DDS-CERBERUS: IMPROVING SECURITY IN DDS MIDDLEWARE USING KERBEROS TICKETS

THESIS

Andrew T. Park, Second Lieutenant, USAF

AFIT-ENG-MS-22-M-052

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-22-M-052

DDS-CERBERUS: IMPROVING SECURITY IN DDS MIDDLEWARE USING KERBEROS TICKETS

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Science

Andrew T. Park, B.S.C.S.

Second Lieutenant, USAF

March 24, 2022

AFIT-ENG-MS-22-M-052

DDS-CERBERUS: IMPROVING SECURITY IN DDS MIDDLEWARE USING KERBEROS TICKETS

THESIS

Andrew T. Park, B.S.C.S.
Second Lieutenant, USAF

Committee Membership:

Maj Richard Dill, Ph.D.
Chair

Dr. Douglas D. Hodson, Ph.D.
Member

Lt Col Wayne C. Henry, Ph.D.
Member

AFIT-ENG-MS-22-M-052

Abstract

The military deploys many Internet of Things (IoT) in battlefield operations to provide information on terrain and enemy combatants. It also deploys automated robots or unmanned aerial vehicles (UAVs) where securing and trusting collected data is essential. Choosing the middleware that handles this message transfer is crucial for real-time operations. Networks with multiple entities, including IoT devices, UAVs, and small computers, require robust middleware facilitating message sending in real-time. Ideally, the middleware would provide Quality of Service (QoS) to handle lost packets and retransmissions in lossy environments, especially between low-power machines. Data Distribution Service (DDS) is a middleware that implements real-time and QoS capabilities by sending messages, not based on endpoints but *topics*. However, DDS nodes are susceptible to impersonation attacks, which compromise integrity and trust. To mitigate these attacks, DDS-Cerberus (DDS-C) is developed as a security layer that integrates with DDS by using Kerberos tickets to identify and authenticate valid DDS nodes. This thesis evaluates DDS-C performance, determining if authentication overhead impedes DDS operations by using Robot Operating System 2 (ROS 2) and Cyclone DDS as testbeds. Additionally, DDS-C is integrated into a commercial network artificial intelligence (AI) provided by Bright Apps as a real-world use case. The results of this research conclude that DDS-C does not impact DDS operations to any significant degree. The added security and minimal middleware impact could help the military ensure node integrity in operational missions.

Acknowledgments

I firstly thank my God for giving me this opportunity to grow. I also thank my family, friends, loved ones, and committee for giving me the support, strength, and motivation to achieve more. They gave me the tools and advice to improve my writing and research. Thank you to my advisor, Major Dill, for the various opportunities to learn more as a researcher and student. The memories I have created here will stay with me for an eternity.

Andrew T. Park

Table of Contents

List of Figures

List of Tables

DDS-CERBERUS: IMPROVING SECURITY IN DDS MIDDLEWARE USING KERBEROS TICKETS

I. Introduction

Technology use from handheld to stationary devices has melded private and public sectors with similar software and architectures. This uniformity provides benefits such as ease of use and interoperability; however, there are more security vulnerability risks. Network and device resilience are of utmost priority to protect from exploitation. This research explores DDS-Cerberus (DDS-C), a novel security layer, mitigating node authentication vulnerabilities in Data Distribution Service (DDS). This chapter goes over the research's problem, motivation, and objectives.

1.1 Problem and Motivation

The military has adopted different technologies ranging from small Internet of Things (IoT) devices to multi-layer software. Many of these technologies cannot function effectively without a framework that provides communications in real-time. The military has adopted the Object Management Group (OMG) DDS standard as an integral part of several systems due to its robustness and easy integration. Different military systems use DDS on reliable real-time performance for decentralized architectures where there are many moving components. One such DDS implementation is Real-Time Innovations (RTI) Connext DDS which has been deployed in several systems such as in General Atomics (GA) Advanced Cockpit Ground Control Stations and the United States (U.S.) Navy's Ship Self Defense System (SSDS) [1, 2]. The U.S. Army also uses DDS which is a part of Robot Operating System-Military

(ROS-M) to support ground vehicle operations [3].

The military trusts their data and systems by using DDS because of continued support with software patches. However, even with this reassurance, there are still security vulnerabilities in old and new DDS implementations. The increase of public-facing IoT devices and their vulnerabilities reveal that there are public-facing government devices found with vulnerabilities. Additionally, multiple access points, routers, and handheld devices on military bases could be outdated or unsecured. One such vulnerability is inadequate authentication due to low computational power and battery life [4]. Attackers can compromise data integrity through impersonation attacks by spoofing nodes. This is a vulnerability faced by DDS and its Robot Operating System 2 (ROS 2) implementation [5].

1.2 Research Objectives

The goal of this research is to explore the impact DDS-C, an authentication layer, has on DDS operations and to conclude if the impact does not hinder DDS operations. DDS-C is a novel security layer developed to mitigate inadequate authentication and impersonation attacks on DDS [6, 7, 8, 9]. It authenticates DDS nodes with Kerberos tickets using *keytabs*, long-term keys, before the nodes can send and receive messages. No research was found that combined DDS and Kerberos. Through various experiments measuring latency and packet capture traffic in different network setups, DDS-C is shown to provide insignificant overhead to regular DDS operations.

1.3 Thesis Overview

This thesis is organized around four scholarly papers, each uniquely experimenting DDS-C through concept and design, testing, and application. The chapters containing papers are labeled by their titles. Chapter II, the background, explains the main

components of DDS-C and its setup which leads into the four papers. Chapter III, the first paper, lays out the concept and design of DDS-C and initial experiments. Chapter IV, the second paper, is the preliminary DDS-C testing measuring latency. Chapter V, the third paper, provides more experiments by conducting statistical analysis of packet capture quantities. Continuing this work, the fourth paper in Chapter VI also captures packet quantities but applies the work to specific real world use cases. Lastly, Chapter VII wraps up the thesis with a conclusion and future works.

II. Background

This chapter provides essential information on two DDS-Cerberus (DDS-C) components: Data Distribution Service (DDS) and Kerberos. In addition to DDS, this chapter also describes the testbeds Robot Operating System 2 (ROS 2) Foxy Fitzroy and Cyclone DDS. It also describes DDS-C and its development.

2.1 Data Distribution Service (DDS)

DDS is a publish-subscribe middleware protocol developed by Object Management Group (OMG) [10]. It is an open standard that other vendors such as Real-Time Innovations (RTI), Eclipse Foundation, and eProsima can use to create implementations [11]. Lying in between the operating system and the applications, DDS facilitates sending messages over a network between different operating systems and architectures. It sends messages not based on source and destination but specified *topics*. *Topic* examples are unique strings that determine where the message is delivered. Quality of Service (QoS) attributes are used in conjunction with these *topics* to determine how messages behavior.

The DDS standard defines different layers to parse and send messages. The research uses the Data-Centric Publish-Subscribe (DCPS) layer because it is the layer where messages are sent and received through a global domain space. Figure 1 shows how messages are sent in DDS. Each node contains the relevant components. They contain at least one *domain participant* which contain data writers, data readers, publishers, and subscribers. Each publisher and subscriber uses a paired data writer and data reader to write and read messages. Different *topics* are included in each message to be sent by a publisher. Subscribers specify a *topic* and only read messages that have the same *topic*. For example, a publisher sends a message with *topic* A,

and a subscriber searching for *topic* A can read that message's data.

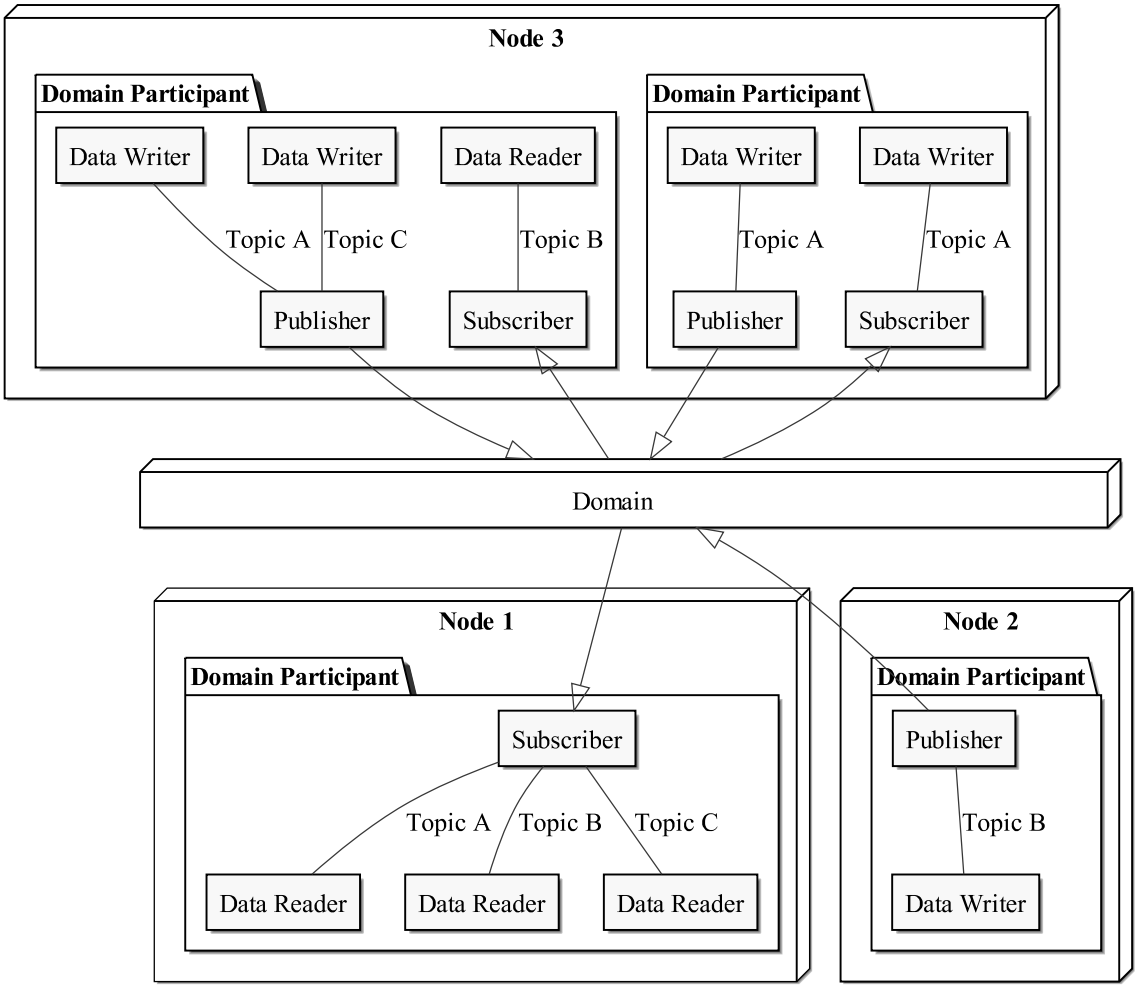

Figure 1: DDS Components and Data Flow [6]

Developers can modify publisher and subscriber behavior using different QoS attributes to affect message liveliness, reliability, and duration. For different network setups and applications, selecting different combinations of QoS allows DDS to handle situations such as operating in lossy environments or with large data files. In addition to different network setups, DDS offers dynamic discovery that allows existing and new DDS applications to communicate with each other during runtime. Dynamic discovery also identifies what publishers and subscribers are doing on the network to help match DDS participants.

Even with different vendors, the core components of DDS are the same across all versions. Two different DDS implementations are the highlight of this research: ROS 2 Foxy Fitzroy and Cyclone DDS. The next two subsections describe these two implementations. Both are installed on Ubuntu Linux and use Python to create DDS components for consistency.

2.1.1 Robot Operating System 2 (ROS 2)

This research uses ROS 2 as the main testbed for preliminary DDS-C testing before experimenting DDS-C with Cyclone DDS for real-world use cases. ROS 2 is chosen because of its popularity, ease of use, and integration with DDS. ROS 2 is built from Robot Operating System (ROS) and introduces real-time capabilities through DDS [12]. It was developed to help robotics development, but in this research, the application is applied to Internet of Things (IoT) and unmanned aerial vehicle (UAV) technologies. The goal is to experiment with the DDS components of ROS 2. There are many ROS 2 distributions, each with their benefits, such as what middleware implementations they support. Due to its long-term support, ROS 2 Foxy Fitzroy is chosen as the testbed [13]. It uses eProsmia Fast-RTPS as the default middleware [14].

2.1.2 Cyclone DDS

Cyclone DDS is an open-source Eclipse IoT project that is akin to ROS 2 [15, 16]. It was developed to support Eclipse IoT projects and existing Eclipse solutions. The Python binding for Cyclone DDS is used in this research [17]. Cyclone DDS is hosted on GitHub for updates and improvements. Bright Apps is the research sponsor using Cyclone DDS in its network architecture [18]. It uses this implementation to control its UAVs and to send video frames for its artificial intelligence (AI) to process.

Integrating DDS-C into the Bright Apps architecture and network is important in experimenting with real-world use cases such as search and rescue and battlefield operations.

2.2 Kerberos

Kerberos is an authentication protocol used to provide authentication for entities on the network by using symmetric key cryptography [19]. It was developed by Massachusetts Institute of Technology (MIT) and is open source. Kerberos handles authentication by providing tickets that allow the authenticated entity to talk to other entities on the network. The Key Distribution Center (KDC) provides the ticket through the Authentication Server (AS), making sure the entity is registered, and Ticket Granting Server (TGS), granting a ticket after verifying and authenticating the entity. This authentication is demonstrated in Figure 2, a sequence diagram where Node A is authenticating to a network to get permission to talk to Node B. Node A first sends a message to the AS to talk to the TGS. Since it is registered, Node A is authenticated and can talk to the TGS using a special message given by the AS. Node A tells the TGS it wants to talk to Node B. The TGS gives Node A a ticket to talk to Node B and a message for Node B. This message allows Node B to receive a ticket, so it also receives permission to talk to Node A.

The Kerberos `kinit` command's function provides a ticket given the correct credentials: username and password [20]. Each Kerberos service has a realm name identifier used to retrieve a ticket. For example, a user uses the username cerby to get a ticket from Kerberos realm EXAMPLE.COM. After running `kinit cerby@EXAMPLE.COM`, the user is prompted to enter the password for the account. If the username does not exist, the user is not prompted. A way to streamline this process is to use *keytabs*, long-term keys that do not expire [21]. *Keytabs* are created on a machine by pro-

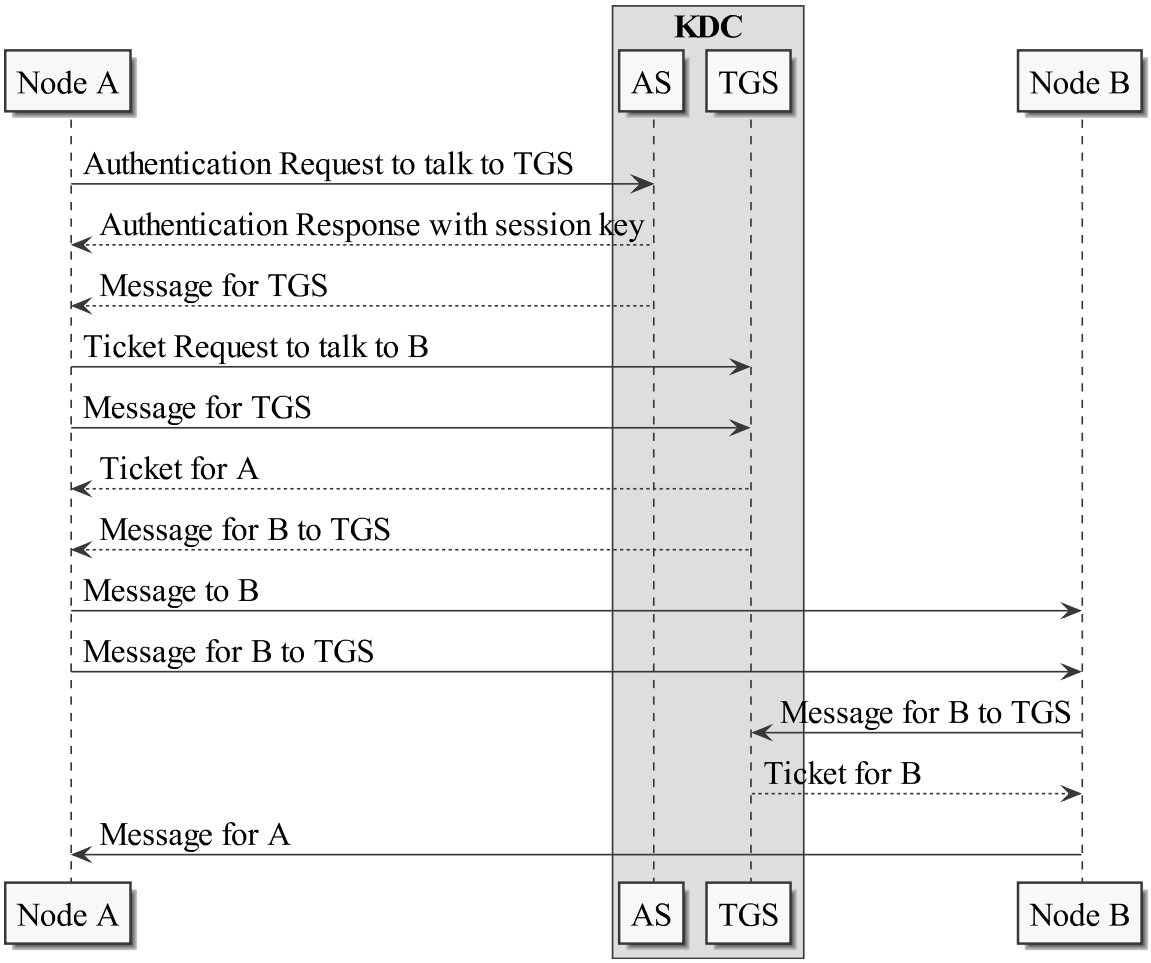

Figure 2: Kerberos Authentication Sequence Diagram [6]

viding the encryption scheme, and the correct credentials for the realm [22]. The resulting .keytab files are stored in a directory, usually in `/tmp`. The command to run `kinit` with a *keytab*: `kinit -k -t /tmp/cerby.keytab cerby@EXAMPLE.COM`. DDS-C utilizes *keytabs* to authenticate DDS participants. It is a crucial component and important to understand when implementing and experimenting with DDS-C.

2.3 DDS-Cerberus (DDS-C)

DDS-C is a layer designed to add additional security to DDS by providing node authentication [6, 7, 8, 9]. Its inception was brought up by papers identifying node

integrity and authentication issues in different DDS and ROS 2 implementations [5, 23, 24]. Attackers were able to perform impersonation attacks on nodes to compromise data integrity. DDS-C mitigates this concern by using Kerberos to authenticate nodes or DDS participants. When a node is executed, and before its normal operation, it calls the `kinit` command to receive a ticket. Each node has a unique *keytab* associated with it, allowing for individual authentication. When the node receives a ticket, it can resume normal operations. Kerberos authentication using *keytabs* prevents attackers from impersonating because an attacker would need to steal or replicate a *keytab* to be authenticated on the DDS domain.

Figure 3 outlines the authentication and message sending process with a publisher and subscriber node. First, each node is authenticated by communicating with the Kerberos Server KDC as seen in A and C; in this case, Publisher1 authenticates first and then Subscriber1. After their authentication, they are able to perform normally in B and D: Publisher1 sends a message with a *topic* that Subscriber1 is to receive. Authentication happens at the beginning of the node lifetime; therefore, in E, the subsequent messages are not interrupted by any DDS-C authentication.

