EAI Endorsed Transactions
on Internet of Things

Review Article **EAI.EU**

DDS-XRCE Standard Performance Evaluation of Different Communication Scenarios in IoT Technologies

Şevval Şolpan[1,*] and Kerem Küçük[2]

[1]Department of Computer Engineering at Kocaeli University, Kocaeli, Turkey
[2]Department of Software Engineering at Kocaeli University, Kocaeli, Turkey

Abstract

Although the increasing number of technological products brings many solutions for Internet of Things (IoT) applications, it also causes some drawbacks, such as whether the product in question would run accordingly to a system structured to enable high-performance like Data Distribution Service (DDS). Therefore, the capabilities of the products must be defined to say that they are compatible enough. This paper aims to evaluate the performance of the DDS-XRCE standard while observing its working mechanism. As test scenarios, we benefit from three DDS-XRCE deployments that occurred due to the kind of receiver and sender, the path that packets follow, and the protocols used. Test conditions were set by switching stream modes, transport profiles, and limiting packet deliveries. We obtained the test environment by creating the DDS and DDS-XRCE objects using several eProsima implementations and tools for the standards. We monitored the network messages in two ways: 1) Using multiple Gnome Terminator terminals for observation via the human eye during testing. 2) Using Wireshark to save the information of the packets for further examination. We conducted 36 experiments focusing on latency, throughput, and packet loss. As a result of our study, the DDS-XRCE standard is deemed suitable for Internet of Things applications.

Keywords: DDS-XRCE, Latency, Packet loss, Performance evaluation, Throughput.

Received on 12 September 2022, accepted on 05 November 2022, published on 23 November 2022

doi: 10.4108/eetiot.v8i4.2691

1. Introduction

The term "Internet of Things (IoT)" comes into question and is used for systems whose objects are established in the physical environment and connected over the Internet [1]. In the last decade, the IoT has gained popularity in several areas, such as the military, automation, and healthcare. It has considerable potential due to its technology based on sensors in the issue of reducing intervention [2]. Thus, it draws the attention of experts even more. It accomplishes this due to technology enabling itself.

An IoT system can be described as a vast network whose devices cooperate and share data. Sensors are the components of the system that generate data, and the system generates a massive amount of data in a short period [3]. Simultaneously, increasing data results in concerns about its handling or its protection. Therefore, technologies and disciplines that enable IoT draw attention in order to improve IoT systems as well.

For instance, to improve human-machine interaction, some researchers want machines to be able to identify emotions and produce their own. The reason is that the IoT does not solely consist of machine-to-machine interaction. It also includes human-machine interaction [4]. Some researchers make a contribution by creating a platform to evaluate a video streaming service that operates in the cloud-server environment in the context of cloud computing [5]. Some researchers focus on the security of IoT systems. Because attackers mostly try to manipulate the network in assorted ways, it causes the network to be at risk [6]. Thus, it is thought that if machine learning algorithms are allowed to learn the operation of the devices and objects in the system, then the

[*]Corresponding author. Email: 205112004@kocaeli.edu.tr

system might be able to prevent corruption caused by unknown origins by detecting abnormal behaviour [7].

It is seen that an IoT system contains various devices of different technologies as it is a vast network. However, maintaining communication between devices becomes quite challenging when IoT is applied in a heterogeneous environment whose objects carry unalike purposes and priorities. The way of communication takes form regarding the kind of devices, purpose and priorities of the implementation, area of functioning, location, protocols, software, and the other end of the connection. The variation appears as a solution for possible problems in challenging issues. On the other hand, the harmonious functioning of things has become difficult as a consequence [8]. There is an increase in the industry's demands regarding IoT applications easily affected by latency and data [9]. It contributes to the burdensome communication issue.

Being able to tell the responsive behaviour of the things to the data is an important issue apart from the compatibility of the devices in an IoT system as well. In other words, the processing time, the amount of data transferred periodically, the amount of data lost, and the retransmissions of the packet must be known. For that reason, for example, one can say if the thing runs at low latency or not. When deciding if the product will perform as expected or not for a certain implementation, knowing these kinds of characteristics is essential.

The IoT structure consists of three layers: the perception layer in which the sensing devices are established for the collection of data; the network layer in which the devices access the network for data transmission; and the application layer in which the applications run actively [2]. IoT applications process request and response operations over application layer protocols. The Constrained Application Protocol (CoAP), Message Queue Telemetry Transport (MQTT), Extensible Messaging and Presence Protocol (XMPP), Advanced Message Queuing Protocol (AMQP), and Data Distribution Service (DDS) that have been adopted by Object Management Group Inc. (OMG) are the most well-known application layer protocols [10], [11]. Although various protocols exist, communication standards do not meet all the needs due to a wide range of demands, environmental conditions, constrained devices, limited resources, budget, and technological limitations. Their usage is a demanding topic in the IoT that causes users to think thoroughly [12].

OMG published the Data Distribution Service for Extremely Resource Constrained Environments (DDS-XRCE) in 2020 as a solution to these limitations and constraints. There are implementations of DDS-XRCE called Micro-XRCE-DDS, Micro-XRCE-DDS-Agent, and Micro-XRCE-DDS-Client by eProsima [13]. “micro-ROS” is a version of ROS2 and runs on microcontrollers. It contains the implementation of DDS-XRCE provided by eProsima [14]. It is possible to see implementations using DDS-XRCE.

Nevertheless, there is a lack of studies concerned with the DDS-XRCE standard. For that reason, the network performance characteristics of the DDS-XRCE remain undetermined, which causes confusion about how to decide whether the DDS-XRCE will function in a system harmoniously or not. Therefore, the problem this study aims to solve is that the DDS-XRCE's unknown performance characteristics remain unrevealed. In this paper, we evaluated the performance of DDS-XRCE on its implementation provided by eProsima. The contributions of this paper are as follows:

- Analysis of the DDS standard, its objects, and working mechanism
- Analysis of the DDS-XRCE standard, its objects, and functioning mechanism
- Analysis of integration between DDS and DDS-XRCE standards
- Performance evaluation experiments and results of the evaluation of the DDS-XRCE standard

This paper is expressed as follows: Recent similar studies mainly concerned with the performance of application layer protocols were examined and summarised in the second section. The DDS and DDS-XRCE standards were examined in the third section. In the fourth section, the experiments were explained in more detail in three phases: 1) the preparation phase, 2) the simulation and data collection phase, and 3) the analysis phase. The information about the tools used in the experiments was given in the fifth section. The results of the experiments were presented as tables in the sixth section. The conclusions that are related to the behaviours of the DDS-XRCE objects were explained in the seventh section.

2. Related Work

In [15], Dehnavi et al. modelled an application of the DDS-XRCE and implemented the model in multi-processor real-time embedded systems. Additionally, they conducted some experiments on the systems, which have a soft real-time side for the DDS-XRCE Agent and a hard real-time side for the DDS-XRCE Client. The worst-case response times of the publisher and subscriber were measured using the Scenario Aware Data Flow (SADF) model, which they proposed to analyse the expected value of throughput in the long term.

Kang et al. were concerned about the problems that IoT applications, which are easily affected by data and latency, cause for the edge and cloud implementations of publish/subscribe utilities [9]. They utilised the DDS and Kubernetes (K8s), which manage containerised applications in the cloud, to come up with a solution to these problems. The evaluation was maintained by running DDS applications in the K8s cluster, and they observed the impact of K8s on the DDS performance by focusing on throughput and latency when different QoS policies were enabled.

Chul-Hwan Kim et al. developed a simulator to evaluate the performance of DDS [16]. The development of the simulator was carried out on the simulation platform called QualNet, which enables the use of several network protocols. The performance metrics that the authors focused on are discovery-completion time, message transmission delay, the quantity of the data messages, and the time spent processing the user data.

In [17], Krinkin et al. evaluated the performance of DDS. The experiments were conducted focusing on latency and jitter in comparison to different open-source implementations of DDS such as OpenDDS by Prismtech, OpenSplice by Vortex, and FastRTPS by eProsima.

Thulasiraman et al. evaluated the performance of DDS, focusing on throughput and latency in a certain scenario [18]. It is because US Naval autonomous systems were in search of a communication protocol that could work with all network assets and the DDS was considered a major candidate. They modelled an experiment system including Satellite Communications (SATCOM) and Wi-Fi links. Also, Mininet was used during network emulation and network parameter arrangement processes.

In [19], Andrei et al. evaluated the performance of DDS and AMQP. The evaluation was conducted using OpenSplice and FastRTPS implementations of DDS and the RabbitMQ implementation of AMQP, focusing on latency, investigation of message queues, and what the message sizes and frequencies are when the throughput has reached its highest level.

Similarly, Profanter et al. evaluated open62541 of OPC UA, ROS C++ of ROS, eProsima FastRTPS of DDS, and Eclipse Paho MQTT C of MQTT implementations comparatively [20]. The evaluation was conducted by measuring the round-trip time of messages when the systems were idle, with high CPU load, and under high network load conditions.

Chen and Khun evaluated the performance of MQTT, CoAP, DDS, and a custom protocol, which relies on UDP, for medical purposes using a network emulator [21]. They focused on the bandwidth consumed by the system, latency, and packet loss.

Web performance of web implementations and IoT protocols were evaluated by experimenting on two test applications by Babovic et al. [22]. In the first application, various Web platform implementations were evaluated in the first application on various metrics. In the second application, MQTT, AMQP, XMPP, and DDS IoT protocols were evaluated, focusing on latency and throughput.

The aim of Chen et al. is to check if DDS works according to real-time essentials [23]. It is a study that evaluates the performance of DDS on the PREEMPT_RT Linux system and Loongson platform in terms of latency, jitter, and data throughput. As a result, there is a relationship between the performance of DDS and the network card.

In [24], MQTT and CoAP protocols were evaluated theoretically and practically by Palmese et al. Another form of MQTT, which is MQTT-SN, works according to the Publish/Subscribe communication scheme. Consequently, some changes were made to CoAP to follow the same communication scheme as MQTT-SN to compare them fairly. Both protocols rely on the UDP protocol in communication.

