3-2022

Considering DDS in the Domain of DIS - Pros and Cons

Nathaniel R. Peck

CONSIDERING DDS IN THE DOMAIN OF DIS - PROS AND CONS

THESIS

Nathaniel R Peck, B.S.E.E., Captain, USSF

AFIT-ENG-MS-22-M-053

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-22-M-053

CONSIDERING DDS IN THE DOMAIN OF DIS - PROS AND CONS

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 Electrical Engineering

Nathaniel R Peck, B.S.E.E., B.S.E.E.

Captain, USSF

March 24, 2022

AFIT-ENG-MS-22-M-053

CONSIDERING DDS IN THE DOMAIN OF DIS - PROS AND CONS

THESIS

Nathaniel R Peck, B.S.E.E., B.S.E.E.
Captain, USSF

Committee Membership:

Douglas D Hodson, Ph.D
Chair

Richard Dill, Ph.D
Member

Michael R Grimaila, Ph.D
Member

Abstract

Distributed Interactive Simulation (DIS) is a legacy IEEE standard for defining and structuring Protocol Data Unit (PDU)s in large scale distributed wargames. Although the standard specifies various Qualities of Service (QoS) appropriate for certain PDUs, a one-size-fits-all transport strategy is traditionally employed via User Datagram Protocol (UDP). Since the inception of DIS, the Object Management Group (OMG) has produced a standard for a Data Distribution Service (DDS) which has been implemented by several middleware vendors. DDS middleware offers an abstraction for network communications that allows applications and developers to easily employ configurable QoS by topic. Adoption and use of these QoS in DIS applications may introduce greater compliance with the IEEE standard and enrich the service features available to distributed wargames and their developers. In this thesis, current use cases of DIS and DDS are examined. The cost, network burden, and performance of DDS is measured and analyzed through experimentation and support DDS's eligibility to promote greater compliance with the IEEE standard for DIS.

AFIT-ENG-MS-22-M-053

To my wife. Without your love, companionship, and support, I would have never seen the completion of this work.

Acknowledgments

I would like to thank my advisor and my committee for supporting my efforts and guiding me through the research process.

Table of Contents

I. Introduction

1.1 Problem Background

This section serves to expand upon the challenges facing DIS today and introduce the key concepts explored throughout this thesis. Fundamentally, this thesis revolves around two key subjects. They are Distributed Interactive Simulation (DIS) and Data Distribution Service (DDS). Each of these subjects is explained in the following sections.

1.1.1 Distributed Interactive Simulation

DIS in the general sense is an overloaded concept described by many synonymous terms. These terms include Distributed Virtual Simulation, Networked Environments, Distributed Virtual Environments, Synthetic Battlespace, etc. A common theme among these terms is an architecture consisting of several interconnected nodes (i.e. simulations) that share data using a network. In military simulations, usually real systems such as aircraft are modeled. Each node is responsible for creating a representation of a shared virtual world to the degree necessary to accomplish the goals of the simulation. Changes made to the world by individual nodes trigger data exchanges between nodes in order to maintain a common shared virtual world.

DIS in the military context implies a specific communication protocol which is described by a family of standards governed by the Institute of Electrical and Electronics Engineers (IEEE). The family is comprised of 4 specific IEEE documents, enumerated below. The most relevant two are the first two because they fully specify the structure of Protocol Data Unit (PDU)s used within the protocol and associated service profiles. Notably, the second document in the family specifies communication classes that users may employ. The third class of profile is reliable unicast, meaning that a transmitted PDU would be resent if not successfully received by an intended recipient.

(1) IEEE-Std-1278.1 - IEEE Standard for Distributed Interactive Simulation - Application Protocols [1]

(2) IEEE-Std-1278.2 - IEEE Standard for Distributed Interactive Simulation - Communication Services and Profiles [2]

(3) IEEE-Std-1278.3 - IEEE Recommended Practice for Distributed Interactive Simulation - Exercise Management and Feedback

(4) IEEE-Std-1278.4 - IEEE Trial-Use Recommended Practice for Distributed Interactive Simulation - Verification, Validation, and Accreditation

In practice, many users opt to use exclusively best-effort service classes through User Datagram Protocol (UDP) because of its deterministic performance, lightweight overhead, and simplicity. Determinism is desired in DIS exercises because real people are interacting with the virtual world. Too much latency in communications would mean remote nodes receive late information and degraded speed through their decision loops. Although this method of using exclusively UDP may satisfy many normal operating requirements when the underlying network is already 95% reliable, it is not without downfall. Not all information sent should have the same Qualities of Service (QoS). Lozes, et. al., suggest reliable QoS would be beneficial to all simulation management PDUs in a DIS exercise [3]. In fact, Holbrook, et. al., investigated the use of a logging server to enable reliable multi-cast [4]. The most common PDU in DIS exercises is the entity state PDU. Throughout normal operation, entity state PDUs are sent repeatedly. If one is lost, a new one will follow shortly. Other PDUs, such as detonations, are not sent as frequently, and a dropped PDU may cause an event to never reach remote nodes. This would certainly disrupt an outcome from the simulation. Although some may suggest solving this challenge by opening a reliable Transmission Control Protocol (TCP) communication channel for PDUs that require

reliability, TCP has been known to induce loss for other UDP traffic [5].

In DIS exercises, it is common for one PDU to be transmitted to many remote nodes. While one method of ensuring the right data gets to the right nodes is via multicast groups, it is often more challenging to establish than simply broadcasting the PDU to all nodes and requiring each node to filter out PDUs it does not need. Basiouni and Chiu explored the challenges introduced by relying on this filtering, particularly in real-time environments such as DIS [6]. Additionally, they showed the connection between traffic size and delay required for receipt of data. In the face of relevance filtering as a fix to broadcasting, Deb Fullford from MAK Technologies highlights bandwidth management as a key element for the future of DIS [7].

1.1.2 Data Distribution Service

DDS is also described by standards. In this case, the standards are provided by the Object Management Group (OMG) [8] [9]. These standards are referenced in chapter VI. Fundamentally, DDS is a middleware aimed at providing real-time communication between nodes of distributed applications. The middleware is based on a publish/subscribe programming pattern. Nodes communicate by publishing to topics to which other nodes subscribe. DDS recognizes that not all data must be sent the same way, so nodes must meet requirements for matching, specified by levels of QoS. These QoS represent contracts between publishers and subscribers. For example, a node may offer best-effort publish messages to a given topic. If another node wishes to subscribe to that topic but requests reliable service, then the node will not be matched with the publisher because the publisher did not offer reliability.

Reliability is not the only QoS supported by DDS. The middleware offers many other QoS such as latency budget, ownership strength, and history policy to name a few. Latency budget allows a subscriber to require a maximum latency for updates

allowed into its data queue. In order for such a subscriber to be matched with a publisher, the publisher must offer less than or equal to the latency budget required by the subscriber. Ownership strength solves a distributed programming challenge where multiple nodes are capable of publishing to the same topic. If two nodes attempt to publish to the same topic, the node possessing a higher ownership strength takes precedence. History policy supplements reliability by specifying how many previous updates will be saved by a publisher. This behavior is useful for instances where nodes may lose connectivity and return to a connected status desiring missed updates.

Research has shown the performance of DDS to meet the high level QoS parameters suggested for DIS by IEEE [10] [11] [12]. Other research has considered using DDS to distribute DIS data and found that DDS offered promising capabilities to applications served by DIS [13]. The research performed by Hakiri, et. al., evaluated DDS in terms of latency and jitter but did not differentiate results based on reliability as is done in this research.

1.2 Research Objectives

The objectives of this thesis are as follows:

1. Evaluate the impact of latency on DIS traffic from using DDS
2. Experiment with publishing and subscribing to topics in DDS
3. Explore means of achieving reliable transmission of DIS PDUs using DDS

1.3 Document Overview

The following chapters present various works either published or expected to be published, leading to chapter VII where final conclusions are discussed. In chapter II,

one of DIS's customer frameworks is explored. An initial exploration of DDS overhead is provided in chapter III. Publications involving a DDS evaluation strategy are provided in chapter IV and chapter V. Finally, chapter VI provides a specific examination of using DDS to distribute DIS PDUs.

II. Paper I: AFSIM's Distributed Communications

The following paper, "AFSIM's Entry into Aircraft Hardware in the Loop," was published at the 2021 World Congress in Computer Science, Computer Engineering, and Applied Computing (CSCE'21). For the purposes of considering DDS in the domain of DIS, AFSIM served as a framework to examine the favorability of DIS in one of its software users. The eXperimental Input Output (XIO) interface was developed by the framework developers for users to use as a simpler alternative to full DIS. The presence and efficacy of the XIO interface illustrates the pull for a simpler application of DIS protocols or an abstract middleware to allow employment of DIS using application layer API such as DDS.

AFSIM's Entry into Aircraft Hardware in the Loop

Nathaniel Peck, Douglas D. Hodson, Richard Dill, and Michael R. Grimaila
Air Force Institute of Technology, WPAFB, OH, USA
email: 2012raptor@gmail.com, doug@sidechannel.net, richard.dill@afit.edu, michael.grimaila@afit.edu

Abstract— *Hardware in the loop (HIL) simulations benefit from a 3D world to visualize their state. This research presents a basic entry point for the Advanced Framework for Simulation, Integration, and Modeling (AFSIM) as a solution to visualize aircraft state. This research highlights AFSIM's XIO capabilities (a network interface) as a potential path to support simulations with integrated hardware.*

Keywords: AFSIM, XIO, HIL

1. Introduction

Hardware-in-the-Loop (HIL) simulations are becoming more widely used in Department of Defense (DoD) test campaigns and in-house demonstrations. This trend is also being adopted by the commercial sector [1]. The authors describe the motivations for HIL experiments in terms of the environment faced by aircraft designers. Architects of HIL simulations must decide what software will interact with their hardware to best support their tests. This decision is based on many factors such as changing requirements. Under tight schedule and budget constraints, software architects are seldom afforded enough time, budget, and detail to survey available options to support HIL development. This can lead to a number of pitfalls which this research aims to help avoid by suggesting a look at a framework owned by the Air Force Research Laboratory (AFRL).

AFRL has acquired a software framework which it hopes will become more ubiquitous in the M&S community [2]. The purpose of this research is to evaluate and illuminate AFSIM as a candidate software product that supports a military aircraft HIL infrastructure. This research seeks to answer some of the questions of interest to HIL architects.

2. Background

The next couple of sections provide an overview of the technologies used to perform this work.

2.1 AFSIM

A high-level overview of AFSIM can be found here [3]. AFSIM is a software product that includes a framework designed to support a broad range of modeling and simulation (M&S) solutions. The AFSIM package includes many products, such as a framework to build new simulation applications and a set of software applications that can be extended for a particular purpose. Two applications of interest are wizard and warlock. This research focuses on using warlock. Section 2.3 discusses that application, where the reason for our focus will become evident. The extensible applications included within the AFSIM package allows DoD customers to add their niche functionality.

Owned and managed by AFRL, AFSIM is a freely available (DoD open-source) solution to HIL architects seeking advanced, capable, flexible, and inter-operable software. While freely available to users, the authors caution readers not to assume it is low quality software product. Development of the AFSIM product is an investment of more than $56M, with an additional commitment of $6M per year [2]. When used as a software development framework, or even an extensible application, [4] details the connection between a conceptual model and expected simulation representation to support experimental objectives. Obtaining AFSIM source code and various supported training courses are described in [2]. In short, AFRL is encouraging the use of AFSIM across the DoD and its partners. They make AFSIM available from version controlled repositories via Information Transfer Agreements (ITAs) and "sweeten the deal" by offering the source code with free user and developer training courses and content. While [2] was published as recent as 2020, AFSIM continues to grow more capable thanks to its growing developer community and research like this.