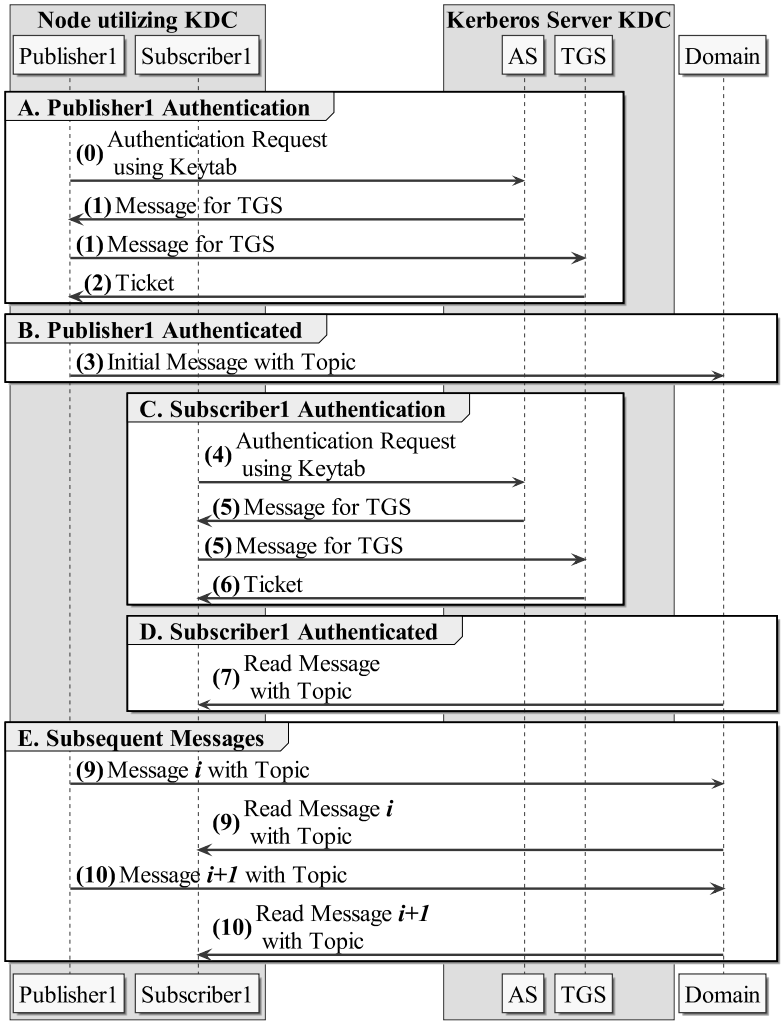

Figure 3: DDS-C Authentication Process [8]

DDS, ROS 2, Cyclone DDS, and Kerberos are important components in understanding the creation of DDS-C. The next four chapters expand on this understanding by providing more background, relevant works, and experimentation. Table 1 outlines the contributions the next four papers make using these components for DDS-C. Chapter III introduces DDS-C by describing DDS, Kerberos, and ROS 2; chapter IV flushes out DDS-C with more information on the three components with experimentation; chapter V quantifies packet traffic in ROS 2; and finally chapter VI introduces Cyclone DDS and offers additional use case experimentation for DDS-C.

Table 1: DDS-C Papers

Chapter	**DDS-C Contribution**
III	DDS-C initial concept by integrating Kerberos with DDS publishers and subscribers. Includes experiment setup using ROS 2.
IV	Experiment execution using ROS 2 and the integration of Kerberos *keytabs*. Measured the average processing times for baseline experiments against DDS-C experiments.
V	On ROS 2 with DDS-C, the packet traffic was captured and categorized into different data categories to compare DDS-C authentication traffic with DDS message traffic.
VI	Continuing the third paper's work, Cyclone DDS was used as the testbed to apply DDS-C in real-world use case experiments.

III. Paper: DDS-Cerberus: Data Distribution via Ticketing

The following paper, "DDS-Cerberus: Data Distribution via Ticketing," was submitted and accepted by the World Congress in Computer Science, Computer Engineering; it was published in July of 2021. This paper is in Institute of Electrical and Electronics Engineers (IEEE) format.

DDS-Cerberus: Data Distribution via Ticketing

Andrew T. Park, Richard Dill, Douglas D. Hodson, Wayne C. Henry
Department of Electrical and Computer Engineering
Air Force Institute of Technology, Wright-Patterson Air Force Base, OH, USA
email: andrew.park@afit.edu, richard.dill@afit.edu, douglas.hodson@afit.edu, wayne.henry@afit.edu

***Abstract*—Data Distribution Service (DDS) has vulnerabilities due to its publish-subscribe model. This paper proposes a novel design, DDS-Cerberus (DDS-C), to increase Internet of Things (IoT) security by integrating Kerberos authentication into ROS 2's (Robot Operating System) DDS communication framework. This method ties Kerberos's key distribution center (KDC) with specific QoS parameters to efficiently prevent common security vulnerabilities such as replay attacks and data manipulation. This research is a work in progress; in this paper, we detail DDS-C and propose functional tests, security evaluations for significant attacks, and performance assessments with varying metrics. The anticipated results are that DDS's vulnerabilities would be mitigated through the KDC even with a large number of entities and constant real-time data flow.**

***Index Terms*—IoT, DDS, Kerberos, distributed system, publish-subscribe**

I. Introduction

Internet of Things (IoT)-based architecture schemes offer consumers different methods to cost-effectively and sustainably process IoT data. Whether it be individuals, companies, or cities, this data introduces new insights into the increasingly connected world.

There are many application-layer protocols to handle these devices and data. Widely used publish-subscribe protocols include Message Queue Telemetry Transport (MQTT), mainly for wireless and low-bandwidth networks [1], and Advanced Message Queuing Protocol (AMQP), for business messaging to establish reliable asynchronous communication [2]. The third protocol and focus of this paper, Object Management Group's (OMG) Data Distribution Service (DDS) [3] [4], has been used by various industries from finance to the government sector for approximately 30 years. It supports smart grids, air-traffic control, healthcare, defense, and robotics [5]. Even though DDS has efficient data transportation, it has security vulnerabilities when sending data over the network such as data type manipulation and unregistered *Publishers* and *Subscribers* [4]. To securely handle these issues, this paper proposes DDS-Cerberus (DDS-C), a novel DDS publish-subscribe solution that uses Kerberos's ticket system. The evaluation experiments assess security and efficiency through modifiable metrics. They are conducted with virtual machines running modified ROS 2 (Robot Operating System) code incorporating Kerberos open-source code.

Section II introduces the background in DDS and Kerberos. Section III covers the DDS-C design approach. Section IV presents the implementation details. Section V captures the assumptions and limitations. Section VI offers conclusions and recommendations for future research.

II. Background

This section describes the Data Distribution Service (DDS) and Kerberos and illustrates why they are used to improve network security for distributed systems. Both have a publish-subscribe architecture; however, they differ in their data implementation and handling methods. DDS's goal is to efficiently transport data, while Kerberos aims to improve security via a ticket granting server.

A. Overview of DDS

DDS transports data with definable quality-of-service (QoS). Two popular implementations of the on Object Management Group's (OMG) DDS standard are Real-Time Innovations (RTI) Connext DDS Secure [6], which adds more security features such as asymmetric keys and certificate authorities, and OpenDDS [7] by Object Computing Incorporated (OCI), which is an open-source C++ implementation [8] [9].

DDS has three layers: Data Local Reconstruction Layer (DLRL), Data-Centric Publish-Subscribe (DCPS), and Real-Time Publish-Subscribe (RTPS). This research focuses on the DCPS layer because it provides core DDS elements [10]. Figure 1 illustrates the following components and data flow:

- ***Domain*:** Main area of data transfer from *Domain Participants*.
- ***Node*:** Contains at least one *Domain Participant*.
- ***Domain Participant*:** The entity that participates in the *Domain*. Contains *Subscribers*, *Publishers*, *Writers*, and *Topics*.
- ***Data Reader*:** An interface for the *Subscriber* to read subscribed data.
- ***Subscriber*:** The entity that reads data from *Publishers*. Associated with at least one *Data Writer*.
- ***Data Writer*:** An interface for the *Publisher* to write data to be published.
- ***Publisher*:** The entity that publishes data. Associated with at least one *Data Reader*.
- ***Topic*:** Defines the data sent in the *Domain*. Has three characteristics: name, data type, and QoS.

Currently, there are 22 different QoS attributes that define communication standards to transported data [9]. This paper focuses on these two QoS attributes:

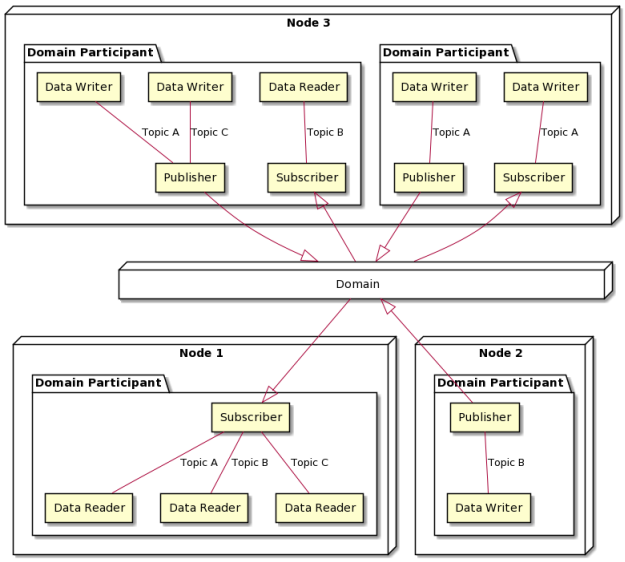

Fig. 1. DDS Components and Data Flow (reproduced from [8])

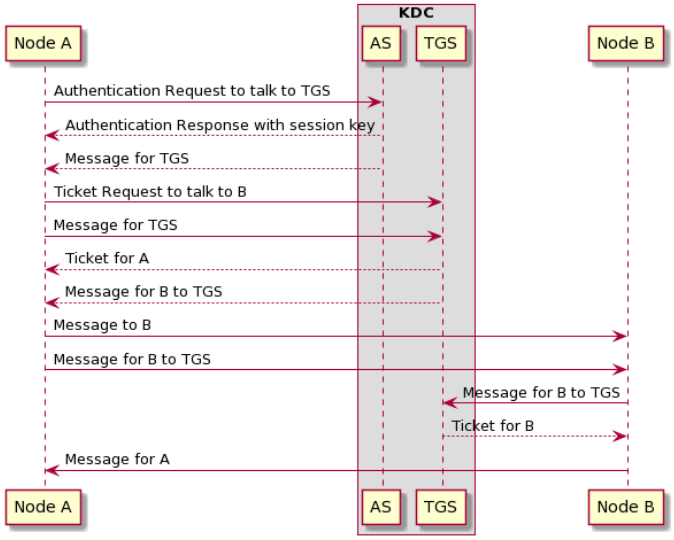

Fig. 2. Kerberos Sequence Diagram [16]

- ***Ownership*:** SHARED *Ownership* - data published by multiple entities at the same time. EXCLUSIVE *Ownership* - data published by a single entity at one time.
- ***Discovery*:** LOCATORLIST environment variable - helps entities discover other *Publishers* or *Subscribers*.

There are two main reasons to use DDS over other protocols. First, in DDS, unlike many other architectures, data is sent, not based on the source and destination but based on specific attributes. The application developer that uses DDS does not need to specify the destination for data sent to other applications. *Domain Participants* need only access the *Domain* to read or write data [11].

Second, DDS has a flexible publish-subscribe system since data is referenced by data type in the *Topic* which helps determine how the data behaves. DDS is also flexible because it has modifiable built-in plugins (e.g. functions for more secure authentication and authorization in the *Domain* space [12]). The flexibility gives developers and researchers more control over features. Since DDS has plugins and a decentralized system that focuses on data type, adding another layer of security by incorporating Kerberos's ticket system using plugin functionality can improve security [4] [8].

B. Overview of Kerberos

Kerberos, managed by the Massachusetts Institute of Technology (MIT), is an open-source network authentication protocol [13]. The protocol allows nodes to communicate over a non-secure network with tickets distributed after verification with a server, the key distribution center (KDC). If a node is not authorized by the KDC or does not have a ticket, it cannot communicate with another node on the same network. The KDC contains an authentication server (AS) and ticket granting server (TGS) [14] [15]. For example in Figure 2, node A wants to talk to node B:

1) **A sends to AS:** unencrypted message to talk to TGS.
2) **AS sends to A:** confirmation message with a session key; encrypted message for TGS.
3) **A sends to TGS:** encrypted message (with session key) that states it wants to talk with B; message for TGS from AS.
4) **TGS sends to A:** A retrieves its ticket from TGS- encrypted message with new session key between A and B; encrypted message for B to send to TGS.
5) **A sends to B:** encrypted message to B with A's time stamp TS_A; message for TGS from A.
6) **B sends to TGS:** B retrieves its ticket from TGS.
7) **B sends to A:** encrypted message to A with $TS_A + 1$.

The messages in Figure 2 have a nonce (n), a time stamp (TS), and time-to-live (TTL). These attributes prevent replay attacks and ensure logging of messages.

C. Motivation for DDS-C

This research uses Kerberos to improve DDS's publish-subscribe system in DDS-Cerberus (DDS-C). There are three main reasons: lasting credentials, more secure communication, and increased data security.

First, the Kerberos credentials do not have to be changed often due to the use of tickets. Administrators of this design do not need to have a separate server to handle network access.

Second, Kerberos can provide a more secure communication protocol. The transport protocol for DDS defaults to User Datagram Protocol (UDP), but it can support Transmission Control Protocol (TCP). DDS does not have a default method to secure these protocols. One security protocol that TCP uses is Secure Sockets Layer (SSL) which creates encrypted keys for both client and server. Al-Ayed and Liu [16] compared Kerberos to SSL and found that Kerberos was more secure since it used the KDC to authenticate the credibility of nodes and server. They found that the TGS creates and encrypts a

ticket that neither the client or AS knows about, therefore, blocking sniffing attacks.

Third, Abdulghani et al. [9] compared security vulnerabilities of well-known protocols DDS, Message Queue Telemetry Transport (MQTT), and Long Range Wide Area Network (LoRaWAN) to discuss main security issues and solutions. They highlighted common vulnerabilities such as eavesdropping, unauthorized access, and replay attacks for all three protocols. They presented DDS's security issues as unauthorized access to data and devices, data manipulation, network disruption, and eavesdropping. Section IV outlines DDS-C tests to address and mitigate these issues.

The following section proposes a complementary novel security scheme that supports efficiency while upholding the CIA (confidentiality, integrity, availability) security principles.

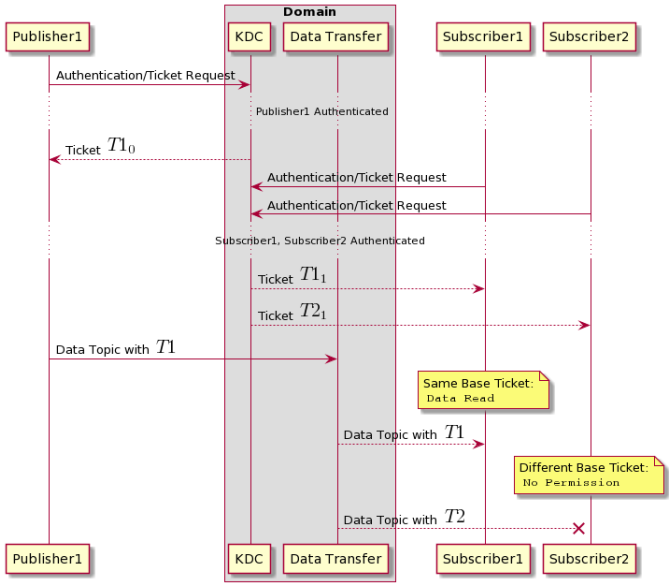

Fig. 3. DDS-C sequence diagram demonstrating *Topic* with ticket functionality

III. Design

DDS-Cerberus (DDS-C) is a novel communication protocol that combines Kerberos security with Data Distribution Service's (DDS) efficiency for Internet of Things (IoT) data. Adding a ticket to the *Topic* protects against data vulnerabilities: sending in plain text, unencryption, and data manipulation. The main focus of DDS-C is Kerberos's key distribution center (KDC) that uses symmetric keys and tickets. This section describes DDS-C's ticket and *Topic* system and states its benefits.

When using DDS-C, the users who register with devices such as IoT devices, external servers, computers reading data are either a *Data Writer* or *Data Reader* assigned to a *Subscriber* or *Publisher*. Each *Subscriber* and *Publisher* has a ticket given by the KDC. *Data Writers* give data for the *Subscriber* to send, and *Data Readers* read data retrieved by the *Publisher*. The DDS framework handles the management of *Data Readers* and *Data Writers* to *Subscribers* and *Publishers*. The KDC framework manages tickets assigned to *Subscribers* and *Publishers*.

A. DDS-C: DDS with Kerberos

The Kerberos ticket function uses tickets linked with quality-of-service (QoS) to ensure the CIA (confidentiality, integrity, availability) security principles. DDS-C is more efficient and sustainable for Internet of Things (IoT) sensors that need to send in continuous streams of data. Tickets base their longevity on specific QoS parameters and defined categories of *Subscribers* and *Publishers*. Kerberos keeps its functionality, but the main change is that certain linked tickets, instead of being separate from each other, receive data from specified end points.

In Figure 3, the KDC is part of the *Domain* to authenticate and distribute tickets. A ticket is added to a *Topic* (name, type, QoS, ticket). Each *Subscriber* and *Publisher* registers with the authentication server (AS). Afterward, they go to the ticket granting server (TGS) to retrieve a ticket. There can be multiple *Data Writers* and *Data Readers* assigned to a *Subscriber* and *Publisher*, respectively. In DDS, each *Subscriber* and *Publisher* can either be associated with *Data Writers* and *Data Readers*; however, DDS-C modifies this rule by only having *Subscribers* associated with *Data Readers* and *Publishers* associated with *Data Writers*.

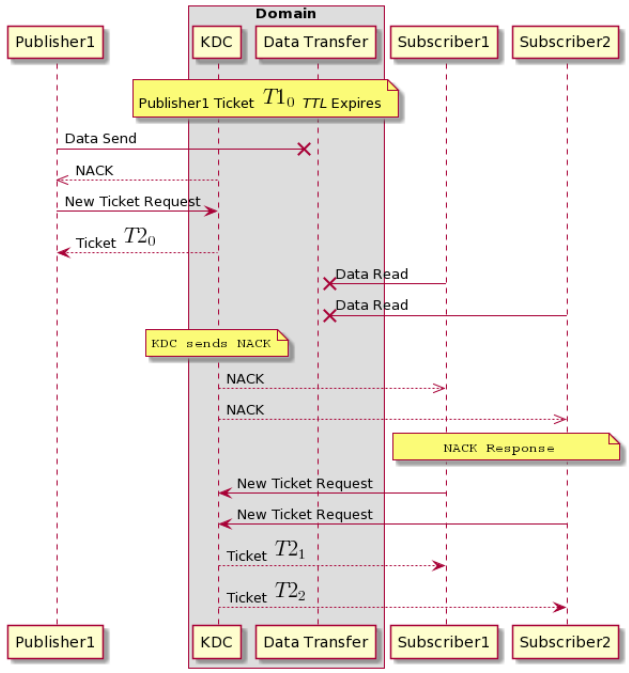

Fig. 4. DDS-C sequence diagram demonstrating ticket reissuing

Instead of only storing tickets in Kerberos, the KDC in DDS-C stores the QoS with its respective ticket. If the *Ownership* is SHARED for a ticket, then any entity can query it. When the *Ownership* is EXCLUSIVE, then only one entity can query it. The KDC tracks who has the ticket and either