According to Sasaki et al., cooperation between IoT protocols and other protocols of OSI layers is a curious topic and worth evaluation. For example, IP and associated protocols perform the work of MQTT. Additionally, MQTT comes with a Quality of Service (QoS). The performance of MQTT-TCP cooperation and the MQTT QoS mechanism on data transmission were analysed [25].

E-health is an area that IoT is crucially interested in, and the performance of e-health applications is particularly thought about. Therefore, Kassem and Sleit examined CoAP and MQTT protocols over e-health scenarios and evaluated their performance comparatively on the past time that the authors chose as a metric [26].

MQTT is a protocol whose messages are brokered by an MQTT Broker between publisher and subscriber. Since messages pass over the broker, the broker is considered the point at which blockage is most likely to happen in the network. Based on that fact, the broker's performance indicates the performance of MQTT. The performance of MQTT v5.0 and its new functionalities were evaluated over its broker using MQTTLoader, which Banno et al. developed for load testing [27].

Bender et al. evaluated the performance of MQTT over its several open-source implementations using a test system that they created, focusing on interoperability, resource consumption, and latency [28]. These implementations are Mosquitto, HiveMQ, EMQX, VerneMQ, MQTT.js, and Paho. The test system they used can work free of MQTT implementations or language.

Protocols of the application layer have critical significance in decreasing network traffic in IoT applications. Choosing them properly might ease the load of network traffic and increase successful message delivery. For that reason, in [29], the performance of CoAP, MQTT, and REST is discussed, which Tandale et al. measured by implementing them on the Raspberry Pi3 as a gateway and evaluated by focusing on the bandwidth that protocols consume and time that operations spend.

In [30], Basavaraju et al. evaluated the AMQP protocol by comparing RabbitMQ and ActiveMQ message brokers, focusing on latency, data rate, different payloads, and the number of messages. One of the message brokers implements AMQP version 0-9-1. The other implements AMQP version 1-0.

Pohl et al. evaluated AMQP, MQTT, and XMPP protocols, focusing on bandwidth usage, reliability, latency, and throughput as performance metrics in a business application. The test system they designed has three layers, along with changeable latency and packet loss rate [31].

Previous studies mostly maintained their evaluation by comparing different protocols or focusing on a single protocol. Studies focusing solely on one protocol evaluated it using its several implementations or features. This paper can be categorised as a study focusing on one protocol, although it examines two protocols: DDS and DDS-XRCE. The DDS-XRCE operates by integrating with the DDS. Hence, the use of DDS-XRCE makes the use of DDS essential in some ways. It is also what distinguishes this study from previous works. A necessary integration between protocols rarely occurs. Even though the DDS appears in the evaluation steps, the main focus is on the DDS-XRCE. The DDS-XRCE standard is the subject of this study as a result. The evaluation proceeded while the DDS-XRCE was operating under several configurations. Since the main goal is to evaluate the performance of a protocol, there are many evaluation criteria. As we researched in this paper, these criteria mostly appear as latency, data packet loss, and throughput. The test system takes shape depending on the evaluation criteria as much as the problem. It is possible to see that some studies contain creations developed or designed by the authors. These creations vary as simulators or applications. Our creations are different topics and clients we produced using code generation tools. In addition, clients create the network of our study.

3. The DDS Standard

Data Distribution Service (DDS), which is known commonly nowadays since it was published in 2006, is also a standard of OMG that provides high-performance communication along with efficient delivery of information. OMG is a standard establishment active in the computer industry [32]. An application in which the DDS is used is Robot Operating System 2 (ROS2). It uses DDS as DDS is more acceptable for real-time embedded systems due to its configurations [33].

3.1. The Structure of the DDS

The structure of the DDS occurs due to the connection of two parts: DDS Global Data Space (DDSGDS) at the centre and DDS Participants, which communicate with each other using the Real-Time Publish-Subscribe (RTPS) protocol only over DDSGDS. Thus, the structure takes the shape of a star topology. The structure is shown in Fig. 1.

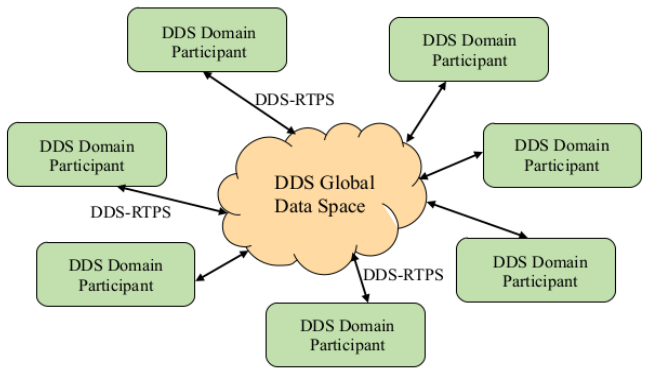

Figure 1. The structure of DDS

3.2. The Objects and Concepts of the DDS

To have a better understanding, common objects and concepts in the DDS are explained.

A publisher is an object that sends data to the endpoint that must be reached. A datawriter is an object that is used by applications to inform the publisher about data and its information. A publication is a relation between a publisher and datawriter. A subscriber is an object that receives data sent from a publisher. A datareader is an object that is used by data-receiving applications by attaching it to subscriber. A subscription is a relation between a subscriber and datareader. A topic is a concept between publication and subscription. Quality of Service is a policy list of adjustable features that manage some actions of the system. Each QoS policy concerns with particular entity or several entities. A domain is a set of conceptual links between the domain members which helps them communicate with each other. A domain participant is an application that is a member of a domain. An application can be a member of more than one domain.

3.3. Data-Centric Publish-Subscribe (DCPS)

The DDS was examined in terms of entities, operations, and functioning mechanisms. DDS defines the Data Centric Publish Subscribe (DCPS) model, which consists of five modules. The DCPS is the object model in the DDS, and the model categorises its objects and interfaces with modules. For example, listener interfaces belong to the Infrastructure Module.

3.3.1. The Infrastructure Module

The Infrastructure Module, which contains Entity, DomainEntity, QosPolicy, Listener, Status, WaitSet, Condition, GuardCondition, and StatusCondition classes and interfaces, helps the middleware to provide notification and wait-based interactions. The classes and interfaces of the Infrastructure Module are all abstract and processed by other modules.

3.3.2. The Domain Module

The Domain Module, working like a factory for many classes, is also to which the DomainParticipant class belongs. The Domain Module contains the DomainParticipantFactory class and the DomainParticipantListener interface in addition to the DomainParticipant class.

3.3.3. The Topic-Definition Module

The Topic-Definition Module contains TopicDescription, Topic, ContentFilteredTopic, MultiTopic, TopicListener, and TypeSupport classes and interfaces. They are the things that will be used during the topic creation process, and the QoS policies of the topic are also attached.

3.3.4. The Publication Module

The Publication Module consists of classes and interfaces that will be used for and help the publication process. These are: Publisher, DataWriter, PublisherListener, DataWriterListener.

3.3.5. The Subscription Module

The Subscription Module consists of classes and interfaces that will be used for and help the subscription process. These are Subscriber, DataReader, DataSample, SampleInfo, SubscriberListener, DataReaderListener, ReadCondition, and QueryCondition.

3.4. The DDS Entity and Interface Creations

The DomainParticipant Entity is produced by DomainParticipantFactory. DomainParticipant creates Publisher, Subscriber, Topic, and MultiTopic Entities as it works like a factory for them. DataWriter is created by the Publisher, and DataReader is created by the Subscriber. Based on the fact that an entity is created by what, it can be said that the creator works like a factory for what it creates. All objects in the tree, except for MultiTopic objects, belong to the Entity class. Fig. 2 shows the creation tree of some of the DDS Entities.

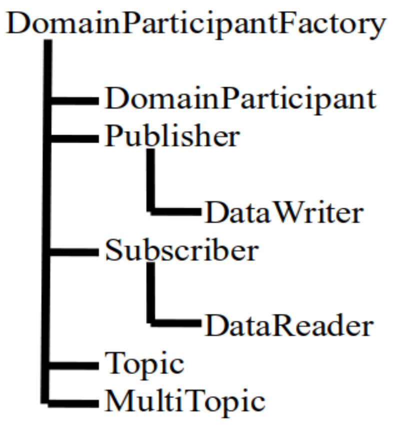

Figure 2. The creation tree of DDS Entities

Fig. 3 shows the creation tree of the Listener interfaces of the DDS.

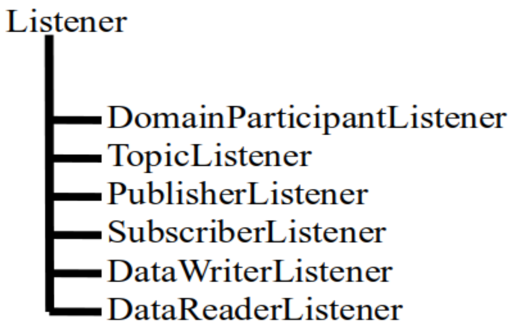

Figure 3. The creation tree of DDS Listener interfaces

The Listener interface, which helps entities monitor network traffic, is exclusive to the DDS and does not exist in the DDS-XRCE. The Listener interface takes place in the Infrastructure Module, and the Listener class of the Infrastructure Module works like an abstract root for other Listener interfaces. DomainParticipantListener, TopicListener, PublisherListener, SubscriberListener, DataWriterListener, and DataReaderListener are all derived from the root Listener and are coupled to the respective entity later. For example, DataReaderListener is attached to the respective DataReader after the creation process.

Although they are not considered entities or listener interfaces, there are objects of other classes that are assistive to the DDS system and operations. Moreover, some of them have no factory and are created directly. For example, the WaitSet object postpones processes of an application until some condition objects, which are coupled with the application, provide the necessary conditions.

3.5. The DDS Message Structure

Communication between DDS DomainParticipants (DDSDP) over DDS Global Data Space is maintained using RTPS protocol, which is a wire protocol for DDS participants, objects, and devices to communicate in a coordinated way [32]. The RTPS message structure consists of two parts: Header and Submessage. The message structure of an RTPS message is shown in Fig 4.