Most notably for this research, AFSIM provides highly flexible operational simulation for aircraft in a wide variety of operational contexts. This flexibility is accessible via a scripting language, interface, and IDE exposed in the user training course and material. Ops researchers can and do perform extensive simulations to understand the effectiveness of one or many aircraft (platforms) in relevant and realistic scenarios ranging from single engagements to full campaigns. It is this range of realistic aircraft operation that makes it attractive as a software foundation for aircraft HIL experiments.

Traditionally, the simulations used by ops researchers are constructive in nature, but AFSIM is capable of much more. [5] describes the capabilities available to support "pseudo" real-time requirements. [6] provides a widely used survey of the simulation categories used by the DoD based on which entities in a simulation are real and which are simulated. This construct typically assigns a category of live, virtual, or constructive to simulations. The authors of [6] suggested the existence of a fourth category called automated simulations and admitted use of a simulation type that does not fit well

within the LVC construct. The authors referred to this type as environmental simulations. The authors connected this new simulation category with the HIL paradigm. Literature review indicates little work has been published using AFSIM to interact with real hardware.

Next, we provide a brief background of HIL, its purpose in the aircraft development cycle, and connects its requirements in the military setting to the capabilities offered by AFSIM.

2.2 HIL

Hardware-in-the-Loop (HIL) is a class of simulation where one or more objects in the simulation are real hardware. As noted by [6], HIL simulations are focused on simulating an environment for the hardware under test and don't fit well into the LVC categorization construct. For example, a HIL simulation designed to test the response of an autonomous vehicle to the inputs from sensors might simulate environmental signals from the sensors rather than relying on say a real test in the real environment. This substitution provides a lower level of risk by allowing the hardware room to fail in a virtual, controlled environment.

With the trend of modern aircraft increasing in capability and sophistication, design margins are narrowing. This narrowing is pushing subsystem designers tasked with providing the right capabilities to better understand the real requirements on their designs. Historically, wider size, weight, and power (SWAP) margins afforded designers the opportunity to rely on conservative over-design to ensure requirements satisfaction. Modern aircraft designers now can rely less on over designing their respective subsystems and must understand more precisely the actual operational demands rather than conservative, rule-of-thumb estimates of experts. In the military context, simulating the operational environment of an aircraft involves more than merely simulating air, weather, and flight characteristics. It involves understanding the tempo and requirements of possible combat maneuvers, weapon deployments, and custom payloads.

There are many reasons to perform HIL experiments at the aircraft level. Suppose an aircraft is about to undergo an avionics modernization program. Managers would likely want to test the effect of a newly developed payload on a digital twin of the aircraft and weigh the cost of a new module against simulated battlefield impact. Managers could select from the operational scenarios with the most, least, and average dynamic load content to allow the hardware the opportunity to respond in a range of operational scenarios. The allows hardware developers advance feedback to better tune control parameters and capacity reserves to fit the right demands. Of course, connecting simulation software to hardware imposes a real-time requirement on the software. In this context, real-time means that one second of simulation time must advance over the span of one second in the physical world.

2.3 Warlock

Warlock is one of the applications available in the AFSIM package. It is a real-time operator in the loop (OITL) application. It can read scenario files used for the constructive application, but execute it with timing commands and execution control to support real-time operation.

Human operators generally operate at reaction speeds much slower than those required of automated control hardware. For that reason, this research aims to stress the maximum throughput of warlock while keeping up with at least the visual appearance of smooth real-time operation.

2.4 XIO

This section provides a brief overview of XIO, AFSIM's eXternal Input/Output (XIO) mechanism - an interface to external input and output. For example, it provides an interface to the network, so that, for example, communications between different AFSIM-based applications can take place. The developers emphasize its ability to communicate between AFSIM applications because it introduces its own serialization and deserialization technology. Since these technologies are specific to AFSIM, simulation applications are compiled and linked to it. As written, XIO implements communications using both User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). As with other AFSIM objects, XIO configurations can be read from user written script files. In one implementation, this input configuration would take the form of a script block declaring the required inputs. An example script input would declare multicast, unicast, or broadcast with an associated IP address, send port, and receive port.

The AFSIM developer training course provided by AFRL includes a module focused on demonstrating the use of XIO. In this module, developers write a plugin for Warlock which uses XIO to listen for operator input in the form of key strokes. These key strokes trigger behavior changes as if the programmer is a pilot of a simulated aircraft.

3. Application Design

AFSIM developers commonly begin with plugins written during the developer training as a launch point for their intended new functionality. For this research, the target is to slightly modify the XIO module from the developer training to take inputs from transmitted UDP packets containing aircraft position state rather than key strokes from a developer representing stick input to a flight dynamics model. Of the modules covered in the training, it was the only one to use the real-time Warlock application. In this way, AFSIM will essentially function as the visualizer for an active HIL simulation.

The XIO module previously relied on an external flight controller application running alongside the primary AFSIM application. This external application constantly waited for

key inputs and responded by sending an XIO packet over the loopback interface at IP address 127.0.0.1. Once the packet reached the AFSIM application, it triggered a custom written callback function to process the input command. The modified flight controller instantiated a socket using the WinSock library which persistently made blocking calls to wait for an input UDP packet containing aircraft state. In an aircraft HIL testbed, this state would come from a flight control computer's believed state. The flight controller application would then transmit an XIO packet to the primary AFSIM application, where the state would get written to the aircraft object in simulation.

To allow repeated tests at various transmission speeds, a Windows Powershell script was used to generate and transmit UDP packets containing serialized aircraft state. This state consisted of latitude, longitude, altitude, roll, pitch, and heading. Each was represented using a 64-bit floating point number. The script uniformly distributed the state across a smooth aircraft trajectory. The trajectory consisted of a northern flight path with a 360 degree roll. Figure 1 illustrates the layout of the applications used for testing.

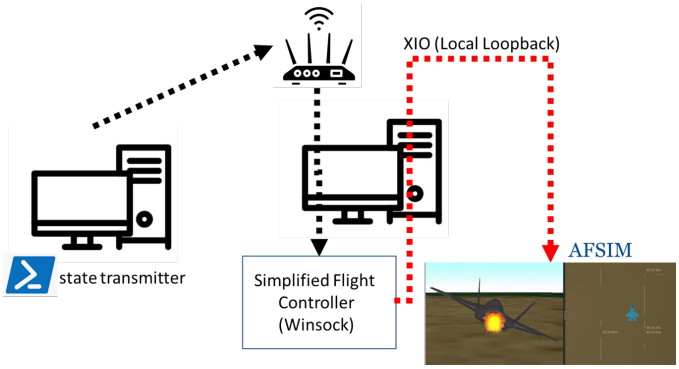

Fig. 1: Test Layout

4. Results

Windows Powershell was used to transmit up to 1,000 state packets over the period of 1 second. The AFSIM simulation successfully maintained the appearance of a smooth execution at the 1 kHz transmission rate.

5. Final Thoughts

This research examined the rate AFSIM could receive and visualize aircraft state. While AFSIM was found to maintain the appearance of smooth operation at 1 kHz state transmission rate, higher rates were prevented by Windows Powershell. AFSIM may be capable of higher input rates with additional testing. Further, the plugin developed during this research did not complete the loop back to the hardware. In this case, the ability of AFSIM to receive 48 bytes of state at a rate of 1 kHz was confirmed. Follow-on steps would involve further modification to the plugin to allow another transmission of an XIO packet from the application back to the hardware. This transmission could involve a request for payload activation. The strength of XIO in this case was the ability to continuously execute UDP listen calls external to the primary AFSIM application.

6. Disclaimer

The views expressed in this paper are those of the authors and do not reflect the official policy or position of the United States Air Force, the Department of Defense, or the U.S. Government.

References

[1] F. Almeida, "Model-based strategies for modern flight test," in *AIAA Atmospheric Flight Mechanics Conference*, 2011, p. 6673.

[2] T. D. West and B. Birkmire, "AFSIM: The air force research laboratory's approach to making M&S ubiquitous in the weapon system concept development process," *Cyber Security & Information Systems Information Analysis Center*, vol. 7, no. 3, 2020.

[3] P. D. Clive, J. A. Johnson, M. J. Moss, J. M. Zeh, B. M. Birkmire, and D. D. Hodson, "Advanced framework for simulation, integration and modeling (AFSIM)," in *Proceedings of the International Conference on Scientific Computing (CSC)*. The Steering Committee of The World Congress in Computer Science, Computer . . . , 2015, p. 73.

[4] D. W. King, D. D. Hodson, and G. L. Peterson, "The role of simulation frameworks in relation to experiments," in *2017 Winter Simulation Conference (WSC)*. IEEE, 2017, pp. 4153–4174.

[5] J. S. Thompson and D. D. Hodson, "AFSIM's pseudo-realtime hybrid simulation software design," *The Journal of Defense Modeling and Simulation*, 2021. [Online]. Available: https://doi.org/10.1177/1548512920985269

[6] C. T. D. West, "Hopes, dreams, and challenges of digital nirvana: The state of the art and the art of the possible in digital twin and digital thread," *Complex Systems Engineering: Theory and Practice*, p. 151.

Author Biographies

NATHANIEL PECK is a 1st Lieutenant in the United States Air Force, currently stationed at Wright Patterson Air Force base. He is a master's student at the Air Force Institute of Technology studying software engineering in modeling and simulation solutions. He is a 2017 graduate of Rochester Institute of Technology, where he studied electrical engineering, robotics. His previous duty station was with AFRL's Aerospace Systems Directorate to develop next generation aircraft power systems.

DOUGLAS D. HODSON is an Associate Professor of Computer Engineering at the Air Force Institute of Technology (AFIT), Wright-Patterson AFB, Ohio USA. He received a B.S. in Physics from Wright State University in 1985, and both an M.S. in Electro-Optics in 1987 and an M.B.A. in 1999 from the University of Dayton. He completed his Ph.D. at the AFIT in 2009. His research interests include computer engineering, software engineering, real-time distributed simulation, and quantum communications. He is also a DAGSI scholar and a member of Tau Beta Pi.

MAJOR RICHARD DILL is an Assistant Professor of Computer Science at the Air Force Institute of Technology (AFIT), Wright-Patterson AFB, Ohio USA. He received a

B.S. in Computer Science from the University of Maryland at College Park in 2004, and both an M.S. in Computer Science in 2008 and a PhD in 2018 from AFIT. Major Dill's research interests include computer security, algorithms, and artificial intelligence.

MICHAEL R. GRIMAILA (BSEE 1993; MSEE 1995; PhD 1999, Texas A&M University) is a professor and head of the Department of Systems Engineering and Management at the Air Force Institute of Technology, Wright-Patterson Air Force Base in Ohio, USA. He is a member of Tau Beta Pi, Eta Kappa Nu, and the Association for Computing Machinery, as well as a Senior Member of the IEEE, and a Fellow of the Information System Security Association. He can be contacted via email at michael.grimaila@afit.edu.

III. Paper II: DDS Network Overhead

The following paper, "Quantifying DDS Network Overhead," was published at the 2021 World Congress in Computer Science, Computer Engineering, and Applied Computing (CSCE'21). The purpose of the work in the scope of this thesis was to perform initial examination of the usage and performance cost of DDS. The periodic traffic required by the DDS discovery mechanism was highlighted.

Quantifying DDS Network Overhead

Nathaniel Peck, Douglas D. Hodson, Richard Dill, and Michael R. Grimaila
Air Force Institute of Technology, WPAFB, OH, USA
email: 2012raptor@gmail.com, doug@sidechannel.net, richard.dill@afit.edu, michael.grimaila@afit.edu

Abstract— *The Data Distribution Service (DDS) is an emerging distributed communication middleware which aims to simplify and enhance the communication between distributed applications. While DDS may simplify the creation of such communication mechanisms that include quality of service (QoS) controls, some developers have expressed concern about introduced runtime overhead. This research seeks to measure some of the network overhead paid as a penalty for supporting this type of communication style with QoS characteristics. We examine the communication traffic associated with DDS over a fixed period of time in various configurations.*

Keywords: DDS, RTPS, Network, Communication, Overhead, Wireshark, JSON, Powershell

1. Introduction

Previous work has been done to compare the runtime performance of DDS to that of raw sockets. This work is presented in a paper [1] and a master's thesis [2]. The work presented DDS (a publish-subscribe pattern) as a simplification for complex network connections in complex environments. What if a designer favors minimum network loads to design simplicity and is willing to spend the time upfront to architect a legacy communication strategy? Perhaps there are use cases where DDS is simply overkill for the simple communication requirements of, say, one-to-one communication instances. Sometimes designers are so motivated to use emerging technologies that they pigeonhole new technologies designed for new problems onto older problems better answered by legacy technologies.

