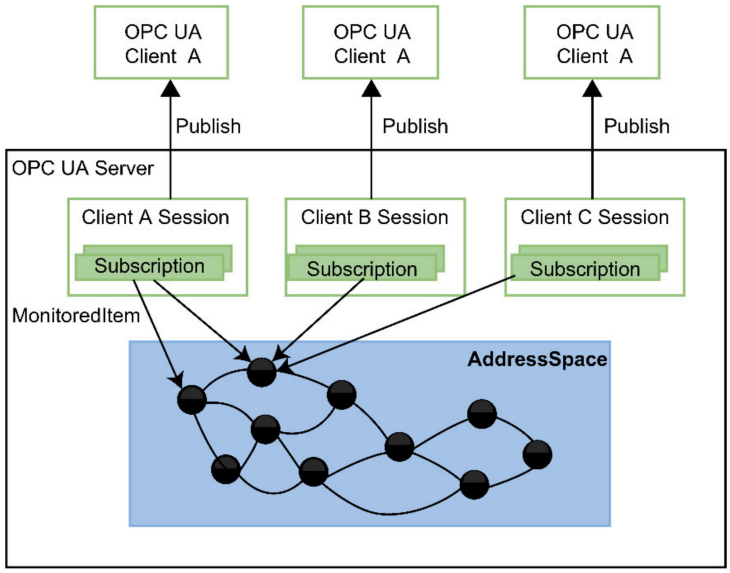

Figure 2. OPC UA PubSub communication model [3].

OPC UA PubSub is a protocol that overcomes this shortcoming. PubSub proposes two communication models: UDP/IP multicast and a broker method using MQTT and AMQP. As the client–server is not directly connected in PubSub, but instead is connected through a broker or IP multicast, an increase in the number of connections has no impact on the communications resources of the server or publisher. Additionally, the protocol stack of the publisher is lightweight and can be applied at the low-specification field level of 16/32 bits. Thus, the addition of the publish–subscribe function to the existing OPC UA allows the OPC UA-based integration of the communication protocols across all system levels, ranging from the field level to the high level. Consequently, an environment, in which all factory automation facilities can be linked to OPC UA and OPC UA PubSub, is created and operates as a single entity.

Figure 3 illustrates the broker communication model of PubSub. MQTT and AMQP are adopted by the OPC UA Part 14 standard as broker communication methods, or IP multicast is used instead of brokers. The transmission to MQTT and AMQP occurs in the forms of "mqtts://<domain name>[:<port>][/<path" and "amqps://<domain name>[:<port>][/<path," and the default port numbers are 8883 and 5671, respectively.

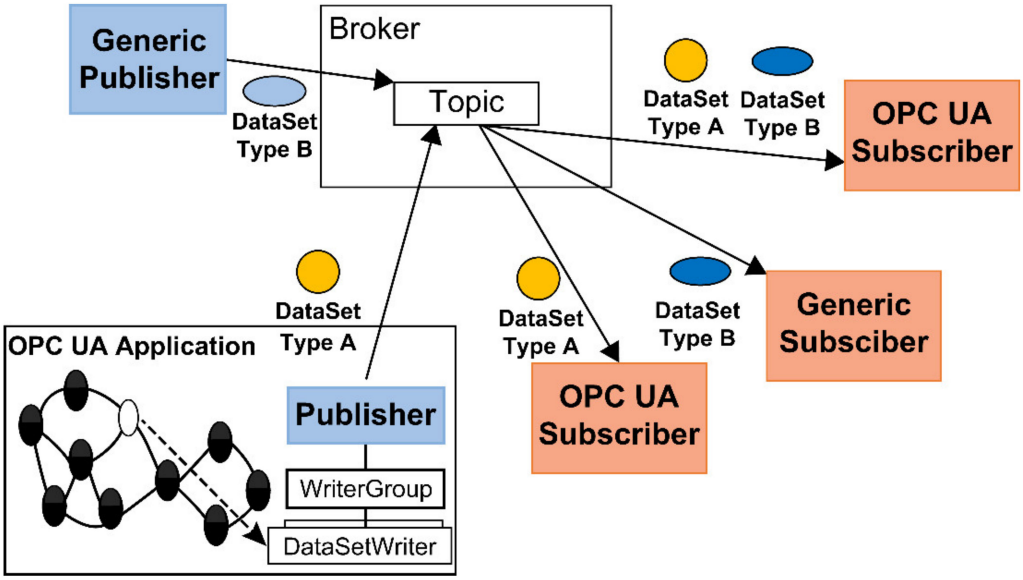

Figure 3. OPC UA PubSub broker communication model.

Two message-encoding methods are adopted in PubSub: UADP and JavaScript object notation (JSON). UADP is an optimized application of UA binary encoding, which also provides a security feature for messages. It is used for communication based on the UDP/IP multicast, Ethernet, and brokers. JSON is a highly readable method in text form and is used as an encoding method for general data exchange.

2.1.2. Data Distribution Service (DDS)

Based on PubSub, the DDS is a data-centric middleware recommended by the Object Management Group [11] for time/mission-critical applications. The DDS provides quality of service (QoS) to ensure communication quality. It sends and receives data based on topics that are virtual data transmission channels between a publisher and a subscriber. Multiple publishers can generate data for a single topic, and the generated data can be passed to all subscribers connected to the topic. Each set of generated data can be assigned a priority level while establishing the QoS for the data. Topic-based application programming interfaces (APIs) for the data-centric publish–subscribe (DCPS) model provided by DDS include the following types [12]:

- Topic: An information exchange unit is defined by using a subset of the interface definition language (IDL), identified by a name. A topic connects the publisher and the subscriber to allow anonymous transparent communication. A topic instance is a data entity separated by a key.
- Domain: A scoping mechanism for establishing a separate virtual network or communication context. Domains provide an optimal communication environment by separating different applications.
- Domain Participant: An entity participating in a domain.
- Partition: A scoping mechanism that logically groups topics and further constructs domains.
- Data Writer: An entity that publishes a topic.
- Data Reader: An entity that subscribes to a topic.
- Publisher: An entity that manages the data writer group.

Subscriber: As an entity that manages the data reader group, DDS dynamically searches for new participants in the system and establishes connections between publishers and subscribers on specific topics.

Figure 4 shows the correlation between the DDS domain participants and topics. Topics A and B exist in the global data space. Each of the two domain participants had a publisher and subscriber. It is not necessary to read only one topic because it is a publisher or subscriber. The topic can be read for each data writer inside the publisher and/or for each data reader inside the subscriber.