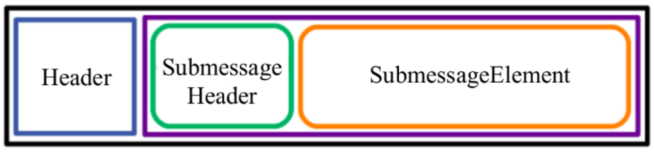

Figure 4. An RTPS message structure

Every RTPS message has to contain the Header part at the start. The Header carries information about the protocol, protocol version, vendorId, and guidPrefix, which is a prefix that is used for reconstruction later. All messages have submessage parts, and the number of submessages is different for messages. Submessage, which is shown as the purple rectangle, also consists of two parts: SubmessageHeader and SubmessageElement. The Header is a part that a submessage has to have and contains information about submessageId, flags, and submessageLength. SubmessageElements are building blocks that the system uses to build submessages. They are predefined.

RTPS version 2.2 defines some submessages that are categorized into two groups: Entity and Interpreter submessages. Entity submessages are summarised as follows: Data, DataFrag, Heartbeat, HeartbeatFrag, Gap, AckNack, and NackFrag. Interpreter submessages are summarised as follows: InfoSource, InfoDestination, InfoReply, InfoTimestamp, and Pad.

4. The DDS-XRCE Standard

The DDS-XRCE is a wire protocol following the client-server paradigm, which is a branch of DDS. The DDS-XRCE is adopted essentially to involve resource-constrained devices in the DDSGDS [32].

4.1. The Objects and Concepts of the DDS-XRCE

The DDS-XRCE has XRCE Clients and Agents in addition to the objects and concepts of the DDS.

An XRCE Client utilises the Agent by requesting publication, subscription, managing resources, etc. For example, XRCE Clients sleep and wake up periodically because they are resource-constrained devices. When an XRCE Client is in the sleep cycle, the Agent connected to the XRCE Client saves the messages to transmit them during its wake-up cycle.

The Agent acts as a server in DDS-XRCE and as a participant in DDS. It maintains the communication between XRCE-Clients and DDS Participants by connecting to the members of DDSGDS over DDSGDS and acts as a bridge. Making connections over DDSGDS is the ability of an Agent in the DDS-XRCE. Distributing resources, converting between protocols while transferring data, configuring parameters and profiles, and maintaining communication within the DDS-XRCE model are other duties and behaviours of the Agent.

4.2. The Integration Between the DDS and DDS-XRCE

Fig. 5 represents the integration between DDS-XRCE and DDS. In DDSGDS, peers of the DDSGDS do not categorise other peers, such as peers of the DDS model or peers of the DDS-XRCE model. Every peer seems solely like a DDS Participant although the Agent of the DDS-XRCE communicates with the other members of the DDSGDS.

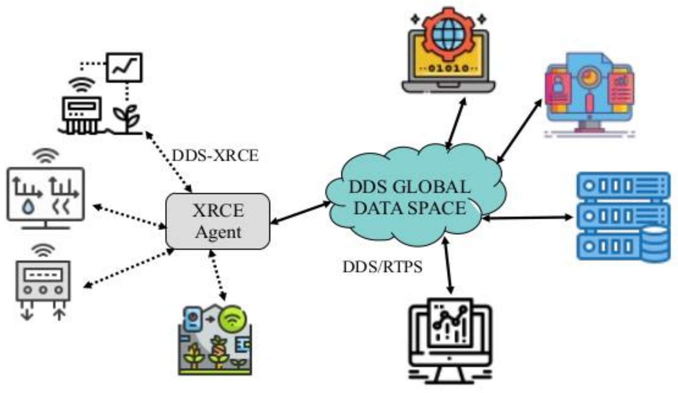

Figure 5. The integration between the DDS-XRCE and DDS

This integration can be explained simply by saying "plug and play" as a solution for compatibility issues. In DDS World, participants also have to be involved in a domain to access topic messages for that domain. The Agent makes the connections by creating a proxy DDSDP in the DDSGDS and keeps communication.

We have mentioned that DDSDPs monitor the network traffic continuously. Monitoring or processing the messages continuously requires a significant amount of resources. Accordingly, the devices which DDS is concerned with can be said to be highly equipped and advanced devices. However, the devices which the DDS-XRCE is concerned with are resource-constrained, as DDS-XRCE stands for Data Distribution Service for Extremely Resource-Constrained Devices. Furthermore, XRCE-Clients sleep and wake up periodically. While XRCE-Clients are in the sleep cycle, an Agent operates for them, such as storing topic messages for XRCE-Clients to transmit messages during their wake cycle. QoS policies, which DDS supports twenty-two of them, are also in common between standards. They are also supported by the DDS-XRCE. DDS-XRCE usually works with DDS. It can be said that the DDS-XRCE even requires work with the DDS to be implemented in some ways when one observes the deployments. Therefore, DDS-XRCE is mostly dependent on the DDS and it has to have the ability to work with the DDS properly. In addition, the DDS-XRCE defines ten profiles that provide configuration abilities to some extent, including the configuration of QoS policies for the XRCE Entities. Some of these profiles provide advanced abilities that give entities the authority to set parameters, such as configuring the QoS policy of the Topic. Having this kind of authority makes the clients advanced. Thus, it shows that although the main purpose of the DDS-XRCE is to provide access to resource-constrained devices from frequently used other devices, the DDS-XRCE is also concerned with advanced devices and/or clients. DDS-XRCE categorises devices as simple devices, more capable devices, advanced clients, and complex clients. Variation of devices and clients occurs due to differentiation in needs of XRCE-Clients.

4.3. The Structure of the DDS-XRCE

While the DDS model is similar to a star topology, it is different in the DDS-XRCE. Six formations that are called "deployments" occur due to the transmission path, kind of sender and receiver, and transmission protocol. Objects communicating with each other, the transmission path focused on deployments, which are illustrated in Fig. 6, are listed in Table 1.

In the transmission path column of Table 1, the ":::" refers to the DDS-XRCE protocol in which objects are used for communication. The "=" refers to the RTPS protocol in which objects are used for communication.

Table 1. Deployments list

\

No	Deployment Details	
	Objects	Transmission Path in Fig. 6
1	XRCE-Client and DDSDP	XRCE-Client-12:::Agent-8 =DDSDP-1
2	XRCE-Clients	XRCE-Client-11:::Agent-8=Agent-7 :::XRCE-Client-10
3	XRCE-Clients	XRCE-Client-10:::Agent-7=Agent-7 :::XRCE-Client-9
4	XRCE-Clients	XRCE-Client-1:::Agent-1:::Agent-3 =Agent-3:::Agent-2:::XRCE-Client-4
5	XRCE-Client and Agent	XRCE-Client-6 of Application1 :::Agent-6 of Application2
6	XRCE-Clients, Agents and DDSDPs	implementation of all or several paths at the same time

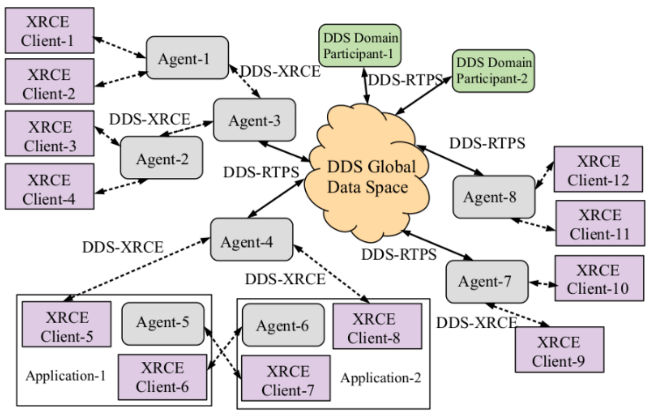

Figure 6. DDS-XRCE deployments and their transmission paths

In deployments, we encounter some characteristics worthy of mention. There is uncertainty about the transmission path of the third. Where the packets travel after being received by the Agent connected to the publisher XRCE Client is unclear. Three possible paths were listed for this uncertainty. 1) The Agent creates proxy DDS Entities for XRCE-Clients separately and maintains clients' communication as if different proxy DDSDPs communicate with each other. 2) The Agent creates one proxy DDSDP and maintains the communication as if the proxy DDSDP communicates with itself over DDSGDS. 3) The Agent creates a short path over itself for XRCE-Clients, which communicate with each other. The first and/or second possible paths are shown in Table 1.

The transmission paths of the third and fourth have one thing in common, which is an Agent managing different clients. Hence, encountering the uncertain transmission path of the third deployment is possible in the same way for the fourth deployment, as well.

In the fifth deployment, an application has to create an XRCE-Client to communicate with the Agents of other applications. Moreover, an application creates an Agent to maintain communication with XRCE-Clients of other applications. Each connection between applications can be accepted as a transmission path, so the transmission path of the fifth deployment may be multiple due to the formation of the fifth deployment on a system. Implementation of the fifth deployment is usually not suitable for resource-constrained devices. Because the application cannot sleep and wake periodically due to the Agent processing messages coming from XRCE-Clients. As an exception, the fifth deployment is the only implementation of the DDS-XRCE without any integration with the DDS model.

4.4. The DDS-XRCE Object Model

When one looks at the object model of DDS-XRCE, there are five classes. They are the Root singleton, ProxyClient, Application, AccessController, and DomainParticipant. In the DDS-XRCE, the object model does not contain modules and it does not have a particular name different from the DDS. At the highest level, it only includes classes.

4.4.1. Root Singleton

The Root singleton works like a factory for all the objects, and the Agent is in charge of these objects. Besides, the Root singleton is an entrance point to the system.

4.4.2. ProxyClient

When the XRCE-Client application and Agent communicate with each other over the XRCE protocol, the ProxyClient class represents the XRCE-Client application. Each Application object obtains the rights of a ProxyClient by being related to a single XRCE ProxyClient.

4.4.3. Application

The Application class represents a software application, which is in charge of the DDS objects used for publication and subscription processes on DDS Domains by associating with the XRCE-Client. An XRCE Application can be related to many DomainParticipants or none. Based on that fact, an XRCE Application can be active on many DDS Domains or none by using proxy objects.

4.4.4. AccessController

An XRCE ProxyClient has limited authority relating to resources and operations to function. This authority is determined and provided by AccessController for an XRCE ProxyClient since it holds the rules relating to a client with rights. These rights give the holders authority, such as choosing the DDS domain when an application intends to create and run proxy entities for a client, and deciding DDS topics when an application wants to publish and subscribe.