The research provided by [1], [2] compare the differences in performance between DDS and raw sockets by varying the size of a common packet transmitted using both DDS and traditional sockets. The research focused on measuring latency and jitter. While the research did test the performance of DDS on a network with extra load, it did not compare the extra load placed on the network by DDS relative to the raw socket approach. [3] also provides evaluation of DDS performance, tracking latency, jitter, and throughput. The formula used in that case to calculate throughput characterized the bandwidth of the WiFi hardware rather than degree of network congestion incurred by using DDS.

This research evaluates the extra load placed on a network by DDS using eProsima's implementation of DDS in their shapes demonstration [4]. While this work does not provide a general-case quantification of the network overhead incurred by all implementations of DDS, it does present the methods used to obtain a measure of overhead given an implementation. This information may be useful to those wanting to understand whether or not their network provides sufficient capability to handle a DDS implementation.

2. Background

The next couple of sections provide an overview of the concepts and technologies used throughout this work.

2.1 Overhead

The subject of overhead can take on many forms. When the subject of overhead occurs in design discussion, it is usually because there is some finite supply of a particular resource, and designers want to understand precisely how much of that resource is consumed to provide a desired feature or benefit. Different metrics are better suited to support discussions of different types of overhead. The point of this paper is not to provide an exhaustive overview of the different types of overhead associated with reaping the numerous benefits of the DDS. Rather, the authors focus on a particular type of network overhead which has not yet gleaned many publications in the community.

Previous research has focused on the performance of the DDS using metrics such as latency, jitter, and throughput [2], [3]. [3] notes that they controlled the amount of traffic on a wireless network to get a measure of throughput. They note network congestion can prove harmful to performance of mission-critical or time-sensitive systems. For this reason, this paper focuses on the price paid in network traffic for the convenience of the services offered by DDS. In some cases, this overhead is measured only in what some refer to as protocol overhead, or the packet size allocated to header or metadata information. While this paper will include some information on protocol overhead, the main focus will be on specifically network overhead. The authors define network overhead as a combination of packet size allocated to DDS wire protocol headers and extra messages required to provide the QoS.

2.2 DDS

The Data Distribution Service is a standard established by the Object Management Group (OMG) to guide implementations of distributed real-time communication middle-

ware using a publish-subscribe pattern [5]. The standard offers high-level data-centric interfaces in lieu of message (object) oriented programming. The standard has received the attention of numerous software development companies, leading to several implementations. Some implementations are free and open, some are free but closed-source, others are licensed. For ease of access, this research uses exclusively the free and open implementation offered by eProsima's shapes demonstration, described in section 2.5.

As a middleware, DDS is implemented in layers. In the case of DDS, it is implemented as an abstraction layer above an operating system. As an abstraction above an operating system, DDS aspires to simplify the creation of and access to communication mechanisms such as sockets. This abstraction layer provides off-the-shelf access to more advanced types of configurations for the runtime behavior of the underlying communication infrastructure. These behaviors are provided through DDS Qualities of Service (QoS).

2.3 QoS

This section is not meant to provide exhaustive coverage of the Qualities of Service (QoS) offered by DDS. Rather, this section focuses on those which lead to additional packet traffic over the communication network. These extra packets will be considered network overhead.

Above the configurable QoS, DDS standard requires dynamic participant discovery as part of the abstraction to operating system communication infrastructure. Vendors may implement this dynamic discovery in different ways, but in general this dynamic discovery must rely on periodic network traffic above that required of a legacy communication strategy by the author's intuition. This intuition was tested by one of the tests described in section 3.

2.4 RTPS Protocol

The Real-Time Publish Subscribe (RTPS) protocol is the wire protocol specified by the DDS standard for interoperability. RTPS sits above the transport layer of choice. RTPS protocol may concatenate multiple payloads in a single packet. When wireshark exports packets with concatenated payloads to JSON, Windows Powershell requires at least version 7.1 to utilize the 'AsHashTable' command switch on the ConvertFrom-Json command to correctly parse the concatenated payloads.

2.5 Shapes Demo

Several DDS implementations provide a standard demonstration illustrating the functionality of the publish-subscribe pattern by exposing users to the creation of subscribers (readers) and publishers (writers) of various shapes moving through a space. eProsima provides a graphical user interface to not only see the shapes in the demonstration, but create new publishers and subscribers with various QoS. Their implementation is open-source and available on GitHub. It provides a simple means of testing interoperability with other DDS implementations. Users are able to create publishers for a variety of shapes with a variety of colors, sizes, and QoS.

2.6 Wireshark

Wireshark is a free application used to monitor and collect network traffic. The free version of wireshark supports recognition and filtering of RTPS packets and allows users to parse fields of its content. The application allows users to export collections to various formats. In this research, JavaScript Object Notation (JSON) format files will be exported for analysis in Windows Powershell.

3. Testing & Results

Wireshark version 3.4.6 was used to collect wireless packet traffic for approximately 1 minute of activity in various stages and configurations of the eProsima shapes demonstration, version 2.3.0. Windows Powershell version 5.1 was used to process Wireshark's exported JSON data files. Because RTPS allows multiple sub-messages to be serialized in a single packet, the exported JSON files from Wireshark contained duplicate keys. These duplicate keys presented a problem when importing into Powershell objects. Only the first or last value associated with a duplicated key was converted to an object. A custom module was written for Powershell to "normalize" the JSON files from Wireshark by incrementing an index at the end of duplicated keys. The first set of collections was executed with the eProsima software using default application settings. All possible application settings are represented using a capture of the settings window, shown in Figure 1. The capture displays the persistent default settings that return once the application is closed and reopened. As shown, default transport occurs using UDP with a domain ID of 0, an update interval for shape movements of 75 milliseconds (ms), and a shape movement speed set to the 7th value from the left starting at 0. The fastest update interval possible was 1 ms and the slowest was 9,999 ms. Subsequent collections added various configurations of publishers and subscribers to the network for observation of network traffic. Figures 2 and 3 display the available configurations for publishers and subscribers, respectively.

First, traffic was monitored for one minute while the shapes demo was initialized with no publishers or subscribers in the domain. This test was executed to test the author's intuition of the discovery network traffic required to offer dynamic participant discovery. In legacy communication designs where the designer knows the desired number of participants, say a simple one-to-one communication, this discovery traffic is present only until the pre-determined number of connections have been made. In the many-to-many environment for which DDS was intended, this traffic continues in order to offer dynamic discovery of additional participants. Initially, the eProsima software sends 6 packets,

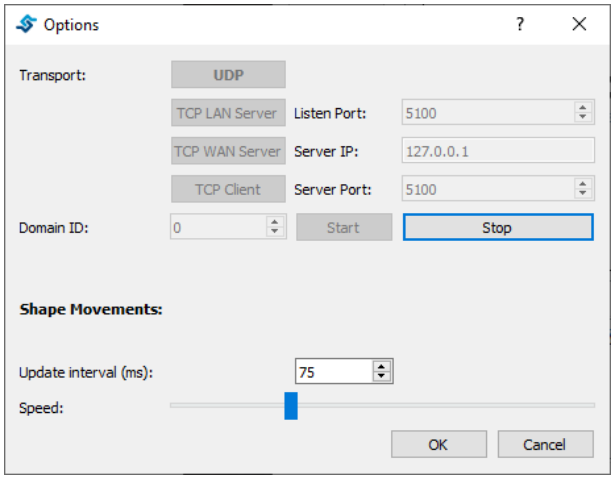

Fig. 1: Application Settings

each being 336 bytes on wire at a rate of one packet roughly every 100 ms. This equates to 12 packets on the network totaling 4012 bytes in the first 500 ms of activity. Each transmission is sent to the traffic controller and then from the traffic controller to the network. Therefore, each packet transmitted by DDS equates to 2 packets on the network. Following this burst of initial packet traffic, the software continues to send 336 byte discovery packets at a slower rate of one every 3 seconds. An additional collect performed using a shape update interval of 1 ms was performed to determine whether this impacted the rate of dynamic discovery packet transmission. This rate was found to be a constant rate, independent of the application settings for shape update interval.

The default publisher QoS included a history of 1 state, reliable transport, volatile durability, and infinite lease duration, announcement period, deadline, and lifespan.

Second, traffic was monitored for one minute with merely one subscriber to the square topic using default QoS. Third, traffic was monitored for one minute with merely one publisher using default QoS on the triangle topic. Fourth, traffic was monitored for one minute with both a publisher and a subscriber of different topics. Lastly, one minute of traffic was collected for the case where a publisher and subscriber exists for the same topic.

Because we are not interested in measuring latency or jitter, the other traffic on the WiFi network did not need to be controlled. We simply apply a Wireshark filter for RTPS traffic so we capture only DDS communication in our packet traffic for summation on size and observation of transmission frequency. In our case, the only caveat to our summation is caution for repeated transmission of dropped packets. These packets should not be double counted in the summation of total network traffic.

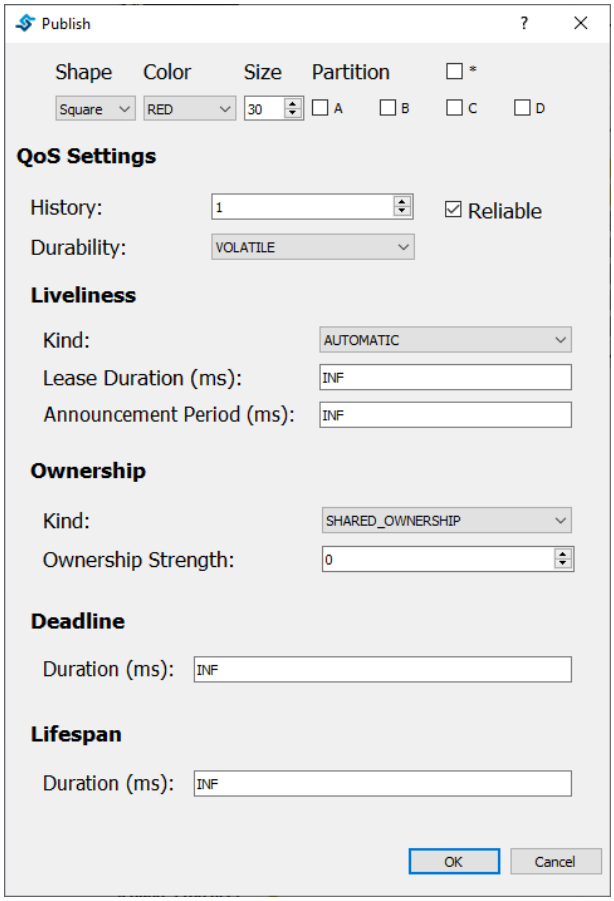

Fig. 2: Publisher QoS Settings

4. Final Thoughts

This research has presented the findings of one metric of network overhead incurred by the eProsima implementation of DDS at various stages of the common shapes demonstration. While this serves as a basic quantification to aid users in understanding their decision to use DDS, it should not be considered a complete or comprehensive quantification of overhead required of DDS as a standard. The methods used to collect and process these measurements of overhead are presented as a means to repeat the process on other DDS implementations or configurations. Not insignificantly, this research has not yet provided the complimentary quantification of the other side of the trade- the extra design cost incurred by opting not to use the simplified interfaces offered by DDS or its QoS. Such a quantification would likely be highly subjective and dependent on the programmer implementing the legacy communication mechanisms. Overall, DDS as a standard provides an agreed upon means to separate the work of designing and implementing a large distributed communication system and may be worth the price paid in overhead to glean access to standardized qualities of service.