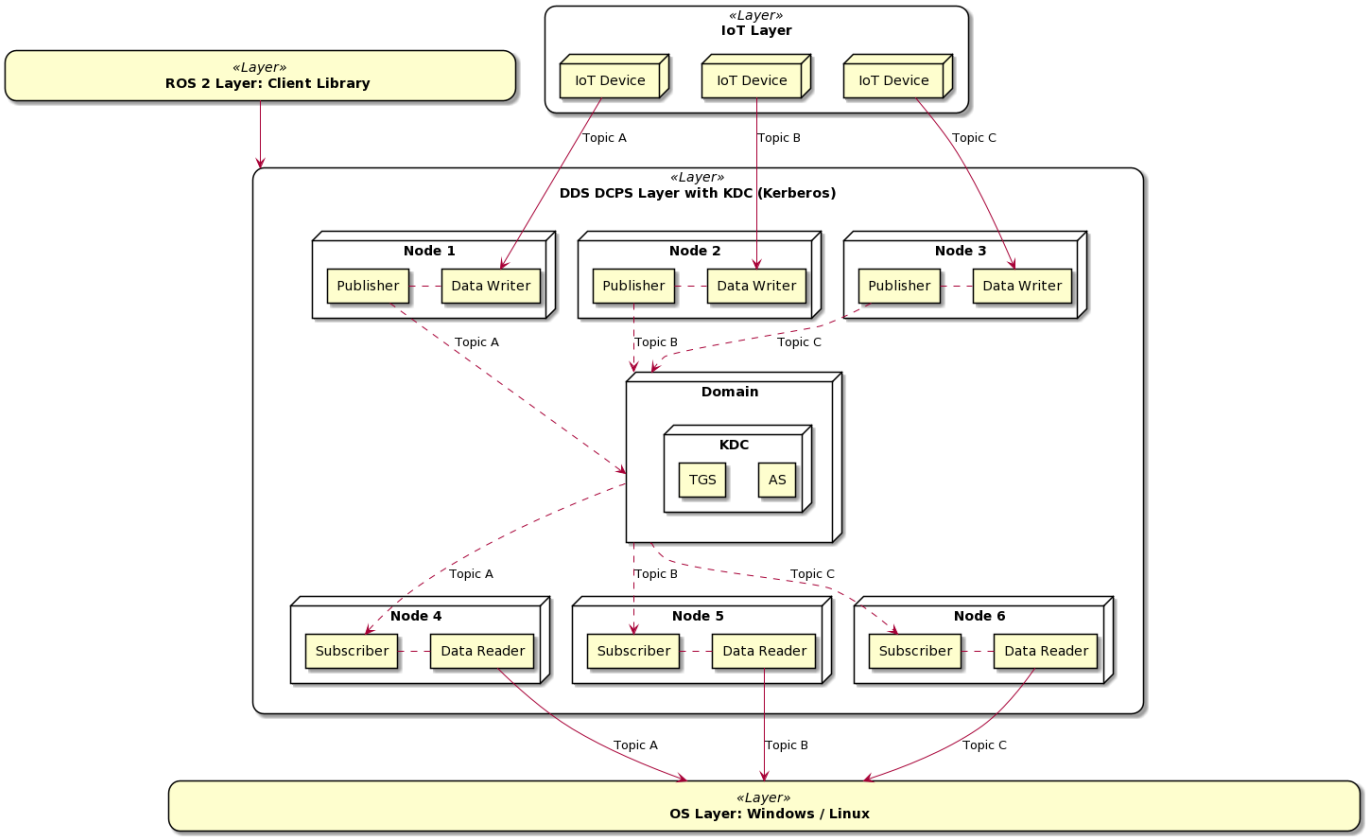

Fig. 5. DDS-C Implementation: Layer Diagram with ROS 2 [8]

prevents or allows other entities from receiving tickets based on the QoS. Let each ticket be T. If a *Subscriber* wants to read data from a *Publisher*, the *Publisher* first needs to query the KDC to get a ticket $T1_0$ as seen in Figure 3. Next, the *Subscriber* identifies that the *Publisher* received a ticket and then queries the KDC to get ticket $T1_1$ that corresponds to the *Publisher* by having the same name but unique subset of the base ticket. Depending on QoS *Ownership*, multiple *Subscribers* can get data from the *Publisher* and have the ticket name and unique subset n: $T1_2, T1_3, T1_4, ..., T1_{n+1}$. In Figure 3, *Subscriber1* can read the data sent by the *Publisher1* since it has the same ticket name $T1$, but *Subscriber2* cannot because this ticket is not tied to *Publisher1's* ticket $T1$.

The ticket's time-to-live (TTL) depends on when the ticket granting server (TGS) distributed it. In Figure 4, the TGS assigns a TTL to a ticket (not an individual data message). After the TTL expires, the ticket is invalid, and when a *Subscriber* or *Publisher* wants to subscribe or publish any data, it needs to query the KDC for a new ticket. The KDC and *Domain* do not send a negative acknowledgment (NACK) when a ticket expires; they send a NACK if a *Subscriber* or *Publisher* sends a message with an expired ticket. If a *Publisher* receives a new ticket due to a NACK and if any *Subscribers* sent messages that were tied to the *Publisher's* expired ticket, the TGS sends out a NACK to these *Subscribers* to get a new ticket. *Publisher1* in Figure 4 receives a new ticket $T2_0$ since ticket $T1_0$'s TTL expired. Therefore, *Subscriber1* and *Subscriber2*, since they are tied to $T1$, are notified by a NACK that their tickets have expired. They respond by requesting a new ticket $T2_1$ and $T2_2$, respectively.

B. Benefits

Kerberos applies symmetric key encryption after the initial public key exchange required for user registering which is faster than exclusively using public key encryption [17]. Furthermore, it has been predicted that quantum computers will be able to break public key encryption in contrast to symmetric keys [18]. Symmetric key cryptography is an inexpensive cryptographic function suitable for low-power Internet of Things (IoT) devices. It does not have to use certificate authorities for security due to session keys and tickets [19]. The use of a symmetric key prevents attackers from changing the quality-of-service (QoS) *Ownership* or *Discovery* to route data to other sources. Even though attackers can sniff network traffic, the messages are encrypted through symmetric keys which are reinforced by the issued tickets. The use of a nonce (n), time-to-live (TTL), and time stamp (TS) are crucial to prevent replay attacks. Implementing a key distribution center (KDC) in the *Domain* only allows authorized *Subscribers* and *Publishers* to access data and talk to other entities. Only authorized entities with valid tickets have access to certain data, and the ticket's TTL ensures that old tickets are not

used for long periods, precluding replay attacks and data manipulation.

IV. Implementation

Implementation testing focuses on three areas: functionality, security, and performance. To test DDS-Cerberus (DDS-C) functionality, ROS 2 (Robot Operating System) is used as the Data Distribution Service's (DDS) model modified with a plugin that adds in Kerberos security. These tests are simulated with different scenarios. The security evaluation tests DDS-C against eavesdropping, Internet Protocol (IP) spoofing, denial-of-service (DoS) / distributed DoS (DDoS) attacks, man-in-the-middle (MITM) attacks, and replay attacks. The performance assessments outline how the functional tests are performed with varying parameters in entity size, data transmission, and file size. This research is a work in progress, and its proposed evaluations will be detailed in a later paper.

A. Setup

Figure 5 illustrates how DDS-C implementation works by having four prominent layers: Internet of Things (IoT), ROS 2, DDS Data-Centric Publish-Subscribe (DCPS), and operating system (OS) layer. A network of virtual machines (VMs) would be set up with one server which hosts ROS 2, DDS, and Kerberos functions and middleware. In the IoT layer, each IoT sensor entity represents a *Data Writer* assigned to a *Publisher*; a function emulates this and sends *Topic* data. The ROS 2 Client Library modifies and complements the DDS DCPS layer; a DDS plugin in C++ code implements the DDS-C's Kerberos modifications. *Data Readers*, one or more virtual clients, receive data *Topics* going to the OS layer. The goal is for DDS-C to allow devices to securely send and receive messages using tickets in various tests.

TABLE I
Simulation Scenarios

	Scenario							
	1	***2***	***3***	***4***	***5***	***6***	***7***	***8***
Number of Subscribers	5	10	10	20	15	30	20	40
Number of Publishers	5	10	10	20	15	30	20	40
Number of Data Writers	100		200		300		400	
Number of Data Readers	100		200		300		400	

B. Simulations

Testing focuses on *Subscribers*, *Publishers*, *Data Writers*, and *Data Readers*. Table I has eight unique scenarios. Each *Subscriber* and *Publisher* in scenarios 1, 3, 5, and 7 are tied to 20 *Data Writers* and 20 *Data Readers*, respectively. Likewise, each *Subscriber* and *Publisher* in scenarios 2, 4, 6, and 8 are tied to 10 *Data Writers* and 10 *Data Readers*, respectively.

Table II outlines the tests done in the three areas: functionality, security, and performance. All tests are simulated with the eight scenarios in Table I.

TABLE II
Simulation Tests

	Tests
Functional Test	Analyze registration processing time for *Subscribers* and *Publishers* in the authentication server (AS).
	Analyze ticket processing time for *Subscribers* and *Publishers* in the ticket granting server (TGS).
	Analyze expired time-to-live (TTL) ticket redistribution processing time for *Subscribers* and *Publishers*.
Security Evaluation	Evaluate against vulnerabilities with a malicious actor: • Eavesdropping attacks. • IP spoofing attacks. • DoS/DDoS attacks. • MITM attacks. • Replay attacks.
Performance Assessments	Analyze data processing time for sending messages: • Send small 10-100 kb files such as text files and small graphics. • Send large 1-5 mb files such as high-resolution JPEG images and mp3 files.

Functional Test analyzes the processing time for authentication, ticket retrieval, and ticket redistribution by measuring the round-trip time (RTT) of sending the message and getting a response from the key distribution center (KDC). The amount of messages sent equals the number of *Subscribers* and *Publishers*.

Security Evaluation evaluates major attacks, eavesdropping, IP spoofing, DoS/DDoS, MITM, and replay attacks, encompassing Abdulghani et al.'s [9] DDS security vulnerabilities. One message is sent for each attack per each test. *Security Evaluation* focuses on the attempting access phase. This phase is selected since keys and tickets are used to prevent and deter outsider access and manipulation of data. Attempting access would be client-side attacks attempting to steal session keys, gain authorization, and manipulate data. The research, however, does not focus on what happens when an attacker registers into the authentication server (AS). Listeners (e.g., Wireshark [20]) collect this data on the same network during transmission.

Performance Assessments analyzes ticket redistribution and the amount of data sent per interval of time. There are five sub-tests for the number of messages sent: 100,000, 200,000, 300,000, 400,000, and 500,000 messages. The following metrics are measured:

- **Samples per second:** the number of messages a *Subscriber* receives per time unit (second).
- **Throughput:** the amount of memory in bytes the *Subscriber* receives per time unit (second).

Each sub-test will be tested with randomly selected 10-100 kb files and 1-5 mb files.

These tests measure DDS-C's ticket system's ability to

handle a small and large number of *Subscribers* and *Publishers* with transferring varying file sizes.

V. Limitations

Even though the system is robust with symmetric keys and a ticket system, there are some vulnerabilities. The initial communication to register an entity in the authentication server (AS) is sent with public key encryption that could be broken and easily attacked with a man-in-the-middle (MITM) attack. Once a malicious actor registers with AS, they are trusted, and there is no inherent mechanism in DDS and Kerberos to identify the intruder. Attackers can also attempt a replay attack by emulating the tickets. Depending on the time-to-live (TTL) of a ticket, ticket redistribution could become a problem if there were numerous *Subscribers* and *Publishers* that needed new tickets at one time which could slow or crash the key distribution center (KDC).

VI. Conclusion

Data Distribution Service (DDS) has many security vulnerabilities in its publish-subscribe model and quality-of-service (QoS) policies. DDS-Cerberus (DDS-C) is a possible novel solution to these vulnerabilities. It focuses on two central QoS policies in DDS and uses symmetric encryption, key distribution center (KDC), and tickets. DDS-C focuses on using ROS 2 (Robot Operating System) as its DDS middleware and configuring it to incorporate Kerberos as a plugin to perform DDS-C tests.

Smart systems in the military, healthcare, and cities would benefit from a robust system like DDS-C. This paper proposes a design that could handle the data-management burden of being smart through its modified publish-subscribe system and *Topics*. ROS 2 simulation testing is planned for future work. Internet of Things (IoT) is a vast field that demands attention to security to prevent common vulnerabilities.

VII. Acknowledgment

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 United States.

References

[1] *MQTT Version 3.1.1*, OASIS Standard, Oct. 29, 2014. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

[2] *OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0 Part 0: Overview*, OASIS Standard, Oct. 29, 2012. [Online]. Available: http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html

[3] S. N. Swamy, D. Jadhav, and N. Kulkarni, "Security threats in the application layer in iot applications," in *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, 2017, pp. 477–480.

[4] G. Nebbione and M. C. Calzarossa, "Security of iot application layer protocols: Challenges and findings," *Future Internet*, vol. 12, no. 3, p. 55, Mar 2020. [Online]. Available: http://dx.doi.org/10.3390/fi12030055

[5] Object Management Group. "OMG Standards for Industries." Accessed Mar. 15, 2021. [Online]. Available: https://www.omg.org/industries/index.htm

[6] Real-Time Innovations. "Connext DDS Secure." Accessed Mar. 15, 2021. [Online]. Available: https://www.rti.com/products/connext-dds-secure

[7] OpenDDS. "OpenDDS." Accessed Mar. 15, 2021. [Online]. Available: https://opendds.org

[8] M. J. Michaud, T. Dean, and S. P. Leblanc, "Attacking omg data distribution service (dds) based real-time mission critical distributed systems," in *2018 13th International Conference on Malicious and Unwanted Software (MALWARE)*, 2018, pp. 68–77.

[9] R. M. Abdulghani, M. M. Alrehili, A. A. Almuhanna, and O. H. Alhazmi, "Vulnerabilities and security issues in iot protocols," in *2020 First International Conference of Smart Systems and Emerging Technologies (SMARTTECH)*, 2020, pp. 7–12.

[10] A. Alaerjan and D. Kim, "Configuring dds features for communicating components in smart grids," in *2017 IEEE International Conference on Smart Energy Grid Engineering (SEGE)*, 2017, pp. 162–169.

[11] G. Pardo-Castellote, "Omg data-distribution service: architectural overview," in *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, 2003, pp. 200–206.

[12] *OpenDDS Developer's Guide*, OpenDDS, Jan. 27 2021. [Online]. Available: http://download.objectcomputing.com/OpenDDS/OpenDDS-latest.pdf

[13] MIT Kerberos. "Kerberos: The Network Authentication Protocol." Accessed Mar. 15, 2021. [Online]. Available: https://web.mit.edu/Kerberos/

[14] B. C. Neuman and T. Ts'o, "Kerberos: an authentication service for computer networks," *IEEE Communications Magazine*, vol. 32, no. 9, pp. 33–38, 1994.

[15] M. v. Steen and A. S. Tanenbaum, *Distributed Systems*, 3rd ed., 2017. [Online]. Available: www.distributed-systems.net

[16] F. Al-Ayed and H. Liu, "Synopsis of security: Using kerberos method to secure file transfer sessions," in *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2016, pp. 1016–1020.

[17] D. E. Denning, "Is quantum computing a cybersecurity threat?" *American Scientist*, vol. 107, no. 2, pp. 83–85, Mar 2019.

[18] V. Mavroeidis, K. Vishi, M. D., and A. Jøsang, "The impact of quantum computing on present cryptography," *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 3, 2018. [Online]. Available: http://dx.doi.org/10.14569/IJACSA.2018.090354

[19] S. Zrelli and Y. Shinoda, "Specifying kerberos over eap: Towards an integrated network access and kerberos single sign-on process," in *21st International Conference on Advanced Information Networking and Applications (AINA '07)*, 2007, pp. 490–497.

[20] Wireshark. "Wireshark." Accessed Mar. 15, 2021. [Online]. Available: https://www.wireshark.org/

IV. Paper: DDS-Cerberus: Ticketing Performance Experiments and Analysis

The following paper, "DDS-Cerberus: Ticketing Performance Experiments and Analysis," was submitted and accepted by the International Conference in Computational Science and Computational Intelligence; it was published in December of 2021. This paper is in IEEE format.

DDS-Cerberus: Ticketing Performance Experiments and Analysis

Andrew T. Park, Richard Dill, Douglas D. Hodson, Wayne C. Henry
Department of Electrical and Computer Engineering
Air Force Institute of Technology, Wright-Patterson Air Force Base, OH, USA
email: andrew.park@afit.edu, richard.dill@afit.edu, douglas.hodson@afit.edu, wayne.henry@afit.edu

***Abstract*—Data Distribution Service (DDS) is a publish-subscribe middleware used to distribute data between real-time systems, production environments, and small embedded platforms. In DDS, *Nodes* have at least one *Publisher* or *Subscriber*. *Publishers* and *Subscribers* use unique *Topics* to send and receive messages. Each *Subscriber* has permission to read the *Publisher's* message if it references the same *Topic* sent from the *Publisher*. This capability supports real-time communication, sacrificing security, such as impersonation attacks.**

This paper details, tests, and evaluates DDS-Cerberus (DDS-C), a novel distributed communication protocol integrating Kerberos ticketing system with DDS. DDS-C integrates Kerberos authentication and *Ticket* retrieval with *Publishers* and *Subscribers*. Experiments have six parameters each with a 2:1 *Publisher* to *Subscriber* ratio. Performance tests modify the message byte size to emulate *.txt* and *.mp3* files: 10 KB, 100 KB, 1 MB, 5 MB, 10 MB, and 20 MB. Experiment metrics for functionality and performance are the messages per second and latency in a wired environment. Experiments utilize ROS 2 (Robot Operating System) as a testbed. Initial tests for a baseline are conducted without DDS modifications and subsequent tests with DDS-C modifications. The results reveal that due to the ticketing component, DDS-C increases DDS security by preventing impersonation attacks while negligibly increasing average processing compared to baseline results.

***Index Terms*—DDS, Kerberos, ROS 2, publish-subscribe, impersonation**

I. Introduction

With the rapid growth of technology and the Internet, interest and development in robots has had constant attention and growth. Cyber-physical systems such as robotic systems, smart grids, and industrial control systems rely on real-time data streams to function correctly. These systems require software that can handle loss or delay of connection from local to long-range communications. They also need to be modifiable so that other products and prototypes can use the same software to test and apply for real-world use.

One such software for real-time use is ROS (Robot Operating System). Beginning as ROS 1, its design goals are summarized as multi-programming language friendly, easy to update and navigate, and open-source [1]. These goals have been integrated into other versions of ROS 1 over the years, rising in popularity. However, with the increasing use of wireless networks, ROS 1's TCP-based communication (Transmission Control Protocol) layer did not meet real-time standards in this environment. ROS 2 with Data Distribution Service (DDS) was developed to meet real-time needs due to its Quality-of-Service, and publish-subscribe interface [2]. Different ROS 2 versions handle the management of *Publishers* and *Subscribers* in different ways, such as through key exchanges and access control [3]. DDS-Cerberus (DDS-C) focuses on modifying DDS and utilizing ROS 2 as the main testbed for experiments for its DDS implementation. It utilizes Kerberos to authenticate every *Publisher* and *Subscriber* by creating *Tickets* with a *keytab*, a long-term key. After authentication, those *Publishers* and *Subscribers* can publish or read data. The experiments are split into evaluation categories for functionality and performance.

The remainder of this work is organized as follows. Section II introduces the background in DDS and Kerberos. Section III presents the implementation framework and analysis of DDS-C performance. Section IV is a security discussion of DDS-C. Section V offers conclusions and recommendations for future research.

II. Background

This section introduces and describes the main functionalities and features of Data Distribution Service (DDS) and Kerberos. These functions and features are highlighted as pillars for the development, motivation, and functionality of DDS-Cerberus (DDS-C). This section also presents related works for experiments and their evaluations.

A. Overview of DDS

DDS uses Quality-of-Service policies that dictate transport behavior, such as message queue size and lifespans. These policies allow User Datagram Protocol (UDP) to be reliable or best-effort as Transmission Control Protocol (TCP). This research focuses on the Data-Centric Publish-Subscribe (DCPS) layer, which contains DDS main components used by ROS 2 (Robot Operating System) [4]. Due to its real-time capability, DDS is widely used, especially with ROS 2. The main components DDS-C focuses on are these two components found in the *Node* component: *Publishers* which receive data from one to many *Data Writers*, and *Subscribers* which receive data for one to many *Data Readers*. Each data published has a unique *Topic* that *Subscribers* can specify to receive.

1) ROS 2: Since its inception, ROS 2 has had several distributions [5]: Dashing Diademata, Galactic Geochelone, and Foxy Fitzroy. It can be built or installed on Linux, macOS,

and Windows. The versatility of ROS 2 is due to Object Management Group's (OMG) DDS [6]. Though DDS does not have extensive wikis or Github repositories, DDS vendors support research towards improving ROS 2. OMG's DDS-Security specification adds Service Plugin Interfaces (SPIs) [7] [8]. There are three specific SPIs utilized by ROS 2 that are mandatory for compliant DDS implementations. This standardization helps with interoperability and cross-vendor support and integration.

- *Authentication:* Utilizes Public Key Infrastructure (PKI) and Certificate Authority (CA), called *DDS:Auth:PKI-DH*, that binds a participant's public key to a specific name.
- *Access Control:* Allows the user to modify participant permissions (i.e., to specific domain, to specific topic).
- *Cryptographic:* Uses Advanced Encryption Standard (AES) in Galois Counter Mode (AES-GCM). Implemented as *DDS:Crypto:AES-GCM-GMAC*.