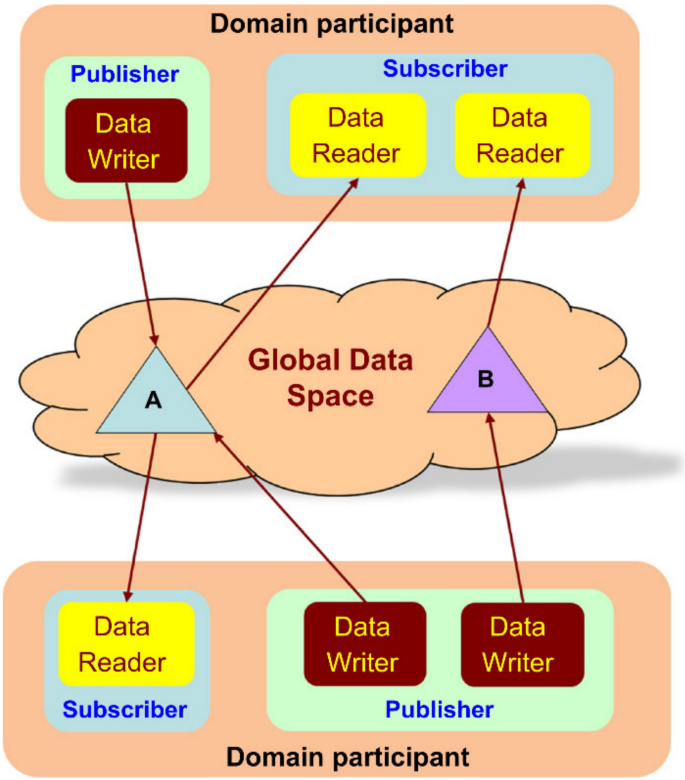

Figure 4. Correlation between DDS domain participants and topics.

Table 1 shows the default QoS policy provided by the DDS [11,12].

Table 1. Data distribution service (DDS) quality of service (QoS) policies.

DDS QoS Policies	
Deadline	Ownership strength
Destination order	partition
Durability	Presentation
Durability service	reader data lifecycle
Entity factory	Reliability
Group data	Resource limits
History	Time-based filter
Latency budget	topic data
Lifespan	Transport priority
Liveliness	user data
Ownership	Writer data lifecycle

2.2. Related Work

Research related to the OPC UA and DDS for interoperability is actively being conducted. Endeley et al. [13] proposed a smart gateway for interoperability between OPC UA and DDS in an IIoT environment. In their system, the OPC UA service set is mapped

through the smart gateway between the OPC UA and DDS. However, it is limited to OPC UA and does not support OPC UA PubSub.

Cheikh et al. [14] proposed the use of DDS middleware to support cooperative vehicle infrastructure systems for vehicle-to-vehicle or vehicle-to-infrastructure systems. Their proposed DDS middleware is based on the publish–subscribe model and has QoS support. Cilden et al. [15] proposed a general distributed architecture based on a DDS as an integrated laboratory architecture for avionics. Their proposed architecture facilitates the testing of modular avionics against the rest of the system consisting of real hardware and simulated participants by bringing together different communication bus standards and improving flexibility through DDS and data gateways. Lee et al. [16] proposed a DDS gateway architecture that enables interconnection between DDS domains for large-scale cyber-physical systems (CPS). Their architecture consists of four main components: interface module, topic manager, routing manager, and network module, and a method to address bottlenecks through token passing.

As the related studies cited above show, DDS is utilized in various fields, including IIoT, vehicles, avionics, and CPS, and interconnection and interoperability between different domains and systems have become more important. Additionally, most existing studies focus on gateways for the OPC UA and DDS, with scant regard for interoperability. To rectify this issue, a PubSub protocol converter model that ensures the interoperability of all devices that support the OPC UA standard is proposed herein.

3. OPC UA PubSub Protocol Converter Model

The protocol converter model for interoperability between the OPC UA publisher and subscriber present in the application layer of the DDS subscriber model consists of a DDS gateway and a DDS bridge. Figure 5 shows the overall system architecture of the proposed protocol converter model.

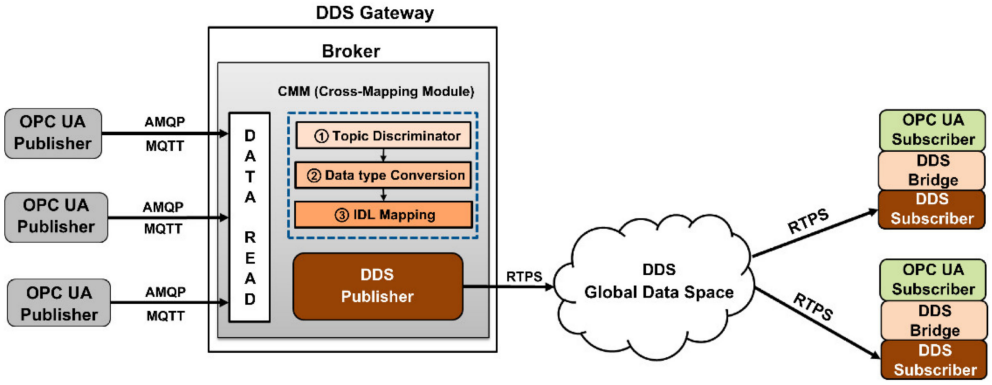

Figure 5. Overall system architecture of the OPC UA protocol converter.

3.1. A-Open62541 PubSub

A-Open62541 PubSub [5] is based on the Open62541 PubSub open source, and it includes additional functions not available in the corresponding source. Open62541 PubSub does not support AMQP, but A-Open62541 PubSub supports two standards: RabbitMQ and Qpid-proton. Figure 6 illustrates the functions supported by A-Open62541 PubSub.

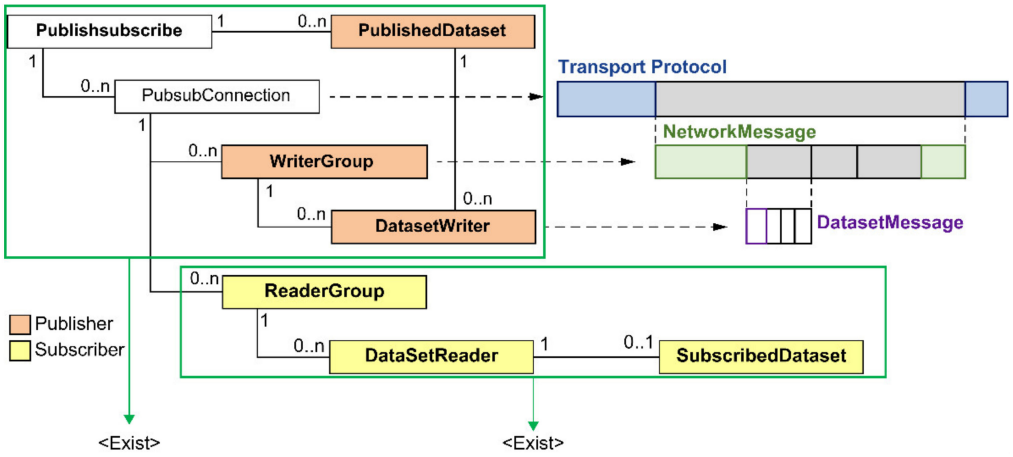

Figure 6. A-Open62541 PubSub UML diagram.