Future enhancements could pursue controlling the subject

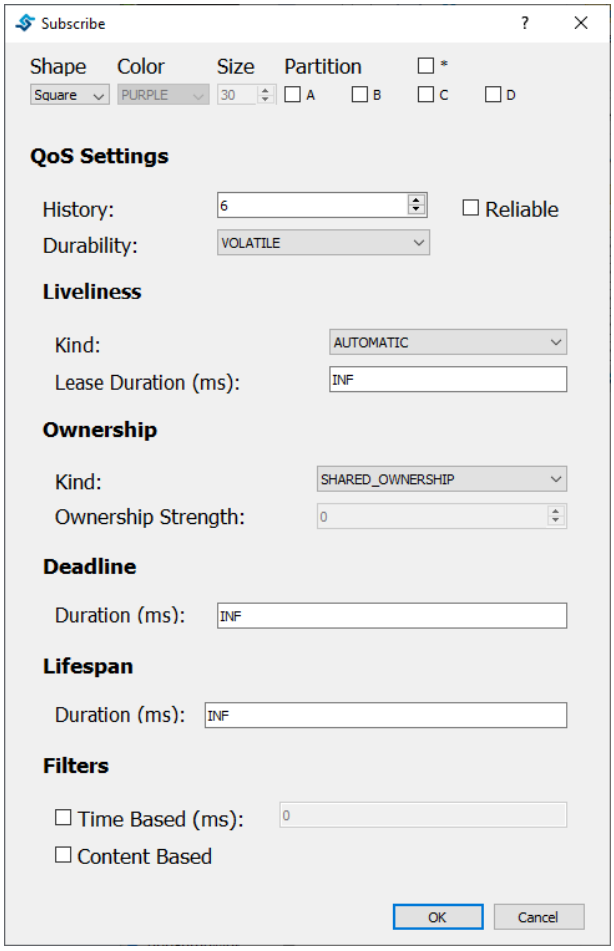

Fig. 3: Subscriber QoS Settings

application via application programming interface to enable a broader range of more consistent collections. Only a fraction of the possible QoS configurations and application settings from the test matrix was observed with a priority based only on author's intuition. A more formal targeting of key setting/configuration combinations could uncover more informative conclusions and offer a more conclusive characterization of the eProsima implementation's network overhead. To supplement those conclusions, a carefully designed legacy communication infrastructure should be tested as a comparison. Without that comparison, designers relying only on these results have only to decide whether their network has enough headroom to pay for the DDS QoS benefits.

5. Disclaimer

The views expressed in this paper are those of the authors and do not reflect the official policy or position of the United States Air Force, the Department of Defense, or the U.S. Government.

References

[1] J. Yang, K. Sandström, T. Nolte, and M. Behnam, "Data distribution service for industrial automation," in *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies Factory Automation (ETFA 2012)*, 2012, pp. 1–8.

[2] J. Yang, "Data distribution service for industrial automation," 2012. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:mdh:diva-15483

[3] B. Almadani, M. N. Bajwa, S.-H. Yang, and A.-W. A. Saif, "Performance evaluation of dds-based middleware over wireless channel for reconfigurable manufacturing systems," *International Journal of Distributed Sensor Networks*, vol. 11, no. 7, p. 863123, 2015.

[4] "eProsima shapes demo," accessed 10-May-2021. [Online]. Available: https://www.eprosima.com/index.php/products-all/eprosima-shapes-demo

[5] "DDS portal," accessed 10-May-2021. [Online]. Available: https://www.dds-foundation.org/

[6] "Wireshark," accessed 10-May-2021. [Online]. Available: https://www.wireshark.org

Author Biographies

NATHANIEL PECK is a Captain in the United States Space Force, currently stationed at Wright Patterson Air Force base. He is a master's student at the Air Force Institute of Technology studying software engineering for modeling and simulation solutions. He is a 2017 graduate of Rochester Institute of Technology, where he studied electrical engineering, robotics. His previous duty station was with AFRL's Aerospace Systems Directorate to develop next generation aircraft power systems.

DOUGLAS D. HODSON is an Associate Professor of Computer Engineering at the Air Force Institute of Technology (AFIT), Wright-Patterson AFB, Ohio USA. He received a B.S. in Physics from Wright State University in 1985, and both an M.S. in Electro-Optics in 1987 and an M.B.A. in 1999 from the University of Dayton. He completed his Ph.D. at the AFIT in 2009. His research interests include computer engineering, software engineering, real-time distributed simulation, and quantum communications. He is also a DAGSI scholar and a member of Tau Beta Pi.

MAJOR RICHARD DILL is an Assistant Professor of Computer Science at the Air Force Institute of Technology (AFIT), Wright-Patterson AFB, Ohio USA. He received a B.S. in Computer Science from the University of Maryland at College Park in 2004, and both an M.S. in Computer Science in 2008 and a PhD in 2018 from AFIT. Major Dill's research interests include computer security, algorithms, and artificial intelligence.

MICHAEL R. GRIMAILA (BSEE 1993; MSEE 1995; PhD 1999, Texas AM University) is a professor and head of the Department of Systems Engineering and Management at the Air Force Institute of Technology, Wright-Patterson Air Force Base in Ohio, USA. He is a member of Tau Beta Pi, Eta Kappa Nu, and the Association for Computing Machinery, as well as a Senior Member of the IEEE, and a Fellow of the Information System Security Association. He can be contacted via email at michael.grimaila@afit.edu.

IV. Paper III: DDS-C Network Overhead

The following paper, "Quantifying DDS-Cerberus Network Control Overhead," is expected to be submitted for publication. Contributions to the following work included establishing the surrogate performance measure using total packet traffic, post-processing collected data, and executing statistical modeling on data to provide conclusions on the significance of overhead for a security enhanced mode of operation relative to base operation without security measures.

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.

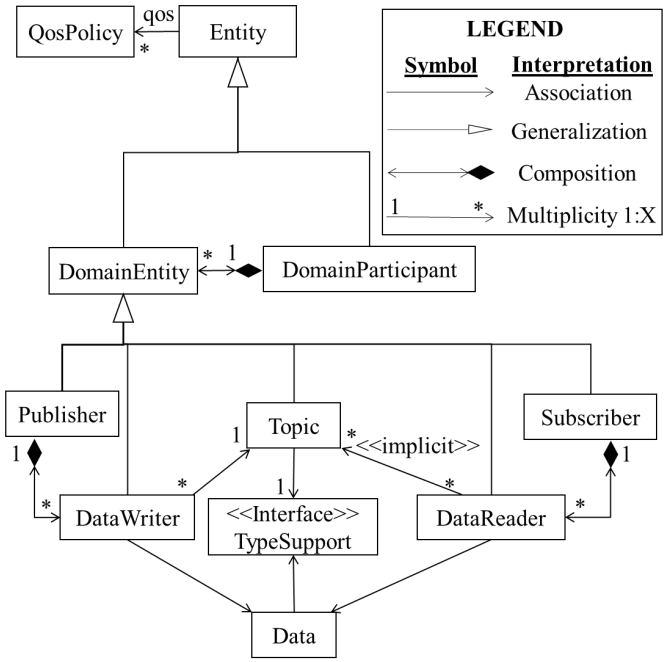

Fig. 1 Partial DDS Entity Model [12]

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:

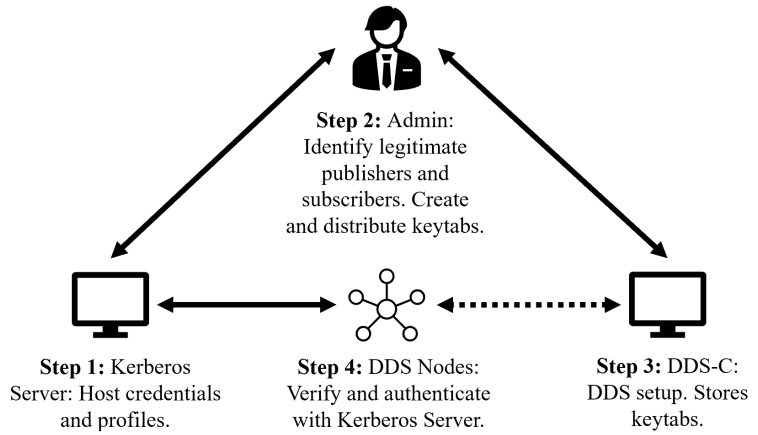

Fig. 2 DDS-C *Keytab* Process [10]

(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.

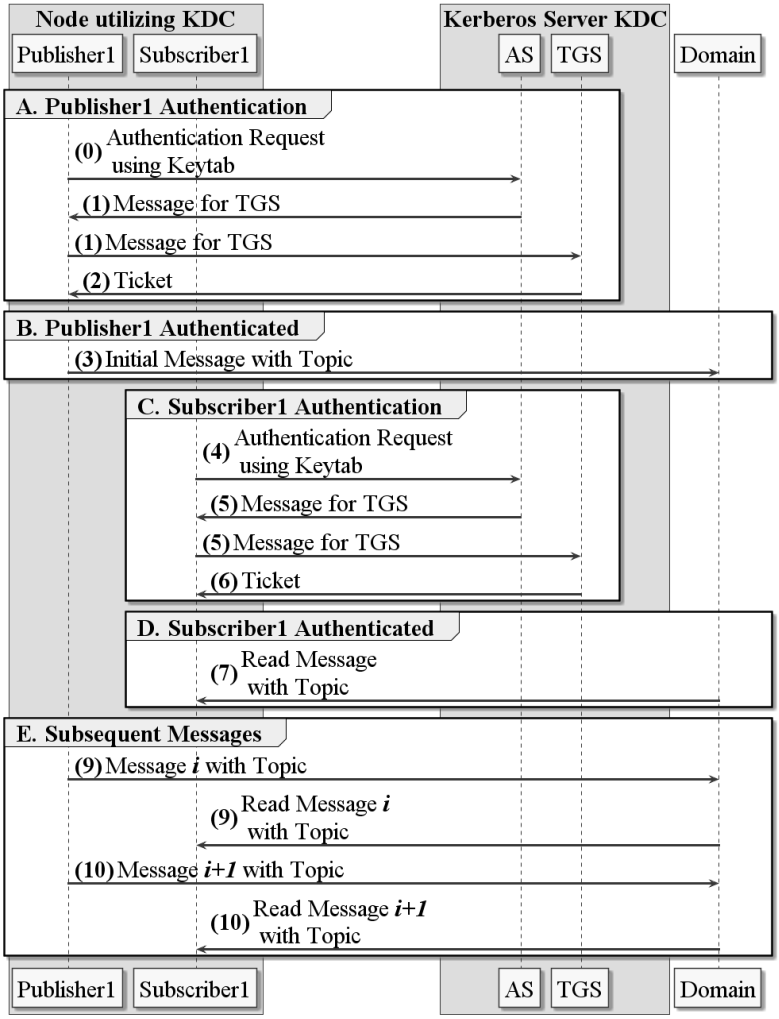

Fig. 3 DDS-C Authentication Process [10]

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.

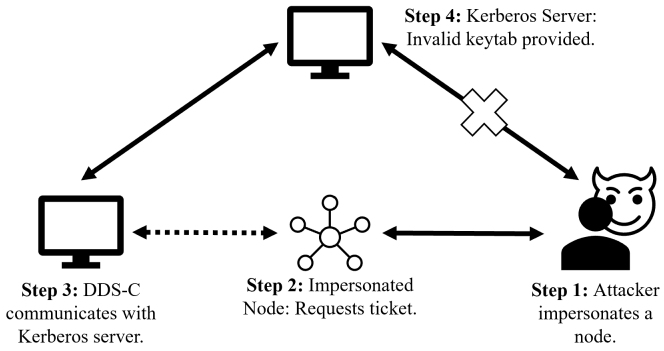

Fig. 4 DDS-C Mitigating Impersonation [10]

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 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*.