In this paper's implementation, ROS 2 serves as the main testbed for DDS-C experiments. The paper does not focus on robots or the effect on robots but on improving DDS.

B. Overview of Kerberos

Kerberos is a network authentication protocol that uses symmetric key cryptography to securely prove identities to a server using *Tickets* [9] [10]. To authenticate with the Key Distribution Center (KDC), which contains the Authentication Server (AS) and Ticket-Granting Server (TGS), a user has to register with the AS via a username and password securely. Subsequent communications after registration and authentication use symmetric keys. Kerberos uses UDP port 88 by default. This experiment uses Kerberos V5 in Linux, *krb5*. To gain a *Ticket*, a user uses `kinit` with a Kerberos name (e.g., `kinit primary/instance@REALM`) [11]. A name contains three parts:

- *Primary:* User's or service's name. Commonly known as the *principal*.
- *Instance:* Used to describe credentials. Can be in a privileged group as "root" or "admin."
- *Realm:* The Kerberos service providing authentication and *Tickets* for a specified *principal*.

These parts are stored in the Kerberos server's *Kerberos database* that has a *Kerberos realm*. After authenticating with the AS, the user obtains credentials to use the *Ticket*, a corresponding *Ticket* and *Ticket* session key.

C. Related Works

This research utilizes performance experiments from J. Kim et al. [12]. The authors evaluated latency and throughput between two machines that have ROS 2 setups. Their experiments had two categories: wired with Ethernet and wireless with a Virtual Private Network over Wi-Fi. They focused on performance with estimated latency and throughput per packet.

Thulasiraman et al. [13] evaluated ROS 2 wireless network performance by measuring latency and message drop rate. The authors modified differing Quality-of-Service settings and security features on a small number of *Nodes* to measure ROS's performance.

C. Kim et al. [14] developed a DDS simulator using QualNet to test the performance of different DDS implementations. Their eight simulations tested the number and performance of sent and received messages between *Data Writers* and *Data Readers*. They focused on DDS's discovery capability and speed of message processing.

These related works evaluate at least one of these areas: experiments and performance. None of these works combine DDS with Kerberos; however, they have techniques and methods beneficial to developing and implementing DDS-C.

III. Implementation

The goal of conducting DDS-Cerberus (DDS-C) experiments is to describe its capabilities and to determine the impact *Ticket* functionality has on the Data Distribution Service (DDS) performance. This section explains DDS-C's design and focuses on its average processing time for *Publishers* and *Subscribers* when executing ROS 2 (Robot Operating System). It compares performance with no modifications as a benchmark compared with DDS-C modifications.

A. Overview of DDS-C

A. T. Park et al.'s *DDS-Cerberus: Data Distribution via Ticketing* [15] is the groundwork for DDS-C's inception and background. This previous paper outlines the background, design, and motivation for DDS-C and subsequent experiments. The current paper refines and improves the DDS-C design and uses the previous paper's plan for experiments as a guideline.

DDS-C is setup with ROS 2 Foxy Fitzroy [16] and Kerberos V5 [10] in a Linux OS. Table I outlines the equipment specifications for experiments. *Foxy*, the Host laptop, maintains ROS 2 domain and workspaces, and the Kerberos server is on a virtual machine in VMware with the name *Kerby*. Both machines have Ubuntu installed. This setup is tested wired through Ethernet with *Foxy* and *Kerby* on the same bridged subnet.

TABLE I
Equipment Specifications

	Host	**Kerberos Server**
Name	*Foxy*	*Kerby*
Machine	Laptop	VMware
OS	Ubuntu 20.04.2 LTS	Ubuntu 20.04.2 LTS
CPU	11th Gen i7-1185G7 8 Cores	11th Gen i7-1185G7 2 Cores
Disk Space	2 TB	20 GB
RAM	31 GB	4 GB

To obtain *Tickets* using Kerberos commands (e.g., `kinit`, `klist`, `kdestroy`), both systems require a Kerberos installation, but all *principals* are stored on *Kerby*. *Tickets* are requested and stored in a cache on *Foxy*. ROS 2 *Publishers* and *Subscribers* request for *Tickets* and *principal* information from

Kerby. *Tickets* have a default time-to-live (TTL) of 24 hours which can be changed depending on the user's preference.

With multiple *Publishers* and *Subscribers*, creating new *Tickets* would require manual credential input; however, Kerberos has a *keytab* functionality that allows creation of long-term keys called *keytabs*. *Keytabs* are created with a *principal* and its password. A *keytab* consists of a timestamp, *principal* name, key version number, encryption type, and encryption key. All *keytabs* are encoded with enctype `-e aes256-cts-hmac-sha1-96`. Regardless of a *Ticket's* status, every time a *Publisher* or *Subscriber* is started up, a new *Ticket* is created using the *keytab* and *principal* (e.g. `kinit -k -t /tmp/keytabs/<username>.keytab primary/instance@REALM`). *Keytabs* do not have to be renewed or changed unless the corresponding *principal's* password is changed, making them favorable for long-term *Publisher* and *Subscriber* use. For the experiments, they are stored in *Foxy*. To ensure additional security, *Tickets* and *keytabs* are stored in directory `/tmp`; therefore, when *Foxy* reboots, all requested data will be destroyed. For multiple systems, the *keytabs* are only stored on the machine that needs to authenticate a specific *Node*. Due to the *keytab's* longevity and function, DDS-C focuses on using them as its main verification tool for *Nodes*.

This setup is illustrated as a diagram in Figures 1 and 2. For each *Node*, the dotted lines in Figure 1 represent *Publisher1*, *Publisher2*, and *Subscriber1*'s communication with the Key Distribution Center (KDC) and the solid lines in Figure 2 represent message sending. They first send a `kinit` with a corresponding *keytab*. The KDC sends back a *Ticket* allowing the *Publisher* or *Subscriber* to interact with messages through the *Topic*. To simulate many *Data Writers* and *Data Readers* sending data, *Publishers* and *Subscribers* send data out at a buffered interval to time performance accurately without creating message queues.

B. Methodology

All scenarios and tests are executed on *Foxy* through bash scripts executing Python scripts in parallel. There are two main experiment components. Both components focus on latency.

- *Functionality:* Collecting processing times for requesting, retrieving, and modifying *Tickets* (samples per second).
- *Performance:* Collecting processing times for messages sending different file sizes.

These test categories will be using six simulations with a 2:1 ratio of *Publishers* and *Subscribers* outlined in Table II. This experiment focuses on using Python to write *Publishers* and *Subscribers*. Each *Publisher* sends a total of 10 messages. Using the Python *time* module's `time.time()`, the processing durations of sent and received messages are recorded for both categories. There is a time buffer of 0.5 seconds between each message sent. Using the setup in Figures 1 and 2, each two *Publishers* in each parameter have a unique *Topic* and *principal*. One *Subscriber* is listening for this *Topic* and also has a unique *principal*.

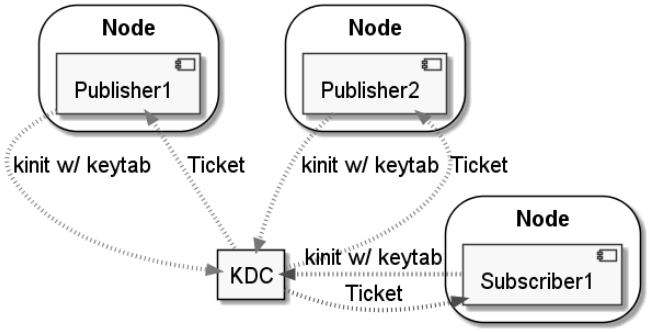

Fig. 1. DDS-C KDC Authentication using *Keytab*

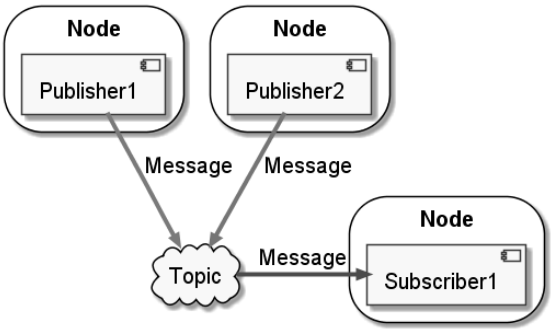

Fig. 2. DDS-C Post-Authentication Message Sending

The experiment components are based on these three scenarios. Benchmark category tests one scenario. DDS-C category tests two *Ticket* scenarios. The benchmark category is considered the baseline in these experiments since it is measuring regular ROS 2 message sending without DDS-C.

- ***Benchmark:***
  - *Scenario 1:* Create a regular *Publisher* and *Subscriber* with no *Ticket* capabilities.
- ***DDS-C:***
  - *Scenario 2:* When a *Ticket* already exists in the `klist` for both *Publisher* and *Subscriber*.
  - *Scenario 3:* Retrieving a new *Ticket* using `kinit` for both *Publisher* and *Subscriber*. This test covers what happens when a *Ticket* is expired since the procedure to get a new *Ticket* is identical.

Results are documented in tables with average *Publisher* and *Subscriber* times in seconds. A total average is calculated to compare benchmark and DDS-C categories.

TABLE II
SIMULATIONS

Simulation	***1***	***2***	***3***	***4***	***5***	***6***
Number of Publishers	2	4	6	8	10	12
Number of Subscribers	1	2	3	4	5	6

C. Tests

The functionality tests measure both benchmark and DDS-C categories with three scenarios and six simulations. The messages sent in these tests contain the 63-byte string "Hello World: <message_id>". Let n be the amount of messages: message_id $= 0, 1, ..., n$.

Performance tests collect times from six different file sizes: 10 KB, 100 KB, 1 MB, 5 MB, 10 MB, and 20 MB. They are emulating files such as *.txt*, *.png*, *.jpg*, and *.mp3* files that could be sent over the network. Each file sent was generated with randomized characters (a-z) for each file size. Both benchmark and DDS-C tests run simulation 1. DDS-C tests only the performance of scenario 2 because the time to run Kerberos commands will not change performance test results since message creation, delivery, and retrieval happen after authentication and *Ticket* retrieval.

D. Results

The functionality tests reveal that Table III benchmark total average times are lower than DDS-C times in Table IV and V. Times in each Avg Pub and Sub Time follow a general upward trend. Between scenarios 1 and 2, average *Publisher* and *Subscriber* times differed by a latency of 0.0507 and 0.0339 seconds, respectively. Between 1 and 3, a latency of 0.4599 and 0.4786 seconds. Between scenarios 2 and 3, a latency of 0.4092 and 0.4447 seconds. The average to check and retrieve a *Ticket* is 0.427 seconds, the average of 0.4092 and 0.4447.

Performance tests showed that Table VI's benchmark average times are lower than Table VII. These results also had an upward trend. Between scenarios 1 and 2, average *Publisher* and *Subscriber* times differed by a latency of 0.0138 and 0.024 seconds, respectively. If accounting for *Tickets* as in scenario 3, the difference would be around 0.4408 and 0.451 seconds.

Through the test results and compared total averages, DDS-C performs on average less than a second slower than an unmodified ROS 2. The difference is a minor setback because *Ticket* checking and retrieval occurs at the creation of a *Publisher* and *Subscriber*; therefore, it does not hinder message sending and retrieval.

TABLE III
BENCHMARK FUNCTIONALITY TEST: SCENARIO 1

Simulation	***1***	***2***	***3***	***4***	***5***	***6***	***Total Avg***
Avg Pub Time (sec)	5.0312	5.0408	5.0446	5.0518	5.0704	5.1055	5.0575
Avg Sub Time (sec)	5.0488	5.0624	5.0941	5.1183	5.1041	5.1684	5.0994

TABLE IV
DDS-C FUNCTIONALITY TEST: SCENARIO 2

Simulation	***1***	***2***	***3***	***4***	***5***	***6***	***Total Avg***
Avg Pub Time (sec)	5.0396	5.0435	5.0494	5.0732	5.0771	5.3662	5.1082
Avg Sub Time (sec)	5.0668	5.0348	5.0907	5.0860	5.1362	5.3854	5.1333

TABLE V
DDS-C FUNCTIONALITY TEST: SCENARIO 3

Simulation	***1***	***2***	***3***	***4***	***5***	***6***	***Total Avg***
Avg Pub Time (sec)	5.6649	5.5044	5.5623	5.3835	5.4111	5.5783	5.5174
Avg Sub Time (sec)	5.6593	5.5844	5.5873	5.4718	5.5328	5.6417	5.5780

TABLE VI
BENCHMARK PERFORMANCE TEST: SCENARIO 1

	Simulation *1*						
	10 kb	***100 kb***	***1 mb***	***5 mb***	***10 mb***	***20 mb***	***Total Avg***
Avg Pub Time (sec)	5.0325	5.0344	5.0358	5.0401	5.0544	5.068	5.0442
Avg Sub Time (sec)	5.0569	5.0472	5.0447	5.0638	5.0687	5.0799	5.0602

TABLE VII
DDS-C PERFORMANCE TEST: SCENARIO 2

	Simulation *1*						
	10 kb	***100 kb***	***1 mb***	***5 mb***	***10 mb***	***20 mb***	***Total Avg***
Avg Pub Time (sec)	5.0434	5.0490	5.0547	5.0582	5.0654	5.0772	5.0580
Avg Sub Time (sec)	5.0664	5.0641	5.0771	5.0784	5.0865	5.1328	5.0842

IV. SECURITY DISCUSSION

An impersonation attack, or also known as a spoofing attack, is when an attacker compromises a system's integrity by impersonating a legitimate node due to weak authentication [17]. DDS-Cerberus (DDS-C) mitigates impersonation attacks, a well-known Data Distribution Service (DDS) and ROS 2 (Robot Operating System) vulnerability.

DDS-C focuses on situations when an attacker compromises a system on the same network DDS and ROS 2 resides. The attacker can perform impersonation attacks by setting up illegitimate *Nodes*. To enforce the integrity of inter-*Node* communication, DDS-C verifies the validity of a *Node's Publisher* and *Subscriber* through *keytabs* before any messages are sent or retrieved. In real-world, large-scale experiments, each *Node* could be on separate systems; therefore, *keytabs* are stored only on the system that requires its use of specific *Publishers* and *Subscribers* through the Key Distribution Center (KDC).

By introducing ticketing capability, DDS-C mitigates impersonation through *keytab* use. Each *Publisher* and *Subscriber* need a unique Kerberos principle and with it a unique *keytab*. The `kinit` command cannot complete if it does not have the correct *keytab* in the same system. In Figure 3, an attacker on the same network, but on a system independent of ROS 2, creates a *Node* with a *Publisher* or *Subscriber*. The top solid black circle is the starting point of both legitimate and illegitimate pub/sub actions, and the bottom solid black circle with a circle is the endpoint, resulting in success or failure. The squares represent the choice the pub/subs are going to make. The dashed path of the illegitimate pub/sub leads to

an unsuccessful authorization because it has the incorrect *keytab* to send to the KDC. The attacker needs to replicate or steal a corresponding *keytab* to impersonate successfully. The legitimate pub/sub ends in a successful authorization to send and receive messages because it has the correct matching *keytab* resulting in a success.

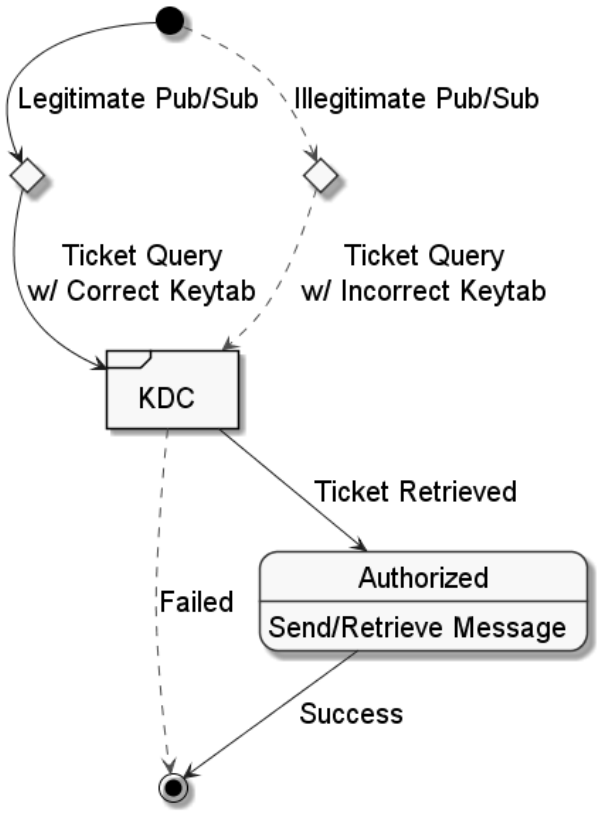

Fig. 3. DDS-C Impersonation Attack States. Solid Line: Legitimate Authentication. Dashed Line: Illegitimate Authentication.

ROS 2 and its other implementations do not have this ticketing capability making DDS-C more secure. The vanilla ROS 2 model does not provide integrity. Main variations such as Secure Robot Operation System (SROS) and ROS-AES-Encryption (Advanced Encryption Scheme) provide integrity but do not provide metadata confidentiality [3]. Foxy Fitzroy's variation, SROS 2 / DDS-Security, also does not provide metadata confidentiality; however, even if an attacker knows the metadata, they would not be able to perform a successful impersonation attack without valid a *keytab*. Thus, DDS-C increases security with only a minor difference in processing time.

V. Conclusion

The novel protocol, DDS-Cerberus (DDS-C), introduces a ticketing capability to Data Distribution Service (DDS) based applications. This paper describes DDS capabilities and characteristics to merit the need of DDS-C. Through experiments with ROS 2 (Robot Operating System), the parameters and scenarios resulted in data to conclude that DDS-C's performance does not significantly interrupt DDS functionality. Future work includes integrating DDS-C into a real-world operational testbed to optimize the DDS-C security features further. Many works evaluate DDS, but none combines Kerberos ticket capabilities with it. Incorporating DDS-C in growing DDS and, subsequently, ROS 2 and its distributions would help to reduce the risk of security vulnerabilities in real-time systems.

VI. Acknowledgment

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 United States.

References

[1] M. Quigley, J. Faust, T. Foote, J. Leibs *et al.*, "ROS: an open-source Robot Operating System," *ICRA Workshop on Open Source Software*, vol. 3, Jan 2009.

[2] P. Bellavista, A. Corradi, L. Foschini, and A. Pernafini, "Data Distribution Service (DDS): A performance comparison of OpenSplice and RTI implementations," in *2013 IEEE Symposium on Computers and Communications (ISCC)*, 2013, pp. 000 377–000 383.

[3] N. Goerke, D. Timmermann, and I. Baumgart, "Who Controls Your Robot? An Evaluation of ROS Security Mechanisms," in *2021 7th International Conference on Automation, Robotics and Applications (ICARA)*, 2021, pp. 60–66.

[4] G. Pardo-Castellote, "OMG Data-Distribution Service: architectural overview," in *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, 2003, pp. 200–206.

[5] Open Robotics. "Distributions." Accessed May 29, 2021. [Online]. Available: https://docs.ros.org/en/foxy/Releases.htmlreleases

[6] W. Woodall. "ROS on DDS." Accessed May 29, 2021. Open Source Robotics Foundation. [Online]. Available: http://design.ros2.org/articles/ros_on_dds.html

[7] K. Fazzari. "ROS 2 DDS-Security integration." Accessed May 29, 2021. Open Source Robotics Foundation. [Online]. Available: http://design.ros2.org/articles/ros2_dds_security.html

[8] *"DDS Security." Accessed May 29, 2021*, Object Management Group. [Online]. Available: https://www.omg.org/spec/DDS-SECURITY/1.1/PDF

[9] Object Management Group. "OMG Standards for Industries." Accessed Mar. 15, 2021. [Online]. Available: https://www.omg.org/industries/index.htm

[10] MIT Kerberos. "Kerberos V5 System Administrator's Guide." Accessed May 29, 2021. [Online]. Available: https://web.mit.edu/kerberos/krb5-1.10/krb5-1.10.7/doc/krb5-admin.html

[11] "kerberos(1) - Linux man page." Accessed May 29, 2021. die.net. [Online]. Available: https://linux.die.net/man/1/kerberos

[12] J. Kim, J. M. Smereka, C. Cheung, S. Nepal, and M. Grobler, "Security and Performance Considerations in ROS 2: A Balancing Act," *arXiv preprint arXiv:1809.09566*, 2018.