4.4.5. DomainParticipant

The DomainParticipant of DDS-XRCE, which works as a proxy in DDSGDS, represents the connection with a DDS Domain and what the Application can do running on that domain.

4.5. The DDS-XRCE Entity Creations

Fig. 7 shows the creation tree of DDS-XRCE objects and entities.

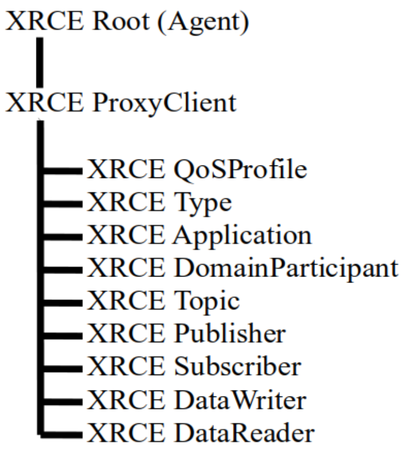

Figure 7. The creation tree of DDS-XRCE Entities

XRCE Root, which represents an agent, creates an XRCE ProxyClient. XRCE ProxyClient creates a QosProfile, Type, Application, DomainParticipant, Publisher, Subscriber, DataWriter, and DataReader. In DDS-XRCE, the Publisher does not create a DataWriter and the Subscriber does not create a DataReader, unlike in DDS. The only object working like a factory in DDS-XRCE is the Root singleton since it is responsible for all the objects controlled by Agent.

4.6. The DDS-XRCE Message Structure

The structure of a DDS-XRCE message is shown in Fig. 8.

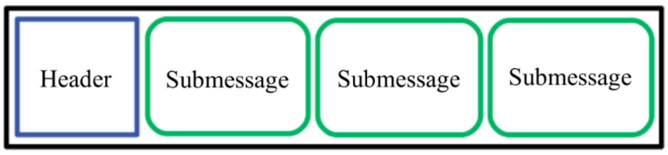

Figure 8. A DDS-XRCE message structure

A DDS-XRCE message contains Header and submessage parts. The Header carries information about the sessionId, streamId, sequence number, and clientKey. A submessage consists of submessageHeader and payload parts. SubmessageHeaders occur with submessageId, flags, and submessageLength. The payload provides information about the submessage according to submessageId. The DDS-XRCE submessage types are as follows: Create_Client, Create, Get_Info, Delete, Status_Agent, Status, Info, Write_Data, Read_Data, Data, Acknack, Heartbeat, Reset, Fragment, Timestamp, and Timestamp_Reply.

5. Methodology

Initially, we need an environment in which the DDS-XRCE has been used. We benefit from deployments for the formation of the test environment to determine the performance of the DDS-XRCE. Thus, we have decided to use the first, second, and third deployments.

The DDS-XRCE is a multi-functional one that has many parameters and provides many options to adjust the quality of communication. We decided to utilise the transport profile, stream mode, and network layer protocol features of DDS-XRCE for the test environment conditions. The transport profile is a choice that the DDS-XRCE provides to the user about the transportation protocol to transmit messages. The user decides if to use UDP/TCP, CAN FD, serial, or custom protocol by utilizing the transport profile feature. We utilised UDP and TCP protocols. As to streams, a stream is an independent flow of topic messages, and there are two kinds of streams in the DDS-XRCE protocol: reliable and best effort. The streams also take place as one of the QoS policies under the name of reliability. In best-effort streams, if the messages have been received or not, it is not controlled, whereas it is controlled in reliable streams. In addition, extra messages are sent to notify the sender of successful delivery in reliable streams. We have utilised both streams. At the network layer, IPv4 and IPv6 are provided for the user to choose from. We have utilised only IPv4.

5.1 Preparation Phase

Layouts of deployments given in the DDS-XRCE specification are provided as examples of the application of the DDS-XRCE. We have reformed the given deployments without corrupting their main focus by adding extra XRCE-Clients, Agents, and DDS participants to obtain a test environment.

Meanwhile, tracking the source, destination, and other information of messages has become difficult because of increasing client numbers and having only the HelloWorld topic, which will cause the same messages to circulate in the network. Thus, we created workspaces for different topics to distinguish messages. Workspaces contain a publisher, a subscriber, and other files for a specific topic. The creation of unalike topics is completed by following the instructions provided by eProsima about the usage of Micro-XRCE-DDS-Gen and Fast-DDS-Gen libraries.

Fig. 9 represents our reformation of the first deployment of DDS-XRCE for our study. Two Agents, four XRCE-Clients, and a DDSDP were utilised for the formation of Deployment-1.

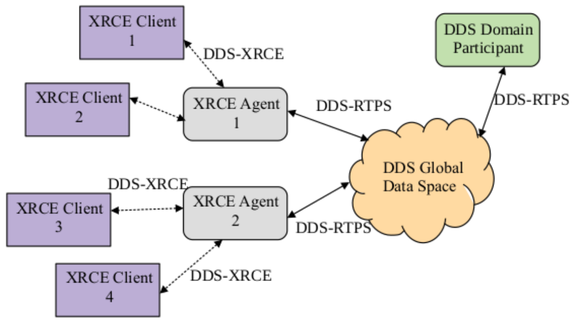

Figure 9. DDS-XRCE Deployment-1 scenario formation for our study

Agents are members of DDSGDS and are directly connected to their respective XRCE-Clients. They communicate over the RTPS protocol with other DDSDPs at DDSGDS and over the DDS-XRCE protocol with respective XRCE-Clients. They have different port numbers that help XRCE-Clients distinguish the respective Agents from each other when connecting. The port numbers of Agent-1 and Agent-2 are 2018 and 2019, respectively. Other objects of Deployment-1 are listed in Table 2 with the topic that they are concerned with, the objects that they connect with, and the objects that they communicate with.

Table 2. The objects of Deployment-1 scenario formation and their details

Objects	Deployment-1 Object Details			
	Pub./Sub.	Topic	Connect	Com. with
DDSDP	Pub.	Humidity	Agent-1 (proxy DDSDP)	XRCE-Client-2
XRCE-Client-1	Pub.	Temperature	Agent-1	-
XRCE-Client-2	Sub.	Humidity	Agent-1	DDSDP
XRCE-Client-3	Pub.	Altitude	Agent-2	-
XRCE-Client-4	Sub.	Pressure	Agent-2	-

We have focused on the communication between DDSDP and XRCE-Client-2 during tests of Deployment-1. They communicate with each other via the Agent connected to the respective XRCE-Client.

Fig. 10 represents our reformation of the second deployment of DDS-XRCE. Two Agents, seven XRCE-Clients, and three DDSDPs have been utilised for the Deployment-2 formation.

The features of the Agents of Deployment-2 are the same as the features of the Agents of Deployment-1. Other objects of Deployment-2 are listed in Table 3. The topics that they are concerned with, the objects that they connect with, and the objects that they communicate with are listed as well.

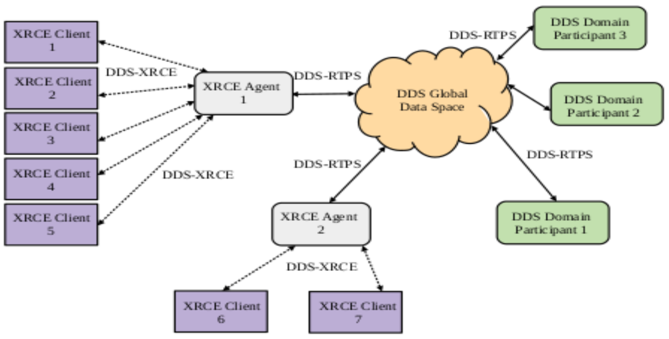

Figure 10. DDS-XRCE Deployment-2 scenario formation for our study

Table 3. The objects of Deployment-2 scenario formation and their details

Objects	Deployment-2 Object Details			
	Pub./Sub.	Topic	Connect	Com. with
DDSDP-1	Sub.	Helloworld	Agent-1 (proxy DDSDP)	XRCE-Client-1, XRCE-Client-2
DDSDP-2	Sub.	Humidity	Agent-1 (proxy DDSDP)	XRCE-Client-5
DDSDP-3	Sub.	Humidity	-	-
XRCE-Client-1	Pub.	Helloworld	Agent-1	DDSDP-1, XRCE-Client-3, XRCE-Client-4
XRCE-Client-2	Pub.	Helloworld	Agent-1	DDSDP-1, XRCE-Client-3, XRCE-Client-4
XRCE-Client-3	Sub.	Helloworld	Agent-1	XRCE-Client-1, XRCE-Client-2
XRCE-Client-4	Sub.	Helloworld	Agent-1	XRCE-Client-1, XRCE-Client-2
XRCE-Client-5	Pub.	Altitude	Agent-1	XRCE-Client-6, DDSDP-2
XRCE-Client-6	Sub.	Altitude	Agent-2	XRCE-Client-5
XRCE-Client-7	Sub.	Humidity	Agent-2	-

We have focused on the communication between XRCE-Client-5 and XRCE-Client-6 during tests of Deployment-2. XRCE-Client-5 and XRCE-Client-6 communicate with each other via their respective Agents.

Fig. 11 represents our reformation of the third deployment of DDS-XRCE for our study. An Agent, five XRCE-Clients, and two DDSDPs have been utilised for the Deployment-3 formation.

The features of the Agent of Deployment-3 are the same as Agent-1's features of Deployment-1. Other objects in Deployment-3 are listed in Table 4 with the topic that they are concerned with, the objects that they connect with, and the objects that they communicate with.