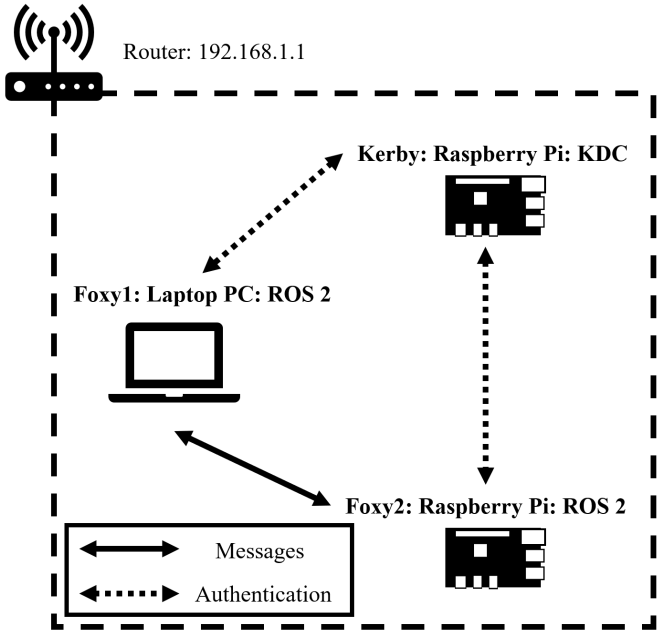

Fig. 5 Experiment Testbed Network Diagram

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.

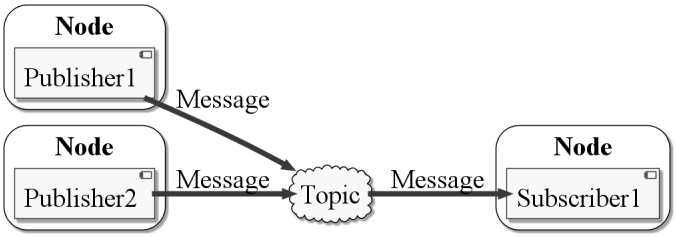

Fig. 6 Experiment Node Layout [10]

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.

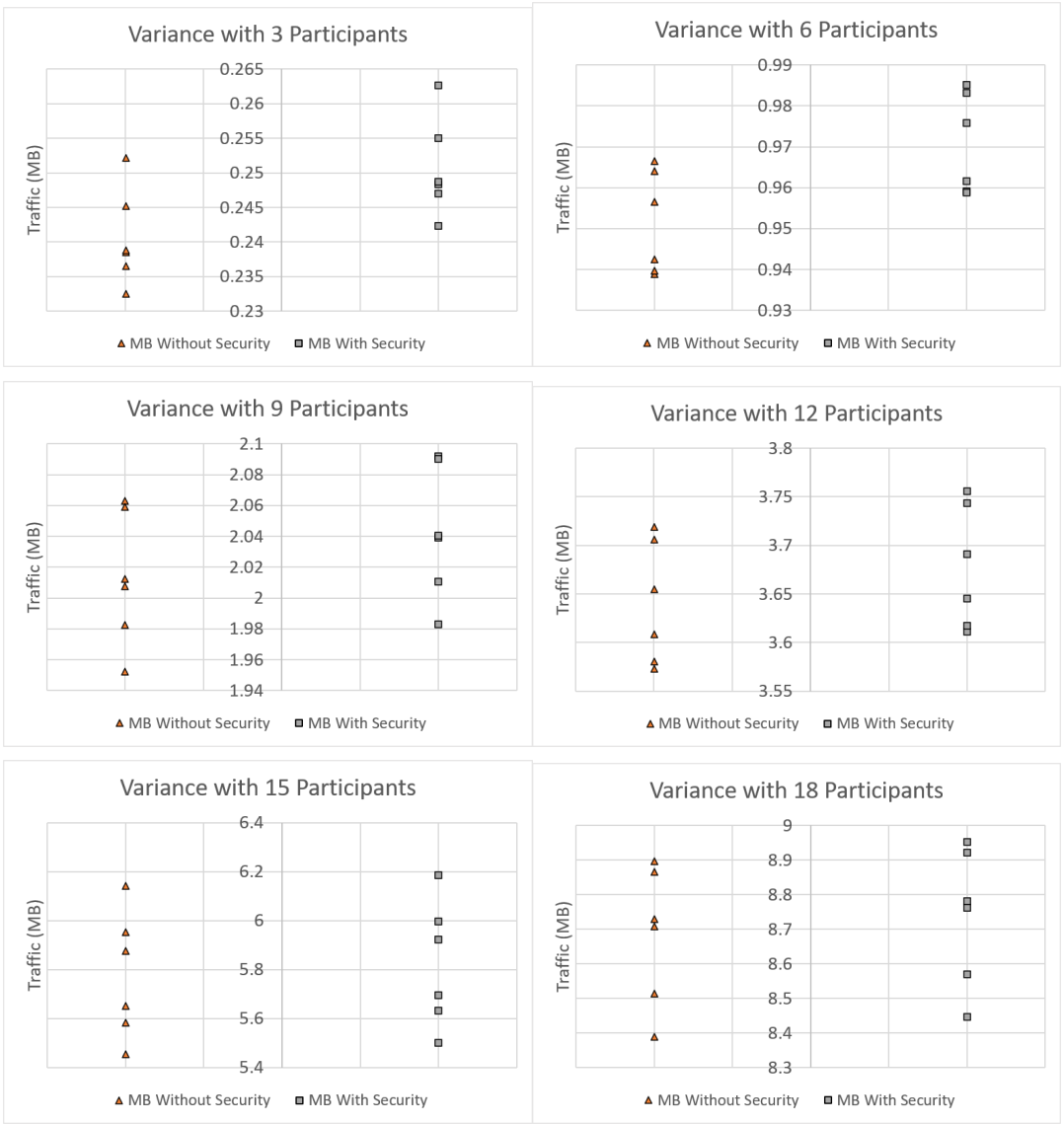

Fig. 7 Experiment Results. Top row: Best Effort Foxy1 with Kerby Variance 3 and 6. Middle row: Best Effort Foxy1 with Kerby Variance 9 and 12. Bottom row: Best Effort Foxy1 with Kerby Variance 15 and 18.

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 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.

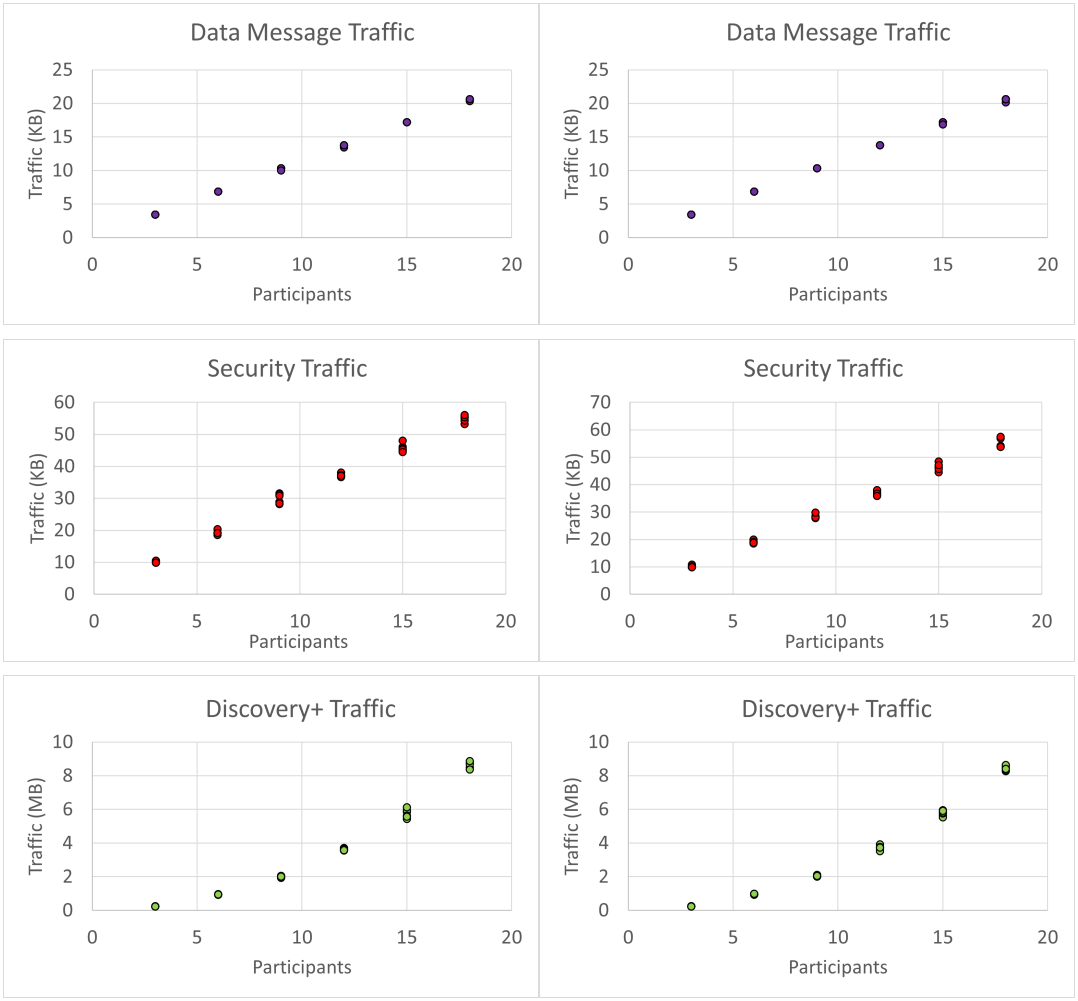

Fig. 8 Experiment Results. Left column: Best Effort Foxy1 with Kerby. Right column: Reliable Foxy1 with Kerby.

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

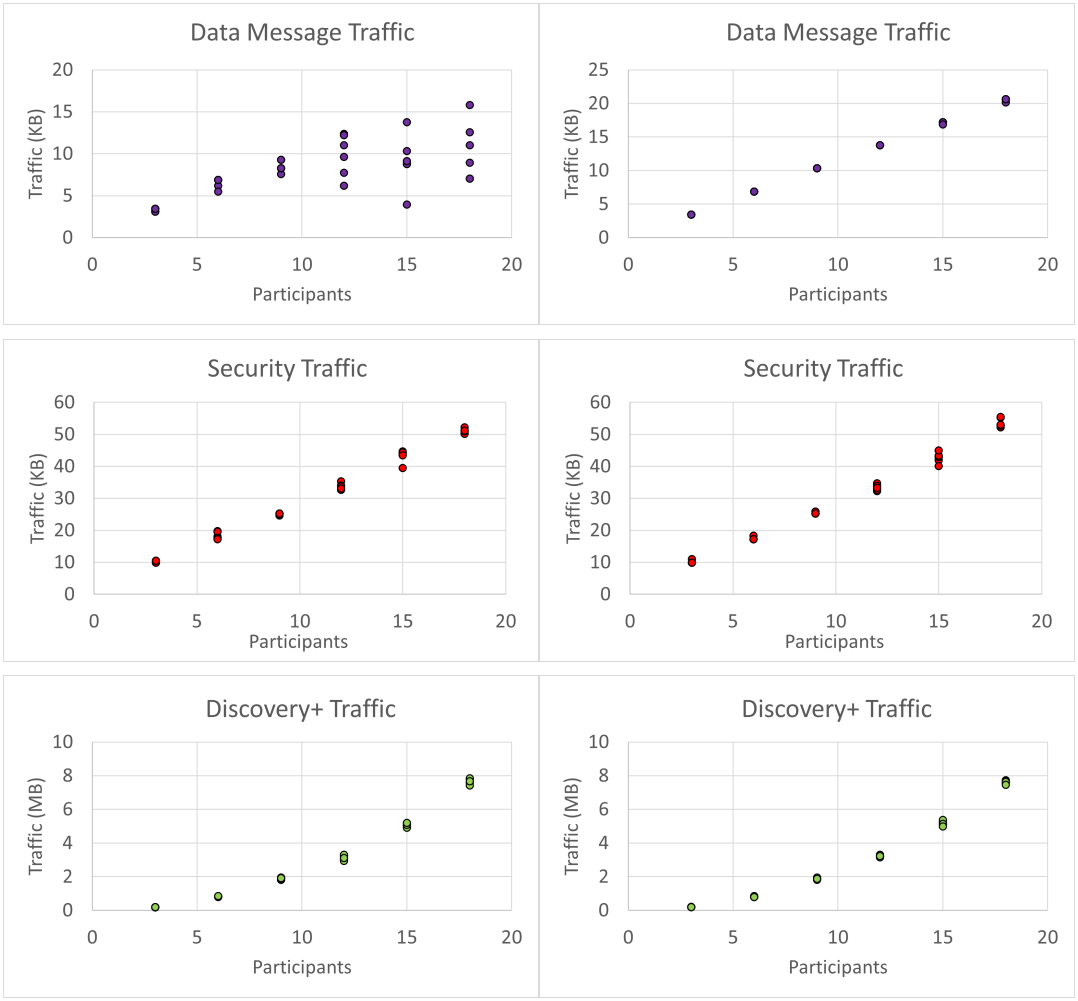

Fig. 9 Experiment Results. Left column: Best Effort Foxy1/Foxy2 with Kerby. Right column: Reliable Foxy1/Foxy2 with Kerby.