3.2. DDS Gateway

A DDS gateway converts the OPC UA published data into DDS messages and sends them to the DDS subscribers. One of the key functions of the DDS gateway is the conversion of native types of OPC UA published data and DDS messages; Table 2 shows the applicable mapping.

Table 2. Mapping of OPC UA PubSub native types to DDS.

OPC UA PubSub Type	DDS Type
Boolean	Boolean
Sbyte	Byte
Byte	Byte
Int16	Int16
UInt16	Uint16
Int32	Int 32
UInt32	Uint32
Float	Float32
Double	Float64
String	String
DateTime	Int64
ByteString	Sequence(Byte)

Data type conversion is performed in the cross-mapping module (CMM) with the following functions:

- Topic discriminator: The data and topics sent to the broker are read to call the "Data Type conversion" function if a topic is "DDS Gateway" or, if not, pass it onto the subscriber connected to the broker.
- Data type conversion: OPC UA publish data are sent in an encoded type, which is converted to DDS IDL before being received by the DDS subscriber.
- IDL mapping: The converted data are substituted into topics containing key values and data, and the key values are set to the length of the publish data.

3.3. DDS Bridge

A DDS bridge is a module that exists between the OPC UA subscriber and DDS subscriber that converts the messages received from the DDS gateway into OPC UA subscriber data and delivers them. In other words, the DDS gateway and bridge provide location transparency for OPC UA publishers and subscribers. Figure 7 illustrates the communication model of the OPC UA publisher and the subscriber. When the publisher issues encoding data such as DataSetField and DataSetMessage, the DDS gateway receives and transforms the data to fit the DDS data stack and the DDS subscriber within the OPC

subscriber receives it. The received DDS data are transformed back into the OPC UA data through the DDS bridge, and the subscriber decodes it. Due to the DDS bridge, it appears that only the OPC UA publisher and subscriber transmit and receive the data.

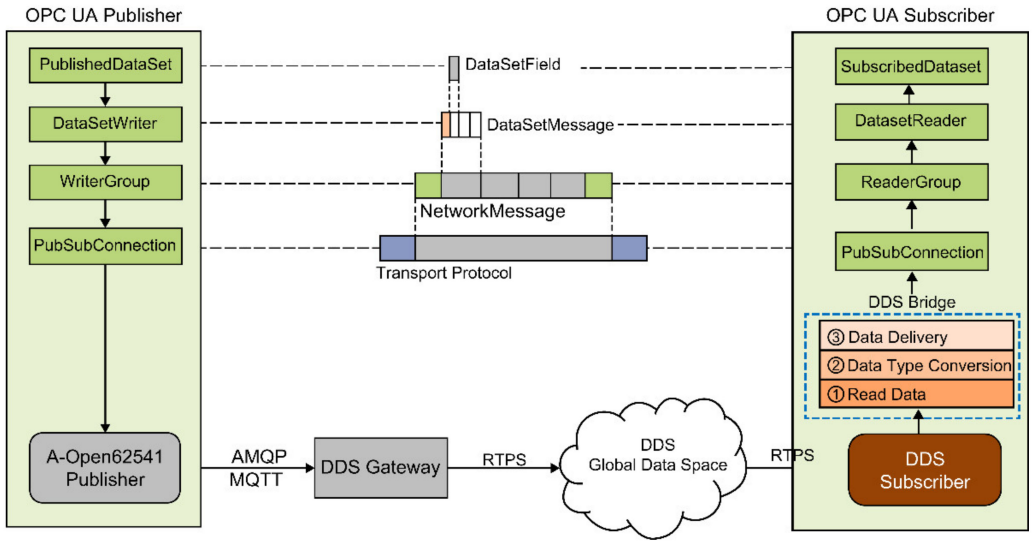

Figure 7. Communication model of OPC UA publisher and subscriber.

4. Implementation and Performance Evaluation

4.1. Implementation

4.1.1. A-Open62541 PubSub

As mentioned in Section 3.1, A-Open62541 PubSub has been implemented based on the Open62541 PubSub open source with the added AMQP function. JSON and MQTT have been implemented in the commonly used JSON-C and Mosquitto. Figure 8 illustrates the protocol stack and API of the A-Open62541 PubSub.

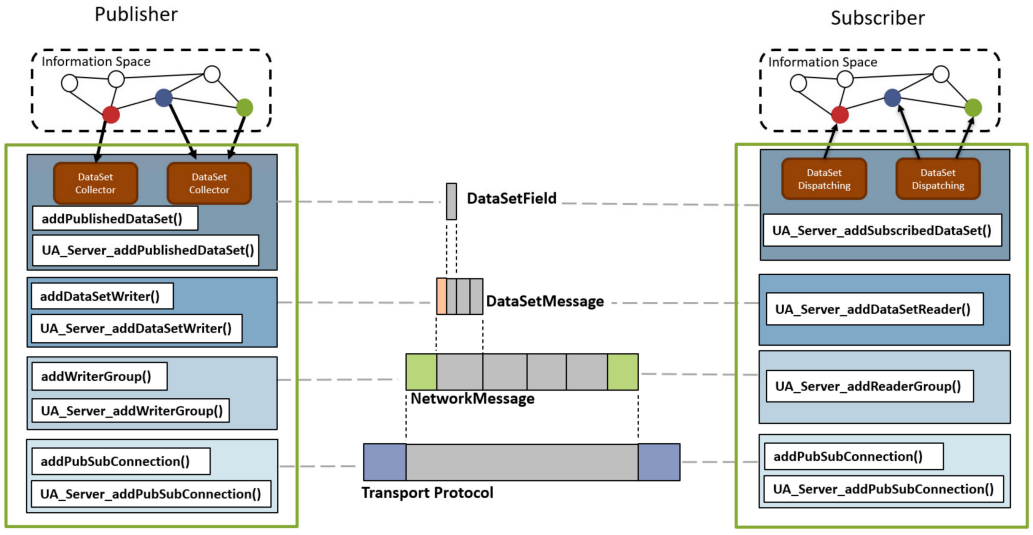

Figure 8. A-Open62541 PubSub protocol stack and API.

Table 3 shows the mapping of JSON used in the Open62541 PubSub open source in the JSON-C API, and Table 4 shows the mapping of MQTT−C in the Mosquitto API. Tables 5 and 6 describe the AMQP API of A-Open62541 PubSub.

Table 3. A-Open62541 PubSub JSON-C API.

Open62541 PubSub	A-Open62541 PubSub (JSON-C)
make_json()	Json_object_new_object() /*create a new Json object*/ Json_object_object_add() /*Add an object field to a json_object of type json_type_object*/
decode_json()	json_tokener_parse() /*Parse a string and return a non-NULL json_object if a valid JSON value is found*/ json_object_object_get() /*Get the json_object associated with a given object field*/ json_object_get_int() /*Get the int value of a json_object*/ json_object_get_string() /*Get the string value of a json_object*/

Table 4. A-Open62541 PubSub Mosquitto API.

