9-7-2022

Quantifying DDS-cerberus Network Control Overhead

Andrew T. Park

Nathaniel R. Peck

Richard Dill
Air Force Institute of Technology

Douglas D. Hodson
Air Force Institute of Technology

Michael R. Grimaila
Air Force Institute of Technology

See next page for additional authors

Recommended Citation

Park, A. T., Peck, N., Dill, R., Hodson, D. D., Grimaila, M. R., & Henry, W. C. (2022). Quantifying DDS-cerberus network control overhead. The Journal of Supercomputing. https://doi.org/10.1007/s11227-022-04770-3

The Journal of Supercomputing
https://doi.org/10.1007/s11227-022-04770-3

Quantifying DDS-cerberus network control overhead

Andrew T. Park[1] · Nathaniel Peck[1] · Richard Dill[1] · Douglas D. Hodson[1] · Michael R. Grimaila[1] · Wayne C. Henry[1]

Accepted: 10 August 2022

Abstract
Securing distributed device communication is critical because the private industry and the military depend on these resources. One area that adversaries target is the middleware, which is the medium that connects different systems. This paper evaluates a novel security layer, DDS-Cerberus (DDS-C), that protects in-transit data and improves communication efficiency on data-first distribution systems. This research contributes a distributed robotics operating system testbed and designs a multifactorial performance-based experiment to evaluate DDS-C efficiency and security by assessing total packet traffic generated in a robotics network. The performance experiment follows a 2:1 publisher to subscriber node ratio, varying the number of subscribers and publisher nodes from three to eighteen. By categorizing the network traffic from these nodes into either *data message*, *security*, or *discovery*+ with Quality of Service (QoS) best effort and reliable, the mean *security* traffic from DDS-C has minimal impact to Data Distribution Service (DDS) operations compared to other network traffic. The results reveal that applying DDS-C to a representative distributed network robotics operating system network does not impact performance.

✉ Andrew T. Park
andrew.park075@gmail.com

Nathaniel Peck
2012raptor@gmail.com

Richard Dill
richard.dill@afit.edu

Douglas D. Hodson
douglas.hodson@afit.edu

Michael R. Grimaila
michael.grimaila@afit.edu

Wayne C. Henry
wayne.henry@afit.edu

[1] Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson Air Force Base, Dayton, OH 45433, USA

Published online: 07 September 2022

Keywords Kerberos · DDS · ROS 2 · Cyclone DDS · QoS · reliability

1 Introduction

Many technologies rely on real-time and efficient communication capabilities across various environments to support consumer, agricultural, and military use cases. Thermostats, audio, and video devices increase consumers' quality of life through smart home environments [1]. Industry depends on low-power IoT devices to monitor crop yield, improve livestock health, and reduce environmental threats to agricultural success [2]. The military depends on a dynamic network connecting air, land, sea, and space assets [3]. Improving the protection of the in-transit data links and the efficient communication between distributed nodes meets the objectives outlined in the 2030 Science and Technology Strategy and protects industry interests [4]. Choosing the correct middleware and security options to support the network of devices in these environments is important.

This paper evaluates DDS-Cerberus (DDS-C) as a viable method to secure a distributed network of devices. DDS-C builds upon Data Distribution Service (DDS) by integrating strong Kerberos authentication into a robust, flexible, open middleware standard (DDS) designed to manage real-time communication between various devices. As a popular middleware, DDS is implemented across a variety of low-power devices across public and private sectors, military, and finance frameworks [5]. DDS offers configurable Qualities of Service (QoS) associated with data. *Topics* are keywords chosen by the user to differentiate and categorize messages. Subscribers that specify the same *topic* can only read that type of message. *Topics* are used in Machine-to-Machine (M2M) communication to effectively allow publishers and subscribers to send and read data in a global space [6]. DDS-C benefits from the DDS robustness, reliability, and efficiency while addressing its main security weakness: impersonation [7–9]. Impersonation allows an adversary to gain unauthorized access to reading and sending data by posing as a trusted entity and node.

With security lacking as a foundation component in the standard, attackers have multiple methods to attack DDS through QoS policies, network participant discovery, and node impersonation. Attackers create rogue DDS nodes to send disruptive messages to other nodes. A solution is to authenticate publisher and subscriber node components before they send messages. DDS-Cerberus adds an additional authentication mechanism to DDS that improve security authentication to prevent impersonation attacks [10, 11]. DDS-C secures the network by integrating DDS node authentication with Kerberos tickets. The motivation of this research to add security stems from the desire to use the real-time communication properties of DDS with DDS-C authentication. No previous work in DDS has used Kerberos tickets to authenticate nodes and measured packet captures for experimentation in lieu of latency.

This research's experiment measures the DDS-C traffic imposed on a network compared to regular DDS operations to determine if incorporating DDS-C into DDS hinders these operations. The goal of the experiment is to characterize the total network traffic to analyze, categorize, and process the number of packets per protocol. The network traffic types of interest include *data message*, *security*, and *discovery*+. The *data message* utilizes the *topic* for message delivery, *security* refers to the DDS-C authentication messages, and *discovery*+ corresponds to the DDS node discovery messages and additional network packets. When testing, the packets are collected for two network configurations. The purpose of the first configuration is to transmit messages on the same system, and the purpose of the second is to send messages through the network. Different QoS settings are selected for each configuration to show that DDS-C authentication traffic does not substantially delay sent DDS messages. The QoS of interest is reliability with two message behaviors, best effort and reliable.

The results of the experiment use a set p-value of α 0.05 to quantify packet traffic statistically. If the results are statistically significant, DDS-C authentication affects DDS message traffic. The various packet protocols are categorized into *data message*, *security*, and *discovery*+ and compared to determine the DDS-C security traffic trends. This paper contributes to existing DDS work in security and performance. It presents a security layer that others can explore and add to their DDS implementations.

This paper is organized as follows. Section 2 outlines DDS and DDS-C. It also lists related research on performance and security for DDS, Kerberos, and ROS 2 (Robot Operating System). Section 3 details the experimental design, the research assumptions and limitations, the methods used to gathering and process network captured packets, and analysis of captured data. Section 4 outlines why the research is important. Section 5 provides future research recommendations.

2 Background

This section provides background information on the functionality and purpose of Data Distribution Service (DDS) and DDS-Cerberus (DDS-C). Understanding how middleware services function is essential to improving security in real-world applications.

Other researchers have compared DDS to various communication protocols, highlighting performance, latency, and throughput differences. What makes DDS-C different is its fusing of both DDS' efficiency and Kerberos' authentication capabilities. Additionally, it is important to focus on the security and efficiency of Kerberos and ROS 2 (Robot Operating System). These past works form the foundation for understanding the research methodology and evaluation of DDS-C in this paper.

2.1 Data distribution service (DDS)

DDS, a standardized specification maintained by the Object Management Group (OMG), is available from the DDS Foundation website and offers both a Platform Independent Model (PIM) and a Platform Specific Model (PSM) [12]. The standard guides vendors to produce compliant implementations using five distinct modules:

infrastructure, domain, *topic*-definition, publication, and subscription modules. The modules with the Real-Time Publish-Subscribe (RTPS) wire protocol collectively define the commonality between vendor implementations that enable interoperability as a distributed middleware solution.

DDS supports distributed applications serving a many-to-many communication architecture. The standard employs a Data-Centric Publish-Subscribe (DCPS) communication pattern between domain participants using *topics*. Figure 1 is a partially reproduced model of the significant domain entities from the DDS specification, version 1.4. All domain participants are either publishers or subscribers to a given *topic*. Communication includes a series of cache change messages accepted into a participant's history cache. Quality of Service (QoS) policies configure the mechanics governing these cache changes and are tied to publishers, subscribers, and *topics*. Comparison of the QoS offered by publishers to those required by subscribers determines whether participants can be matched for communication. The standard defines the methods to meet QoS levels between publishers and subscribers.