is insignificant for use cases involving anything more than small numbers of participants sharing a few messages of small size. These results indicate that DDS-C applied to other Data Distribution Service (DDS) implementations adds extra benefit without substantial performance costs. Understanding this information is crucial in applying DDS-C to future research.

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

Concerning analysis, future work could include regression tests to estimate model parameters for linear and 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/

V. Paper IV: Application of DDS-C

The following paper, "Distribution of DDS-Cerberus Authenticated Facial Recognition Streams," is expected to be submitted to a journal for publication. Contributions include repeated application of previously developed surrogate measure, post-processing methods, and statistical analysis techniques. Repeated use of the aforementioned contributions demonstrate the portability and generality of the approach for multiple applications.

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, reliability

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.

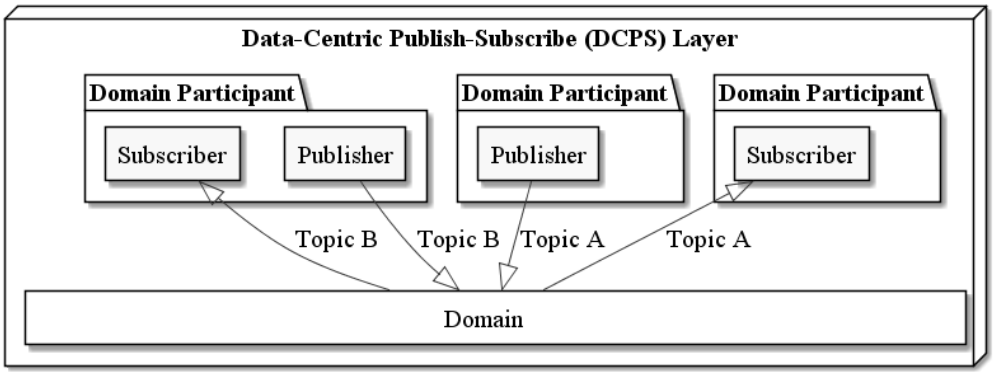

Fig. 1 DCPS Layout [8]

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

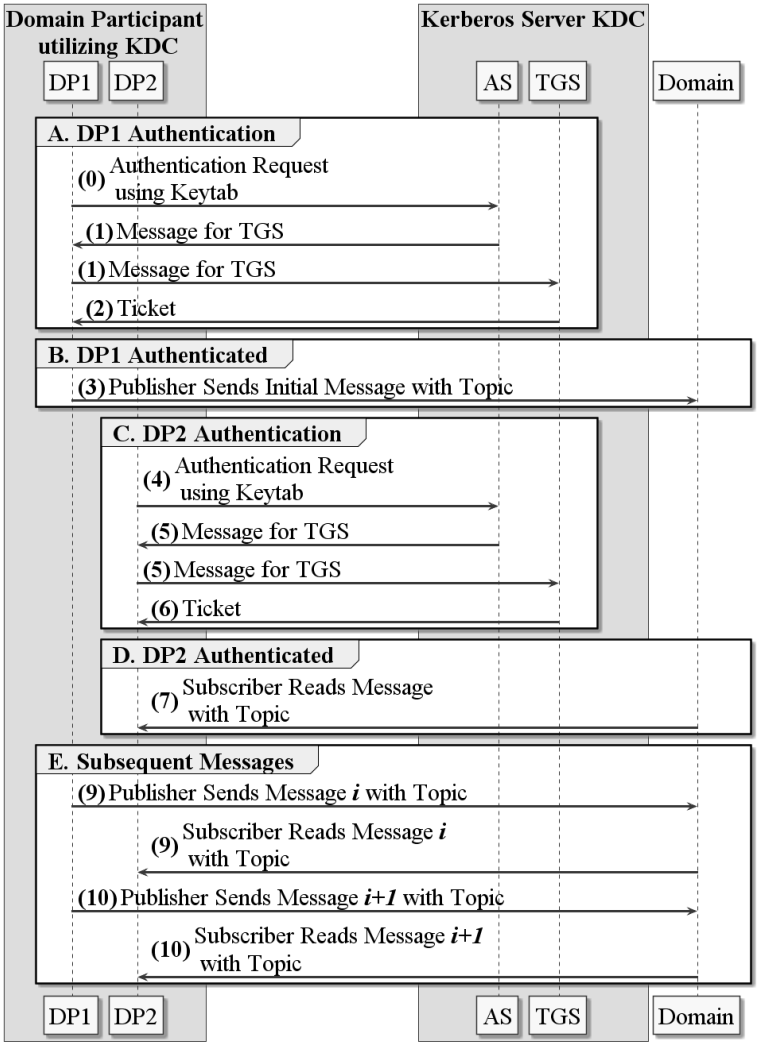

Fig. 2 DDS-C Authentication Process with *domain participants* [10]

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.

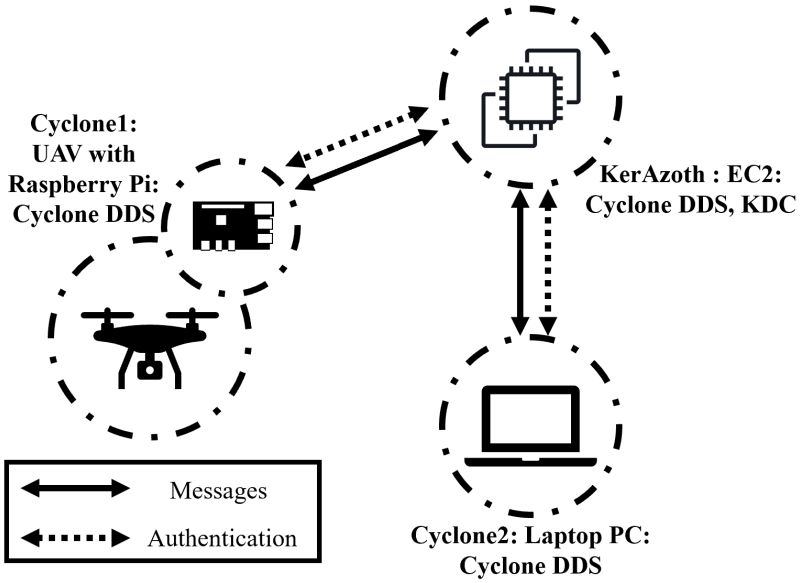

Fig. 3 Experiment Network Diagram

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

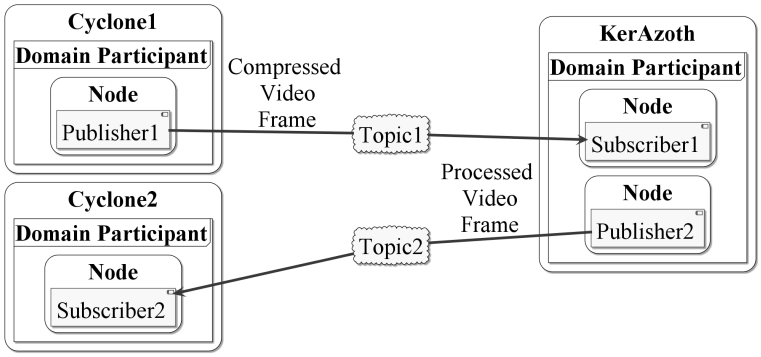

Fig. 4 Experiment Node Layout

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

send the processed video frame with a *topic* that only Cyclone2's subscriber uses.

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.

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

3.2 Assumptions and Limitations

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

- *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

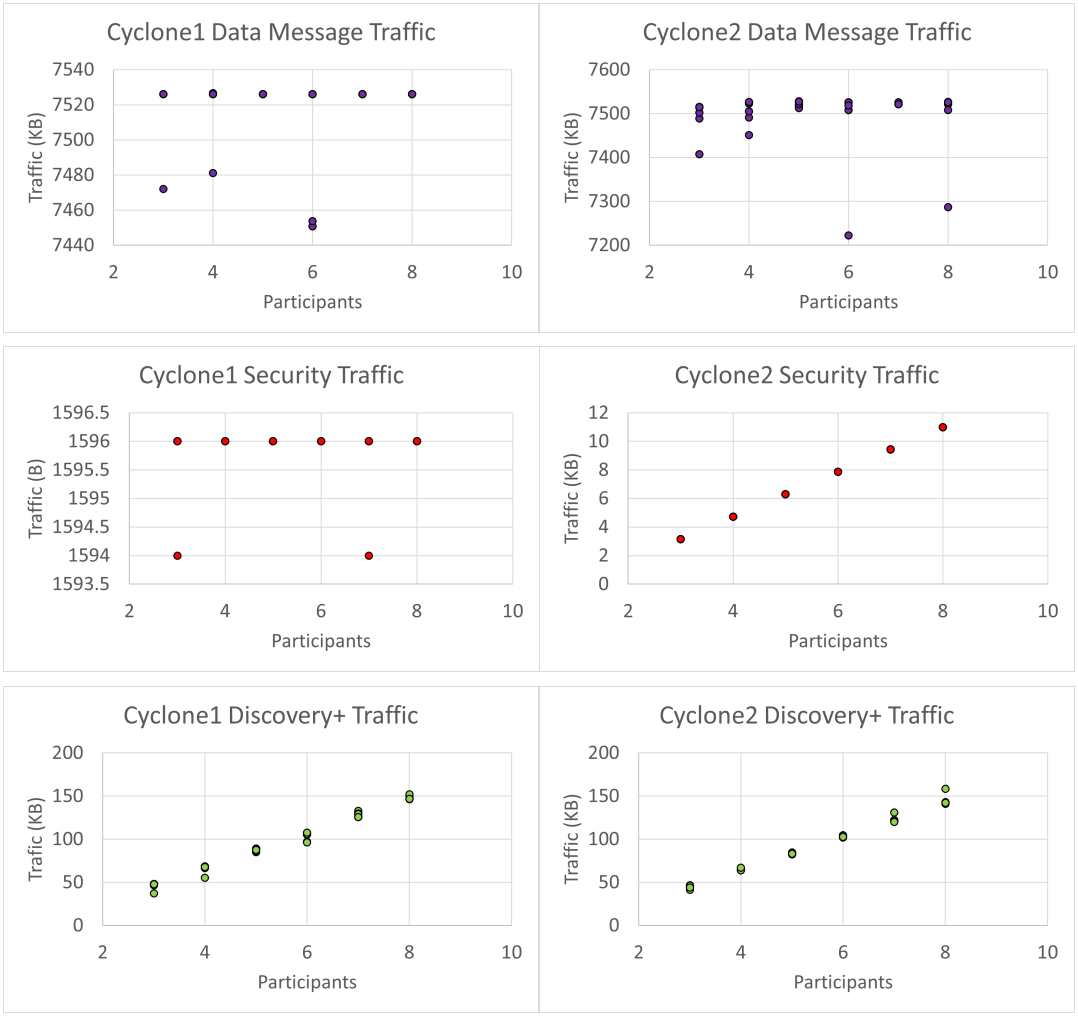

Fig. 5 Experiment results. Left column: Cyclone1. Right column: Cyclone2.