Open62541 PubSub	A-Open62541 PubSub(Mosquitto)
UA_PubSubChannelMQTT_open()	mosquitto_lib_init() /* Must be called before any other mosquitto functions.*/ mosquitto_new() /* Create a new mosquitto client instance. */
UA_PubSubChannelMQTT_regist()	mosquitto_subscribe() /* Subscribe to a topic. */
UA_PubSubChannelMQTT_send()	mosquitto_publish() /* Publish a message on a given topic. */
UA_PubSubChannelMQTT_close()	mosquitto_disconnect() /* Disconnect from the broker. */

Table 5. A-Open62541 PubSub RabbitMQ API.

Open62541 PubSub	A-Open62541 PubSub (RabbitMQ)
UA_PubSubChannelAMQP_open()	amqp_new_connection() /*Allocate and initialize a new amqp_connection_state_t object*/ amqp_tcp_socket_new() /*A TCP socket connection. Create a new TCP socket.*/
UA_PubSubChannelAMQP_regist()	amqp_bytes_malloc_dup() /*Duplicates an amqp_bytes_t buffer.*/ amqp_queue_bind() /*amqp bind*/ amqp_basic_consume() /*Subscribe to a message from the broker*/
UA_PubSubChannelAMQP_send()	amqp_basic_publish() /*Publish a message to the broker*/ amqp_bytes_free() /*Frees an amqp_bytes_t buffer*/
UA_PubSubChannelAMQP_close()	amqp_channel_close() /*Closes a channel*/ amqp_connection_close() /*Closes the entire connection*/ amqp_destroy_connection() /*Destroys an amqp_connection_state_t object*/

Table 6. A-Open62541 PubSub Qpid-proton API.

Open62541 PubSub	A-Open62541 PubSub (Qpid-proton)
UA_PubSubChannelAMQP_open()	pn_event_connection() /*Get the connection associated with an event.*/
	pn_session() /*Factory for creating a new session on a given connection object.*/ pn_connection_set_container() /*Set the AMQP Container name advertised by a connection object.*/ pn_connection_open() /*Open a connection.*/
UA_PubSubChannelAMQP_regist()	pn_event_delivery() /*Get the delivery associated with an event.*/ pn_link_recv() /*Receive message data for the current delivery on a link.*/
UA_PubSubChannelAMQP_send()	pn_message_body() /*Get and set the body of a message.*/ pn_data_put_binary() /*Puts a PN_BINARY value.*/ pn_data_exit() /*Sets the current node to the parent node and the parent node to its own parent.*/ pn_message_send() /*Encode and send a message on a sender link.*/
UA_PubSubChannelAMQP_close()	pn_connection_close() /*Close a connection.*/

4.1.2. DDS Gateway

The DDS gateway (Figure 9) runs inside the MQTT and AMQP brokers. The main APIs of the DDS gateway are as follows:

- handle_connect(): This checks data and topics by receiving the data sent from the publisher to the Mosquitto broker.
- amqp_basic_get(): This checks data and topics by receiving the data sent from the publisher to the RabbitMQ broker.
- queue_receive(): This checks data and topics by receiving the data sent from the publisher to the Qpid-proton broker.
- check_gateway_topic(): If a topic is "DDS Gateway," the received data are passed to the change_type() API parameter.
- change_type(): Encoded OPC UA published data are converted into IDL types that can be used in DDS.
- input_IDL_data(): DDS topics are generated using IDL.

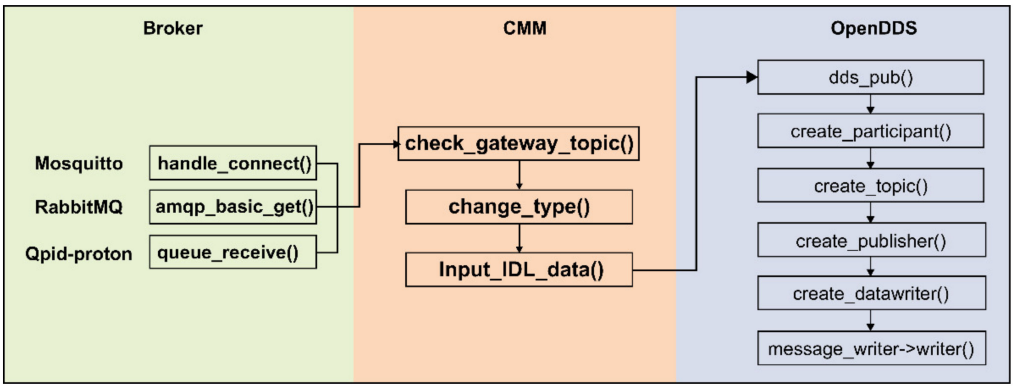

Figure 9. Overall software architecture of DDS gateway.

Figure 10 and Algorithm 1 show an example of the IDL mapping process and the QoS setting, respectively.

Algorithm 1: DDS gateway QoS setting

Initialization

1. publisher -> get_default_datawriter_qos(pin_qos)

Task 1: Modify QoS settings

2. pin_qos.history.kind = DDS::KEEP_ALL_HISTORY_QOS
3. pin_qos.ownership.kind = DDS::SHARED_OWNERSHIP_QOS
4. pin_qos.destination_order.kind = DDS::BY_SOURCE_TIMESTAMP_DESTINATION_QOS
5. pin_qos.reliability.kind = DDS::RELIABLE_RELIABILITY_QOS
6. pin_qos.reliability.max_blocking_time.sec = 0
7. pin_qos.reliability.max_blocking_time.nanosec = 100,000