Developers using DDS have already accepted a degree of network control overhead to access the rich set of QoS available for tuning communication behavior between distributed entities. The overhead is configurable beyond mandatory headers and allows developers to add canned behaviors by allocating network resources to the *topics* that require them. After developers have elected to use DDS as a middleware, they may add a layer of security to the distributed communication. That

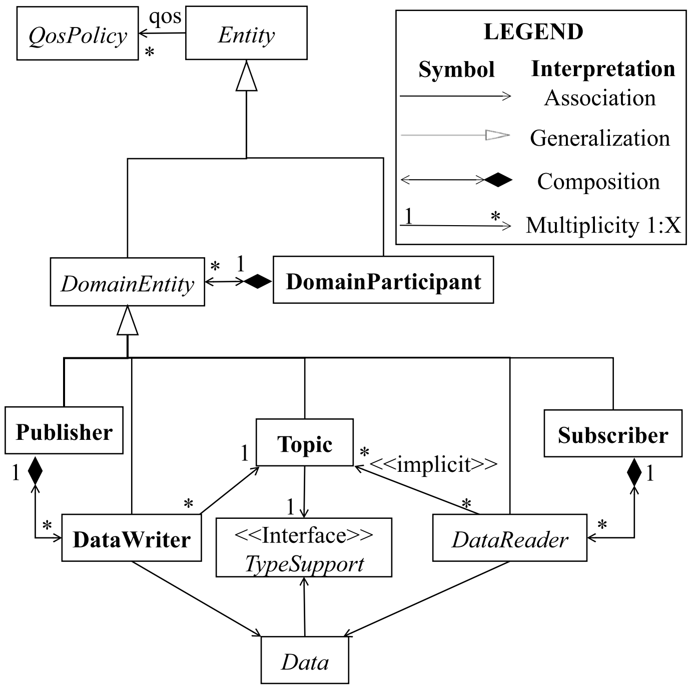

Fig. 1 Partial DDS entity model [13]

layer is not without its overhead additions and is the subject of the comparisons made in this research. While DDS delivers the correct data at the right time, security can be viewed as a possible QoS not yet included in the standard list, ensuring the right participants receive the data rather than actors.

2.2 DDS-cerberus (DDS-C)

DDS-C is a novel security layer incorporating Kerberos ticketing with DDS publishers and subscribers, developed by two previous papers [10, 11]. It provides additional security by validating nodes and preventing impersonation attacks. The first DDS-C paper presented the initial DDS-C design proposal, and the second paper was based on latency measurements with ROS 2 [10, 11]. The latency collections in the second paper determined that DDS-C does not hinder message sending and retrieval. This paper's experiments focuses on packet captures to determine if the amount of packets correlate with the data sent from DDS-C authentication would significantly hinder message sending.

Kerberos is an open authentication protocol that uses tickets to control communication in a network. Each Kerberos setup has a specific realm name. Users who want to authenticate using a network need to know the realm's name and have a registered principal, basically a username.

DDS-C utilizes long-term keys, named *keytabs* that Kerberos provides to create tickets. These tickets are the products of the successful authentication of publishers and subscribers. The benefit of using DDS-C is that once a node is registered and authenticated, there is no extra need to communicate with the central Kerberos server. For example, in a real-time operational network with IoT devices, this authentication would happen before a node publishes or subscribes.

Figure 2 presents the process for creating, storing, and using *keytabs*. In step 1, the Kerberos server is responsible for the credentials corresponding to each or a set of publishers and subscribers. A Kerberos server consists of a Key Distribution Center (KDC) that includes two main components: the Authentication

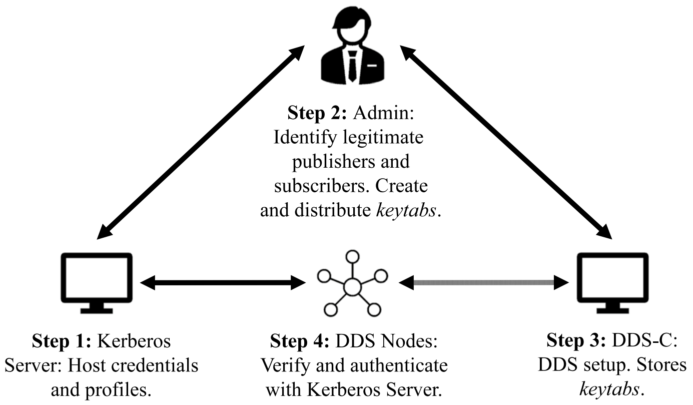
Step 2: Admin: Identify legitimate publishers and subscribers. Create and distribute *keytabs*.
Step 1: Kerberos Server: Host credentials and profiles.
Step 4: DDS Nodes: Verify and authenticate with Kerberos Server.
Step 3: DDS-C: DDS setup. Stores *keytabs*.

Fig. 2 DDS-C *Keytab* process [11]

Server (AS) and Ticket Granting Server (TGS). An admin would create credentials that nodes use to authenticate. When authenticating, the node first messages the AS to receive a ticket from the TGS by using a Ticket Granting Ticket (TGT). The resulting ticket has a default time-to-live of 24 hours; however, an admin can change this to a shorter or longer time.

During step 2, an admin queries and saves the *keytabs* to the appropriate machine where DDS resides before a node can send data. The *keytabs* do not expire, which is essential in operations where time is sometimes not determined.

In step 3, the DDS-C device has a Kerberos server to communicate with the central server. Additionally, an admin can host the Kerberos server in the cloud to provide authentication for the nodes and support *keytab* generation.

At step 4, publishers and subscribers use a *keytab* for authentication. This *keytab* would preferably be created just for a single node to use. The node containing the publishers and subscribers would receive the Kerberos server's response. If a ticket is received, the node is authorized to send and read data. Otherwise, the node is not permitted to send or access any data.

Figure 3 is a sequence diagram outlining the flow of the authentication messages transmitted when Publisher1 and Subscriber1 publish and read messages. This figure outlines the Kerberos authentication process. The leftmost gray area, "Node utilizing KDC," represents the *keytabs* that were created and stored for Publisher1 and Subscriber1. The DDS node leverages the *keytabs* to request and receive tickets from the rightmost gray area, the central Kerberos server "Kerberos Server KDC." Messages flow as follows:

A. Publisher1 authentication:

   (0) Publisher1 requests to authenticate and starts the process to receive a ticket using a *keytab*. The AS receives Publisher1's request.
   (1) The AS sends a message back that Publisher1 is able to authenticate.
   (2) Publisher1 sends its *keytab* to the AS.
   (3) The AS sends a TGT and its shared key for TGS after authenticating Publisher1. The shared key is used by the node to encrypt messages for the TGS.
   (4) Publisher1 sends the TGT and message request to the TGS to get a ticket.
   (5) The TGS grants a ticket to Publisher1.

B. Publisher1 authenticated:
   (6) Afterward, Publisher1 is successfully authenticated and can send its messages to the DDS domain.

C. Subscriber1 Authentication:

   (7) Subscriber1 requests to authenticate and starts the process to receive a ticket using a *keytab*. The AS receives Subscriber1's request.
   (8) The AS sends a message back that Subscriber1 is able to authenticate.
   (9) Subscriber1 sends its *keytab* to the AS.

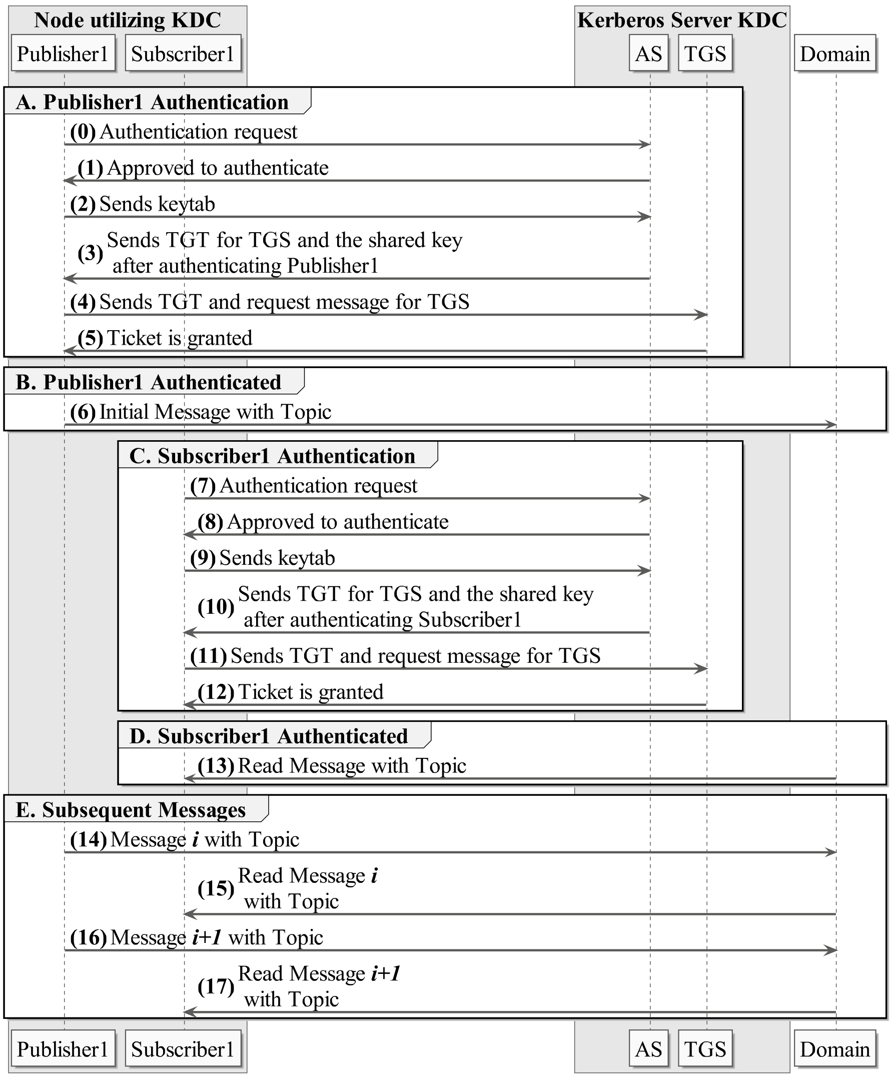
Node utilizing KDC
Publisher1
Subscriber1
Kerberos Server KDC
AS
TGS
Domain
A. Publisher1 Authentication
(0) Authentication request
(1) Approved to authenticate
(2) Sends keytab
(3) Sends TGT for TGS and the shared key after authenticating Publisher1
(4) Sends TGT and request message for TGS
(5) Ticket is granted
B. Publisher1 Authenticated
(6) Initial Message with Topic
C. Subscriber1 Authentication
(7) Authentication request
(8) Approved to authenticate
(9) Sends keytab
(10) Sends TGT for TGS and the shared key after authenticating Subscriber1
(11) Sends TGT and request message for TGS
(12) Ticket is granted
D. Subscriber1 Authenticated
(13) Read Message with Topic
E. Subsequent Messages
(14) Message i with Topic
(15) Read Message i with Topic
(16) Message i+1 with Topic
(17) Read Message i+1 with Topic
Publisher1
Subscriber1
AS
TGS
Domain