[13] P. Thulasiraman, Z. Chen, B. Allen, and B. Bingham, "Evaluation of the Robot Operating System 2 in Lossy Unmanned Networks," in *2020 IEEE International Systems Conference (SysCon)*. IEEE, pp. 1–8.

[14] C.-H. Kim, G. Yoon, W. Lee, J. Park, and H. Choi, "A performance simulator for DDS networks," in *2015 International Conference on Information Networking (ICOIN)*, 2015, pp. 122–126.

[15] A. T. Park, R. Dill, D. D. Hodson, and W. C. Henry, "DDS-Cerberus: Data Distribution via Ticketing," in *The 2021 World Congress in Computer Science, Computer Engineering, Applied Computing (CSCE'21)*.

[16] Open Robotics. "ROS 2 Foxy Fitzroy." Accessed May 29, 2021. [Online]. Available: https://docs.ros.org/en/foxy/Releases/Release-Foxy-Fitzroy.html

[17] P. M. Jawandhiya, M. M. Ghonge, M. Ali, J. Deshpande *et al.*, "A Survey of Mobile Ad Hoc Network Attacks," *International Journal of Engineering Science and Technology*, vol. 2, no. 9, pp. 4063–4071, 2010.

V. Paper: Quantifying DDS-Cerberus Network Control Overhead

The following paper, "Quantifying DDS-Cerberus Network Control Overhead," is submitted to the Journal of Supercomputing. This paper is in Springer journal format.

Quantifying DDS-Cerberus Network Control Overhead

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

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

*Corresponding author(s). E-mail(s): andrew.park075@gmail.com;
Contributing authors: 2012raptor@gmail.com; richard.dill@afit.edu; douglas.hodson@afit.edu; michael.grimaila@afit.edu; wayne.henry@afit.edu;

Abstract

Selecting a secure and efficient middleware to process data is critical to assure the security properties and quality of service for communications between nodes within modern distributed systems and applications. Researchers have determined that Data Distribution Service (DDS) (a popular middleware used in industry, government, and military applications) is vulnerable to data and node integrity attacks. In contrast, DDS-Cerberus (DDS-C) is a novel security layer with foundational DDS components designed to mitigate impersonation attacks by requiring node authentication with Kerberos. This research provides an analysis of DDS-C, evaluating the layer's overhead and security features, by assessing total packet traffic generated in a robotics network. The experiment has a 2:1 publisher to subscriber node ratio, varying the number of subscribers and publisher nodes from three to eighteen. By categorizing the 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 DDS operations when compared to the other traffic. The results reveal that applying DDS-C to a representative distributed network does not substantially impact performance.

Keywords: Kerberos, DDS, ROS 2, Cyclone DDS, QoS, reliability

1 Introduction

Internet of Things (IoT) technologies and cyber-physical systems 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]. In addition to reliability and security challenges, these networks need to account for connectivity and power issues.

Data Distribution Service (DDS) is a robust, flexible, open middleware standard designed to manage real-time communication between various cyber-physical devices. Its popularity is evident from the wide adoption in public and private sectors, including military and finance frameworks [4]. 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 [5]. While DDS meets robustness, reliability, and efficiency requirements, it lacks some security features. The main security vulnerability in DDS, impersonation, allows a motivated adversary to gain unauthorized access to reading and sending data by posing as a trusted entity and node [6–8].

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. This research focuses on node impersonation through impersonation attacks. 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 (DDS-C), a novel secure distributed communication layer, adds this additional authentication mechanisms to DDS that improve security authentication to prevent impersonation attacks [9, 10]. 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.

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* has the *topic*, *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. Section 3 explains the set up for the experiment, research assumptions and limitations, and gathering and processing captured packet data. It also explores and analyzes the data. Section 4 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) [11]. The standard guides for 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 results for comparisons between QoS levels so that publishers match subscribers for which they are overqualified but never match with subscribers that promise less than the service required by those subscribers.

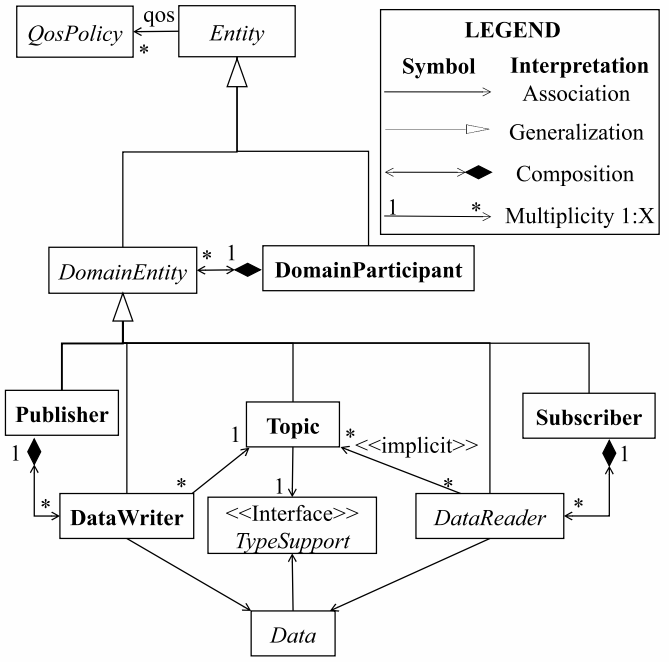

Fig. 1 Partial DDS Entity Model [12]

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 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 [9, 10]. It provides additional security by validating nodes and preventing impersonation attacks.

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 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. A 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. 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:

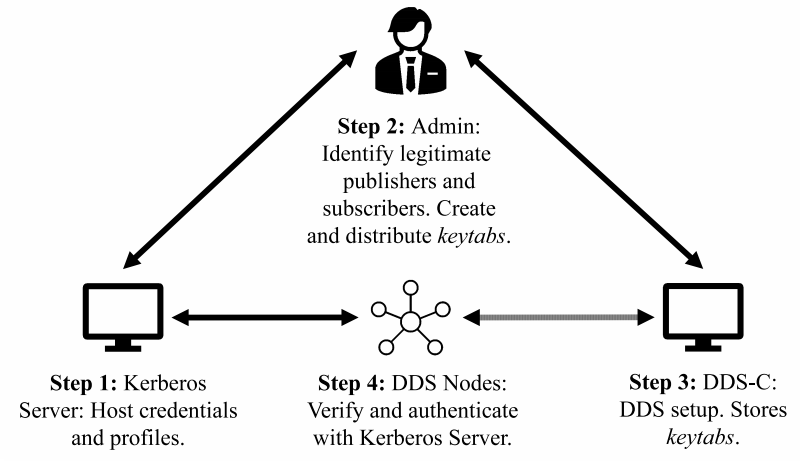

Fig. 2 DDS-C *Keytab* Process [10]

(0) Publisher1 authenticates and requests a ticket using a *keytab*. The AS receives Publisher1's message.
(1) The AS sends a message back for the TGS. A shared key, only known between the AS and TGS, encrypts this message. Publisher1 sends this message to the TGS to get a ticket.
(2) The TGS sends a ticket to the Kerberos Server KDC.

B. Publisher1 Authenticated:
(3) Afterwards, Publisher1 is successfully authenticated and can send its messages to the DDS domain. Server KDC.

C. Subscriber1 Authentication:
(4) Subscriber1 authenticates and requests a ticket using a *keytab*. The AS receives Subscriber1's message.
(5) The AS sends a message back for the TGS. A shared key, only known between the AS and TGS, encrypts this message. Subscriber1 sends this message to the TGS to get a ticket.
(6) The TGS sends a ticket to the Kerberos Server KDC.

D. Subscriber1 Authenticated:
(7) Afterwards, Subscriber1 is successfully authenticated and can read messages. In this case, it would be reading data sent from Publisher1.

E. Subsequent Messages:
(8) Since Publisher1 and Subscriber1 authenticated, no further authentication is needed.
(9) Message i with *Topic* is sent from Publisher1 and received by Subscriber1.
(10) Message $i + 1$ with *Topic* is sent from Publisher1 and received by Subscriber1.

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.

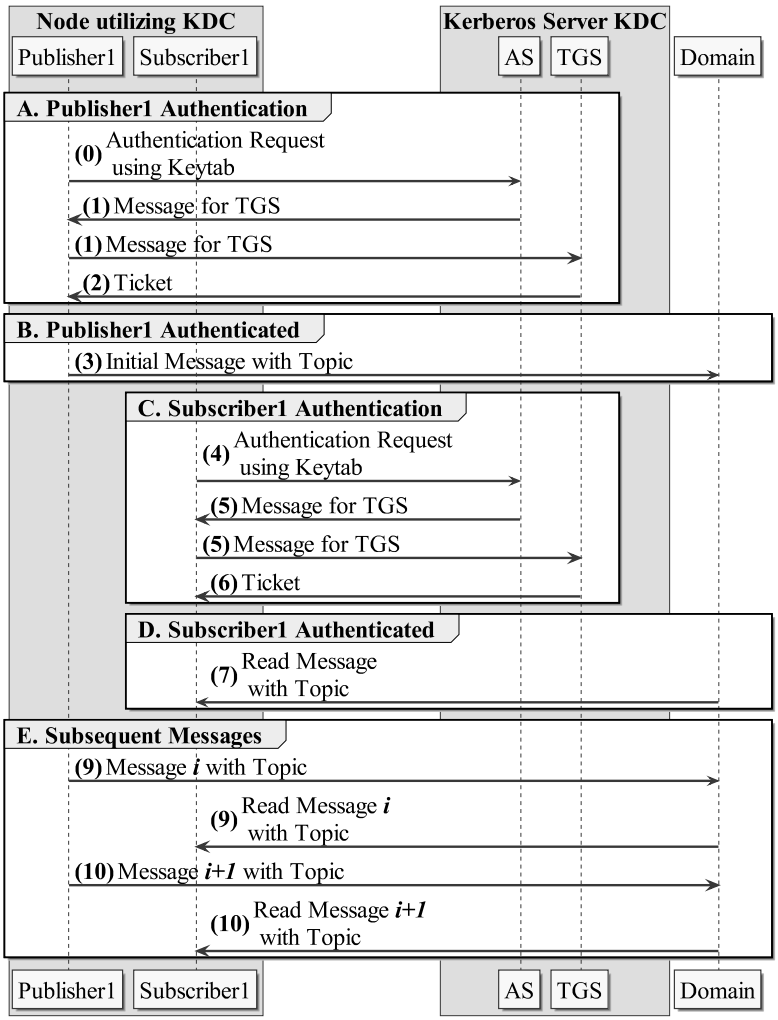

Fig. 3 DDS-C Authentication Process [10]

The inability to validate nodes is a security concern for DDS [6–8]. 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 sent 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 gets on 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.

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.

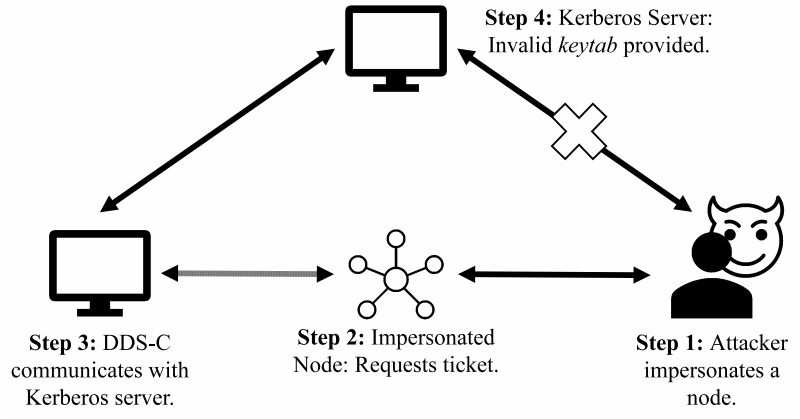

Fig. 4 DDS-C Mitigating Impersonation [10]

2.3 DDS Performance Evaluations

Other researchers have measured the performance qualities of DDS, such as latency [13–16]. 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 [13]. 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, 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) [14]. 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. 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) [15]. 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 [16]. These authors selected eProsima's FastDDS 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 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).

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 [5]. Comparing these protocols reinforces the benefits of choosing the proper lightweight communication protocol for low-power IoT devices, reliability, network traffic, and latency. No one protocol is universally used. Zorkadis presented the OSI security architecture guidelines [17]. 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 [18–20]. 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 [21]. 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 [22]. 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 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.

2.5 ROS 2 Evaluations

ROS 2 evolved from ROS 1, and both primarily differ at the communication layer. [23] ROS 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 [24]. 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 [25] 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 [26]. 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 [27]. 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 [28]. 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.

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.

3 Experiment

Table 1 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). Afterwards, the assumptions and limitations are listed. The final step is to process captured packets using scripts on a Windows machine.

Table 1 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.

3.1 Statistical Approach

Design of Experiment (DoE) methods provide experimenters with an unbiased, mathematical framework to evaluate the significance of statistical results [29].

DoE offers statistical mechanisms to test on hypotheses concerning response variables of different types. Many 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 [30]. 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, especially when compared to itself.

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 [16].

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 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 p-value from the calculated test statistic is compared to alpha 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 [31, 32]. Foxy Fitzroy was selected because of its long-term support and its use of eProsima Fast-RTPS [33]. 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 2 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 [34]. 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.

Table 2 Equipment Specifications

	Laptop PC: ROS 2	**Raspberry Pi: ROS 2**	**Raspberry Pi: KDC**
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

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 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 3. These three are set to ensure different node and message behaviors. ROS 2 sets all other QoS settings to their default values [35]. The experiment modifies reliability, switching between best effort and reliable. Queue size is 10 messages, and durability is transient local. Every

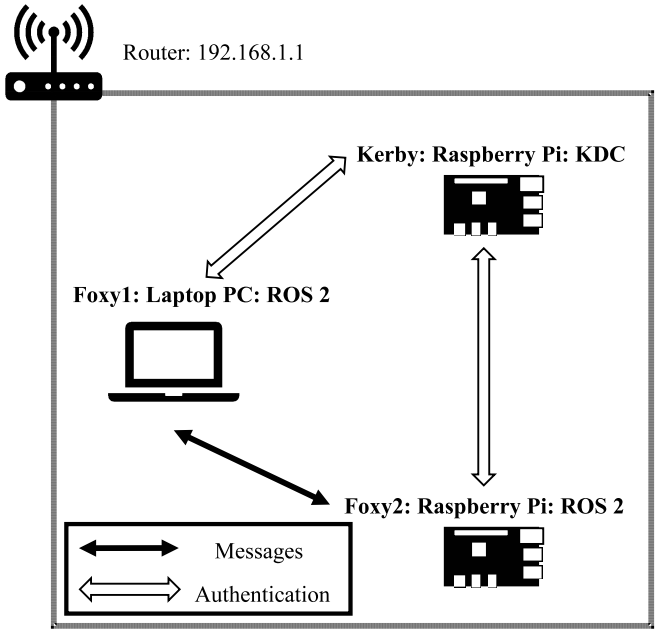

Fig. 5 Experiment Testbed Network Diagram

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*.

Table 3 Experiment QoS Settings

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

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 4.

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.

Table 4 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.

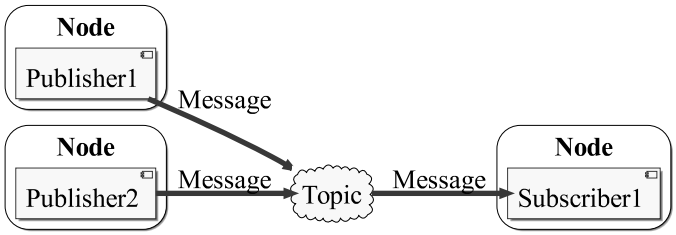

Fig. 6 Experiment Node Layout [10]

3.3 Assumptions and Limitations

This subsection outlines the experiment's assumptions and limitations, byproducts 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 [36, 37]. The .pcap files are then sent to a Windows machine for processing.

The Power Shell Tabluation Script, as shown in Listing 1, filters the packet capture files into columns of data fields via tshark [38, 39]. 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.

Listing 1 Tabulation Script

```
ForEach( $file in $list_of_files )
{
    tshark.exe -2 -r $file -T fields ...
        -E "Separator=," ...
        -e "frame.protocols" ...
        -e "frame.len"
}
```

The research 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 [40]. The `SciPy.Stats` module provides the `stats.t.cdf` function to evaluate the p-values given the test statistic and degrees of freedom [41]. To better understand the software used, Table 5 presents information about the names, locations, versions, and descriptions of all the software.

Table 5 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

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+*, resulting from increased participants. 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, Figure 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 6.

Table 6 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

Table 6 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[1]	0.027[1]	0.007[1]	0.007[1]
6	0.015[1]	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

[1]Statistically significant p-values with α 0.05.

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 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

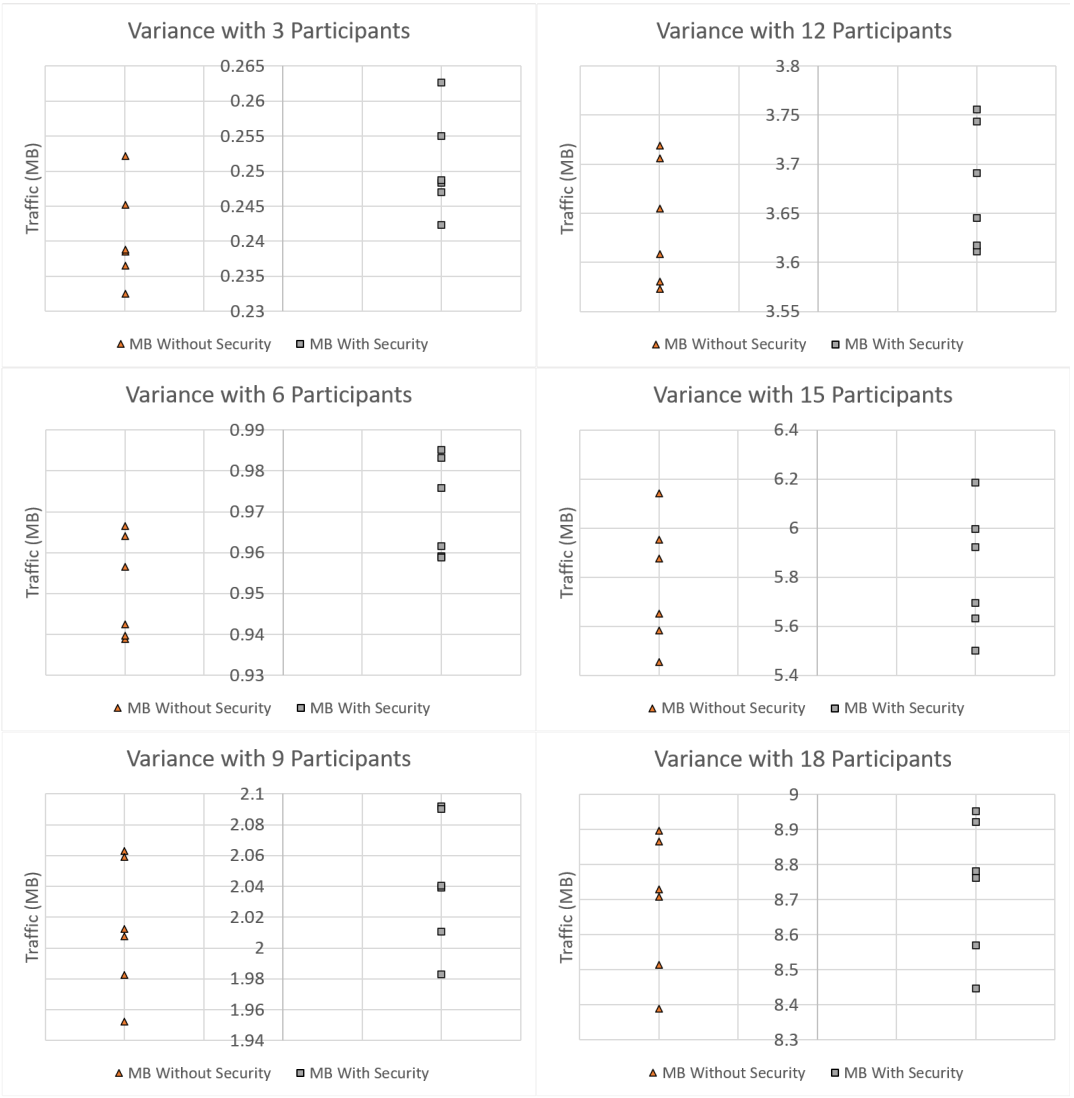

Fig. 7 Experiment Results. Top row: Best Effort Foxy1 with Kerby Variance 3 and 6. Middle row: Best Effort Foxy1 with Kerby Variance 9 and 12. Bottom row: Best Effort Foxy1 with Kerby Variance 15 and 18.