Task 2: Apply QoS settings

8. DDS::DataWriter_var writer = publisher->create_datawriter(topic, pin_qos, 0, 0)

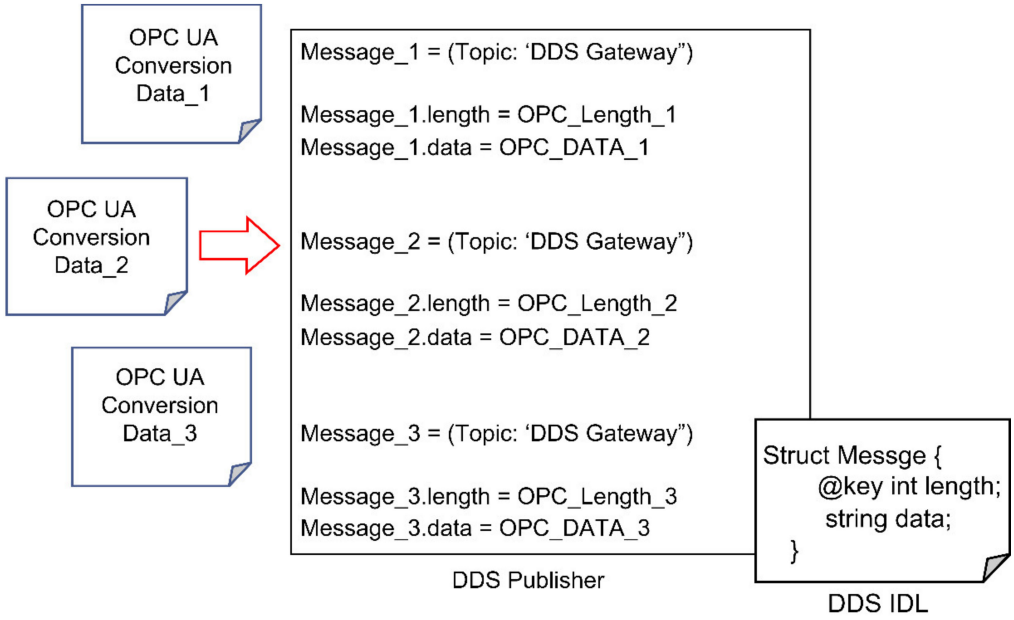

Figure 10. Example of DDS gateway IDL mapping process.

4.1.3. DDS Bridge

Figure 11 shows the architecture of the DDS bridge, which consists of the following APIs.

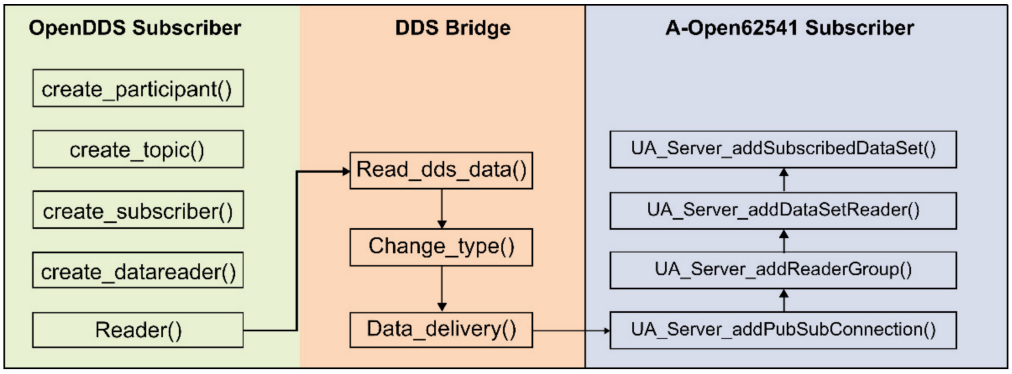

Figure 11. DDS bridge APIs.

- Read_dds_data(): This receives and reads DDS messages.
- Change_type(): This converts DDS messages to OPC UA subscriber data types according to the rules in Figure 9.
- Data_delivery(): This passes the information model converted to OPC UA subscriber data types to UA_Server_addPubSubConnection(), which provides the "PubSubConnection" function.

4.2. Performance Evaluation

4.2.1. Testbed and Scenario

To evaluate the performance of the proposed protocol converter, a testbed was built and a test scenario executed (see Figure 12).

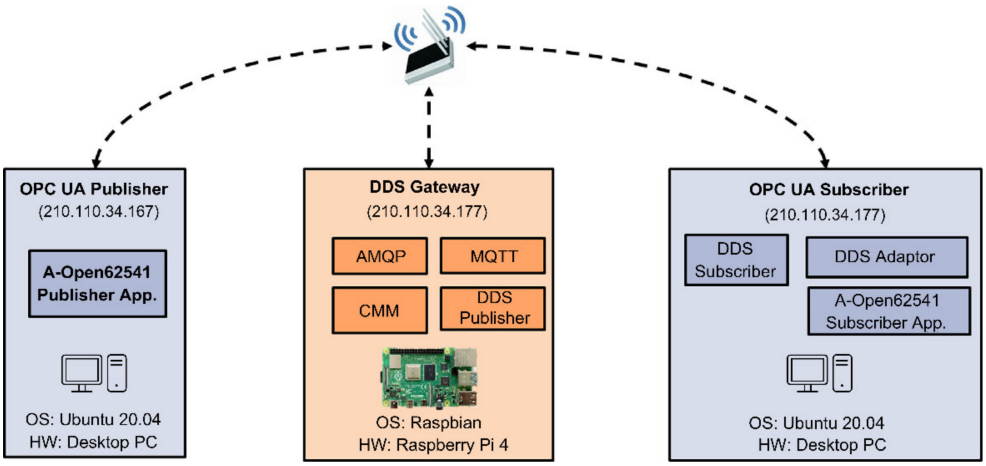

Figure 12. Testbed configuration used for proof of concept.

The testbed used for proof of concept (POC) comprised a DDS gateway, an OPC UA publisher, and a DDS subscriber. The DDS gateway ran on Raspberry Pi 4 whereas the OPC UA publisher and DDS subscriber ran on Ubuntu virtual machines. Figure 13 and Table 7 illustrate the testbed and corresponding system specifications, respectively.

Table 7. Proof of concept testbed system specifications.

Component	OPC UA Publisher	DDS Gateway	OPC UA Subscriber
OS	Ubuntu 20.04	Raspbian GNU/Linux 10	Ubuntu 20.04
Tool	A-Open62541 Pub	Mosquitto v2.0.4 RabbitMQ v3.8.2 Qpid-proton v0.33.0 OpenDDS v3.17	OpenDDS v3.17
Language	C	C, C++	C, C++
RAM	4 GB	4 GB	4 GB
Capacity	30 GB	32 GB	30 GB
CPU	Intel Core i7	ARM v7 Processor rev3	Intel Core i7

The test scenario for the POC was as follows:

- AMQP/MQTT-based OPC UA publisher data publication (source IP: 210.110.34.167).
- Broker reception (destination IP: 210.110.34.177).
- Publication of RTPS messages to UDP by the DDS publisher inside broker (source IP: 210.110.34.177).
- UDP-based RTPS messages received by DDS subscriber (destination IP: 239.255.0.2).

To analyze the performance of the protocol converter, a test bed was configured as a local environment based on Ubuntu Linux-based virtual machines. Figure 13 and Table 8 illustrate the configuration of the test bed and detailed H/W and S/W specifications.