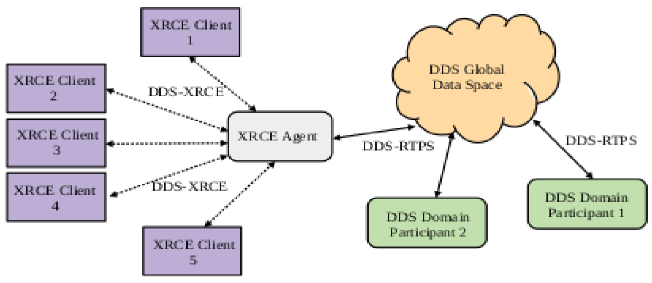

Figure 11. DDS-XRCE Deployment-3 scenario formation for our study

Table 4. The objects of Deployment-3 scenario formation and their details

\

Objects	Deployment-3 Object Details			
	Pub./Sub.	Topic	Connect	Com. with
DDSDP-1	Sub.	Helloworld	Agent (proxy DDSDP)	XRCE-Client-1, XRCE-Client-5
DDSDP-2	Sub.	Humidity	Agent (proxy DDSDP)	XRCE-Client-2
XRCE-Client-1	Pub.	Helloworld	Agent	DDSDP-1, XRCE-Client-3
XRCE-Client-2	Pub.	Humidity	Agent	DDSDP-2, XRCE-Client-4
XRCE-Client-3	Sub.	Helloworld	Agent	XRCE-Client-1, XRCE-Client-5
XRCE-Client-4	Sub.	Humidity	Agent	XRCE-Client-2
XRCE-Client-5	Pub.	Helloworld	Agent	DDSDP-1, XRCE-Client-3

We have focused on the communication between XRCE-Client-2 and XRCE-Client-4 during tests of Deployment-3. XRCE-Client-2 and XRCE-Client-4 communicate with each other via the same Agent, to which they are connected.

It will come to one's attention that some XRCE-Clients will not receive any messages on their respective topics. All subscribers may not receive messages continuously, even in real-time applications. Consequently, XRCE-Clients who do not receive any messages will not affect experiments.

We emphasise that transport profile options, streams, and network layer protocols are configurable via source codes of publisher and subscriber of DDS-XRCE using respective functions. These options have to be chosen and configured before compilation. Thus, we have created publishers and subscribers for each topic to send and receive messages over the UDP protocol at the transport layer and the IPv4 protocol at the network layer in the reliable stream. Also, it goes the same for transport profile-stream mode pairs like UDP-BE-IPv4, TCP-R-IPv4, and TCP-BE-IPv4.

Since these are test environment conditions, it means that each deployment will be tested according to four different conditions. We indicate that while tests were carried out under a respective condition, for example, UDP-R-IPv4, all XRCE-Clients in the deployment have been configured to transport messages over UDP protocol at the transport layer and IPv4 protocol at the network layer in the reliable stream.

However, we have encountered a function dissimilarity at further stages of the UDP-BE-IPv4 tests. Because of the function dissimilarities, we have written additional codes for publishers and subscribers, which we focused on their communication, running according to the UDP-BE-IPv4 pair. The purpose of these actions will be explained in the analysis phase of the methodology.

5.2 Simulation and Data Collection Phase

We have twelve different scenarios in total, and all software products are ready to run. Fig. 12 shows the steps of the simulation. All experiments have been conducted on a computer operated by Ubuntu 18.04.6 LTS.

As the first step of the simulation, source code directories of Publishers, Subscribers, Agents, and DDSDPs were set on multiple Gnome Terminator windows, and all commands, which will make Wireshark and all units of the deployment work, have been written on each respective terminal.

In the second step, Wireshark was run through the command line to track messages in the network and configured through its interface to record them for analysis.

We run Agents, later XRCE-Clients, which we have not focused on, and DDSDPs in order when it comes to the third step.

In the fourth step, we ran Subscriber and Publisher, in which we focused on their communication with each other, in order through the command line.

Then we stopped Publisher and Subscriber through the command line after enough messages had been received.

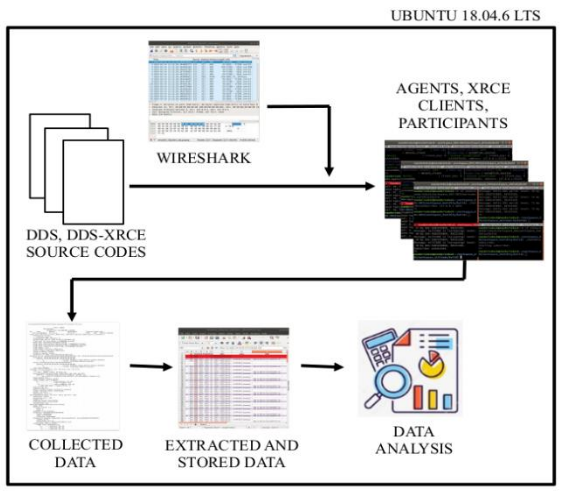
UBUNTU 18.04.6 LTS
WIRESHARK
AGENTS, XRCE CLIENTS, PARTICIPANTS
DDS, DDS-XRCE SOURCE CODES
COLLECTED DATA
EXTRACTED AND STORED DATA
DATA ANALYSIS

Figure 12. Steps of our evaluation study

We have limited the messages to be transferred to 50, 100, and 150 packets to decide if enough messages were received. In the final step, we saved the simulation data for analysis. We obtained data from thirty-six tests shown in Table 5 by the end of the simulation and data collection phase.

Table 5. Tests and their limits

Cond.	Deployments								
	Deployment-1			Deployment-2			Deployment-3		
UDP-R	50	100	150	50	100	150	50	100	150
UDP-BE	50	100	150	50	100	150	50	100	150
TCP-R	50	100	150	50	100	150	50	100	150
TCP-BE	50	100	150	50	100	150	50	100	150

5.3 Analysis Phase

Latency is described as the time needed to transfer some data from one end to another. Hence, we needed to know when to start and end the transfer of each message to measure network latency. We used Wireshark to monitor the network and could track network packets by knowing their timestamp and size.

However, Wireshark adds the beginning time of sending each packet as a timestamp with various formats. Still, we had to know the ending time to measure latency. As a function of reliable streams, when a message is successfully transferred, the receiver sends a confirmation message back to the sender. Since to send a confirmation message, another message needs to be transferred successfully earlier. Based on that fact, we assumed that we could utilise the timestamp of the receiver's confirmation message that the receiver sends back to the sender as the ending time of the data transfer. We observed that confirmation messages still exist in TCP-BE scenarios, although they use a best-effort stream. The time of the received message refers to the future relative to the time of the sent message. We calculated the duration between the times of the received messages and the sent messages. Each duration value represents the latency of packets. The latency of a packet is shown as follows,

$$L = T_{CM} - T_{DP} \tag{1}$$

T_{CM} is the time of the confirmation message, which the receiver sends back to the sender; T_{DP} is the time of the data packet sent initially; and L is the latency of a packet, which the difference between T_{CM} and T_{DP} gives in Equation (1). Nevertheless, there are UDP-BE scenarios that need to be tested. During the simulations, it is observed that there is no confirmation message for UDP-BE scenarios to determine the ending time of messages. This is what we mentioned in the preparation phase as the function dissimilarity of the UDP-BE tests. As a solution, initially, we analysed the source codes of publishers and subscribers and added some code between specific lines. When it is run, the code returns the time in seconds and minor values than seconds since the Epoch.

For publisher source code, extra code calculating the time was written right before the code line sent the data message, and extra code returning the time, which was calculated earlier, was written right after the code line sent the data message. For subscriber source code, extra code calculating and returning the time was written right after the code line that publishes the message that the subscriber received. Aiming to look like data messages come first, and timestamps of the data messages come second in the terminal window.

Again, one timestamp refers to the future according to the other. The duration between the time of the sent and the received messages was calculated for all transferred packets. Thus, we obtained the latency of each message in UDP-BE scenarios.

Throughput answers the question of how much data is transferred successfully from one point to the other in the network for a particular period. It is measured in bits per second. Nevertheless, there is confusion when it comes to throughput and bandwidth. The bandwidth corresponds to the maximum throughput.

Wireshark monitored network traffic during all of our scenarios. It has herewith provided for our study the size of packets as bytes for each message. However, a packet size may change several times on the transmission path due to protocol conversions. Fig. 13 shows Deployment-2 with some points on the transmission path we have focused on and information about the thirty-fourth packet transferred in Deployment-2 with TCP-R-50-p. The first row in the table is packet information that was transferred from Point A to Point B over the DDS-XRCE protocol. The second row in the table is information on the confirmation message, which was sent back from B to A for the packet that was sent earlier. The third row in the table is packet information that was transferred from Point B to Point C over the RTPS protocol.

The fourth row in the table is packet information that was transferred from Point C to Point D over the DDS-XRCE protocol. The fifth row in the table is the confirmation message, which informs the sender about successful delivery. Point A is where the transmission starts. Point B is where the DDS-XRCE protocol is converted to the RTPS protocol. Point C is where the RTPS protocol is converted to the DDS-XRCE protocol. Point D is where the transmission ends.

D2 TCP-RELIABLE 34/50

PROTOCOL	SOURCE	DESTINATION	SIZE
TCP	A	B	185
TCP	B	A	66
RTPS	B	C	166
TCP	C	D	110
TCP	D	C	66

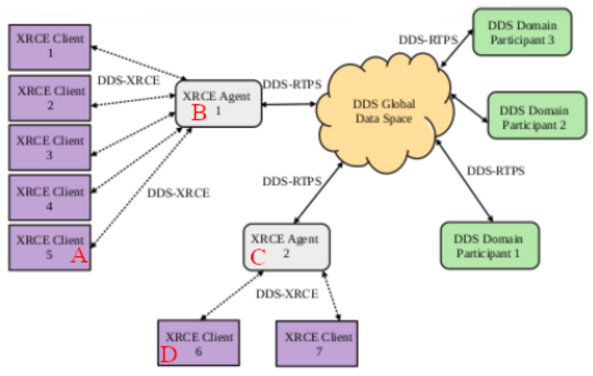

Figure 13. Protocol and size information of a packet when a message is published

The values of the “size” column show the sizes of the packets transferred from Point A to Point B, B to C, and C to D, respectively. The change in the size of the packet occurs due to protocol conversion. A successful delivery has to happen between a sender and a receiver, and the packet size must be known to calculate throughput. If the latency of a delivered packet can be thought of as the time spent by a vehicle that goes from one point to another. If total packet size can be thought of as the total length of the road that the vehicle goes on. The calculation of throughput can be associated with the calculation of the vehicle's velocity. Therefore, the throughput has been calculated using this similarity. The throughput is shown as follows in Equation (2),

$$Th = \sum_{n=1}^{N} \left(\frac{S_{AB} + S_{BC} + S_{CD}}{L} \right) \qquad (2)$$

The total size of the packet for the calculation of the throughput is obtained by summing up all sizes of the packet between protocol conversions. S_{AB} is the size of the packet transferred from A to B. S_{BC} is the size of the packet transferred from B to C. S_{CD} is the size of the packet transferred from C to D. L is the latency of the packet, and the total size of the packet is divided by the latency of the respective packet to obtain a value in bits per second (bps). We have calculated the bps values for each successfully transferred packet on the transmission path on which we have focused. In this context, N refers to the number of successful deliveries on the transmission path. Eventually, the sum of the bps values of the packets on the transmission path gives the total throughput, as shown as Th in Equation (2).