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 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.

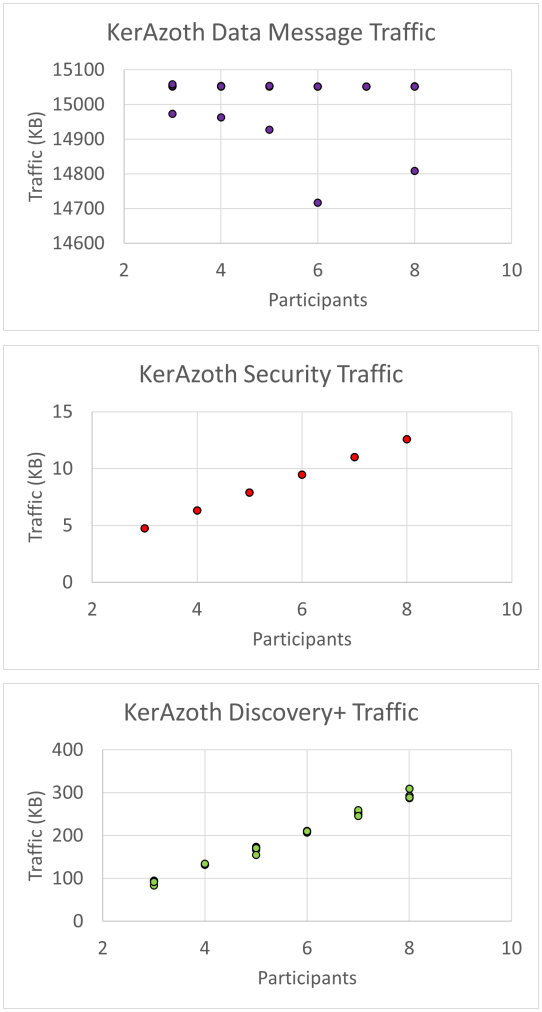

Fig. 6 Experiment results for KerAzoth.

4 Conclusion

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

VI. Paper V: DDS Distribution for DIS

The following paper, "Distributing DIS PDUs via UDP vs DDS," is expected to be submitted to a journal for publication. As noted in its future work section, the testing will be broadened to include additional QoS combinations and exploration for reducing discovery traffic.

Distributing DIS PDUs via UDP vs DDS

Nathaniel Peck, Douglas Hodson, Michael Grimaila, and Richard Dill
Air Force Institute of Technology, WPAFB, OH, USA
emails: 2012raptor@gmail.com, doug@sidechannel.net, michael.grimaila@afit.edu richard.dill@afit.edu

Abstract—*As computer programs and simulations scale in reach and capability, developers must ensure efficient employment of computing resources and segregation of complexity to avoid self-imposed scalability limitations. As improved technologies become available, legacy software systems must be updated or replaced to maintain competitive standing in the domain of their applications. The IEEE Distributed Interactive Simulation (DIS) standard defines a protocol and semantics for communication in large scale distributed simulations, which are widely used in the Department of Defense. The Object Management Group's (OMG's) Data Distribution Service (DDS) is a standard that seeks to simplify and expand capability in the domain of distributed communications. This research examines DDS capabilities, resource cost, and potential for scale against the benchmark of DIS today. Findings indicate that DDS offers a rich set of Quality of Service (QoS) configurations at a minimal cost which simplify the creation and operation of large scale DIS exercises, all while maintaining the primary specifications for DIS by IEEE 1278.1 - Application Protocols and introducing the configurability for greater compliance with IEEE 1278.2 - Communication Services and Profiles.*

Keywords: Distributed, Interactive, Simulation, Data, Distribution, Service, DIS, DDS, Network, Control, Overhead, Qualities, Service, QoS, Reliable, UDP

1. Introduction

Since the emergence of modeling and simulation (M&S), the Department of Defense has aspired to best utilize its advantages to acquire more advanced capabilities and provide more competitive training at reduced cost and risk and accelerated schedules. While early efforts to leverage M&S resulted in many isolated strategies, the need for efficient use of resources drove the formation and push for standards. One such standard is the IEEE Distributed Interactive Simulation (DIS). The standard defines a protocol and the semantics for how distributed simulation applications should interact with each other.

Initially, DIS offered a basic set of words or Protocol Data Units (PDUs) that cooperating simulations could send to each other. Over time, demand for increased fidelity led to the creation of additional PDUs. The growth in fidelity eventually illuminated a communication bottleneck requiring a change. The easiest and fastest way to send data was for all entities to broadcast all updates and interactions. While initially manageable, growth in traffic led to filtering. While careful filtering and multicast groups have helped ensure the right information gets to the right entities, the use of a publish subscribe communication pattern may help overcome one of the major survival challenges facing DIS.

Further, IEEE Standard 1278.2 specifies communication profiles that should be used for given PDUs. While certain PDUs should be delivered reliably, blanket use of UDP limits packet delivery to best effort. In clean network environments, best effort transport can achieve near reliable performance, but in larger simulations with congested networks packets can be dropped. For most DIS PDUs a periodically dropped packet does not cause significant disruption. In the case of interaction PDUs such as detonations, a dropped packet can significantly disrupt the outcome of a simulation and require correction. To overcome this challenge, a middleware abstraction of the transport reliability would enable seamless interchangeability between reliability levels of PDUs. DDS appears to offer a well suited solution of this type.

2. Background

This section provides a shallow dive into the basic elements of this research. Beginning with an introductory explanation of the concept of networking layers, this section provides a brief background in key protocols and QoS at the transport layer and above as well as an entry level background to DDS and DIS.

2.1 OSI Model

The concept of layers is central to understanding the comparison between UDP and DDS distribution. DDS does not compete at the same conceptual layer as UDP and therefore, does not present an either/or comparison. Rather, it exists at a higher layer of abstraction to underlying infrastructure which could include UDP. The Open Systems Interconnection (OSI) model, depicted in Figure 1, is a common conceptual framework used to divide the plethora of networking tasks into manageable layers. This division reduces the scope of knowledge required of individual engineers to cohesively integrate highly complex systems spanning multiple engineering disciplines. For example, an electrical engineer need only focus on defining the concept of a bit and offering an interface for interacting with bits to engineers at higher layers.

In the OSI model, the concept of a bit exists at the first and lowest layer- the Physical Layer. A total of 7 layers were

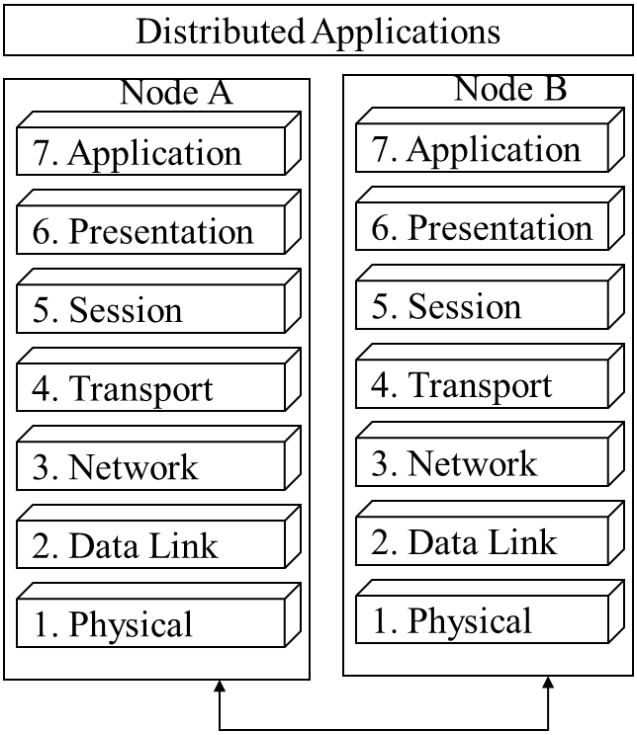

Fig. 1: OSI 7-Layer Model

defined, each introducing definitions of increasingly abstract concepts up to the application layer. The first layer deals with physical interconnection of distributed, communicating nodes. It defines the concept of a bit and offers interfaces for using bits up to the second layer. The Data Link layer resides above the physical layer and introduces the concepts of bit streams and frames being sent over a physical channel in finite sets limited by a Maximum Transmission Unit (MTU). The Network layer further abstracts the functions of the Data Link layer by introducing the concept of a packet. The IP protocol resides at the Network layer.

Above the Network layer lies the Transport layer. At this layer, packets may have types and are not all associated with data. For example, transport protocols such as TCP send acknowledgement type packets (ACK), which do not carry data. Rather, they carry metadata in support of layer functions or Qualities of Service (QoS) like reliability. Transport headers include machine port numbers which enable more granular communication. Beyond machine-to-machine, the presence of port numbers enables parallel communication between sub-processes on a single machine. The session layer provides the concept of a connection between two communicating nodes capable of sending and receiving messages which exceed a channel's MTU. This layer provides nodes with a service for accepting or rejecting messages that belong to a particular session or connection.

In DDS, the session layer's services are implemented by the publish/subscribe behavior and node discovery. At the presentation layer, structures are offered to applications and presented in a common data format. This is achieved via serialization. At the application layer, much of the verbiage and concerns of networking has been fully abstracted and applications share information using data types custom tailored to their purposes.

As noted by Kumar in 2014, the number of layers used was a balancing point [1]. While the addition of layers further subdivided responsibilities, too many layers can over inflate the processing overhead experienced by applications. For this reason, some applications have opted to only use up to layer 4 in their communication protocol stacks. In practice, protocols don't always have a one-to-one correlation with OSI layers. Some protocols span multiple layers. Middleware often spans layers 4 through 7 and above.

2.2 User Datagram Protocol

One of the most common transport layer protocols is the User Datagram Protocol (UDP). Its popularity, in part, stems from its simplicity. Packets are sent on a best-effort basis, offering minimal error detection and no correction in addition to that already provided by lower layers. UDP does not rely on the formation of a connection between communicating nodes. It is said to be connectionless. This fact simplifies the ability to send one packet to multiple recipients via broadcasting or multicasting.

Since UDP minimizes the added functionality, it is able to minimize the amount of overhead required by header fields, making it an extremely lightweight transport protocol option. Figure 2 shows the header structure for UDP datagrams. The structure displays the protocol's reliance on lower layers by only requiring port numbers rather than the address information already included by the underlying network protocol. The length field enables efficient use of network time and resources by allowing variable datagram sizes to be sent and received without relying on a fixed packet size and large padding. The unit of the length field is in bytes, meaning the maximum amount of padding, or wasted bits, is 7. The benefit of this field arises when considering the possible variance of datagrams. In the general sense, the variance of datagrams is likely to exceed 2 bytes. Therefore, the length field allows UDP to fit user needs without imposing a largely unused padding requirement. UDP presents a good example of a protocol embodying the principles of layering and fitting within a single conceptual layer.

Offsets	Octet	0								1							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	Source Port															
2	16	Destination Port															
4	32	Length															
6	48	Checksum															

Fig. 2: UDP Header Structure

2.3 Transmission Control Protocol

A direct contradiction to the use of UDP is the Transmission Control Protocol (TCP). TCP offers an alternative higher layer protocol that relies on an underlying network protocol. This reliance is revealed by the protocol's header structure in Figure 3. Similar to UDP, TCP headers do not require lower layer addressing because it must be provided by the underlying protocol. While TCP acts as an alternative to other transport layer protocols like UDP, it is not an apples-to-apples trade because TCP as a protocol combines functions from multiple OSI conceptual layers into one protocol, namely the session layer. TCP is a connection-oriented protocol and therefore makes broadcasting more difficult.