Fig. 3 DDS-C authentication process [11]

(10) The AS sends a TGT and its shared key for TGS after authenticating Subscriber1. The shared key is used by the node to encrypt messages for the TGS.
(11) Subscriber1 sends the TGT and message request to the TGS to get a ticket.
(12) The TGS grants a ticket to Subscriber1.

D. Subscriber1 Authenticated:

(13) Afterward, Subscriber1 is successfully authenticated and can read messages. In this case, it would be reading data sent from Publisher1.

E. Subsequent Messages:

(14) Since Publisher1 and Subscriber1 authenticated, no further authentication is needed. Publisher1 sends Message *i* with *Topic*.
(15) Subscriber1 receives the Message *i*.
(16) Publisher1 sends Message $i + 1$ with *Topic*.
(17) Subscriber1 receives the Message $i + 1$.

The Publisher1 and Subscriber1 authentication sequence can be redone as many times as needed. The admin has the choice to re-authenticate new tickets at any interval of time—for example, a check with the central Kerberos server after 24 hours for all nodes; however, this research does not go into this use case and is considered for future work.

The inability to validate nodes is a security concern for DDS [7–9]. By implementing DDS-C, all nodes need to authenticate with the Kerberos server before sending or receiving messages. DDS-C invalidates a node if Kerberos sends back an error message resulting in no delivered ticket. Additionally, an attacker wanting to send or read data would have to communicate with the Kerberos Server to get a ticket. Figure 4 presents DDS-C mitigating an attacker using an impersonation attack. In step 1, an attacker accesses the same network where DDS-C resides. In step 2, the attacker creates an impersonated node; however, any node on the server needs to get a ticket before performing any operations. In steps 3 and 4, since the attacker did not provide the correct *keytab*, it cannot get a valid ticket; therefore, DDS-C prevents the unauthenticated node from interacting with other nodes. Kerberos stores the *keytabs* and tickets in \ `tmp` and when the system shuts down, those files are deleted.

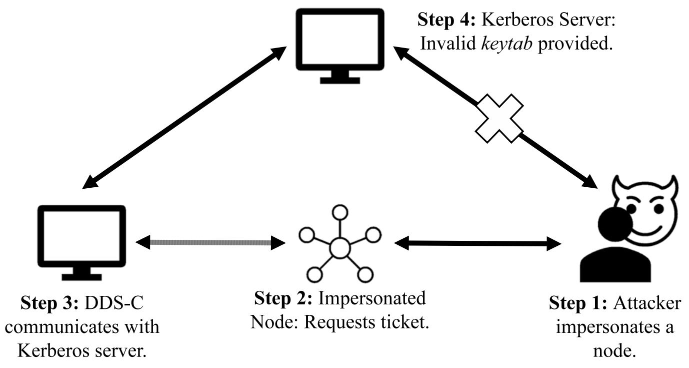
Step 4: Kerberos Server: Invalid *keytab* provided.
Step 3: DDS-C communicates with Kerberos server.
Step 2: Impersonated Node: Requests ticket.
Step 1: Attacker impersonates a node.

Fig. 4 DDS-C mitigating impersonation [11]

DDS-C is a security layer added onto DDS to authenticate DDS nodes with Kerberos tickets. The following three subsections explore other pieces of literature that aid in understanding DDS-C experiments.

2.3 DDS performance evaluations

Other researchers have measured the performance qualities of DDS, such as latency [14–17]. While measuring latency is an essential benchmark for real-time communication middleware, there are methods to collect the information and many other factors that influence end-to-end latency. The cited works all measure latency, but they collect slightly different information that provides unique insights into the middleware's performance in various environments and configurations.

Relatively early works used wired networks to conduct experiments. In 2012, Yang et al. compared DDS communication performance to that obtainable using traditional sockets [14]. They used the OpenSplice DDS implementation provided by Prism Tech to accomplish distributed communication mapped within the IEC 61499 standard. The authors examined the impact of message size, network load, and QoS configurations on latency. They also provide the distribution of latency observed over 1 million iterations. The experiment measured latency by placing timestamps in a message making a round-trip to and back from a node on an Ethernet network connected via a switch. They defined latency as half the measured round-trip time. The test environment used real-time patched Ubuntu operating systems on all nodes. The authors performed tests in this environment to gain insight relevant to distributed industrial control systems which can be realized using similar environments. The results measured roughly 10 times the latency standard deviation of DDS compared to sockets (109 microseconds compared to 10) and a smaller message size before the rapid growth of latency. The authors concluded that DDS offered more favorable simplification for complex network architectures than a traditional socket implementation. DDS began to incur rapid latency growth after message sizes exceeded 2048 bytes but were less sensitive to network load than traditional sockets. Finally, the results illustrated the successful capability of DDS to tailor communication performance according to latency budget and transport priority QoS.

Later, works began to include wireless topology in DDS evaluation experiments. In 2015, Almadani et al. evaluated DDS-based middleware over a wireless channel for re-configurable manufacturing systems (RMS) [15]. With Real-Time Innovations (RTI) DDS implementation, the authors measured latency, jitter, and throughput for payload and headers sent over industrial-grade WiFi and Bluetooth wireless channels. Rather than a simple one-to-one communication architecture, these authors used one-to-many and many-to-many. The experimental setup used simulation to mimic the endpoint behavior of an RMS and measured traffic over the physical channels. The results illustrated that although DDS over WiFi obtained lower latency and tighter jitter, Bluetooth enabled much greater throughput because the peer-to-peer communication strategy was not funneled through an access point. Notably, the processing speed of the access point was not provided and could be the source of some throughput limitation. Although the WiFi throughput was lower, it achieved roughly 7 Mbps and may be sufficient for some applications.

Other works used virtual networks to collect performance in deliberately degraded environments. In 2016, Chen and Kunz compared the performance of DDS to other IoT protocols, including Constrained Application Protocol (CoAP), Message Queuing Telemetry Transport (MQTT), and a custom User Datagram Protocol (UDP) [16]. The intended environment for evaluation in this work was a constrained network used for medical monitoring of multiple sensors. The test environment consisted of various sensors connected to an Arduino in series with a Raspberry Pi device connected to a Linksys router with a laptop acting as a central server. They used virtual networking software to simulate various packet loss, bandwidth, and system latency conditions. Testing observed bandwidth consumption, experienced latency, and experienced packet loss over multiple combinations of environment settings. The authors selected OpenDDS as the implementation of DDS. They also compared the protocols by their quantity of control overhead as a percentage of the payload size. The research showed that DDS experienced the most significant portion of control overhead, but the payload size was held constant at a relatively small 409 bytes. Again, latency was measured as half the round trip time experienced by a single message.

As recent as 2019, works began comparing DDS performance while examining the effects of network and computational loads. Profanter et al. conducted performance comparisons between DDS, Open Platform Communications Unified Architecture (OPC UA), ROS, and MQTT [17]. These authors selected eProsima's Fast-DDS implementation of DDS. They began by examining the traffic required in bytes to connect the listed protocols. ROS and DDS required the most traffic to connect with 8915 and 8348 bytes, respectively. For DDS, this number resulted from a summation over discovery traffic before publisher/subscriber matching. The authors continued to measure the impact of network and CPU loads on Round Trip Time (RTT) for the various protocols over increasing message sizes. DDS latency was dependent on CPU and network load, but the significance of their impact was not statistically evaluated. All latency measurements appeared relatively constant for small message sizes but exhibited a rise when message sizes surpassed a fixed point, potentially related to the Maximum Transmission Unit (MTU).