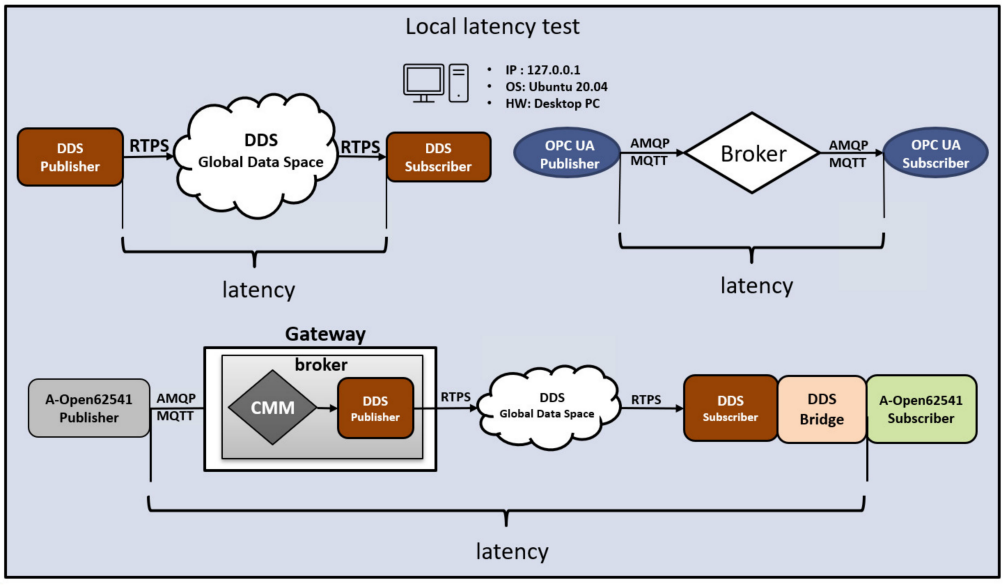

Figure 13. Testbed configuration for the performance analysis.

The specifications of the performance analysis testbed are as follows.

Table 8. System specifications for the performance analysis testbed.

OS	Tool	RAM	Language	Capacity	CPU
Ubuntu 20.04	A-Open62541 OpenDDS v3.17 Mosquitto v2.0.4 RabbitMQ v3.8.2 Qpid-proton v0.33.0	4 GB	C, C++	30 GB	Intel Core i7

The test scenario for the performance analysis was as follows:

- A measure of the OpenDDS latency obtained without the protocol converter – ①;
- A measure of the OPC UA PubSub latency obtained without the protocol converter – ②;
- A measure of the A-Open62541 PubSub latency obtained with the protocol converter – ③;
- The comparison of latency ③ and the sum of latency ① and ②.

4.2.2. Evaluation Results

The throughput and latency were measured and analyzed based on the test scenario to confirm the interoperability and verify the implementation accuracy and usability of the implemented protocol converter.

To verify the implementation accuracy, Figure 14 shows the flow of AMQP messages from the OPC UA publisher to the DDS gateway monitored using Wireshark, and Figure 15 shows the data flow from the DDS gateway to the DDS subscriber. As Figures 14 and 15 show, when the real number 58.0 was sent as random data from the OPC UA publisher to the JSON type, the corresponding type was mapped to the DDS type at the DDS gateway, and an identical value of 58 was passed to the DDS subscriber.

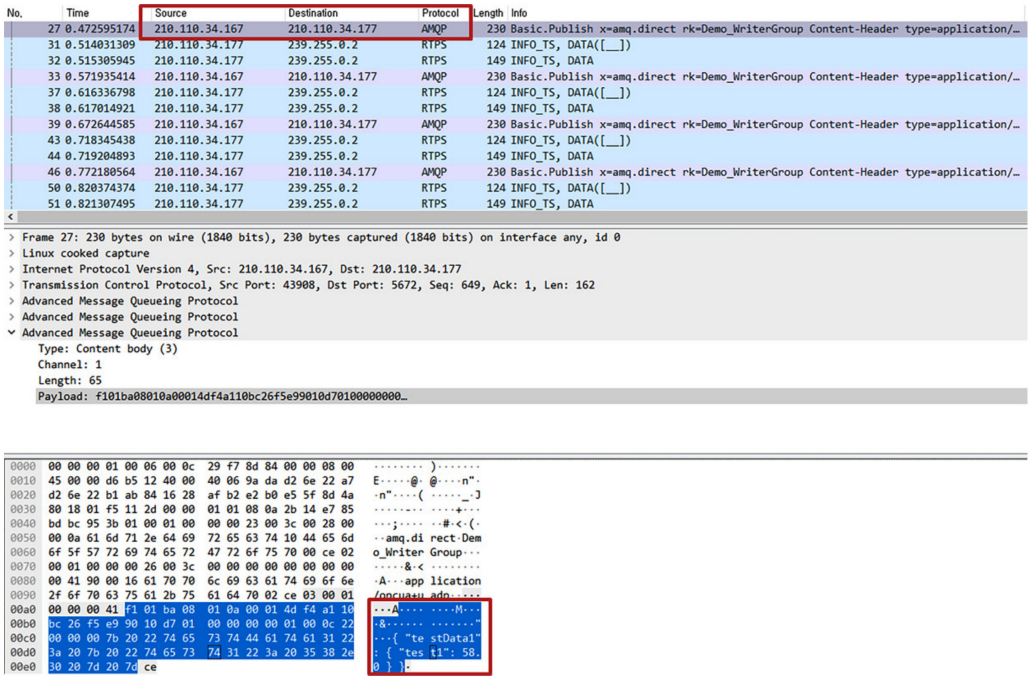

Figure 14. Monitoring screen between the OPC UA publisher and DDS gateway (AMQP).

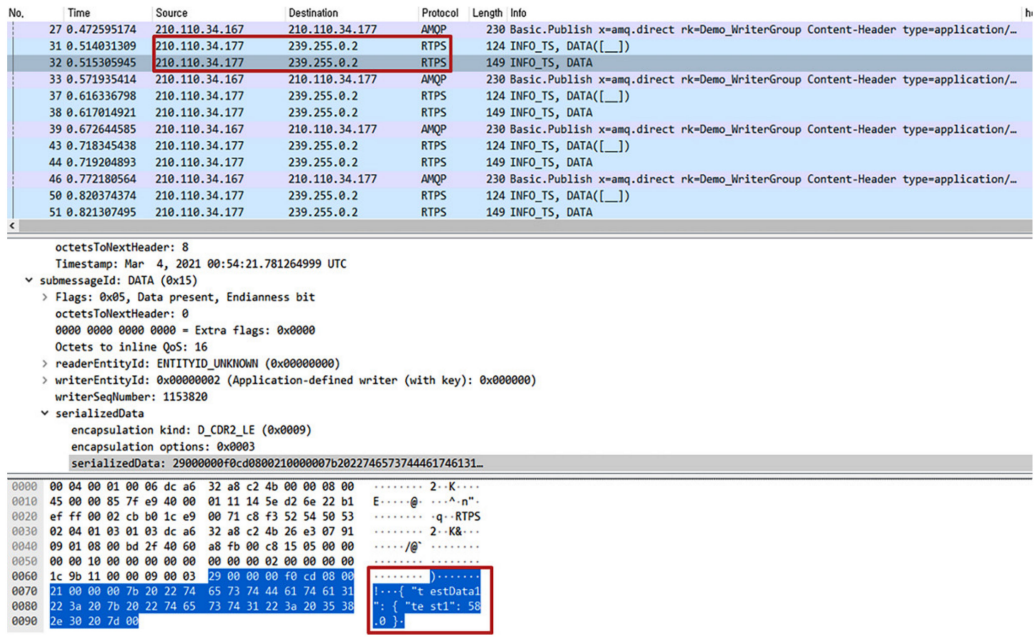

Figure 15. Monitoring screen between the DDS gateway and OPC UA subscriber (AMQP).