6. Used Tools

6.1. eProsima Software Products

eProsima provides networking, high-performance middleware solutions. As a member of OMG, eProsima creates and implements middleware standards. These are DDS, RTPS, DDS-XRCE, CDR, RPC over DDS, etc. In addition, eProsima publishes the source codes of some products they provide on Github under the Apache 2.0 licence.

The DDS-XRCE is the standard we mainly focused on in this study. While evaluating the performance of the DDS-XRCE on network latency and throughput, we have used some of the other products eProsima provides. It is important to state that we ran all performance evaluation tests in an environment that is run by the Ubuntu 18.04 LTS operating system. Thus, the installation process of all eProsima products is followed according to the installation manual for Ubuntu [34].

6.1.1. eProsima Micro-XRCE-DDS-Agent v2.0.0

Client and server communication is essential in the DDS-XRCE protocol. The server is represented by an agent whose function is to become a bridge between XRCE-Clients and the DDS world to ensure safe and secure communication. In more detail, the agent receives messages from the DDS world to transmit them to XRCE-Clients and receives messages from XRCE-Clients to transmit them to the DDS world.

The library implementation of the agent in the DDS-XRCE protocol is the Micro-XRCE-DDS-Agent source code, which is provided by eProsima on Github. This implementation allows devices like microcontrollers and microsensors to communicate with the DDS world. Also, a feature of agent library implementation is the ability to provide some built-in transports such as UDPv4, UDPv6, TCPv4, TCPv6, and Serial communication. Communication between the XRCE-Client library and Agent library is implemented via built-in transports aforementioned [13]. In our study, we used the Micro-XRCE-DDS-Agent source code without making any changes.

6.1.2. eProsima Micro-XRCE-DDS-Client v2.0.0

We mentioned that client and server communication is essential in the DDS-XRCE protocol and what the function of the Agent is. Thus, it is clear that XRCE-Clients are the other transmission end while the Agent is communicating with the DDS world. The Agent publishes and subscribes to topics on behalf of XRCE-Clients according to their requests. Entities like Topics, Publishers, Participants, and Subscribers that may be needed by XRCE Clients in the DDS world are created by an assigned ProxyClient.

The Micro-XRCE-DDS-Client library is configurable. It has changeable features such as profile options, which can be enabled or disabled by changing some CMake flags. It has some other parameters that can be used to control the capabilities of the library. Moreover, the XRCE-Client library provides the built-in transports as the Agent library does [13].

In our study, we have benefited from these changeable features. We needed publishers and subscribers to publish and subscribe to different topics for our scenarios. As a result, it is necessary to make some changes to the source code for our study. We utilised the Micro-XRCE-DDS-Client library to get a better understanding of how the library works and to decide which parameters to choose or use during our trial process.

6.1.3. eProsima Micro-XRCE-DDS-Gen

Micro-XRCE-DDS-Gen is a tool that is used to generate topics and some supplement files. The code generated by using Micro-XRCE-DDS-Gen cannot be generated without the Micro CDR library. The tool generates the topics by using an IDL file as a source file and a Micro CDR library. Thus, the only dependency this library has is on the Micro CDR library [13].

We created IDL files for different topics to create different topic workspaces. These are humidity, temperature, pressure, altitude, and helloworld. After the generation was completed, publisher and subscriber files were edited as needed and compiled. We had applications on different topics whose publishers and subscribers have different transportation profiles and stream modes in the end for the DDS-XRCE.

6.1.4. eProsima Fast-DDS v2.4.0

Fast-DDS is the source code that eProsima provides for the DDS. Previously, it was known as Fast RTPS, where RTPS stands for Real Time Publish Subscribe. RTPS is a wire protocol that maintains communication over some transports and was produced for DDS. The Fast-DDS library provides the implementation of the RTPS protocol and full access to its full functionalities [13]. In our study, this implementation was used to understand how the product works.

6.1.5. eProsima Fast-DDS-Gen

Similar to the Micro-XRCE-DDS-Gen library, this software product is a code generator tool from an IDL file [13]. The difference between them is that while the Micro-XRCE-DDS-Gen tool is for the DDS-XRCE, the Fast-DDS-Gen tool is for the DDS. The code obtained from the Fast-DDS-Gen implementation by generating can work in every Fast-DDS application without having any extra features [35].

Creating DDS workspaces is similar to creating DDS-XRCE workspaces. Firstly, we made an IDL file which will be the source of the topic workspace. The Fast-DDS-Gen tool was run by indicating from which IDL file to generate. After the generation was completed, we obtained the publisher, subscriber, and some other files. As the last step, we edited and compiled publisher and subscriber files. Finally, we created different topic workspaces with publishers and subscribers as needed for the DDS.

6.2. Wireshark

Wireshark is a network protocol analyser that can run on various operating systems and other platforms [36]. It captures packets from the network and allows us to monitor the network traffic in real-time. We can save what we captured as a pcapng file, which Wireshark 1.8 and later generate by default. Pcapng files can be used to store packet details in it as other formatted files or for any other purpose.

Wireshark was used to retrieve network traffic data during the simulation. After saving what we retrieved as a pcapng file, we printed all captured packets to extract the necessary features. These are the protocol, source port, destination port, length of the packet, and timestamp. The extracted features were initially used to tell messages we focused on from other messages in the network traffic and, secondly, used to calculate network latency and throughput.

6.3. Gnome Terminator

It is a command prompt that was developed as a Python script. It has the functionality of running multiple terminals in the same window. The user can benefit from the multiple terminal features by splitting the window. Also, the split windows can be combined later [37].

In our study, we needed an environment to be able to show multiple topic messages at the same time. Hence, we used Gnome Terminator to run many Agents, Publishers, and Subscribers and see network traffic messages. After the ordering layout of the terminals was completed, we ran the codes in the terminals accordingly to our test plan and saw the network traffic messages as the participants saw them.

7. Results

We measured the latency of packets, the packet sizes, the number of packets sent, and the number of packets received. The other criteria were calculated according to the information obtained from the experiments.

The Min in the tables refers to the minimum of the respective measurement. The Max refers to the maximum of the respective measurement. The Avg. refers to the average of the respective measurements. Std. Dev. refers to the standard deviation of the respective measurement. Var. refers to the variance of the respective measurement. The Total in the tables refers to the total packet size of successful delivery. The number sent in the tables refers to the number of packets that were sent. The number of recv. in the tables refers to the packets that were received. In other words, it refers to the number of successful deliveries. The results are shown in the tables.

Latency results of Deployment-1 are presented in Table 6. The test with UDP-BE-150p conditions had the lowest average latency of all, which is interpreted as possibly performing the fastest among the tests with 150-packet deliveries. The test with TCP-R-100p conditions had the highest average latency of all, which is interpreted as possibly performing the slowest among the tests with 100-packet deliveries. The lowest minimum latency was observed during the test with UDP-BE-150p conditions. The highest maximum latency was observed during the test with TCP-R-100p conditions. Latency results of Deployment-3 are presented in Table 8. The test with TCP-BE-150p conditions had the lowest average latency of all, which is interpreted as possibly performing the fastest among the tests with 150-packet deliveries. The test with TCP-R-100p conditions had the highest average latency of all, which is interpreted as possibly performing the slowest among the tests with 100-packet deliveries. The lowest minimum latency was observed during the test with TCP-BE-50p conditions. The highest maximum latency was observed during the test with TCP-R-100p conditions. The test with TCP-R-150p conditions had the lowest average throughput of all, which is interpreted as possibly transmitting the least data periodically among the tests with 150-packet deliveries. The test with UDP-BE-150p conditions had the highest average throughput of all, which is interpreted as possibly transmitting the most data periodically among the tests with 150-packet deliveries. The lowest minimum throughput was observed during the test with TCP-R-100p conditions. The highest maximum throughput was observed during the test with UDP-BE-150p conditions. Throughput results of Deployment-1 are presented in Table 9. Throughput results of Deployment-2 are presented in Table 10. The test with TCP-R-100p conditions had the lowest average throughput of all, which is interpreted as possibly transmitting the least data periodically among the tests with 100-packet deliveries. The test with TCP-BE-50p conditions had the highest average throughput of all, which is interpreted as possibly transmitting the most data periodically among the tests with 50-packet deliveries. The lowest minimum throughput was observed during the test with TCP-R-50p conditions. The highest maximum throughput was observed during the test with TCP-R-100p conditions. The test with UDP-BE-50p conditions had the lowest average throughput of all, which is interpreted as possibly transmitting the least data periodically among the tests with 50-packet deliveries. The test with TCP-BE-150p conditions had the highest average throughput of all, which is interpreted as possibly transmitting the most data periodically among the tests with 150-packet deliveries. The lowest minimum throughput was observed during the test with TCP-R-100p conditions. The highest maximum throughput was observed during the test with TCP-BE-50p conditions. Throughput results of Deployment-3 are presented in Table 11. Packet length results of Deployment-1 are presented in Table 12. The tests with UDP-R and UDP-BE conditions had the lowest average data packet of all, which are interpreted as transmitting the least data among the tests with their packet deliveries. The test with TCP-R-50p conditions had the highest average data packet of all, which is interpreted as transmitting the most data among the tests with 50-packet deliveries. Any packet loss was not observed during the tests of Deployment-1. The tests with UDP-R-50p, UDP-BE-50p, and UDP-BE-100p conditions had the lowest average data packet of all, which are interpreted as transmitting the least data among the tests with their packet deliveries. The test with TCP-R100p conditions had the highest average data packet of all, which is interpreted as transmitting the most data among the tests with 100-packet deliveries. The test with TCP-R-150p conditions had the highest packet loss of all. The loss is three packets. The packet loss of the other tests varies between zero and two. Packet length results of Deployment-2 are presented in Table 13. The tests with UDP-R and UDP-BE conditions had the lowest average data packet of all, which are interpreted as transmitting the least data among the tests with their packet deliveries. The test with TCP-R-150p conditions had the highest average data packet of all, which is interpreted as transmitting the most data among the tests with 150-packet deliveries. All tests of Deployment-3 resulted in one packet loss. Packet length results of Deployment-3 are presented in Table 14.