Table 1 lays out the four works to highlight the emphasis on latency. These works represent the majority of DDS research that measure latency when

Table 1 Measuring DDS latency

Paper	Description
Yang et al. [14]	Compared DDS communication performance to using sockets by measuring latency
Almandani et al. [15]	Measuring latency and jitter for RMS through different wireless channels
Chen and Kunz [16]	Comparing DDS to other protocols over different environment settings
Profanter et al. [17]	Examined the traffic in bytes to measure the impact of DDS an other protocols

evaluating DDS performance. This paper's research focuses on quantifying packet captures offering another perspective in determining DDS-C's impact on DDS operations.

2.4 Kerberos evaluations

Al-Masri et al. surveyed various IoT messaging protocols that reside on the application layer of the Open System Interconnection (OSI) model [6]. Comparing these protocols reinforces the benefits of choosing the proper lightweight communication protocol for low-power devices, reliability, network traffic, and latency. No one protocol is universally used. Zorkadis presented the OSI security architecture guidelines [18]. There are five classes of security: authentication, access control, confidentiality, integrity, and non-repudiation. Zorkadis explained performance costs due to security by using the queuing theory. The author offered optimization recommendations for securing these communication protocols.

Any added security to DDS should not interrupt real-time communication performance. Also, security features added should not hinder the performance of Kerberos. Kirsal et al. coauthored and published three papers that proposed increasing Kerberos security by using frequent key renewal for a local area network [19–21]. They utilized CASPER for the first paper's security analysis. Subsequent papers used Markov Reward models to illustrate Kerberos states. The papers provide a methodology for understanding a novel protocol in Kerberos; however, they do not contain substantial information on what applications and setup they used to gather such data.

Researchers Harbitter and Menascé evaluate public-key performance in Kerberos with Cross-Realm (PKCROSS) and Public Key Utilizing Tickets for Application Servers (PKTAPP) with a five-step approach in the server and network [22]. They measured both proposals by their messages with the KDC. The first step was to create a testbed with code that monitored service times and message sizes. Then, they developed a closed queuing network to represent public key extensions. They compared the testbed results with the queuing model to determine the accuracy with several realms and servers. Finally, they analyzed the changes in service time and network delay to understand dependencies. The results from comparing the two proposals showed that PKCROSS outperformed PKTAPP.

Evaluating existing Kerberos implementations is essential for research, but the development of new Kerberos mechanisms is also equally important. Eum and Choi proposed a new authentication mechanism in Extensible Authentication Protocol (EAP) named EAP-Kerberos II [23]. This protocol mitigated three security concerns of wireless local area networks (WLANs) for an 802.11 network: rogue access points (APs), unprotected messages, and message delay. 802.11i has existing security measures using Transport Layer Security (TLS) and Authentication and Key Agreement (AKA) over EAP. Instead of TLS or AKA, EAP-Kerberos II utilized Kerberos's function as a trusted third party in a mutual authentication by adapting it into EAP. The reason to use Kerberos tickets is that Kerberos does not require significant computational power or memory space to store a certificate. They measured

Table 2 Different Kerberos authentication methods

Paper	Description
Kirsal et al. [19–21]	Frequent key renewal for a local area network
Harbitter and Menascé [22]	Evaluating service time and network delay for public-key performance with PKCROSS and PKTAPP
Eum and Choi [23]	Measuring RTT of a proposed new authentication mechanism named EAP-Kerberos II and comparing it to EAP-TLS and EAP-AKA

the number of messages sent between EAP-TLS, EAP-AKA, and EAP-Kerberos II. They also compared the message's round trip times (RTT), processing delay in clocks per message, and RTTs when the access point is far from the Authentication Server. They concluded that EAP-Kerberos II is more efficient than the other two protocols since it requires fewer authentication servers and sends fewer RTTs.

Table 2 presents three papers that introduce different methods in improving Kerberos authentication. One of the main highlights of Kerberos research is focusing and improving Kerberos security; however, no other Kerberos work aims to combine DDS and Kerberos. The experiments take these insights to Kerberos authentication to better understand the future work avenues DDS-C can improve Kerberos with.

2.5 ROS 2 evaluations

ROS 2 evolved from ROS 1, and both primarily differ at the communication layer. [24] ROS 1 and ROS 2 both support robotics and IoT communication use cases. They can be used and set up together, but the main difference is that ROS 2 has the capability for real-time communication between devices. This paper uses ROS 2 for its real-time capability and recent development.

Kronauer et al. measured latency on ROS 2 middleware to provide guidelines on designing ROS 2 architectures and reducing traffic overhead. They utilized three DDS implementations, eProsima FastRTPS, Eclipse Cyclone DDS, and RTI Connext, using ROS 2 Foxy Fitzroy [25]. Their selected best effort QoS does not require re-transmitting lost frames since the majority will go through; this is emulating their use case of using sensors. They measured latency via node scalability on localhost using a ping-pong scenario with payloads of 128 B and 500 KB sent over UDP. Afterward, they provided a list of techniques that affect latency.

In addition to the previously mentioned DDS implementations, Maruyama et al. compared ROS 1 and ROS 2 by measuring latency, throughput, number of threads, and memory consumption [26] across three different DDS implementations: Connext, OpenSplice, and FastRTPS. They choose different QoS policies to get varied results for each DDS implementation.

Other research measured latency and throughput in different network settings. Park et al. compared ROS 1 and ROS 2 characteristics by measuring the real-time performance of the software stack and communication [27]. Utilizing various nodes, they collected message loss rates and latency times and represented them through statistical mean, maximum, minimum, and standard deviation. The authors also utilized a multi-agent service robot to verify the real-time performance. Their results showed that ROS 1 did not meet real-time requirements.

In addition to measuring latency, Thulasiraman et al. set up a small network of two and five nodes in ROS 2 to measure performance in a lossy wireless environment [28]. They utilized NS-3, an open-source network simulator, to measure latency and message drop rate. By varying QoS and security configurations, they concluded that enabling more security features leads to a higher messaged drop rate with any QoS policies and that scaling with more nodes leads to increased message latency.

Researching the impact of other security implementations should be considered when experimenting DDS-C. Kim et al. concentrate on the performance of additional security implementations on top of default ROS 2 and DDS security features since the default DDS middleware in ROS 2 does not conform to security specifications set by OMG [29]. They have two performance metrics: estimated latency and estimated throughput. Additionally, they configured them into both wired and wireless configurations when setting up performance benchmark scenarios. The three security situations include using no security, cryptographic algorithms, and Secure Sockets Layer (SSL)/TLS through OpenVPN. The authors also used Cppcheck, a static analysis error checking tool, to conduct further security analysis. They concluded that using a VPN is a secure choice in simple system architectures.

Table 3's six papers play an important role in shaping the ROS 2 evaluation of this paper's experiments. ROS 2 experiments aim to measure latency with different DDS implementations, but this experiment focuses on implementing DDS-C and capturing packet traffic quantity. These papers offer other implementations DDS-C can be integrated into and experimented with.

This section explained DDS and DDS-C architecture and core functions. It also presented other pieces of literature to support the motivation for testing DDS performance and security. This information helps understand the research experiment setup, execution, and analysis.

Table 3 Measuring ROS 2 latency

Paper	Description
Kronauer et al. [25]	Using three DDS implementations, they measured ROS 2 latency to find techniques to reduce traffic overhead
Maruyama et al. [26]	Compared ROS 1 and ROS 2 across three different DDS implementations
Park et al. [27]	Measured the real-time performance of ROS 1 and ROS 2 by collecting message loss rates and latency times
Thulasiraman et al. [28]	Set up a small network of nodes in a lossy environment with different QoS policies
Kim et al. [29]	Measuring latency and throughput for additional security implementations on top of ROS 2

Table 4 Experiment parts

Subsection	Step	Description
3.1	Statistical approach	DoE theory used to draw statistical conclusions regarding the significance of the burden imposed by security
3.2	Apparatus	The equipment, Kerberos, and ROS 2 setup experimentation
3.3	Assumptions and limitations	Considering what are the research assumptions and limitations of the experiments
3.4	Data processing	The general steps to collect and process the data

3 Experiments

Table 4 outlines sequential experimental steps measuring the *security* packet traffic from DDS-Cerberus (DDS-C). First, the statistical approach for the Design of Experiments (DoE) is determined. Next is setting up the experiment testbed with the appropriate software which includes Kerberos and ROS 2 (Robot Operating System). Afterward, the assumptions and limitations are listed. The final step is to process captured packets using scripts on a Windows machine.