The protocol converter supports AMQP and MQTT; MQTT shows the same results as AMQP, which verifies that the function works correctly. The performance analysis and results are as follows. To analyze the performance of the protocol converter, it was tested locally with the delay time unit in microseconds (μs). The final delay time was calculated as the average value of 20 publish–subscribe cycles executed at 100 ms intervals. As in the performance analysis scenario described above, the delay time of OPC UA PubSub and that of OpenDDS with no DDS converter are shown in Figure 16.

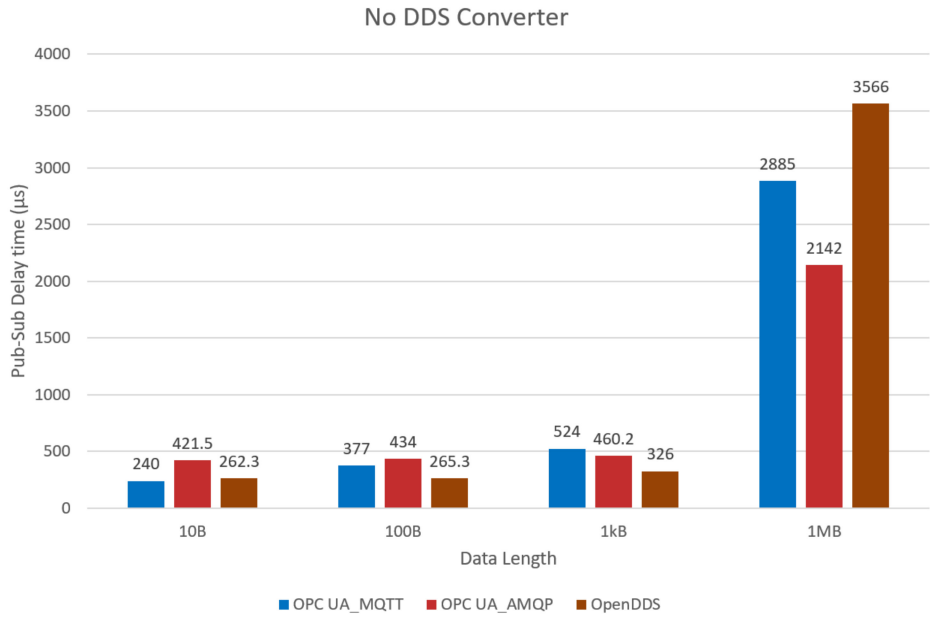

Figure 16. OPC UA PubSub and OpenDDS latencies without converter.

Figure 17 compares the sum of the OPC UA and OpenDDS latencies to that obtained when using the MQTT-based converter while Figure 18 compares it to that obtained using the AMQP-based converter.

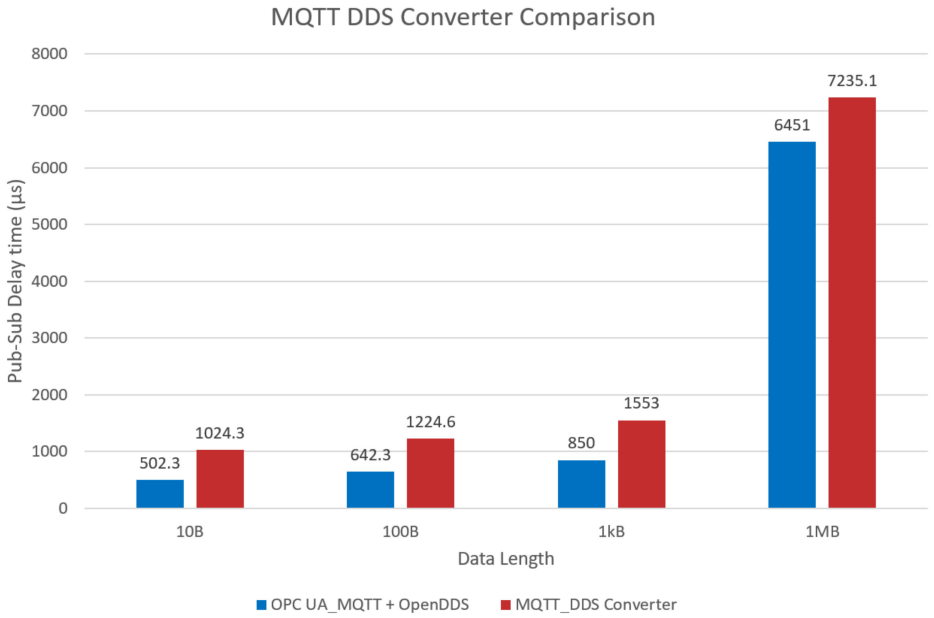

Figure 17. Comparison of latencies with and without the MQTT-based converter.

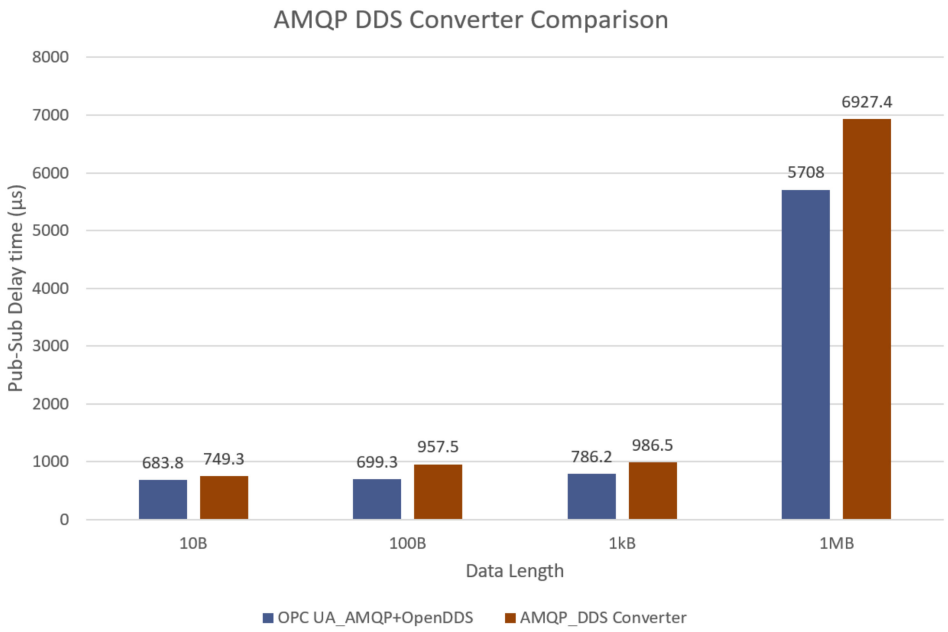

Figure 18. Comparison of latencies with and without the AMQP-based converter.