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, Figure 9 best effort *data message* traffic is different from the relative reliable plot. This reliable plot is similar to Figure 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.

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

This section outlined how the statistical model, network and ROS 2 setup, and processing software support the experiment results. It illustrated the defined process and setup to efficiently acquire, process, and analyze data, and examined 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 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

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

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 non-linear 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. [30].

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.

5 Acknowledgments

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 United States.

References

[1] El-Hajj, M., Fadlallah, A., Chamoun, M., Serhrouchni, A.: A survey of internet of things (IoT) authentication schemes. Sensors **19**(5), 1141 (2019)

[2] Ahmed, N., De, D., Hussain, I.: Internet of Things (IoT) for smart precision agriculture and farming in rural areas. IEEE Internet of Things Journal **5**(6), 4890–4899 (2018)

[3] White, T., Johnstone, M.N., Peacock, M.: An investigation into some security issues in the DDS messaging protocol. Proceedings of the 15th Australian Information Security Management Conference, AISM 2017, 132–139 (2017)

[4] Object Management Group: "OMG Standards for Industries." Accessed Oct 11, 2021. https://www.omg.org/industries/index.htm

[5] Al-Masri, E., Kalyanam, K.R., Batts, J., Kim, J., Singh, S., Vo, T., Yan, C.: Investigating messaging protocols for the Internet of Things (IoT). IEEE Access **8**, 94880–94911 (2020)

[6] Abdulghani, R.M., Alrehili, M.M., Almuhanna, A.A., Alhazmi, O.H.: Vulnerabilities and Security Issues in IoT Protocols. In: 2020 First International Conference of Smart Systems and Emerging Technologies (SMARTTECH), pp. 7–12 (2020). IEEE

[7] Michaud, M.J., Dean, T., Leblanc, S.P.: Attacking OMG data distribution service (DDS) based real-time mission critical distributed systems. In: 2018 13th International Conference on Malicious and Unwanted Software (MALWARE), pp. 68–77 (2018). IEEE

[8] Goerke, N., Timmermann, D., Baumgart, I.: Who Controls Your Robot? An Evaluation of ROS Security Mechanisms. In: 2021 7th International Conference on Automation, Robotics and Applications (ICARA), pp. 60–66 (2021). IEEE

[9] Park, A.T., Dill, R., Hodson, D.D., Henry, W.C.: DDS-Cerberus: Data Distribution via Ticketing. In: The 2021 World Congress in Computer Science, Computer Engineering, & Applied Computing (CSCE'21)

[10] Park, A.T., Dill, R., Hodson, D.D., Henry, W.C.: DDS-Cerberus: Ticketing Performance Experiments and Analysis. In: The 2021 International Congress on Computational Science and Computational Intelligence (CSCI'21)

[11] DDS Portal. Accessed 16-September-2021. https://www.dds-foundation.org/

[12] Object Magagement Group: OMG Data Distribution Service (DDS). (2015). Object Magagement Group. Version 1.4

[13] Yang, J., Sandström, K., Nolte, T., Behnam, M.: Data Distribution Service for industrial automation. In: Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies Factory Automation (ETFA 2012), pp. 1–8 (2012). https://doi.org/10.1109/ETFA.2012.6489544

[14] Almadani, B., Bajwa, M.N., Yang, S.-H., Saif, A.-W.A.: Performance evaluation of DDS-based middleware over wireless channel for reconfigurable manufacturing systems. International Journal of Distributed Sensor Networks **11**(7), 863123 (2015)

[15] Chen, Y., Kunz, T.: Performance evaluation of IoT protocols under a constrained wireless access network. In: 2016 International Conference on Selected Topics in Mobile & Wireless Networking (MoWNeT), pp. 1–7 (2016). IEEE

[16] Profanter, S., Tekat, A., Dorofeev, K., Rickert, M., Knoll, A.: OPC UA versus ROS, DDS, and MQTT: performance evaluation of industry 4.0 protocols. In: 2019 IEEE International Conference on Industrial Technology (ICIT), pp. 955–962 (2019). IEEE

[17] Zorkadis, V.: Security versus performance requirements in data communication systems. In: European Symposium on Research in Computer Security, pp. 19–30 (1994). Springer

[18] Kirsal, Y., Gemikonakli, O.: 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, pp. 1–6 (2008). IEEE

[19] Ever, E., Kirsal, Y., Gemikonakli, O.: 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), pp. 1–6 (2009). IEEE

[20] Kirsal-Ever, Y., Kirsal, Y., Polzonetti, A., Mostarda, L., Sule, C., Shah, P., Ever, E.: Challenges of Kerberos Variance with High QoS Expectations. In: Proceedings of the International Conference on Security and Management (SAM), p. 1 (2013). The Steering Committee of The World Congress in Computer Science, Computer ...

[21] Harbitter, A.H., Menascé, D.A.: Performance of public-key-enabled Kerberos authentication in large networks. In: Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001, pp. 170–183 (2000). IEEE

[22] Eum, S.-H., Choi, H.-K.: EAP-Kerberos II: An adaptation of Kerberos to EAP for mutual authentication. In: 2008 8th International Conference on ITS Telecommunications, pp. 78–83 (2008). IEEE

[23] Erős, E., Dahl, M., Bengtsson, K., Hanna, A., Falkman, P.: A ROS2 based communication architecture for control in collaborative and intelligent automation systems. Procedia Manufacturing **38**, 349–357 (2019)

[24] Kronauer, T., Pohlmann, J., Matthe, M., Smejkal, T., Fettweis, G.: Latency Analysis of ROS2 Multi-Node Systems (2021)

[25] Maruyama, Y., Kato, S., Azumi, T.: Exploring the performance of ROS2. In: Proceedings of the 13th International Conference on Embedded Software, pp. 1–10 (2016)

[26] Park, J., Delgado, R., Choi, B.W.: Real-time characteristics of ROS 2.0 in multiagent robot systems: An empirical study. IEEE Access **8**, 154637–154651 (2020)

[27] Thulasiraman, P., Chen, Z., Allen, B., Bingham, B.: Evaluation of the Robot Operating System 2 in Lossy Unmanned Networks. In: 2020 IEEE International Systems Conference (SysCon), pp. 1–8 (2020). IEEE

[28] Kim, J., Smereka, J.M., Cheung, C., Nepal, S., Grobler, M.: Security and performance considerations in ros 2: A balancing act. arXiv preprint arXiv:1809.09566 (2018)

[29] Montgomery, D.C.: Design and analysis of experiments. John wiley & sons (2017)

[30] Sadjadi, S.M., Shimizu, S., Figueroa, J., Rangaswami, R., Delgado, J., Duran, H., Collazo-Mojica, X.J.: 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 (2008). https://doi.org/10.1109/IPDPS.2008.4536214

[31] Open Robotics: "ROS 2 Foxy Fitzroy." Accessed Oct 11, 2021. https://docs.ros.org/en/foxy/Releases/Release-Foxy-Fitzroy.html

[32] MIT Kerberos: "Kerberos V5 System Administrator's Guide." Accessed Oct 11, 2021. https://web.mit.edu/kerberos/krb5-1.10/krb5-1.10.7/doc/krb5-admin.html

[33] Arguedas, M., Ragnarok, S., Thomas, D.: "ROS 2 Releases and Target Platforms." Accessed Oct 11, 2021 (2020). https://ros.org/reps/rep-2000.html

[34] "Releases." Accessed Oct 11, 2021. https://github.com/ros2/ros2/releases

[35] Open Robotics: "About Quality of Service settings." Accessed Oct 11, 2021. https://docs.ros.org/en/foxy/Concepts/About-Quality-of-Service-Settings.html

[36] Wireshark: "Wireshark." Accessed Oct 11, 2021. https://www.wireshark.org/

[37] The Tcpdump Group: "TCPDUMP/LIBPCAP public repository." Accessed Oct 11, 2021. https://www.tcpdump.org/

[38] Wireshark: "tshark." Accessed Oct 11, 2021. https://www.wireshark.org/docs/man-pages/tshark.html

[39] "PowerShell." Accessed Oct 11, 2021. https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/powershell

[40] "Python 3.0 Release." Accessed Oct 11, 2021. https://www.python.org/download/releases/3.0/

[41] "SciPy." Accessed Oct 11, 2021. https://scipy.org/

VI. Paper: Distribution of DDS-Cerberus Authenticated Facial Recognition Streams

The following paper, "Distribution of DDS-Cerberus Authenticated Facial Recognition Streams," is submitted to the Journal of Supercomputing. This paper is in Springer journal format.

Distribution of DDS-Cerberus Authenticated Facial Recognition Streams

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

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

*Corresponding author(s). E-mail(s): andrew.park075@gmail.com;
Contributing authors: 2012raptor@gmail.com; richard.dill@afit.edu; douglas.hodson@afit.edu; michael.grimaila@afit.edu; wayne.henry@afit.edu;

Abstract

Whether it be for humanitarian or military purposes, mission success often relies upon information and communication technologies. Securing and selecting the middleware that handles the messages sent between network nodes and applications is essential. One such middleware is Data Distribution Service (DDS) which employs a publish-subscribe model. However, researchers have found several security vulnerabilities in DDS implementations. DDS-Cerberus (DDS-C) is a novel security layer implemented into DDS to mitigate impersonation attacks using Kerberos authentication and ticketing. This paper extends our previous work by analyzing the performance of DDS-C in a use case implementation. The use case covers an artificial intelligence (AI) scenario that connects edge sensors across a commercial network. Specifically, it characterizes DDS-Cerberus in unmanned aerial vehicles (UAV), the cloud, and video streams for facial recognition. An evaluation of network traffic using DDS-C revealed that it was not statistically significant compared to DDS for the majority of the configuration runs. The results demonstrate that DDS-C provides security benefits without significantly hindering the overall performance.

Keywords: Kerberos, DDS, Cyclone DDS, UAV, AI, QoS

1 Introduction

Networked sensor devices typically follow the paradigm where one node tasks and receives input from multiple Internet of Things (IoT) devices. For example, a command node sends operational messages and receives sensor data from multiple unmanned aerial vehicles to conduct a search and rescue operation. These messages, ranging from simple commands to video frames, could have Quality of Service (QoS) attributes such as retransmitting unreceived messages to ensure nodes that joined late or have re-started receive all messages. For example, a UAV requires specific messages to navigate search and rescue missions correctly in lossy environments. Remotely operated bases use forward-deployed unmanned aerial vehicles (UAV) to support battlespace surveillance in contested environments [1]. The inter-communication links between UAVs and external links to cloud support services need to be robust enough to send and process video and images given terrain diversity and secure enough to thwart adversary attacks [2, 3]. Other messages could have QoS as best effort when a system can handle not receiving every message. For example, artificial intelligence (AI) facial recognition software on an IoT device may not require all frames from a live video feed to detect entities correctly. These use cases are essential in understanding DDS-Cerberus's (DDS-C) impact on real-world operations.

Data Distribution Service (DDS) is an open-source middleware that has been used in many sectors like finance, healthcare, and the military [4]. For real-time communication, DDS messages do not need to include the intended recipient but have a *topic*, represented as a unique string, from publisher to subscriber. The subscribers receive messages based on the associated *topic*. The messages have QoS properties to determine the sender and messages' behavior. Despite its efficient and real-time message sending capabilities, DDS is prone to impersonation attacks which allow an attacker to gain unauthorized access to messages [5–7].

DDS-C, a security layer for DDS, handles the authentication of DDS participants using Kerberos tickets [8–10]. This authentication mitigates impersonation attacks by verifying the identity of authenticated participants. This research's experiment captures network traffic from DDS and DDS-C to assess if DDS-C significantly impacts regular DDS performance. It uses the Bright Apps cloud architecture and network layout to evaluate DDS-C [11].

The experiment testbed relies on Cyclone DDS (an implementation of the DDS Standard) and the a commercial network infrastructure. The goal is to demonstrate that DDS-C is mature enough to support commercial artificial intelligence (AI) applications, specifically evaluating the impact on transmitting video frames. This impact is quantified by capturing total network traffic. The experiment emulates a network of unmanned aerial vehicles (UAV) with Raspberry Pi devices that send video frames. In conjunction with Bright Apps, this experiment aims to support UAV deployment in the field with DDS-C, such as in search and rescue. There are three use-cases detailing real-world scenarios for Bright Apps network infrastructure applications. To address the

three use-cases, the experiment has one network configuration whose goal is to send video frames processed by AI over a Virtual Private Network (VPN). The network setup consists of a Raspberry Pi device, Elastic Compute Cloud (EC2), and laptop PC. First, the Raspberry Pi device sends video frames to the EC2 for facial recognition AI processing, and then the PC displays the processed frames. The same message types of interest are selected. The QoS of interest is best effort to mimic use case scenarios. The data collected is categorized by equipment to determine the traffic impact on each device.

This research builds on the previous paper, Park et al.'s *Quantifying DDS-Cerberus Network Control Overhead*, by collecting network packet quantities for facial recognition streams [10]. The collected traffic is split into three categories: *data message*, *security*, and *discovery+*. The research determines if DDS-C *security* traffic has a significant impact on the other DDS traffic by comparing the three through statistical analysis. This paper aims to contribute to other DDS research in use case applications.

This paper is organized as follows. Section 2 outlines DDS, DDS-C, and related works. Section 3 contains the experiment setup, assumptions and limitations, and results. Section 4 provides future research recommendations.

2 Background

This section provides background information on the design and implementation of DDS-Cerberus (DDS-C) by explaining Data Distribution Service (DDS) and Kerberos. Additionally, it presents other similar application works that support the development of this research's experiment.

2.1 DDS-Cerberus (DDS-C)

DDS-C is a security layer that mitigates impersonation attacks [8–10]. It is integrated into DDS to provide participant authentication through Kerberos.

DDS is managed by Object Management Group (OMG) and is open-source, allowing for several implementations from different vendors. Its primary function is to handle message delivery between communicating entities. The communication is done through *topics*, or unique strings, that are sent by publishers and received by subscribers. Subscribers receive a message by specifying a unique string a message has. Quality of Service (QoS) policies dictate publisher and subscriber behavior on how to send messages. The policies are adapted to different network setups, such as having subscribers only read the most recent message.

The research focuses on the Data-Centric Publish-Subscribe (DCPS) layer containing the following components: publishers, subscribers, and *domain participants*. The components are seen in Figure 1 where *domain participants* can contain any number of publishers and subscribers. The messages are sent with *topics* to the DDS domain to be read by subscribers. Previous DDS-C research focused on authenticating publisher and subscriber nodes, but this experiment focuses on authenticating *domain participants*. Two reasons to use *domain*

participants are adding and authenticating nodes becomes less of a hassle, and they allow for easy integration into the Bright Apps architecture. *Domain participants* assist with executing publishers and subscribers in parallel, which helps send multiple video frames.

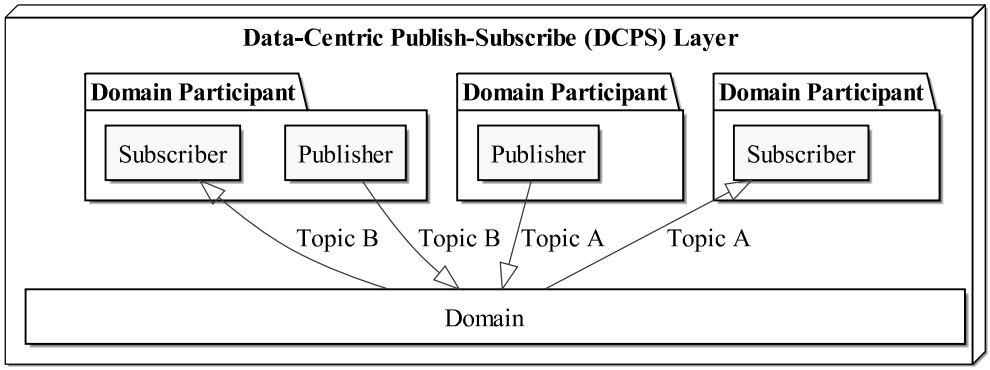

Fig. 1 DCPS Layout [8]

Kerberos is an authentication protocol used on distributed networks to authenticate users or nodes who request to talk on the network [12]. Kerberos servers have a realm name that is used to specify where the authentication is taking place. The `kinit` command grants tickets with the correct principal, or username, realm, and password. When the command runs, the requesting device or node communicates with Kerberos's Key Distribution Center (KDC), which has two main components: the Authentication Server (AS) and Ticket Granting Server (TGS). The requester authenticates with the AS if their credentials are in the server. If credentials match, the authenticated can gain a ticket from the TGS by a special message granted by the AS. The lifetime of a ticket is default of 24 hours, but it is possible to change the lifetime to add ticket security.

One important function of Kerberos that DDS-C leverages are *keytabs*, long-term keys to aid in creating tickets. Each *domain participant* in DDS is paired with a unique *keytab* for seamless authentication. These *keytabs* are encrypted using AES-256. When running the `kinit` command, the password has to be manually entered; however, manually typing the password is not required if run with passing in the long-term key. This authentication is important in mitigating impersonation attacks because if the attacker does not have access to the keytab, DDS-C does not allow them to impersonate a node or *domain participant* [5–7]. The attacker would have to either replicate or steal the long-term key, which would be difficult due to the key's encryption and additional network security.

Figure 2 shows these *keytabs* in action with a sequence diagram of two *domain participants*, DP1 and DP2, authenticating with a KDC. The leftmost gray area, "Domain Participants utilizing KDC", represents the *keytabs* that were created and stored for DP1 and DP2. DP1 contains one publisher, and DP2 contains one subscriber. Messages flow as follows:

A. DP1 Authentication:
   (0) DP1 authenticates and requests a ticket using a *keytab*. The AS receives DP1's message.
   (1) The AS sends a message back for the TGS. A shared key, only known between the AS and TGS, encrypts this message. DP1 sends this message to the TGS to get a ticket.
   (2) The TGS sends a ticket to the Kerberos Server KDC.

B. DP1 Authenticated:
   (3) Afterwards, DP1 is successfully authenticated, and the publisher can send its messages to the DDS domain. Server KDC.

C. DP2 Authentication:
   (4) DP2 authenticates and requests a ticket using a *keytab*. The AS receives DP2's message.
   (5) The AS sends a message back for the TGS. A shared key, only known between the AS and TGS, encrypts this message. DP2 sends this message to the TGS to get a ticket.
   (6) The TGS sends a ticket to the Kerberos Server KDC.

D. DP2 Authenticated:
   (7) Afterwards, DP2 is successfully authenticated, and the subscriber can read messages. In this case, it would be reading data sent from DP1's publisher.

E. Subsequent Messages:
   (8) Since DP1 and DP2 authenticated, no further authentication is needed.
   (9) Message i with *Topic* is sent from DP1's publisher and received by DP2's subscriber.
   (10) Message $i + 1$ with *Topic* is sent from DP1's publisher and received by DP2's subscriber.

DDS-C authentication executes at the beginning of a *domain participant's* lifecycle; however, this authentication can run more than once based on an administrator's needs. Additionally, this can be performed in conjunction with shorter ticket lifespans. This research does not integrate these scenarios with the experiment and is possibly integrated into future work.

2.2 Related Use Case Applications

DDS-C's application and use cases are inspired by search and rescue and battlefield operations. Many papers provide solutions to these complex problems, but this research focuses on those that offer solutions using unmanned aerial vehicles (UAV). Understanding the other researchers' proposed designs and experiments helps craft the experiment use cases and real-world application.