3.1 Statistical approach

Design of Experiment (DoE) methods provide experimenters with an unbiased, mathematical framework to evaluate the significance of statistical results [30]. DoE offers statistical mechanisms to test on hypotheses concerning response variables of different types. Most research measure latency as Round Trip Time (RTT) for Data Distribution Service (DDS); however, this approach is not consistent in different network environments. Instead, using a more portable response variable such as packet traffic overhead provides standardized results. Sadjadi et al. introduce the need for environment agnostic performance measures, particularly in the distributed system arena [31]. They introduce a statistical model to estimate the execution time for a task at a distributed node. The following two paragraphs expound upon the deficiencies of RTT to evaluate the performance of DDS across environments.

Several vendors provide DDS implementations. ROS 2 supports several of these vendors. Unless vendors use the same code, their implementations require different instructions to execute standard behavior. While a given implementation affects one component of the end-to-end latency experienced by a DDS application, the overall RTT depends on more factors. Profanter et al. showed that central processing unit (CPU) and network loads also impact the RTT experienced by a DDS message [17].

At each stage of the end-to-end process, the RTT experienced is proportional to the amount and size of traffic, the computational hardware's performance, and the efficiency of the software controlling the hardware. Further, the actual RTT of a message is influenced by the distance it must travel through the communication medium. For these reasons, RTT can make a reasonable response variable when comparing DDS to other communication solutions in a fixed environment. However, this research compares the performance of DDS to itself with a change in security. To increase the portability of these results to other environments, RTT is not used. Since the standard specifies the behavior of the middleware to be interoperable, the message quantity and content are expected to be far less variable between environments and implementations than RTT. Therefore, the response variable is the total network traffic in bytes required to send a fixed quantity of published messages containing a fixed size payload between a set number of participants.

The Student's t test is one of the tools used in DoE. It is uniquely suited to test hypotheses on means where the population variance is unknown. Testing whether the population mean traffic in bytes generated by DDS to execute a fixed quantity of

Table 5 Equipment specifications

Name	Foxy1	Foxy2	Kerby
Machine	XPS 13 9310	Raspberry Pi 4B	Raspberry Pi 4B
OS	Ubuntu 20.04.3 LTS	Ubuntu 20.04.3 LTS	Ubuntu 20.04.3 LTS
CPU	11th Gen i7-1185G7	ARM Cortex-A72	ARM Cortex-A72
Disk space	2 TB	64 GB	256 GB
RAM	31 GB	8 GB	8 GB

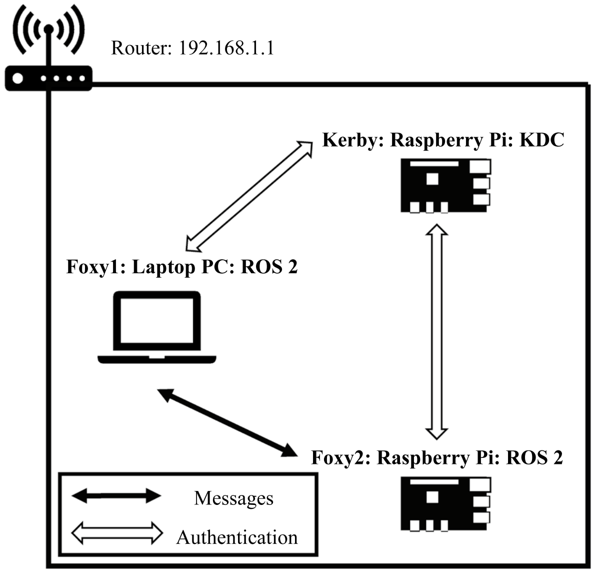

Fig. 5 Experiment testbed network diagram

published messages without authentication, μ_0, is significantly different than the population mean traffic required to complete the same communication with authentication, μ_1. The inputs for the t test are based on the publisher and subscriber ratio which has six runs completed for each total amount of messages where with and without authentication can be measured and extracted. The p-value from the calculated test statistic is compared to α to determine whether a null hypothesis, H_0, can be rejected. α is commonly set to 0.05 and 0.01, an acceptable probability for an incorrect rejection. The null hypothesis is that there is no difference between the population means. If the null hypothesis is rejected, sufficient evidence suggests a difference in population mean traffic in bytes generated by DDS to execute a fixed quantity of published messages with and without authentication.

3.2 Experiment apparatus

The experiment testbed for DDS-C utilizes ROS 2 Foxy Fitzroy and Kerberos [32, 33]. Foxy Fitzroy was selected because of its long-term support and its use of eProsima Fast-RTPS [34]. Four pieces of apparatus are used—a Netgear R6100 router, a Dell XPS 13 Laptop personal computer (PC), and two Raspberry Pi 4B devices. Table 5 lists the main equipment and its specifications. The names from the table distinguish the three main pieces of equipment: Foxy1, Foxy2, and Kerby. All three devices need Kerberos installed; however, Kerby's Kerberos is the main KDC of interest for the experiments. Foxy1 and Foxy2 additionally have ROS 2 Foxy Fitzroy installed, architectures amd64 and arm64, respectively [35]. Figure 5 is the testbed network diagram. The three devices connect wirelessly to the same router and are logically on the same network subnet. Foxy1 and Foxy2's nodes have to request and receive tickets from Kerby to authenticate prior to sending messages to each other.

Each ROS 2 node has either one publisher or subscriber. Each publisher to one subscriber sends a total of 10 messages at 0.5-second intervals. For scalability, there are six sets of publisher and subscriber nodes with a two publisher to one subscriber

Table 6 Experiment QoS settings

QoS	Selected	Description
Depth	Queue size = 10	Queues messages for a subscriber based on message traffic to it
Reliability	Best Effort	Some messages may be lost due to the network
	Reliable	Messages are guaranteed to be sent through retries
Durability	Transient Local	Publisher persists messages for subscribers that join the network late

Table 7 Configuration-dependent variables

Variables	Description
QoS	Option to lose some messages with best effort or guarantee all messages are sent with reliable
Node count and ratios	Increasing number of nodes with each larger ratio increases number of messages
With and without DDS-C	Run experiments with and without DDS-C to analyze its *security* traffic impact

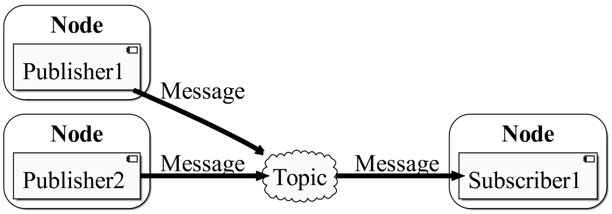

Fig. 6 Experiment node layout [11]

ratio: 2:1, 4:2, 6:3, 8:4, 10:5, and 12:6. The total amount of messages for each ratio: 20, 40, 60, 80, 100, 120. Each subscriber node receives 10 messages from two publisher nodes for a total of 20 messages, as shown in Figure 6. Every 2:1 node pairing has a unique *topic*. The message payload is a "Hello World: *i*" string where *i* is the message counter. Other payload sizes are not experimented with because they do not impact authentication, starting at the beginning of a node's life cycle.

All nodes have set Quality of Service (QoS) policies for queue size, reliability, and durability as shown in Table 6. These three are set to ensure different node and message behaviors. ROS 2 sets all other QoS settings to their default values [36]. The experiment modifies reliability, switching between best effort and reliable. Queue size is 10 messages, and durability is transient local. Every node has a unique credential that is created and managed by Kerby. When authenticating, a node needs to know their Kerberos principal and realm and access their respective *keytab*.

There are two different network configurations. The first configuration uses only Foxy1 and Kerby, and the second configuration uses Foxy1, Foxy2, and Kerby. Each configuration is tested with the dependent variables listed in Table 7.

1. *Foxy1 with Kerby*: Foxy1's publisher and subscriber nodes are on the same laptop PC and authenticate with Kerby. Before each node operation, they authenticate through Kerby by receiving a ticket. Afterward, the publishers send messages to the subscribers.
2. *Foxy1/Foxy2 with Kerby*: All publisher nodes are on Foxy1, and all subscribers are on Foxy2. Once the nodes authenticate through Kerby, the publishers send messages, and the subscribers read them.

3.3 Assumptions and limitations

This subsection outlines the experiment's assumptions and limitations, by-products of the setup, configurations, and processing. The list of assumptions are as follows:

- For ROS 2, nodes do not fail authentication and that an attacker does not compromise nodes.
- All publishers send all 10 messages, and all subscribers receive the specified messages.
- No Kerberos principals were renewed with new keys or *keytabs*; the same ones were used in all test iterations.
- For data processing, only pertinent captured packet protocols such as Real-Time Publish-Subscribe (RTPS) were included in packet analysis. Protocols such as NetBIOS Name Service (NBNS), which Wireshark sends out when it starts to sniff, and Simple Service Discovery Protocol (SSDP), discovery of plug and play devices, are excluded and deemed extraneous due to low packet captures and low relevancy to DDS-C security.
- All RTPS packets without the predefined publish payload were categorized as *discovery+*.