Based on the performance analysis results, when the data for all cases are averaged, it is clear that there is an increase in latency by a factor of 47% compared to the case without the converter. Table 9 summarizes these results.

Table 9. Performance analysis results/latencies for various data lengths.

	Data Lengths and Corresponding Latencies			
	10 B (µs)	**100 B (µs)**	**1 kB (µs)**	**1 MB (µs)**
OPC UA MQTT	240	377	524	2885
OPC UA AMQP	421.5	434	460.2	2142
OpenDDS	262.3	265.3	326	3566
MQTT DDS converter	1024.3	1224.6	1553	5235.1
AMQP DDS converter	749.3	957.5	986.5	4927.4

In the case of DDS, there is no broker, but A-Open62541 satisfies all the criteria of OPC UA Part 14, adding AMQP and MQTT communication. As shown in the performance analysis results in Table 9, when the length of the transmission/reception data are 1 kB, the latency with the AMQP DDS converter is approximately 1.2 times that without, and for 1 MB, the latency with the MQTT DDS converter is approximately 1.12 times that without. Therefore, it is recommended to convert the type of broker according to the data length and use it.

5. Conclusions

This paper proposed a protocol converter that provides interoperability between the OPC UA publisher and DDS subscriber. The proposed protocol converter primarily converts the data types passed from the publisher into DDS messages. The A-Open62541 PubSub used in this study is based on the Open62541 PubSub and implemented the AMQP function. In contrast, the JSON parser and MQTT were implemented by substituting JSON-C and Mosquitto, which are commonly used in industry. Through performance analysis, the interoperability of the proposed technique was confirmed, and the implementation accuracy and usability were verified. OPC UA PubSub is a protocol that supports real-time at the field level and allows the integration of OPC UA and the entire factory automation levels into the identical communication protocol of the OPC UA. It also facilitates the

construction of cloud-based IIoT applications by connecting to MQTT or AMQP. Most existing studies focused on gateways using OPC UA and DDS. The results from existing studies and the technique presented here allow all devices supporting OPC UA and PubSub, as OPC UA standards, to acquire interoperability with DDS subscribers. In a smart grid, the OPC UA (IEC 62541) is used as a protocol for common information model communication, which is a standard information model for smart grids, and the application scope of IEC62541 can be significantly expanded by adding OPC UA PubSub. As the current implementation of A-Open62541 PubSub does not support the security key service (SKS) recommended by OPC UA Part 14, there is a limit to data security for transmission and reception. SKS will be implemented in the future based on an open-source secure socket layer (SSL) [17] and OAuth2.0 [18]. Additionally, an embedded system [19] that supports a TSN will be ported to validate the technique in a real-time Ethernet communication environment.

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

Funding: This research was funded by the Ministry of Environment of Korea under a grant to the Korea Environmental Industry and Technology Institute, grant number 2020002970006.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscrip, or in the decision to publish the results.

References

1. Prinz, F.; Schoeffler, M.; Eckhardt, A.; Lechler, A.; Verl, A. Configuration of Application Layer Protocols within Real-time I4.0 Components. In Proceedings of the 2019 IEEE 17th International Conference on Industrial Informatics (INDIN), Helsinki, Finland, 22–25 July 2019; pp. 971–976. [CrossRef]
2. Eckhardt, A.; Muller, S.; Leurs, L. An Evaluation of the Applicability of OPC UA Publish Subscribe on Factory Automation Use Cases. In Proceedings of the 2018 IEEE 23rd International Conference on ETFA, Turin, Italy, 4–7 September 2018; pp. 1071–1074. [CrossRef]
3. OPC Foundation, OPC Unified Architecture Specification Part 14: PubSub. 2019. Available online: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-14-pubsub/ (accessed on 19 July 2021).
4. Open62541. Available online: https://open62541.org (accessed on 19 July 2021).
5. Song, B.K. A-Open62541 PubSub Technical Memorandum. 2020. Available online: https://github.com/SimWB/A-62541 (accessed on 19 July 2021).
6. GitHub. Available online: https://github.com/LiamBindle/MQTT-C (accessed on 19 July 2021).
7. Mosquitto Eclipse Foundation. Available online: https://mosquitto.org (accessed on 19 July 2021).
8. RabbitMQ. Available online: https://www.rabbitmq.com/ (accessed on 19 July 2021).
9. Apache Qpid. Available online: https://qpid.apache.org/proton/ (accessed on 19 July 2021).
10. OpenDDS. Available online: https://opendds.org/ (accessed on 19 July 2021).
11. Object Management Group (OMG). Available online: https://www.omg.org/omg-dds-portal/ (accessed on 19 July 2021).
12. Etxeberria-Agiriano, I.; Calvo, I.; Pérez, F. Providing Soft Real-Time Capabilities to Business Applications. In Proceedings of the 7th Iberian Conference on Information Systems and Technologies, Madrid, Spain, 20–23 June 2012; pp. 1–6.
13. Endeley, R.; Fleming, T.; Jin, N.; Fehringer, G.; Cammish, S. A Smart Gateway Enabling OPC UA and DDS Interoperability. In Proceedings of the 2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation, Leicester, UK, 19–23 August 2019; pp. 88–93. [CrossRef]
14. Cheikh, F.B.; Mastouri, M.A.; Hasnaoui, S. Implementing a Real-Time Middleware Based on DDS for the Cooperative Vehicle Infrastructure Systems. In Proceedings of the 2010 6th International Conference on Wireless and Mobile Communications, Valencia, Spain, 20–25 September 2010; pp. 492–497. [CrossRef]
15. Cilden, E.; Gültekin, E.; Poyraz, D.; Haluk Canberi, M. A Generic Distributed Architecture to Integrate Simulated Participants with Modular Avionics. In Proceedings of the 2018 IEEE/ACM 22nd International Symposium on Distributed Simulation and Real Time Applications (DS-RT), Madrid, Spain, 15–17 October 2018; pp. 1–2. [CrossRef]
16. Lee, W.; Chung, S.; Choi, M.; Cho, S.; Joe, I.; Park, J.; Lee, S.; Kim, W. A Robust Inter-Domain DDS Gateway Based on Token Passing for Large-Scale Cyber-Physical Systems. In Proceedings of the 16th International Conference on Advanced Communication Technology, Pyeongchang, Korea, 16–19 February 2014; pp. 868–871. [CrossRef]

17. OpenSSL. Available online: https://www.openssl.org/ (accessed on 19 July 2021).
18. OAuth 2.0. Available online: https://oauth.net/2/ (accessed on 19 July 2021).
19. TTTech Industrial. Available online: https://www.tttech-industrial.com/products/slate/edge-ip-solution/ (accessed on 19 July 2021).