The first paper to inspire the experiment design was Munir et al.'s research on proposing FogSurv, a fog-assisted architecture to be used in urban areas for real-time surveillance using artificial intelligence (AI) [13]. They constructed a centralized cloud server with fog nodes to offload communication and computation power burdens. Their use cases mention battlefield applications for

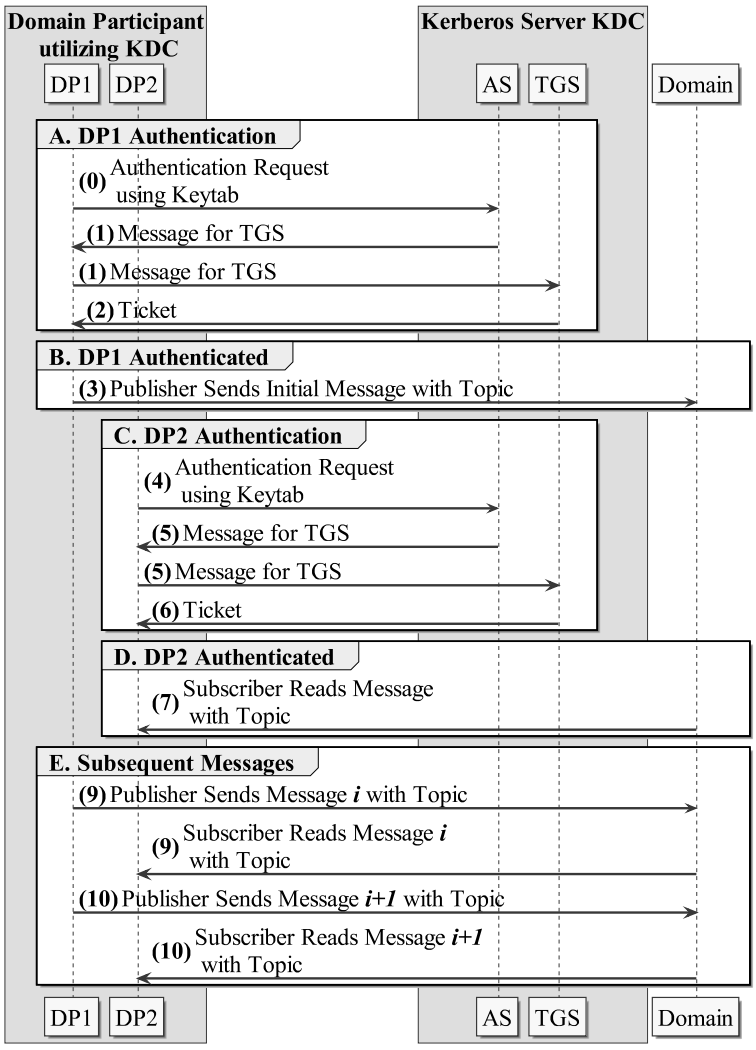

Fig. 2 DDS-C Authentication Process with *domain participants* [10]

security and control, which this paper's research does on a more specific scale with DDS-C. The experiment has two scenarios with an Internet of Things (IoT) device where in the first scenario, it offloads tasks to a fog node and the second where it offloads it to the cloud server. They measure latency in different experiment runs with data fusion and AI. The results revealed that offloading to a fog node is 37% more efficient than to the cloud. The use cases and design for broad low-power surveillance helped motivate DDS-C use case research.

In 2017, Ribeiro et al. conducted simulated and physical experiments with UAVs and DDS [14]. They also used a cloud architecture but focused on using a DDS communication infrastructure. They designed a two-layer UAV network for UAVs closer to the ground and those far away from the ground. They selected sensors ranging from those that work near the ground to those far away. The types of sensors on the UAV categorized what layer it would operate in. The simulated experiments tested different network links for low bandwidth and lossy environments in wired and wirelessly configurations. They tested

with both QoS best effort and reliable and found more throughput with reliable QoS acknowledgments. The physical experiment only utilized one UAV with ROS (Robot Operating System) for DDS communication with a base station on the cloud. They observed high signal attenuation and loss of connectivity since they used default DDS QoS policies. For future work, they plan to extend the experiment to four UAVs.

ROS is a middleware with two versions: ROS 1 and ROS 2. The difference is that ROS 2 uses DDS for real-time communication. In 2019, Sandoval and Thulasiraman's research goal was to use simulated experiments to test ROS 2's ability to protect against cyber attacks for UAV communication [15]. This work was to help support the integration of ROS 2 into the U.S. Navy's UAV swarms. Since ROS 1 was still in use for Naval UAV control, they simulated an environment where ROS 1 and ROS 2 were connected with a bridge to control three UAVs. The first two UAVs were susceptible to rogue node attacks, unwanted disabling and landing, due to ROS 1, but the third UAV used the bridge with ROS 2 and its security plugins to prevent these attacks. Even though the plugins mitigated the attacks, there was significant latency overhead due to the bridge setup. This work contributes to DDS-C by highlighting the need for node authentication when controlling UAVs.

The related works relate to DDS-C design and experiments regarding AI, network environments, and security. The following experiment combines these three elements to measure DDS-C's performance in a cloud-based network.

3 Experiment

Bright Apps developed Azoth artificial intelligence (AI) with UAVs for facial recognition in real-world use cases like search and rescue [11]. It uses Cyclone DDS, a variation of Data Distribution Service (DDS) developed by the Eclipse Foundation, to send live video frames in lossy environments to be processed by Azoth AI [16, 17]. Cyclone DDS is related to ROS 2 (Robot Operating System) because it is a tier-1 ROS 2 Middleware Interface (RMW). It uses a python binding which helps integrate DDS-C and the AI [18]. Bright Apps uses unmanned aerial vehicles (UAV) connected to and controlled by Raspberry Pi devices. These devices communicate with Amazon Web Services (AWS) Elastic Compute Cloud (EC2) instances for facial recognition processing by Azoth AI [19]. This configuration is to mimic operations where UAVs send live video feeds. OpenVPN connects this framework by providing additional security and network maintenance [20]. DDS-Cerberus (DDS-C) authenticates *domain participants* to allow for multiple node executions. Integrating it with Bright Apps technology is still in development, and this research presents initial work in this integration with a real-world commercial network infrastructure. The experiment's results aim to support this paper's previous experiment results and if DDS-C authentication traffic adds negligible latency overhead to affect normal DDS message traffic significantly.

3.1 Experiment Apparatus

The experiment testbed for DDS-C uses Cyclone DDS and Kerberos. There are three pieces of apparatus-one Raspberry Pi 4B device, a Dell XPS 13 Laptop personal computer (PC), and one EC2 instance. These labels distinguish the three pieces of equipment: Cyclone1, Cyclone2, and KerAzoth. Table 1 lists the main equipment and its specifications. All devices have Kerberos installed. Additionally, to communicate with each other, Cyclone1 and Cyclone2 are OpenVPN clients, and KerAzoth is the OpenVPN server; all traffic routes through KerAzoth from the other two. KerAzoth is located in the AWS region code us-west-2a within Oregon. Figure 3 is this equipment's testbed network diagram. All components are connected wirelessly through OpenVPN and use Cyclone DDS to communicate.

Table 1 Equipment Specifications

	Raspberry Pi: Cyclone DDS	**Laptop PC: Cyclone DDS, KDC**	**EC2: Cyclone DDS, KDC**
Name	*Cyclone1*	*Cyclone2*	*KerAzoth*
Machine	Raspberry Pi 4B	XPS 13 9310	t3.2xlarge [21]
OS	Ubuntu 20.04.3 LTS	Ubuntu 20.04.3 LTS	Ubuntu 20.04.3 LTS
CPU	ARM Cortex-A72	11th Gen i7-1185G7	Intel Xeon E5-2676 v3
Disk Space	64 GB	2 TB	58 GB
RAM	8 GB	31 GB	32 GB

Cyclone1's *domain participants* authenticate with KerAzoth's Key Distribution Center (KDC) before sending messages. Similarly, KerAzoth's *domain participants* authenticate with Cyclone2's KDC. Cyclone1 represents the UAV with Raspberry Pi device, Cyclone2 represents command and control (C2), and KerAzoth represents a network bridge and AI processing.

This experiment covers three main use cases that encompass the apparatus used in the commercial network infrastructure.

- ***Use Case 1***: Perform DDS-C authentication on a Raspberry Pi device and EC2, and after authentication, both devices communicate using Cyclone DDS. This tests communication from a Raspberry Pi device to an EC2 on the cloud.
- ***Use Case 2***: Authenticate using DDS-C over a Virtual Private Network (VPN). OpenVPN clients utilize unique credentials to communicate with the OpenVPN server. This tests communication using a VPN between devices and the cloud.
- ***Use Case 3***: Send a video feed to be processed by AI for face recognition. The video feed is sent over DDS with compressed video frames.

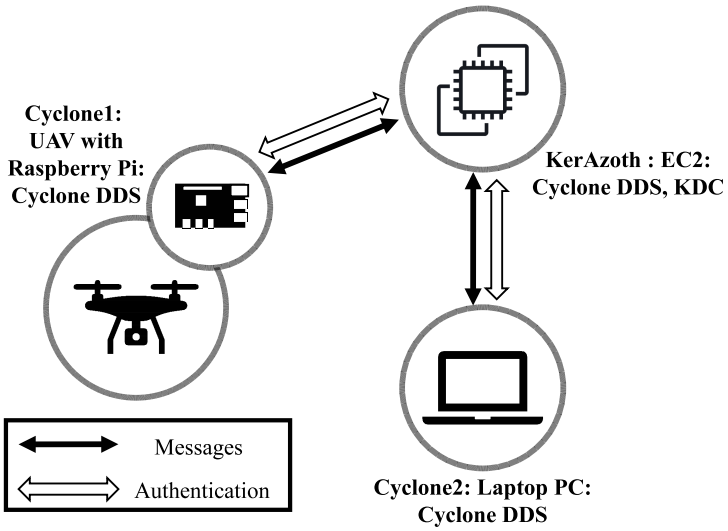

Fig. 3 Experiment Network Diagram

These frames are sent with best effort reliability QoS to handle lossy environments. This tests sending video frames over DDS for AI processing.

There is one network configuration to encompass these three use cases. It is run with QoS best effort to mimic real-world UAV use when sending video frames.

- ***Cyclone with KerAzoth***: All *domain participants* authenticate either with Cyclone2 and KerAzoth when starting up. Cyclone1 sends compressed video frames to KerAzoth for AI processing, and when finished, KerAzoth sends the processed frames to Cyclone2.

The scalability goal of Cyclone with KerAzoth is to increment the number of *domain participants* in KerAzoth to increase DDS-C authentication traffic and highlight the use of Azoth AI. Figure 4 and Table 2 illustrate how the configuration participants are set up and how the frames are passed. The figure illustrates Set 1 from the table. Cyclone1's *domain participant* has one publisher in the figure but increases, as seen in the table, by one as each set is tested to handle video frame publishing. KerAzoth's publisher and subscriber node count also increases based on the experimented set. KerAzoth has one publisher to one subscriber increasing with subsequent runs: 1:1, 2:2, 3:3, 4:4, 5:5, 6:6. Each of these ratios has a unique *domain participant*. Cyclone2's one subscriber with one *domain participant* does not increase in number, and it receives all AI processed video frames to display on the laptop screen.

Cyclone1's publishers send 100 messages with frames as data to Cyclone2. Figure 4's Publisher1 sends a frame with a *topic*, and as more publishers are added to Cyclone1, they send different frames with unique *topics*. The subscribers in KerAzoth receive these *topics*. Subscriber1 receives Topic1 and applies facial recognition AI to the frame. Other subscribers would be waiting for their respective *topics*. Afterward, Publisher2 sends the processed video frame with Topic2 to Subscriber2. In this case, the publishers in KerAzoth send the processed video frame with a *topic* that only Cyclone2's subscriber uses.

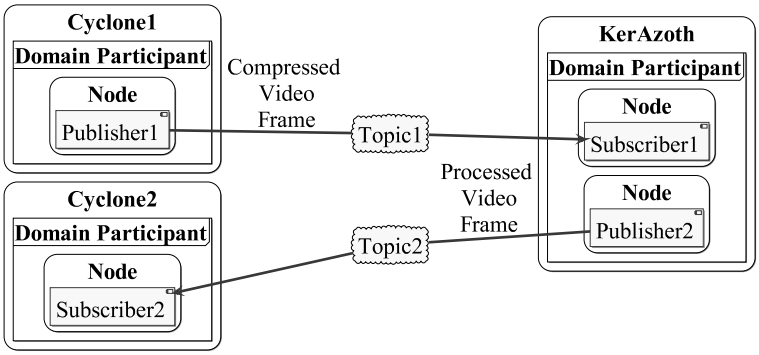

Fig. 4 Experiment Node Layout

Table 2 Experiment Participants

		Set 1	Set 2	Set 3	Set 4	Set 5	Set 6
Cyclone1	*Domain participants*	1	1	1	1	1	1
	Nodes (Pub/Sub)	1	2	3	4	5	6
Cyclone2	*Domain participants*	1	1	1	1	1	1
	Nodes (Pub/Sub)	1	1	1	1	1	1
KerAzoth	*Domain participants*	1	2	3	4	5	6
	Nodes (Pub/Sub)	2	4	6	8	10	12

To incorporate the AI, the configuration was designed to parallelize AI processsing with *domain participants*. With the number of messages set to 100 for every experiment iteration, the messages were divided up so each publisher in Cyclone1 sends a unique frame to the subscriber in Cyclone2. For example, Table 2's Set 2 has two publishers in Cyclone1's *domain participant*. One publisher would publish only odd frames and the other even frames. For subsequent Sets, the frames are divided by every third or every fourth frame for the newly added publishers. This parallel processing is an important aspect of this network configuration to showcase Azoth AI while also increasing participant count for DDS-C authentication.

On all three pieces of equipment, tcpdump captured the use case experiment's data and was sent to a separate Windows machine to be processed [22]. The data is first run with Windows Powershell scripts involving tshark, a Wireshark filtering tool [23, 24]. Afterward, the Student's t-test is used to analyze the results with a α of 0.05 using Python and SciPy [25, 26]. Since the population variance is unknown, this test is applicable to the population of DDS use cases discussed in this paper. Table 3 lists all the software mentioned for the experiment.

3.2 Assumptions and Limitations

These are the experiment's assumptions and limitations to execute the specified network setup. The assumptions are as follows:

Table 3 Software Information

Name	Version	Location
Cyclone DDS	0.8.1	Cyclone1, Cyclone2, KerAzoth
OpenVPN	2.4.7	Cyclone1, Cyclone2, KerAzoth
Kerberos	V5	Cyclone1, Cyclone2, KerAzoth
tcpdump	4.9.3	Cyclone1, Cyclone2, KerAzoth
tshark	3.4.7	Windows
PowerShell	5.1.19041.1237	Windows
Python	3.9.7	Cyclone1, Cyclone2, KerAzoth, Windows
SciPy	1.7.0	Windows

- *Domain participants* do not fail authentication and that an attacker does not compromise them.
- Cyclone1 publishers send all 100 video frames, and KerAzoth subscribers receive the specified messages.
- *Keytabs* were not renewed or changed between experiment runs.
- Relevant packet protocols such as Real-Time Publish-Subscribe (RTPS) and Kerberos (KRB5) were selected. Protocols such as Simple Service Discovery Protocol (SSDP) were excluded because they did not contribute to any of the three traffic categories in analyzing DDS-C.
- All RTPS packets without the video frame payload were categorized as *discovery+*.
- Azoth AI detected only one human face during experimentation. Other experiments may incorporate more faces to analyze the AI's processing load on the EC2.

The limitations are as follows:

- The experiment is only performed with the us-west-2a zone. If other zones were used, the experiment may differ with a lossier environment.
- RTPS messages containing video frames were fragmented, resulting in more packet traffic.
- The experiment only experimented with reliability QoS of best effort. Best effort fits the use cases; however, future experimentation could include other QoS policies.
- Collecting the total packet traffic is only one factor in determining DDS-C's impact on DDS. Other factors could include latency and location of equipment. The appartus in this experiment were in the same immediate area while the EC2 was not.
- The default discovery protocols were used. The *discovery+* traffic could differ if other discovery protocols were invoked.

3.3 Experiment Results

The data is organized based on the three used equipment as seen in Figure 5 for Cyclone1 and Cyclone2 and 6 for KerAzoth. The figures use the three data categories: *data message*, *security*, and *discovery+* traffic. The figures' independent variable uses the total number of *domain participants* for each participant set in Table 2, and the dependent variable is based on the traffic amount for each data category. The *security* traffic bytes in the use case experiment are indistinguishable compared to the greater *data message* and *discovery+* traffic.

Table 4 shows the p-values for all three equipment. Overall, the quantity of participants was not statistically significant; however, for seven participants, two cases were statistically significant for Cyclone2 and KerAzoth. The network and experiment setup could have influenced this situation. The experiment setup and best effort QoS use resulted in a more lossy environment and no packet retransmissions. The use of an EC2 brings possible unreliability with its network. With the addition of using a VPN, the sent video frames could be lost over the network. Cyclone1 did not have a statistically significant p-value because it is the starting point for all message traffic by sending captured video frames; only the components receiving the messages were affected. Even though the p-values were statistically significant, the results reveal that choosing the correct network and equipment setup is important in ensuring all components function as intended. Also, the other participant p-values show that this significance is uncommon and that DDS-C's additional *security* traffic did not impose a statistically significant change in the overall traffic. Instead, the Cyclone DDS and network setup contributed to this change.

Table 4 Configuration p-values

Participants	Cyclone1	Cyclone2	KerAzoth
3	0.911	0.888	0.785
4	0.9	0.77	0.751
5	0.09	0.114	0.8
6	0.946	0.905	0.899
7	0.315	0.002[1]	0.003[1]
8	0.234	0.82	0.8

[1]Statistically significant p-values with α 0.05.

Figure 5 and 6's *data message* traffic for all three components show no dramatic change overall because Cyclone1 sends out 100 video frames regardless of participant count. Cyclone1 and Cyclone2 send traffic through KerAzoth's OpenVPN server; therefore, the average traffic of both Cyclone1 and Cyclone2 should be roughly equal to KerAzoth's. For Cyclone1 and Cyclone2 the average is around 7,509 KB, which doubled is 15,018 KB. This byte average is roughly equal to KerAzoth's *data message* traffic's byte average of 15,028 KB. The

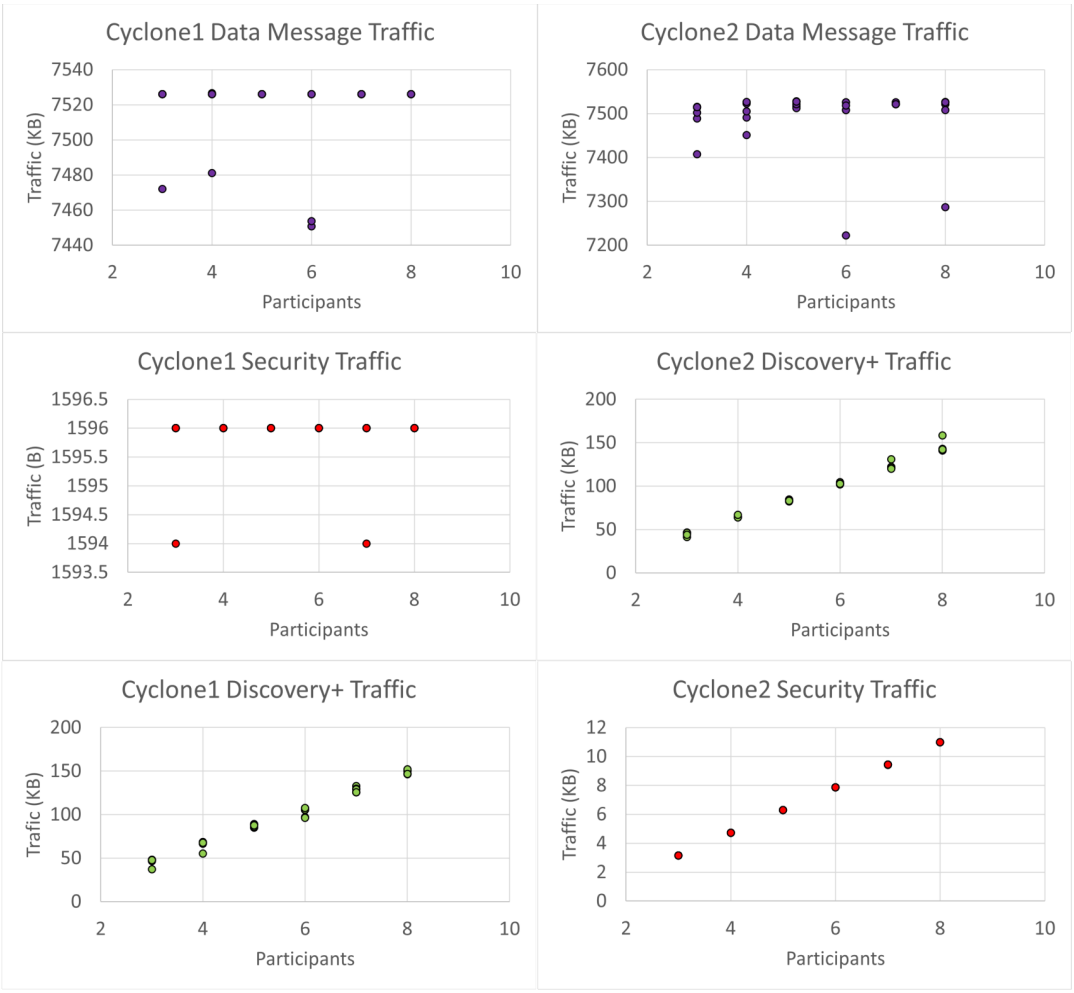

Fig. 5 Experiment results. Left column: Cyclone1. Right column: Cyclone2.

figures' *security* traffic is consistent with participants and their DDS-C authentication. Cyclone1 has only one *domain participant* to authenticate; therefore, overall traffic has no substantial change. Cyclone2 and KerAzoth's *domain participant* count increases for each experiment run, resulting in increased authentication and a strong positive linear correlation. Due to the consistent participant counts, all three components' *discovery+* traffic have positive linear correlations.