Limitations of the experiment include:

- The experiments occur in a local area network with the same subnet, thereby confining the nodes to a controlled network with less outside packet noise. In future work, more packet noise could be desired if DDS-C is tested in a more lossy environment or different networks.
- Nodes send fixed size payloads with a set time interval of 0.5 seconds for all network configurations, which is appropriate since packet quantity was measured regardless of latency.
- Selected QoS limits the message's behavior, and more combinations could be implemented. Using the reliability QoS is essential because it allows for message retransmissions. Still, the scope could widen to other QoS properties if other DDS-C properties were explored.
- Selection of total categorized network traffic as the response variable for statistical testing provides one component of the overall overhead of using DDS-C. The remaining overhead components are environment-dependent.
- Default usage of simple discovery protocols changes the total traffic compared to other discovery methods. Other discovery methods may change the sensitivity of statistical tests to the mean difference in traffic-induced by authentication.

3.4 Data processing

Data processing is the final step. The data is successfully collected first on Foxy1 and Foxy2 and then transferred to a separate Windows machine for processing and formatting.

The ROS 2 `launch` command executes a modifiable script that specifies which nodes to run simultaneously at the start of each configuration. When the nodes run, Wireshark, used on Foxy1, and tcpdump, used on Foxy2, collect the packets sent from Kerberos and ROS 2 [37, 38]. The .pcap files are then sent to a Windows machine for processing.

The PowerShell Tabluation Script, as shown in Listing 1, filters the packet capture files into columns of data fields via tshark [39, 40]. Next, it automatically sums the total bytes captured for each category, dumping the results to a comma-separated value (CSV) files. This example pseudocode does not display all the column fields extracted but includes two to show that the command can accept additional fields.

Algorithm 1 Tabulation Script

```
for each $file in $list_of_files do
    tshark.exe -2 -r $file -T fields ...
    -E "Separator=," ...
    -e "frame.protocols" ...
    -e "frame.len"
end for
```

Table 8 Experiment software information

Name	Version	Location
ROS 2	Foxy Fitzroy	Foxy1, Foxy2
Kerberos	V5	Foxy1, Foxy2, Kerby
Wireshark	3.2.3	Foxy1
tcpdump	4.9.3	Foxy2
tshark	3.4.7	Windows
PowerShell	5.1.19041.1237	Windows
Python	3.9.7	Foxy1, Foxy2, Windows
SciPy	1.7.0	Windows

The PowerShell script extracted message sizes to identify and categorize messages transmitting the published payload. The published data had a fixed message size of 44 bytes in these experiments. This size was unique to data publish messages and presented a suitable criterion to categorize a packet as a *data message*. The protocols field identified packets belonging to the *security* category as they were the only packets sent using either Domain Name System (DNS) or Kerberos protocol. All other packets sent using the Real-Time Publish-Subscribe (RTPS) protocol were categorized as *discovery+*. This category represented the traffic associated with typical DDS network traffic overhead.

Python was used to apply Student's T tests to the summed traffic for each configuration's participant count [41]. The `SciPy.Stats` module provides the `stats.t.cdf` function to evaluate the *p*-values given the test statistic and degrees of freedom [42]. To better understand the software used, Table 8 presents information about the names, locations, versions, and descriptions of all the software.

3.5 Experiment results

This section summarizes the experiment's results. Plots illustrate the growth of three categories of network traffic, *data message*, *security*, or *discovery+*,

Table 9 Configuration *p*-values

Participants	Best Effort Foxy1 with Kerby	Reliable Foxy1 with Kerby	Best Effort Foxy1/Foxy2 with Kerby	Reliable Foxy1/Foxy2 with Kerby
3	0.022[a]	0.027[a]	0.007[a]	0.007[a]
6	0.015[a]	0.134	0.170	0.197
9	0.218	0.249	0.445	0.357
12	0.290	0.614	0.651	0.274
15	0.742	0.603	0.440	0.546
18	0.610	0.482	0.532	0.339

[a]Statistically significant *p*-values with α 0.05

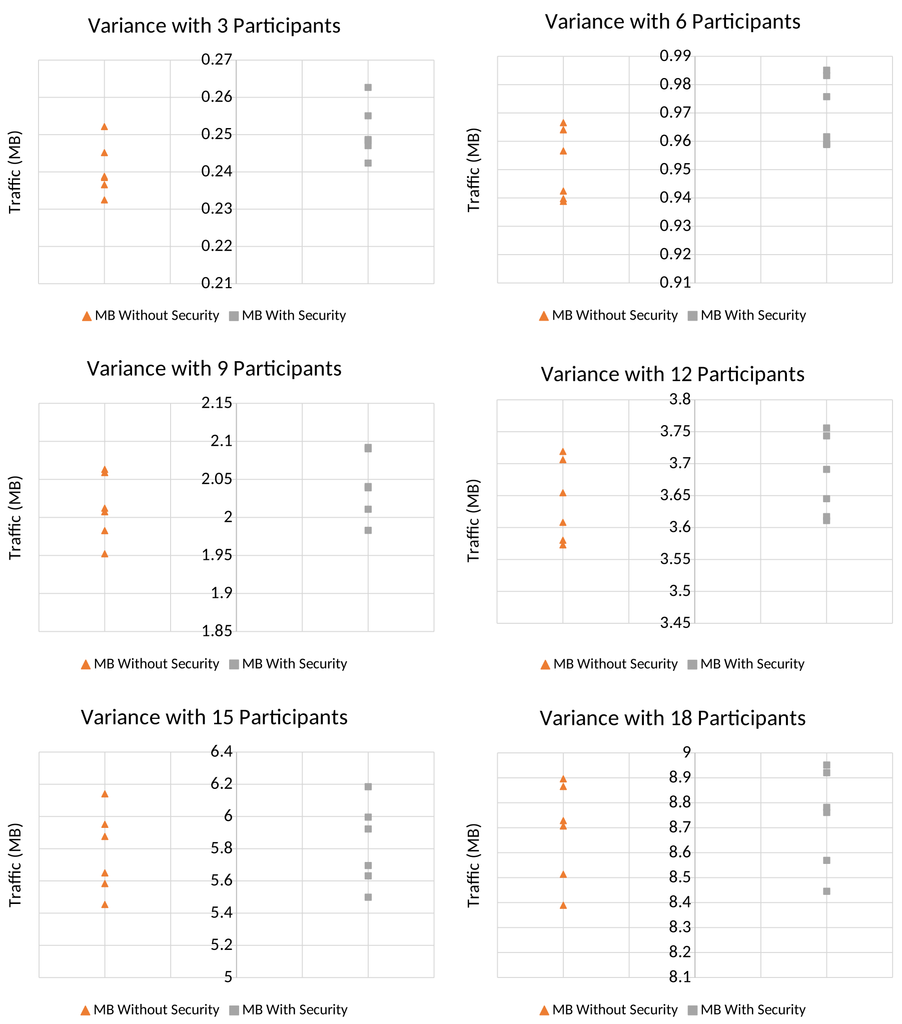

Fig. 7 Experiment Results. Top row: Best Effort Foxy1 with Kerby Variance 3 and 6. Middle row: Best Effort Foxy1 with Kerby Variance 9 and 12. Bottom row: Best Effort Foxy1 with Kerby Variance 15 and 18

resulting from increased participants. Each set of data points for a x-value represents six runs executed. Although nodes sent relatively small data amounts, *security* traffic was indistinguishable due to the dominant *discovery+* traffic and its associated variance.

To illustrate the magnitude of the differences in means relative to the sample variances required to reject the null hypothesis, Fig. 7 plots the observed spread of the traffic quantity observed in MB for each participant count with and without security. The relative magnitude of the difference erodes as more participants enter the domain. These values are used to calculate the *p*-values in Table 9.

Table 9 lists the *p*-values for the two different configurations with the best effort and reliable QoS. In all cases with three participants, the addition of *security* imposed a statistically significant change in mean traffic on the network. However, in most cases, the difference in mean traffic set by *security* was not statistically significant by six participants. For the statistically significant values, the discovery traffic growth with each participant dominated the other traffic sources. Although one of the configurations with six participants indicated significant traffic due to *security*, the significance was diminished by nine participants.