Table 7. Latency results of Deployment-2

Criteria	The Conditions											
	UDP-R	UDP-R	UDP-R	TCP-R	TCP-R	TCP-R	UDP-BE	UDP-BE	UDP-BE	TCP-BE	TCP-BE	TCP-BE
Limit (packet)	50	100	150	50	100	150	50	100	150	50	100	150
Min (ms)	0.387	0.301	0.274	0.249	0.248	0.238	0.272	0.318	0.248	0.251	0.272	0.236
Max (ms)	0.873	1.175	0.902	48.129	48.427	48.800	1.467	2.661	1.385	0.630	0.917	1.124
Avg. (ms)	0.656	0.615	0.617	14.792	15.002	14.689	0.558	0.731	0.573	0.404	0.570	0.555
Std. Dev. (ms)	0.111	0.135	0.128	20.985	20.537	20.435	0.188	0.242	0.193	0.085	0.120	0.136
Var. (ms2)	0.012	0.018	0.016	440.379	421.788	417.588	0.035	0.058	0.037	0.007	0.014	0.019

Table 8. Latency results of Deployment-3

\

Criteria	The Conditions											
	UDP-R	UDP-R	UDP-R	TCP-R	TCP-R	TCP-R	UDP-BE	UDP-BE	UDP-BE	TCP-BE	TCP-BE	TCP-BE
Limit (packet)	50	100	150	50	100	150	50	100	150	50	100	150
Min (ms)	0.261	0.252	0.252	0.242	0.238	0.253	0.303	0.256	0.254	0.167	0.201	0.169
Max (ms)	1.440	0.846	0.890	48.112	48.589	47.084	0.808	0.807	1.469	0.682	0.761	0.983
Avg. (ms)	0.576	0.571	0.579	15.058	15.274	14.584	0.628	0.606	0.550	0.450	0.493	0.444
Std. Dev. (ms)	0.200	0.115	0.111	21.105	20.885	20.099	0.133	0.148	0.183	0.106	0.100	0.128
Var. (ms2)	0.040	0.013	0.012	445.402	436.164	403.954	0.018	0.022	0.033	0.011	0.010	0.016

Table 9. Throughput results of Deployment-1

\

Criteria	The Conditions											
	UDP-R	UDP-R	UDP-R	TCP-R	TCP-R	TCP-R	UDP-BE	UDP-BE	UDP-BE	TCP-BE	TCP-BE	TCP-BE
Limit (packet)	50	100	150	50	100	150	50	100	150	50	100	150
Min (Mbps)	3.294	3.181	2.648	0.051	0.048	0.049	3.608	3.040	4.782	3.146	3.686	3.210
Max (Mbps)	9.997	14.233	12.447	17.357	15.117	13.771	16.027	14.506	18.885	15.326	17.225	15.747
Avg. (Mbps)	5.436	6.115	5.897	5.441	4.744	4.636	6.602	6.353	8.638	7.615	6.796	7.062
Std. Dev. (Mbps)	1.234	1.901	2.010	4.683	3.772	3.578	2.213	1.839	2.130	2.365	1.948	2.361
Var. ((Mbps)2)	1.523	3.613	4.039	21.931	14.232	12.803	4.896	3.383	4.537	5.592	3.794	5.575

Table 10. Throughput results of Deployment-2

\

Criteria	The Conditions											
	UDP-R	UDP-R	UDP-R	TCP-R	TCP-R	TCP-R	UDP-BE	UDP-BE	UDP-BE	TCP-BE	TCP-BE	TCP-BE
Limit (packet)	50	100	150	50	100	150	50	100	150	50	100	150
Min (Mbps)	3.096	2.300	2.998	0.077	0.064	0.063	1.843	1.016	1.952	4.901	3.368	2.747
Max (Mbps)	6.992	8.997	9.869	12.426	12.462	12.999	9.947	8.491	10.883	12.295	11.368	13.062
Avg. (Mbps)	4.263	4.683	4.651	4.259	4.220	4.554	5.247	3.988	5.248	7.995	5.693	5.999
Std. Dev. (Mbps)	0.874	1.377	1.372	3.601	3.319	3.581	1.381	1.188	1.718	1.795	1.430	1.919
Var. ((Mbps)2)	0.765	1.897	1.881	12.964	11.016	12.821	1.908	1.411	2.951	3.221	2.045	3.681

Table 11. Throughput results of Deployment-3

\

Criteria	The Conditions											
	UDP-R	UDP-R	UDP-R	TCP-R	TCP-R	TCP-R	UDP-BE	UDP-BE	UDP-BE	TCP-BE	TCP-BE	TCP-BE
Limit (packet)	50	100	150	50	100	150	50	100	150	50	100	150
Min (Mbps)	0.956	1.626	1.547	0.049	0.036	0.037	1.704	1.705	0.937	2.580	2.311	1.791
Max (Mbps)	5.264	5.469	5.459	7.262	7.403	8.070	4.544	5.368	5.425	10.543	8.753	10.445
Avg. (Mbps)	2.680	2.541	2.494	2.458	2.431	2.610	2.335	2.468	2.773	4.233	3.810	4.371
Std. Dev. (Mbps)	0.987	0.700	0.640	1.999	1.890	2.144	0.701	0.859	0.917	1.482	1.249	1.544
Var. ((Mbps)2)	0.974	0.490	0.409	3.996	3.573	4.599	0.492	0.738	0.841	2.197	1.561	2.383

Table 12. Packet length results of Deployment-1

\

Criteria	The Conditions											
	UDP-R	UDP-R	UDP-R	TCP-R	TCP-R	TCP-R	UDP-BE	UDP-BE	UDP-BE	TCP-BE	TCP-BE	TCP-BE
Limit (packet)	50	100	150	50	100	150	50	100	150	50	100	150
Avg. (bit)	2144	2144	2144	2357.6	2336	2345.6	2144	2144	2144	2336	2336	2336
Total (bit)	107200	214400	321600	117880	233600	351840	107200	214400	321600	116800	233600	350400
Number of Sent	50	100	150	50	100	150	50	100	150	50	100	150
Number of Recv.	50	100	150	50	100	150	50	100	150	50	100	150

Table 13. Packet length results of Deployment-2

\

Criteria	The Conditions											
	UDP-R	UDP-R	UDP-R	TCP-R	TCP-R	TCP-R	UDP-BE	UDP-BE	UDP-BE	TCP-BE	TCP-BE	TCP-BE
Limit (packet)	50	100	150	50	100	150	50	100	150	50	100	150
Avg. (bit)	2704	2706.6	2704.9	3287.2	3296.5	3296.2	2704	2704	2706	3088	3088	3088
Total (bit)	132496	267952	400320	164360	326352	484536	132496	267696	403200	151312	305712	460112
Number of Sent	50	100	150	50	100	150	50	100	150	50	100	150
Number of Recv.	49	99	148	50	99	147	49	99	149	49	99	149

Table 14. Packet length results of Deployment-3

\

Criteria	The Conditions											
	UDP-R	UDP-R	UDP-R	TCP-R	TCP-R	TCP-R	UDP-BE	UDP-BE	UDP-BE	TCP-BE	TCP-BE	TCP-BE
Limit (packet)	50	100	150	50	100	150	50	100	150	50	100	150
Avg. (bit)	1376	1376	1376	1963.3	1961.2	1970.2	1376	1376	1376	1760	1760	1760
Total (bit)	67424	136224	205024	96200	194160	293560	67424	136224	205024	86240	174240	262240
Number of Sent	50	100	150	50	100	150	50	100	150	50	100	150
Number of Recv.	49	99	149	49	99	149	49	99	149	49	99	149

[34] enounces that TCP-BE streams perform similar behaviour to that of UDP-R streams. Thus, for example, when one looks at the latency levels of deployments on different transport profile-stream mode pairs, the proximate measures in TCP-BE and UDP-R of respective deployments support the accuracy of results.

8. Conclusions

After some experiments, we came to a few conclusions by observing the behaviour of XRCE-Clients, Agents, and DDSDPs and analyzing the results.

If we think of the transmission path we focused on in Deployment-1, we tested it in the aspect of subscription due to the direction of the transmission. Because XRCE-Client is a subscriber. When we think of the successful delivery, every delivery for all experiments of Deployment-1 was completed successfully. Consequently, it can be said that packets go finely to the destination without encountering any problems or loss in the aspect of subscription for the first conclusion.

As to the second conclusion, we need to remember the transmission path of Deployment-2. The transmission path is the same from the standpoints of both ends. The connection between XRCE-Clients is symmetrical due to the formation of Deployment-2, and it allows us to test the DDS-XRCE from the perspective of publication and subscription. It means there is an XRCE-Client that sends the packets, and there is another XRCE-Client that receives them. When we look at the number of successful deliveries, a few packets were not received by the subscriber. Although all packets were sent by the publisher XRCE-Client in the experiments of Deployment-2. When we examined the messages, the lost packets were lost after they were received by the Agent connected to the publisher XRCE-Client. Hence, the first conclusion we obtained from experiments of Deployment-1 is also valid for experiments of Deployment-2. Because the subscriber side would have received all the packets if the publisher side could have sent all of them. Furthermore, the lost packets were always in the first three. It seems as if the Agent does not realise that it has to send the packets at the moment that it receives the first few packets. When it starts to send, the Agent sends the rest of the packets. Thus, it can be said that the first few packets might not be sent by the Agent of publisher XRCE-Client in the aspect of publication for the second conclusion.