The use case experiment results show that DDS-C's security overhead is not statistically significant enough to hinder normal DDS operations with sending video frames. Using DDS-C in a real-world environment with architecture similar to Bright Apps will benefit DDS security and expand its integration in more use cases.

4 Conclusion

This research experimented DDS-Cerberus with use cases around unmanned aerial vehicles (UAV), the cloud, and video streams. The background on Data

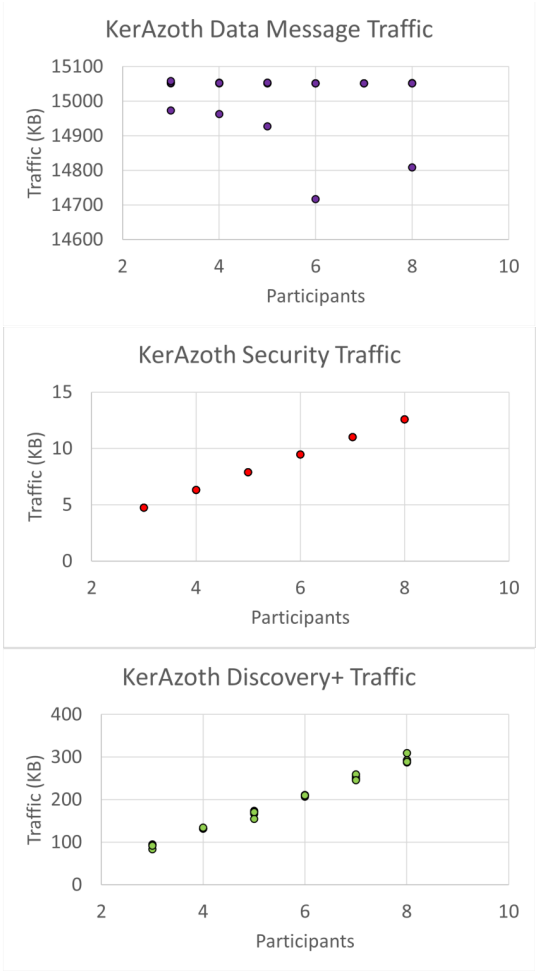

Fig. 6 Experiment results for KerAzoth.

Distribution Service (DDS), Kerberos, and related works on UAV-related papers culminated into the three use cases. The experiment tested these use cases by connecting devices through a Virtual Private Network (VPN) and used DDS-C to authenticate *domain participants*. The experiment's configuration emulated use cases for real-world operations such as using unmanned aerial vehicles (UAV) for search and rescue. It used Cyclone DDS as its testbed where the nodes in the *domain participants* deal with sending and receiving video frames, emulating UAVs sending their video feeds for artificial intelligence (AI) processing. The Quality of Service (QoS) used was best effort to mimic an operational environment where some frames are not needed for the facial recognition AI. To analyze the collected traffic from the configuration, the packets were divided into three message categories: *data message*, *security*, and *discovery+* traffic. The *security* traffic quantity was low enough to not be statistically significant for the majority of the configuration runs. The results show that the mean traffic from DDS-C overhead is insignificant when constant video frames are sent over the network. The experiment shows that

DDS-C applied to other DDS implementations or even in conjunction with other software adds security benefits without hindering overall performance.

Future work would incorporate middleware handling multiple Internet of Things (IoT) devices for integration into real-time systems. The node authentication provided by DDS-C would be beneficial for search and rescue and battlefield operations. Future research aims to integrate the use case experiment setup with multiple UAVs and more diverse AI. Additionally, as DDS-C evolves with better functionality and features, future work could also include experimenting with cyber attacks against it. These future work ideas could develop, extend, and add to the use cases in this paper.

Governments, companies, and people are looking to improve existing technologies through future works. DDS-C is still in development and requires more real-world experimentation before operational use to improve DDS security and development.

5 Acknowledgments

This research was supported by Bright Apps. The team there provided the resources and support to conduct these experiments.

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 United States.

References

[1] Sánchez, R., Evans, J., Minden, G.: Networking on the battlefield: Challenges in highly dynamic multi-hop wireless networks. In: MILCOM 1999. IEEE Military Communications. Conference Proceedings (Cat. No. 99CH36341), vol. 2, pp. 751–755 (1999). IEEE

[2] Munir, A., Kwon, J., Lee, J.H., Kong, J., Blasch, E., Aved, A., Muhammad, K.: FogSurv: A Fog-Assisted Architecture for Urban Surveillance Using Artificial Intelligence and Data Fusion. IEEE Access (2021)

[3] Nobre, J., Rosario, D., Both, C., Cerqueira, E., Gerla, M.: Toward software-defined battlefield networking. IEEE Communications Magazine **54**(10), 152–157 (2016). https://doi.org/10.1109/MCOM.2016.7588285

[4] Object Management Group: "OMG Standards for Industries." Accessed Oct 11, 2021. https://www.omg.org/industries/index.htm

[5] Abdulghani, R.M., Alrehili, M.M., Almuhanna, A.A., Alhazmi, O.H.: Vulnerabilities and Security Issues in IoT Protocols. In: 2020 First

International Conference of Smart Systems and Emerging Technologies (SMARTTECH), pp. 7–12 (2020). IEEE

[6] Michaud, M.J., Dean, T., Leblanc, S.P.: Attacking OMG data distribution service (DDS) based real-time mission critical distributed systems. In: 2018 13th International Conference on Malicious and Unwanted Software (MALWARE), pp. 68–77 (2018). IEEE

[7] Goerke, N., Timmermann, D., Baumgart, I.: Who Controls Your Robot? An Evaluation of ROS Security Mechanisms. In: 2021 7th International Conference on Automation, Robotics and Applications (ICARA), pp. 60–66 (2021). IEEE

[8] Park, A.T., Dill, R., Hodson, D.D., Henry, W.C.: DDS-Cerberus: Data Distribution via Ticketing. In: The 2021 World Congress in Computer Science, Computer Engineering, & Applied Computing (CSCE'21)

[9] Park, A.T., Dill, R., Hodson, D.D., Henry, W.C.: DDS-Cerberus: Ticketing Performance Experiments and Analysis. In: The 2021 International Congress on Computational Science and Computational Intelligence (CSCI'21)

[10] Park, A.T., Peck, N., Dill, R., Hodson, D.D., Grimaila, M.R., Henry, W.C.: Quantifying DDS-Cerberus Network Control Overhead. Unpublished

[11] Bright Apps LLC: "Bright Apps LLC" Accessed Oct 11, 2021. https://brightappsllc.com/

[12] MIT Kerberos: "Kerberos V5 System Administrator's Guide." Accessed Oct 11, 2021. https://web.mit.edu/kerberos/krb5-1.10/krb5-1.10.7/doc/krb5-admin.html

[13] Munir, A., Kwon, J., Lee, J.H., Kong, J., Blasch, E., Aved, A.J., Muhammad, K.: Fogsurv: A fog-assisted architecture for urban surveillance using artificial intelligence and data fusion. IEEE Access **9**, 111938–111959 (2021). https://doi.org/10.1109/ACCESS.2021.3102598

[14] Ribeiro, J.P., Fontes, H., Lopes, M., Silva, H., Campos, R., Almeida, J.M., Silva, E.: Uav cooperative perception based on dds communications network. In: OCEANS 2017 - Anchorage, pp. 1–8 (2017)

[15] Sandoval, S., Thulasiraman, P.: Cyber security assessment of the robot operating system 2 for aerial networks. In: 2019 IEEE International Systems Conference (SysCon), pp. 1–8 (2019). https://doi.org/10.1109/SYSCON.2019.8836824

[16] Eclipse Foundation: "Eclipse Cyclone DDS" Accessed Oct 11, 2021. https://github.com/eclipse-cyclonedds/cyclonedds

[17] Eclipse Foundation: "Eclipse Cyclone DDS" Accessed Oct 11, 2021. https://projects.eclipse.org/projects/iot.cyclonedds

[18] Eclipse Foundation: "Python binding for Eclipse Cyclone DDS" Accessed Oct 11, 2021. https://github.com/eclipse-cyclonedds/cyclonedds-python

[19] "Amazon EC2." Accessed Oct 11, 2021. https://aws.amazon.com/ec2/

[20] "OpenVPN." Accessed Oct 11, 2021. https://openvpn.net/

[21] "Amazon EC2 Instance Types." Accessed Oct 11, 2021. https://aws.amazon.com/ec2/instance-types/

[22] The Tcpdump Group: "TCPDUMP/LIBPCAP public repository." Accessed Oct 11, 2021. https://www.tcpdump.org/

[23] "PowerShell." Accessed Oct 11, 2021. https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/powershell

[24] Wireshark: "tshark." Accessed Oct 11, 2021. https://www.wireshark.org/docs/man-pages/tshark.html

[25] "Python 3.0 Release." Accessed Oct 11, 2021. https://www.python.org/download/releases/3.0/

[26] "SciPy." Accessed Oct 11, 2021. https://scipy.org/

VII. Conclusion

This research introduced DDS-Cerberus (DDS-C) and its subsequent design, testing, and application. The initial DDS-C design was the important groundwork that led to more robust features and diverse experiments. Integrating DDS-C into Robot Operating System 2 (ROS 2) and Cyclone DDS widened the scope for different use case tests and also possible integration with other Data Distribution Service (DDS) implementations. The various experiments measured latency and packet captures to understand DDS-C's impact on simulated and real-world environments. Each experiment helped determine that DDS-C does not hinder DDS operations and adds extra benefit without impeding performance. The four papers are the foundation to DDS-C's continued development and application in improving DDS middleware and enforcing node and data integrity.

7.1 Future Work

DDS-C has many avenues for integration and applicability since the testbeds used are open-source. These potential ideas could expand research on DDS-C:

- Create a DDS-C node or equivalent that would handle all authentication operations. This node is responsible for managing and categorizing tickets for nodes allowing for centralized control of DDS-C. For example, the DDS-C node acts as a firewall or switch that routes other nodes' authentication requests. This node enforces node registration through ticket pairing to identify rogue nodes.

- Add a Quality of Service (QoS) security policy that utilizes DDS-C tickets to increase message security. Since many QoS policies determine how publishers send out messages, DDS-C tickets can be used to be unique identifiers in ad-

dition to the *topic*. The tickets can also be used to route messages to certain subscribers.

- Integrating DDS-C into the architecture of DDS source code. This integration is also extended to other DDS implementations such as ROS 2 and Cyclone DDS. For example, a unique Kerberos credential is required when creating and using any publisher or subscriber. DDS also prompts the user to have a Kerberos setup when using DDS-C.

- Experiment DDS-C with the Real-Time Publish-Subscribe (RTPS) layer's data writers and data readers. Since this layer is a lower level than Data-Centric Publish-Subscribe (DCPS), working with it offers avenues for more refined control and implementation of DDS-C. One possibility is to discover what other DDS components can be compromised through impersonation attacks and mitigate with DDS-C authentication.

- Perform impersonation attacks against an entity that is authenticated with DDS-C to test DDS-C's mitigation of the attack. An idea would be to use multiple unmanned aerial vehicles (UAVs) that are authenticated with DDS-C and send malicious commands to certain UAVs. Another idea would be to use simulation software to create real-world situations. In addition to experimenting with impersonation attacks, DDS-C can have additional features using Kerberos tickets to mitigate other cyber attacks such as spoofing and replay attacks.

Bibliography

1. RTI. RTI in Aerospace and Defense. `https://info.rti.com/hubfs/Datasheets/RTI_Datasheet_20002_Aerospace-Defense_V11_Web_0718.pdf`. Online; accessed Nov 17, 2021.

2. RTI. General Atomics Aeronautical Systems, Inc. `https://www.rti.com/hubfs/docs/GenAtomics_Snapshot.pdf`. Online; accessed Nov 17, 2021.

3. Jerry Towler and Matthew Bries. ROS Military: Progress and Promise. In *Proc. 2018 Ground Vehicle Systems Engineering and Technology Symposium (GVSETS)*, 2018.

4. Nataliia Neshenko, Elias Bou-Harb, Jorge Crichigno, Georges Kaddoum, and Nasir Ghani. Demystifying iot security: An exhaustive survey on iot vulnerabilities and a first empirical look on internet-scale iot exploitations. *IEEE Communications Surveys Tutorials*, 21(3):2702–2733, 2019.

5. Niklas Goerke, David Timmermann, and Ingmar Baumgart. Who Controls Your Robot? An Evaluation of ROS Security Mechanisms. In *2021 7th International Conference on Automation, Robotics and Applications (ICARA)*, pages 60–66. IEEE, 2021.

6. Andrew T. Park, Richard Dill, Douglas D. Hodson, and Wayne C. Henry. DDS-Cerberus: Data Distribution via Ticketing. In *The 2021 World Congress in Computer Science, Computer Engineering, & Applied Computing (CSCE'21)*.

7. Andrew T. Park, Richard Dill, Douglas D. Hodson, and Wayne C. Henry. DDS-Cerberus: Ticketing Performance Experiments and Analysis. In *The 2021 International Congress on Computational Science and Computational Intelligence (CSCI'21)*.

8. Andrew T. Park, Nathaniel Peck, Richard Dill, Douglas D. Hodson, Michael R. Grimaila, and Wayne C. Henry. Quantifying DDS-Cerberus Network Control Overhead. Unpublished.

9. Andrew T. Park, Nathaniel Peck, Richard Dill, Douglas D. Hodson, Michael R. Grimaila, and Wayne C. Henry. Distribution of DDS-Cerberus Authenticated Facial Recognition Streams. Unpublished.

10. DDS Foundation. What is DDS? `https://www.dds-foundation.org/what-is-dds-3/`. Online; accessed Nov 17, 2021.

11. Object Management Group. DDS Vendor Directory Listing. `https://www.omg.org/dds-directory/vendor/list.htm`. Online; accessed Nov 17, 2021.

12. Endre Erős, Martin Dahl, Kristofer Bengtsson, Atieh Hanna, and Petter Falkman. A ROS2 based communication architecture for control in collaborative and intelligent automation systems. *Procedia Manufacturing*, 38:349–357, 2019.

13. Open Robotics. ROS 2 Foxy Fitzroy. `https://docs.ros.org/en/foxy/Releases/Release-Foxy-Fitzroy.html`. Online; accessed Nov 17, 2021.

14. Arguedas, Mikael and Ragnarok, Steven! and Thomas, Dirk. ROS 2 Releases and Target Platforms. `https://ros.org/reps/rep-2000.html`, 2020. Online; accessed Nov 17, 2021.

15. Eclipse Foundation. Eclipse Cyclone DDS. `https://github.com/eclipse-cyclonedds/cyclonedds`. Online; accessed Nov 17, 2021.

16. Eclipse Foundation. Eclipse Cyclone DDS. `https://projects.eclipse.org/projects/iot.cyclonedds`. Online; accessed Nov 17, 2021.

17. Eclipse Foundation. Python binding for Eclipse Cyclone DDS. https://github.com/eclipse-cyclonedds/cyclonedds-python. Online; accessed Nov 17, 2021.

18. Bright Apps LLC. Bright Apps LLC. https://brightappsllc.com/. Online; accessed Nov 17, 2021.

19. MIT Kerberos. Kerberos: The Network Authentication Protocol. https://web.mit.edu/Kerberos/. Online; accessed Nov 17, 2021.

20. die.net. kerberos(1) - Linux man page. https://linux.die.net/man/1/kerberos. Online; accessed Nov 17, 2021.

21. MIT Kerberos. keytab. https://web.mit.edu/kerberos/krb5-1.12/doc/basic/keytab_def.html. Online; accessed Nov 17, 2021.

22. MIT Kerberos. Encryption types. https://web.mit.edu/kerberos/krb5-devel/doc/admin/enctypes.html. Online; accessed Nov 17, 2021.

23. Raghad M Abdulghani, Marwa M Alrehili, Abrar A Almuhanna, and Omar H Alhazmi. Vulnerabilities and Security Issues in IoT Protocols. In *2020 First International Conference of Smart Systems and Emerging Technologies (SMARTTECH)*, pages 7–12. IEEE, 2020.

24. Michael James Michaud, Thomas Dean, and Sylvain P Leblanc. Attacking OMG data distribution service (DDS) based real-time mission critical distributed systems. In *2018 13th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 68–77. IEEE, 2018.

Acronyms

AI artificial intelligence. iv, 6, 1

AS Authentication Server. 7

DCPS Data-Centric Publish-Subscribe. 4, 71

DDS Data Distribution Service. iv, 1, 2, 4, 5, 6, 8, 9, 10, 11, 70, 71, 1

DDS-C DDS-Cerberus. iv, 1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 70, 71, 1

GA General Atomics. 1

IEEE Institute of Electrical and Electronics Engineers. 12, 19

IoT Internet of Things. iv, 1, 2, 6, 1

KDC Key Distribution Center. 7, 9

MIT Massachusetts Institute of Technology. 7

OMG Object Management Group. 1, 4

QoS Quality of Service. iv, 4, 5, 70, 1

ROS Robot Operating System. 6

ROS-M Robot Operating System-Military. 1

ROS 2 Robot Operating System 2. iv, 2, 4, 6, 9, 10, 11, 70, 71, 1

RTI Real-Time Innovations. 1, 4

RTPS Real-Time Publish-Subscribe. 71

SSDS Ship Self Defense System. 1

TGS Ticket Granting Server. 7

U.S. United States. 1

UAV unmanned aerial vehicle. iv, 6, 71, 1

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
24–03–2022	Master's Thesis	July 2020 — Mar 2022

4. TITLE AND SUBTITLE

DDS-Cerberus: Improving Security in DDS Middleware Using Kerberos Tickets

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Andrew T. Park, 2d Lt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-22-M-052

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Bright Apps LLC
2872 Ygnacio Valley Road 146
Walnut Creek, CA 94598
COMM 925-864-7746
Email: greg@brightappsllc.com

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The military deploys many IoT in battlefield operations to provide information on terrain and enemy combatants. It also deploys automated robots or UAVs where securing and trusting collected data is essential. Choosing the middleware that handles this message transfer is crucial for real-time operations. Networks with multiple entities, including IoT devices, UAVs, and small computers, require robust middleware facilitating message sending in real-time. Ideally, the middleware would provide QoS to handle lost packets and retransmissions in lossy environments, especially between low-power machines. DDS is a middleware that implements real-time and QoS capabilities by sending messages, not based on endpoints but *topics*. However, DDS nodes are susceptible to impersonation attacks, which compromise integrity and trust. To mitigate these attacks, DDS-C is developed as a security layer that integrates with DDS by using Kerberos tickets to identify and authenticate valid DDS nodes. This thesis evaluates DDS-C performance, determining if authentication overhead impedes DDS operations by using ROS 2 and Cyclone DDS as testbeds. Additionally, DDS-C is integrated into a commercial network AI provided by Bright Apps as a real-world use case. The results of this research conclude that DDS-C does not impact DDS operations to any significant degree. The added security and minimal middleware impact could help the military ensure node integrity in operational missions.

15. SUBJECT TERMS

DDS, Kerberos, ROS 2, Cyclone DDS, QoS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Major Richard Dill, AFIT/ENG
U	U	U	UU	86	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636, ext 3652; richard.dill@afit.edu