Multiple factors could have influenced the delayed insignificance experienced by the best effort configuration using Foxy 1 with Kerby. The best effort configurations generally resulted in less traffic, making the conclusion more sensitive to minor differences. Additionally, the configuration using only Foxy 1 with Kerby was less lossy than the configuration involving Foxy 2. The reduced loss resulted in less variance, further sensitizing the test to smaller differences in means. Combining these effects required more participants before the *security* traffic could be

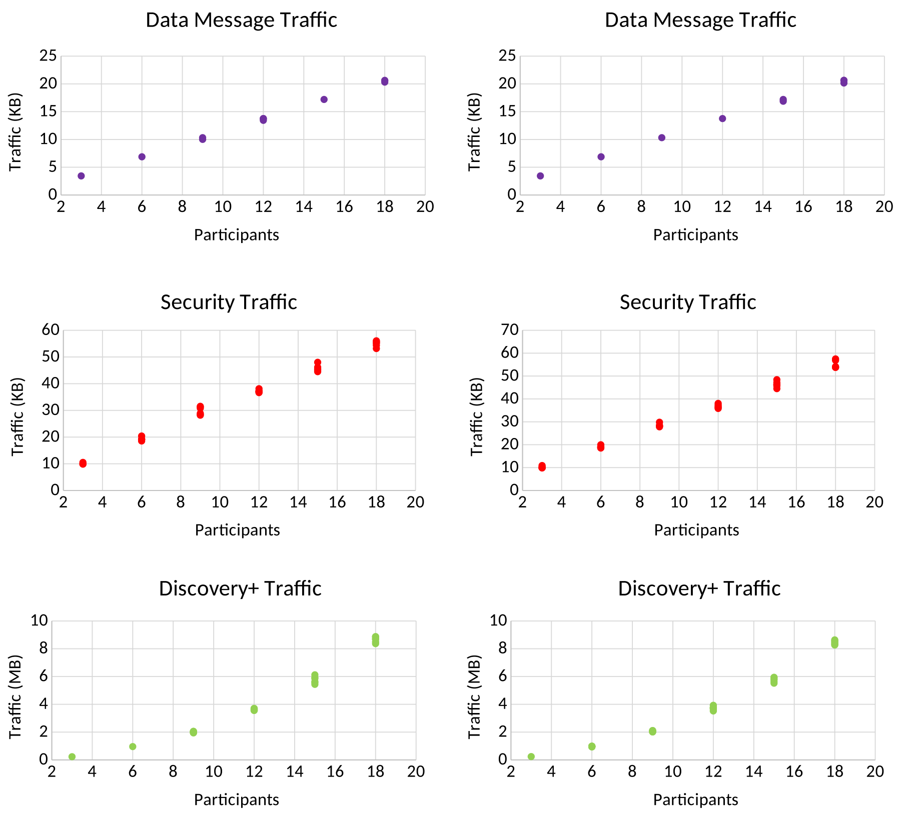

Fig. 8 Experiment results. Left column: Best Effort Foxy1 with Kerby. Right column: Reliable Foxy1 with Kerby

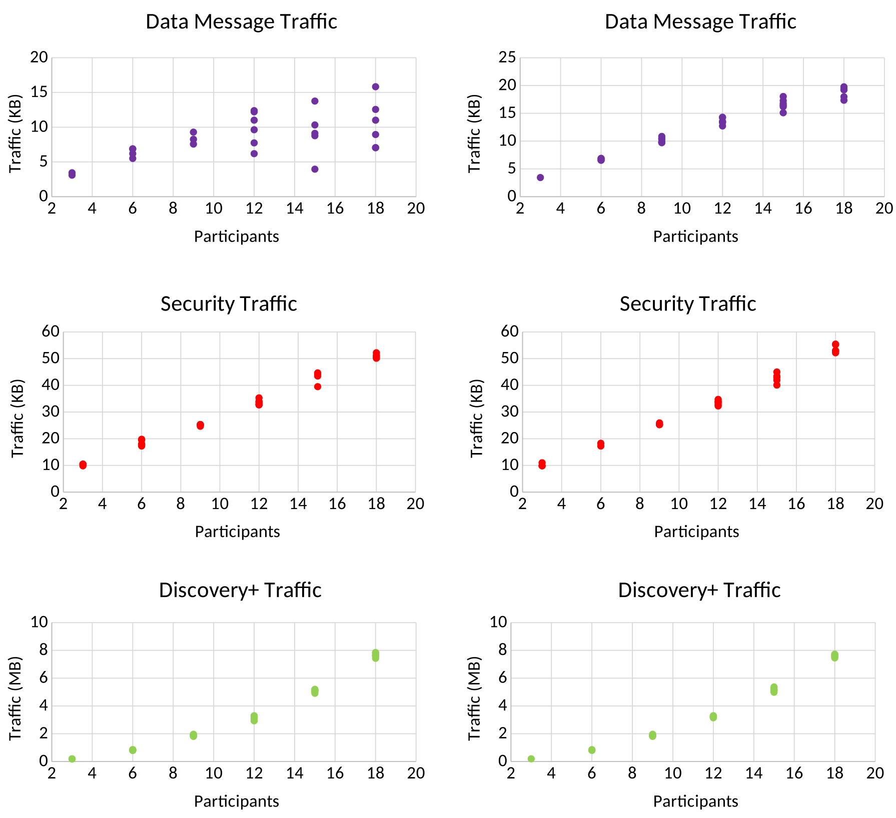

Fig. 9 Experiment results. Left column: Best Effort Foxy1/Foxy2 with Kerby. Right column: Reliable Foxy1/Foxy2 with Kerby.

considered insignificant. The *p*-values show that adding DDS-C requires statistically insignificant additional traffic for reasonably sized experiments.

Figures 8 and 9 layout both configurations, Foxy1 with Kerby and Foxy1/Foxy2 with Kerby, with QoS best effort and reliable. They plot the packet traffic categorized as *data message*, *security*, and *discovery+*:

- *Data message*: traffic represents the captured packets for messages sent from publishers to subscribers.
- *Security traffic*: represents packets for Kerberos server communication.
- *Discovery+ traffic*: includes all additional traffic that consists of a majority of but is not limited to DDS node discovery messages. Other traffic categorized as *discovery+* has meta traffic used by DDS to ensure QoS, such as heartbeat messages and acknowledgments.

In both figures, the traffic grows with increased participants. Visually, *discovery+* traffic is about two orders of magnitude greater than *data message* and *security* traffic. It also has a steeper slope than the other two categories and could fit a higher-order model. Notably, the plotted *discovery+* traffic uses units of MB while the other two are in KB. If not considering *discovery+* traffic in the statistical calculations, the *security* traffic would be statistically significant for all participant configurations. This observation would be accurate if nodes sent messages with User Datagram Protocol (UDP) rather than RTPS as provided by DDS. However, in this case, due to the overwhelming collection of *discovery+* messages, the *security* overhead is shown to not be statistically significant for the majority of all participant sets. Due to a lossy network configuration and reliability QoS, Fig. 9 best effort *data message* traffic is different from the relative reliable plot. This reliable plot is similar to Fig. 8's *data message* traffic plots for both best effort and reliable. Reliability QoS does not significantly change the amount of traffic in all performed configurations. Nonetheless, even with a lossy environment, the overall trend indicates that *security* traffic does not produce enough traffic overhead to significantly deter the use of security mechanisms in both QoS reliabilities.

This section outlines how the statistical model, network and ROS 2 setup, and processing software support the experiment results. It illustrates the defined process and setup to efficiently acquire, process, and analyze data, and examines the results collected by these methods and software. DDS-C is not statistically significant enough, as seen with the majority of configurations, to hinder DDS.

4 Discussion

This research, unlike previous research and related works, combines two existing technologies to create a more secure product. Many of the Data Distribution Service (DDS) and ROS 2 (Robot Operating System) related works, previously mentioned, measure DDS and ROS 2 latency times and not on experimenting them with other software. Many Kerberos research improved authentication methods, as seen in the related works, but not on combining technologies. This research is unique by combining DDS and Kerberos to create DDS-Cerberus (DDS-C). DDS-C's security capability leverages Kerberos creating a new security layer to determine its affect on DDS. The results concluded that DDS-C traffic does not hinder DDS. This is important because it can be implemented in other DDS frameworks and ROS 2 implementations without hindering performance and improving security through authentication. In the future, it can be improved through more experimentation with different use cases and mission sets. It can also be expanded to other middleware protocols.

5 Conclusion

This research explored the cost of using DDS-Cerberus (DDS-C) to provide security. The experiment hosted DDS-C in a local subnet by authenticating publisher and subscriber nodes. The results revealed the mean *security* traffic incurred by DDS-C to send a given amount of data between authenticated nodes is indistinguishable from traffic quantity observed from comparable experiments without authentication.

Analyzing results from both Quality of Service (QoS) best effort and reliable show that the difference in mean traffic is insignificant for use cases involving anything more than small numbers of participants sharing a few messages of small size. These results indicate that DDS-C applied to other Data Distribution Service (DDS) implementations adds extra benefit without substantial performance costs. Understanding this information is crucial in applying DDS-C to future research.

Future research could improve the existing DDS-C design and integrate it into real-time systems. For instance, creating a Kerberos node that facilitates ticket retrieval to handle a more significant number of nodes. This idea can also lead to experimenting with re-authentication throughout the lifetime of a node to observe the authentication traffic impact. Another proposal could integrate DDS-C into a QoS policy or experimenting with other QoS policies besides reliability. Also, DDS-C can be experimented with integrating authentication with other ROS 2 (Robot Operating System) components: services and actions. DDS-C is still in development and requires more real-world use case experimentation before operational use.