Moreover, we encountered an integration pattern during Deployment-2 experiments by examining network messages, and it is related to the topic on whose messages we focused. When we think of DDSGDS at the centre, there are two Agents and a DDSDP connected over DDSGDS due to the concern of the topic on whose messages we focused. We added the DDSDP as a subscriber to the experiments of Deployment-2 to observe the topic messages circulating in the DDSGDS. After the examination of the network messages, we observed that the Agent connected to the publisher XRCE-Client sends the packets in two different series of messages to two different DDSDPs over DDSGDS. It is important to remember for the next conclusion.

The first and second conclusions are also valid for Deployment-3. However, the transmission path was unclear. We added a DDSDP as a subscriber to the experiments of Deployment-3 for observation. During analyzing the network messages, we encountered only one series of messages, which carries the data of the respective topic and carries the data using the RTPS protocol. However, there were two series of messages for experiments in Deployment-2 due to two subscribers, and two subscribers existed in Deployment-3 as well. It means the Agent never transferred the packets over DDSGDS for two subscriber members of DDSGDS while maintaining the communication between XRCE-Clients. It did for one member of DDSGDS, which is DDSDP. It maintained communication between XRCE-Clients by creating a short path. The path that the Agent chose is the third conclusion of our study.

When all the conclusions are considered one more time, the existence of the fourth conclusion is highly likely. After the Deployment-2 tests, the Agent's packet loss is possible for other scenarios. Observation of the Agent losing packets during Deployment-3 tests makes this possibility more realistic. Therefore, although Deployment-1 has not been tested while the Agent is connected to a publisher XRCE Client, the Agent might act the same way during Deployment-

1 tests. The Agent's possible behaviour regarding packet loss for Deployment-1 is the fourth conclusion.

References

[1] J. Holler, V. Tsiatsis, C. Mulligan, S. Karnouskos, S. Avesand and D. Boyle. From Machine-to-Machine to the Internet of Things - Introduction to a New Age of Intelligence. UK: Academic Press; 2014. pp. 14.

[2] J. J. Wang, R. Payne. A survey of Internet of Things in Healthcare. EAI Endorsed Transactions on Internet of Things. 2022; 7(27).

[3] A. A. Laghari, K. Wu, R. A. Laghari, M. Ali, A. Ayub Khan. A Review and State of Art of Internet of Things (IoT). Archives of Computational Methods in Engineering. 2021; 29(2).

[4] Sheng Huang, Yu-Hsuan Lu, M. Shafiq, A. A. Laghari, and R. Yadav. A Generative Adversarial Network Model Based on Intelligent Data Analytics for Music Emotion Recognition under IoT. Mobile Information Systems. 2021; 2021(1).

[5] A. A. Laghari, Hui He, A. Khan, R. A. Laghari, Shoulin Yin, and Jiachi Wang. Crowdsourcing Platform for QoE Evaluation for Cloud Multimedia Services. Computer Science and Information Systems. 2022; 00:38-38.

[6] M. Waqas, K. Kumar, A. A. Laghari, U. Saeed, M. M. Rind, A. A. Shaikh, F. Hussain, A. Rai, and A. Q. Qazi. Botnet attack detection in Internet of Things devices over cloud environment via machine learning. Concurrency and Computation. 2021; 34(5):1-23.

[7] A. H. Farea and K. Küçük. Detections of IoT Attacks via Machine Learning-Based Approaches with Cooja. EAI Endorsed Trans IoT. 2022; 7(28): e1.

[8] G. Bouloukakis, N. Georgantas, A. Kattepur, and V. Issarny. Timed protocol analysis of interconnected mobile IoT devices. Journal of Internet Services and Applications. 2021; 12(12): 1-31.

[9] Z. Kang, K. An, A. Gokhale and P. Pazandak. A Comprehensive Performance Evaluation of Different Kubernetes CNI Plugins for Edge-based and Containerized Publish/Subscribe Applications. 2021 IEEE International Conference on Cloud Engineering (IC2E); 2021; USA.

[10] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari and M. Ayyash. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. IEEE Communications Surveys & Tutorials. 2015; 17(4): 2347-2376.

[11] J. M. Schlesselman, G. Pardo-Castellote and B. Farabaugh. OMG data-distribution service (DDS): architectural update. IEEE MILCOM 2004. Military Communications Conference; 2004; USA. IEEE; 2004. p. 961-967.

[12] C. Bayılmış, M. A. Ebleme, Ü. Çavuşoğlu, K. Küçük, A. Sevin. A survey on communication protocols and performance evaluations for Internet of Things. Digital Communications and Networks. 2022.

[13] eProsima. “Repositories”. Available from: https://github.com/orgs/eProsima/repositories

[14] P. Phueakthong and J. Varagul. A Development of Mobile Robot Based on ROS2 for Navigation Application. 2021 International Electronics Symposium (IES); 2021; Indonesia. IEEE; 2021. p. 517-520.

[15] S. Dehnavi, D. Goswami, M. Koedam, A. Nelson and K. Goossens. Modeling, implementation, and analysis of XRCE-DDS applications in distributed multi-processor real-time embedded systems. 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE); 2021; France. Institute of Electrical and Electronics Engineers; 2021. p. 1148-1151.

[16] Chul-Hwan Kim, Gunjae Yoon, Wonjoon Lee, Jungdo Park and Hoon Choi. A performance simulator for DDS networks. 2015 International Conference on Information Networking (ICOIN); 2015; Cambodia. IEEE; 2015. p. 122-126.

[17] K. Krinkin, A. Filatov, A. Filatov, O. Kurishev and A. Lyanguzov. Data Distribution Services Performance Evaluation Framework. 2018 22nd Conference of Open Innovations Association (FRUCT); 2018; Russia. IEEE; 2018. p. 94-100.

[18] P. Thulasiraman, Y. K. D. Cheng and B. Allen. Evaluation of the Data Distribution Service for a Lossy Autonomous Hybrid System. 2022 IEEE International Systems Conference (SysCon); 2022. Canada: IEEE; 2022, p. 1-8.

[19] G. Andrei, B. Marlen, T. Sergey and K. Krinkin. Industrial Messaging Middleware: Standards and Performance Evaluation. 2020 IEEE 14th International Conference on Application of Information and Communication Technologies (AICT); 2020; Uzbekistan. IEEE; 2020. p. 1-6.

[20] S. Profanter, A. Tekat, K. Dorofeev, M. Rickert and A. Knoll. OPC UA versus ROS, DDS, and MQTT: Performance Evaluation of Industry 4.0 Protocols. 2019 IEEE International Conference on Industrial Technology (ICIT); 2019; Australia. IEEE; 2019. p. 955-962.

[21] Y. Chen and T. Kunz. Performance evaluation of IoT protocols under a constrained wireless access network. 2016 International Conference on Selected Topics in Mobile & Wireless Networking (MoWNeT); 2016; Egypt. IEEE; 2016. p. 1-7.

[22] Z. B. Babovic, J. Protic and V. Milutinovic. Web Performance Evaluation for Internet of Things Applications. IEEE Access. 2016; 4: 6974-6992.

[23] X. Chen, X. Kong, Y. Ling and X. Cao. DDS Performance Evaluation for PREEMPT_RT Linux. 2021 International Conference on Computer, Blockchain and Financial Development (CBFD); 2021; China. IEEE; 2021. p. 84-89.

[24] F. Palmese, E. Longo, A. E. C. Redondi and M. Cesana. CoAP vs. MQTT-SN: Comparison and Performance Evaluation in Publish-Subscribe Environments. 2021 IEEE 7th World Forum on Internet of Things (WF-IoT); 2021; USA. IEEE; 2021. p. 153-158.

[25] Y. Sasaki, T. Yokotani and H. Mukai. Comparison with Assured Transfer of Information Mechanisms in MQTT. 2018 International Japan-Africa Conference on Electronics, Communications and Computations (JAC-ECC); 2018; Egypt. IEEE; 2019. p. 95-98.

[26] I. Kassem and A. Sleit. Elapsed Time of IoT Application Protocol for ECG: A Comparative Study Between CoAP and MQTT. 2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE); 2020; Turkey. IEEE; 2020. p. 1-6.

[27] R. Banno, K. Ohsawa, Y. Kitagawa, T. Takada and T. Yoshizawa. Measuring Performance of MQTT v5.0 Brokers with MQTTLoader. 2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC); 2021; USA. IEEE; 2021. p. 1-2.

[28] M. Bender, E. Kirdan, M. -O. Pahl and G. Carle. Open-Source MQTT Evaluation. 2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC); 2021; USA. IEEE; 2021. p. 1-4.

[29] U. Tandale, B. Momin and D. P. Seetharam. An empirical study of application layer protocols for IoT. 2017 International Conference on Energy, Communication, Data Analytics and

Soft Computing (ICECDS); 2017; India. IEEE; 2018. p. 2447-2451.

[30] N. Basavaraju, N. Alexander and J. Seitz. Performance Evaluation of Advanced Message Queuing Protocol (AMQP): An Empirical Analysis of AMQP Online Message Brokers. 2021 International Symposium on Networks, Computers and Communications (ISNCC); 2021; United arab Emirates. IEEE; 2021. p. 1-8.

[31] M. Pohl, J. Kubela, S. Bosse and K. Turowski. Performance Evaluation of Application Layer Protocols for the Internet-of-Things. 2018 Sixth International Conference on Enterprise Systems (ES); 2018; Cyprus. IEEE; 2018. p. 180-187.

[32] Object Management Group Inc. “Specifications”. Available from: https://www.omg.org/spec/About

[33] Y. Maruyama, S. Kato and T. Azumi. Exploring the performance of ROS2. 2016 International Conference on Embedded Software (EMSOFT); 2016; USA. IEEE; 2016. p. 1-10.

[34] eProsima. “eProsima Micro-XRCE-DDS”. Available from: https://micro-xrce-dds.docs.eprosima.com/en/latest/

[35] eProsima. “Introduction”. Available from: https://fast-dds.docs.eprosima.com/en/latest/fastddsgen/introduction/introduction.html

[36] HyunHo Kim, HoonJae Lee and HyoTaek Lim. Performance of Packet Analysis between Observer and WireShark. 2020 22nd International Conference on Advanced Communication Technology (ICACT); 2020; Korea (South). IEEE; 2020. p. 268-271.

[37] Gnome Terminator Organization. “About”. Available from: https://gnome-terminator.org/about/