Concerning analysis, future work could include regression tests to estimate model parameters for linear and nonlinear models. As the effect of authentication was found to diminish to insignificance for reasonably sized domains, its parameter was not estimated. Instead, future work could further investigate the *discovery+* category of traffic and any factors affecting its component of the response variable. These parameters would facilitate the application of these results to predict performance in other scenarios. The process of applying the predictive power of this response variable could be refined and validated in the following work, similar to that of Sadjadi et al. [31].

Technologies and middleware are constantly evolving. Further research is needed to improve DDS security and performance. DDS-C is one option that provides that extra security to any DDS implementation, increasing data integrity and node trust.

Acknowledgements The views expressed in this document are those of the author and do not reflect the official policy or position of the United States Air Force, the United States Department of Defense or the United States Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the USA.

Data availability The datasets generated during and/or analyzed during the current study are available from the corresponding author on reasonable request.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

References

1. El-Hajj M, Fadlallah A, Chamoun M, Serhrouchni A (2019) A survey of internet of things (IoT) authentication schemes. Sensors 19(5):1141
2. Ahmed N, De D, Hussain I (2018) Internet of Things (IoT) for smart precision agriculture and farming in rural areas. IEEE Internet Things J 5(6):4890–4899
3. White T, Johnstone MN, Peacock M (2017) An investigation into some security issues in the DDS messaging protocol. In: Proceedings of the 15th Australian Information Security Management Conference, AISM 2017, pp 132–139
4. Force, U.S.A. (2019) Science and Technology Strategy Strengthening USAF Science and Technology For 2030 and Beyond. United States Air Force
5. Object Management Group: OMG Standards for Industries. https://www.omg.org/industries/index.htm. Accessed 11 Oct 2021
6. Al-Masri E, Kalyanam KR, Batts J, Kim J, Singh S, Vo T, Yan C (2020) Investigating messaging protocols for the Internet of Things (IoT). IEEE Access 8:94880–94911
7. Abdulghani RM, Alrehili MM, Almuhanna AA, Alhazmi OH (2020) Vulnerabilities and Security Issues in IoT Protocols. In: 2020 First International Conference of Smart Systems and Emerging Technologies (SMARTTECH). IEEE, pp 7–12
8. Michaud MJ, Dean T, Leblanc SP (2018) Attacking OMG data distribution service (DDS) based real-time mission critical distributed systems. In: 2018 13th International Conference on Malicious and Unwanted Software (MALWARE). IEEE, pp 68–77
9. Goerke N, Timmermann D, Baumgart I (2021) Who Controls Your Robot? An Evaluation of ROS Security Mechanisms. In: 2021 7th International Conference on Automation, Robotics and Applications (ICARA). IEEE, pp 60–66
10. Park AT, Dill R, Hodson DD, Henry WC. DDS-Cerberus: Data Distribution via Ticketing. In: The 2021 World Congress in Computer Science, Computer Engineering, and Applied Computing (CSCE'21)
11. Park AT, Dill R, Hodson DD, Henry WC. DDS-Cerberus: Ticketing Performance Experiments and Analysis. In: The 2021 International Congress on Computational Science and Computational Intelligence (CSCI'21)
12. Foundation, D.: DDS Portal. https://www.dds-foundation.org/. Accessed 16 Sep 2021
13. Object Magagement Group: OMG Data Distribution Service (DDS). (2015). Object Magagement Group. Version 1.4
14. Yang J, Sandström K, Nolte T, Behnam M (2012) Data Distribution Service for industrial automation. In: Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies Factory Automation (ETFA 2012), pp 1–8. https://doi.org/10.1109/ETFA.2012.6489544
15. Almadani B, Bajwa MN, Yang S-H, Saif A-WA (2015) Performance evaluation of DDS-based middleware over wireless channel for reconfigurable manufacturing systems. Int J Distrib Sens Netw 11(7):863123
16. Chen Y, Kunz T (2016) Performance evaluation of IoT protocols under a constrained wireless access network. In: 2016 International Conference on Selected Topics in Mobile and Wireless Networking (MoWNeT). IEEE, pp 1–7
17. Profanter S, Tekat A, Dorofeev K, Rickert M, Knoll A (2019) OPC UA versus ROS, DDS, and MQTT: performance evaluation of industry 4.0 protocols. In: 2019 IEEE International Conference on Industrial Technology (ICIT). IEEE, pp 955–962
18. Zorkadis V (1994) Security versus performance requirements in data communication systems. In: European Symposium on Research in Computer Security. Springer, pp 19–30
19. Kirsal Y, Gemikonakli O (2008) Improving kerberos security through the combined use of the timed authentication protocol and frequent key renewal. In: 2008 7th IEEE International Conference on Cybernetic Intelligent Systems. IEEE, pp 1–6

20. Ever E, Kirsal Y, Gemikonakli O (2009) Performability modelling of a Kerberos server with frequent key renewal under pseudo-secure conditions for increased security. In: 2009 International Conference on the Current Trends in Information Technology (CTIT). IEEE, pp 1–6
21. Kirsal-Ever Y, Kirsal Y, Polzonetti A, Mostarda L, Sule C, Shah P, Ever E (2013) Challenges of Kerberos Variance with High QoS Expectations. In: Proceedings of the International Conference on Security and Management (SAM), p 1. The Steering Committee of The World Congress in Computer Science, Computer
22. Harbitter AH, Menascé DA (2000) Performance of public-key-enabled Kerberos authentication in large networks. In: Proceedings 2001 IEEE Symposium on Security and Privacy. S &P 2001. IEEE, pp 170–183
23. Eum S-H, Choi H-K (2008) EAP-Kerberos II: An adaptation of Kerberos to EAP for mutual authentication. In: 2008 8th International Conference on ITS Telecommunications. IEEE, pp 78–83
24. Erős E, Dahl M, Bengtsson K, Hanna A, Falkman P (2019) A ROS2 based communication architecture for control in collaborative and intelligent automation systems. Procedia Manuf 38:349–357
25. Kronauer T, Pohlmann J, Matthe M, Smejkal T, Fettweis G (2021) Latency analysis of ROS2 multinode systems
26. Maruyama Y, Kato S, Azumi T (2016) Exploring the performance of ROS2. In: Proceedings of the 13th International Conference on Embedded Software, pp 1–10
27. Park J, Delgado R, Choi BW (2020) Real-time characteristics of ROS 2.0 in multiagent robot systems: an empirical study. IEEE Access 8:154637–154651
28. Thulasiraman P, Chen Z, Allen B, Bingham B (2020) Evaluation of the Robot Operating System 2 in Lossy Unmanned Networks. In: 2020 IEEE International Systems Conference (SysCon). IEEE, pp 1–8
29. Kim J, Smereka JM, Cheung C, Nepal S, Grobler M (2018) Security and performance considerations in ros 2: a balancing act. arXiv preprint arXiv:1809.09566
30. Montgomery DC (2017) Design and analysis of experiments. Wiley, London
31. Sadjadi SM, Shimizu S, Figueroa J, Rangaswami R, Delgado J, Duran H, Collazo-Mojica XJ (2008) A modeling approach for estimating execution time of long-running scientific applications. In: 2008 IEEE International Symposium on Parallel and Distributed Processing, pp 1–8. https://doi.org/10.1109/IPDPS.2008.4536214
32. Open Robotics: ROS 2 Foxy Fitzroy. https://docs.ros.org/en/foxy/Releases/Release-Foxy-Fitzroy.html. Accessed 11 Oct 2021
33. MIT Kerberos: Kerberos V5 System Administrator's Guide. https://web.mit.edu/kerberos/krb5-1.10/krb5-1.10.7/doc/krb5-admin.html. Accessed 11 Oct 2021
34. Arguedas M, Ragnarok S, Thomas D (2020) ROS 2 Releases and Target Platforms. https://ros.org/reps/rep-2000.html. Accessed 11 Oct 2021
35. Github: Releases. https://github.com/ros2/ros2/releases. Accessed 11 Oct 2021
36. Open Robotics: About Quality of Service settings. https://docs.ros.org/en/foxy/Concepts/About-Quality-of-Service-Settings.html. Accessed 11 Oct 2021
37. Wireshark: Wireshark. https://www.wireshark.org/. Accessed 11 Oct 2021
38. The Tcpdump Group: TCPDUMP/LIBPCAP public repository. https://www.tcpdump.org/. Accessed 11 Oct 2021
39. Wireshark: tshark. https://www.wireshark.org/docs/man-pages/tshark.html. Accessed 11 Oct 2021
40. Microsoft: PowerShell. https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/powershell. Accessed 11 Oct 2021
41. Python: Python 3.0 Release. https://www.python.org/download/releases/3.0/. Accessed 11 Oct 2021
42. SciPy: SciPy. https://scipy.org/. Accessed 11 Oct 2021