Offsets	Octet	0								1							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	Source Port															
2	16	Destination Port															
4	32	Sequence Number															
6	48																
8	64	Acknowledgement (ACK) Number															
10	80																
12	96	Data Offset				Reserved				Flags							
14	112	Window Size															
16	128	Checksum															
18	144	Urgent Pointer															
20	160	Options															
⋮	⋮																
62	496																

Fig. 3: TCP Header Structure

Transmission Control Protocol (TCP) offers a reliable quality of service (QoS). The reliability offered by TCP is essentially built into the protocol and does not allow for much tailoring beyond its strict definition of reliability. This strict interpretation of reliability causes the protocol to continuously attempt to send packets until all sent information is at least received. In the default application of the protocol, sent packets are also required to be received in order such that ordering of delivery is guaranteed. The options in TCP headers allow for a slightly relaxed interpretation of reliability in which packets may be selectively acknowledged. This option enables greater efficiency in the communication channel because packets sent after a dropped packet may not have to be resent. However, even this relaxed interpretation of reliability is still relatively strict because all sent packets are delivered even if by the time of delivery the information is outdated or irrelevant. In many cases (e.g., interaction with websites, email exchange, etc.) strict reliability is a desired feature. In others, it is an overly burdensome interpretation of reliability. An expansion of the reliability concept is deferred to the next section.

2.4 Reliability

Although it has been widely employed, the topic of reliability is still a matter of exploration. Some applications require more strict interpretation of reliability than others. For example, a user would demand strict reliability of a banking application, particularly in the form of guaranteed delivery and sequenced delivery. If a user makes a deposit, then makes a purchase using those funds and the order is flipped at the processing center, the user may be wrongly subject to overdraft fees. On the other hand, the distributed simulation and gaming communities requires a more flexible interpretation and heterogeneous application of reliability. Strict reliability may actually degrade performance relative to a more relaxed interpretation due to increased latency.

In the case of visual representation of entities in virtual worlds, state is usually sent periodically, and in moments more current information is sent again. In the case of interactions between entities, a single interaction may be all that is sent from one entity to another. This information should be sent with a greater guarantee of delivery because without it, the outcome of an interaction could be affected.

Perhaps one of the central details to levels of reliability is when to require or allow a packet to be resent. The ability to offer reliability hinges on an application's ability to store state on the information being sent reliably. If a distributed simulation or networked game allows users to join late and receive earlier state from the shared virtual environment, developers may need to consume memory to store history. The freedom to choose how many states or how much history is one method of allowing scalable interpretations of reliability.

Another point of interpretation for reliability is assumptions regarding acknowledgement. A sender may assume a packet to be dropped unless it receives a positive acknowledgement (ACK) from a receiver. Alternatively, the sender could assume a packet to have been delivered unless it receives a negative acknowledgement from a perspective receiver. Each method has its pros and cons. In an inherently reliable network, packets may be infrequently dropped, leading to an overabundance of positive ACK packets congesting the network. Alternatively, a negative ACK system requires an alternative means of notifying listeners of what sequence numbers to expect. This is commonly termed a heartbeat. By reducing the ratio of heartbeat messages to data transmissions, networks can be less congested at the cost of reduced response rates to dropped packets.

Not all data should be sent the same way. Glenn Fiedler details the pitfall of assuming the ability to use two separate transport mechanisms in his short article on gaming protocols [2]. In short, the downside stems from the negative impact TCP has on UDP traffic. Still, the ability to have heterogeneous reliability QoS would be desirable. This is where a higher layer protocol could rely on UDP as a transport protocol and build a reliability QoS above it. This

is the case with the Real-Time Publish Subscribe Protocol (RTPS).

2.5 Real-Time Publish Subscribe Protocol

The central documentation for the Real-Time Publish Subscribe (RTPS) protocol exists in an OMG Standard for DDS Interoperability, DDSI-RTPS [3]. It was established as an interoperability wire protocol for DDS. RTPS is a higher layer communication protocol that relies on an underlying transport protocol. This fact is evident from the RTPS headers, displayed in Figure 4. The headers lack any source port or destination port information because they must be provided by the underlying transport. As such, the transport is specified in the standard as intended to be UDP. Certain vendors like eProsima have enabled switching between transport options such as UDP, TCP, or other user transports.

Offsets	Octet	0								1							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	"RTPS"															
2	16																
4	32	Protocol Version (Major)								Protocol Version (Minor)							
6	48	Vendor ID															
8	64	GUID Prefix															
⋮	⋮																
18	144																
20	160	Submessages															
⋮	⋮																
MTU - N - v																	

Fig. 4: RTPS Header Structure

As shown in Figure 4, RTPS headers are not the only overhead paid to send a payload. Rather, a payload is contained in a DATA submessage which has its own structure. Using RTPS, a data payload is not simply sent using a DATA submessage, but it requires a timestamp submessage with each DATA submessage. The sizes of these submessages are shown in Figure 5. If UDP over IP is used as the underlying network stack on an Ethernet network, Figures 2, 4, and 5 can be used to calculate the minimum overhead required to send an RTPS payload. The IP, UDP, and RTPS contribute 34, 8, and 56 bytes of headers, respectively, totaling 98 bytes of header overhead for each RTPS payload sent.

2.6 Data Distribution Service

DDS is a middleware standard for an abstraction layer above RTPS. Its main describing document is an OMG standard [4]. While RTPS layer entities include readers and writers, DDS abstracts these classes with publishers and subscribers. Various works have been done to evaluate the performance of DDS compared to other means. Most use measures of latency, throughput, and jitter. In these regards,

Offsets	Octet	0								1							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	INFO_TS Submessage (Timestamp)															
⋮	⋮																
10	80																
12	96	DATA Submessage Headers															
⋮	⋮																
34	272																

Fig. 5: RTPS Minimum Submessage Header Structure

DDS has been shown to provide comparable performance to other means in the domain. In 2012, Yang et al. mapped DDS entities to IEC 61499 and compared DDS performance to that of raw sockets [5].

2.7 Distributed Interactive Simulation

Distributed Interactive Simulation (DIS) in the general sense is the connection of a set of individual simulations to create a common virtual environment (i.e., world, situation) in which multiple hosted entities can interact. DIS is an overloaded concept; for example, these systems are also referred to as Distributed Virtual Environments (DVE) or Networked Virtual Environments (NVE). IEEE Std 1278 for DIS defines a specific protocol for how to construct and manage such a geographically distributed interactive system. The standard exists in 4 separate documents covering different pieces of the whole. The four documents comprising the standard are as follows, of which the first and second are the most important for this research.

(1) IEEE-Std-1278.1 - IEEE Standard for Distributed Interactive Simulation - Application Protocols [6]
(2) IEEE-Std-1278.2 - IEEE Standard for Distributed Interactive Simulation - Communication Services and Profiles [7]
(3) IEEE-Std-1278.3 - IEEE Recommended Practice for Distributed Interactive Simulation - Exercise Management and Feedback
(4) IEEE-Std-1278.4 - IEEE Trial-Use Recommended Practice for Distributed Interactive Simulation - Verification, Validation, and Accreditation

DIS currently specifies the data in terms of Protocol Data Units (PDUs) and the semantics for how each individual simulation should (act or react) transmit and receive such data items. In 1996 all PDUs were broadcast to all entities. [8] Various methods of managing network traffic were suggested, including DIS Lite and a publish subscribe pattern through multicast traffic management. In 2010, researchers compared the performance of HLA to that of DDS to send DIS data [9].

3. Trade Space

Application developers usually must consider the desires of their intended customers. Unfortunately, meeting these desires usually imposes conflicting requirements. For example, coding techniques which produce faster performance or lightweight products often conflict with techniques aimed at producing robust performance or feature-rich products. This trade-off is apparent in the comparison between UDP and TCP. While these these two protocols seem to offer two levels of trade space configurations at the transport layer, DDS at the application layer enables many more levels to the trade-off.

4. Test Setup

To minimize human error in results, a computer automated test routine was employed. Capture of transmitted data using Wireshark required the use of at least two separate computing environments. Without separate environments, DDS discovery caused the communicating endpoints to use shared memory rather than remote transmission protocols. In order to control two separate computing environments, one environment served as a master and the other a slave. The slave received commands from the master via a dedicated TCP control socket. This TCP control stream existed on port 1234 and was filtered from Wireshark collections. A Powershell script was used on both master and slave machines to send and interpret the control commands.

5. Results

The test system described in Section 4 was used to send 100 payloads of 4 bytes using all four configurations listed in Table 1. The total amount of traffic in bytes observed over the entire collection is reported. Of note, the total quantity of data sent in each case was 400 bytes. Figure 6 provides a graphical comparison of the cost of reliability when implemented at the transport layer versus the application layer using DDS. Notably, the price of reliability in overhead is less when paid at the application layer using DDS compared to the transport layer using TCP.

Table 1: Observed Traffic by Configuration

Layer of Implementation	Reliability	Total Bytes on Wire
Transport	Best Effort	6000
	Reliable	10618
DDS	Best Effort	12200
	Reliable	15400

6. Discussion

While applications communicating using a maximum of UDP, such as plain DIS, are simple to troubleshoot because each packet corresponds to a single message transmission of data, introduction of middleware overhead reduces the likelihood of needing to troubleshoot by preemptively sending the required information for the applications to use for error corrections and re-transmissions where necessary.

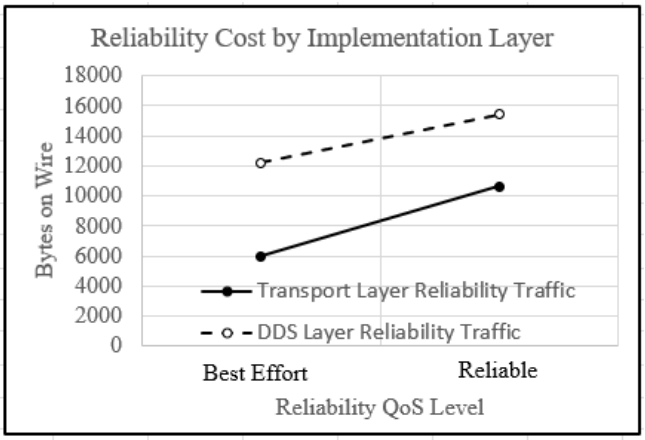

Fig. 6: Plotted Effects from Table 1

7. Conclusions

RTPS is a very capable wire protocol provided by DDS which could help bring DIS simulations into greater compliance with IEEE 1278.2 via reliability QoS. The protocol requires minimal overhead which still provides ample payload capacity in most common MTU sizes to accommodate DIS PDUs without fragmentation. However, a custom middleware designed for DIS could more efficiently utilize channel bandwidth by avoiding redundancies and more appropriately sizing fields. For example, the Entity State PDU only requires a timestamp field with 4 bytes, but RTPS requires each packet to contain a timestamp field with 12 bytes.

8. Future Work

Future work could include broader exploration of additional QoS offered by DDS. Additionally, a wider range of payload sizes could be used to run a mock simulation using payloads mimicking the sizes of DIS PDUs. Further, the total amount of overhead could be further reduced by fine tuning the discovery phase and matching. In traditional DIS simulations, participants are organized prior to simulation. This could enable static endpoint discovery in lieu of dynamic discovery to save overhead traffic.

9. Disclaimer

The views expressed in this paper are those of the authors and do not reflect the official policy or position of the United States Air Force, the Department of Defense, or the U.S. Government.

References

[1] S. Kumar, S. Dalal, and V. Dixit, "The osi model: Overview on the seven layers of computer networks," *International Journal of Computer Science and Information Technology Research*, vol. 2, no. 3, pp. 461–466, 2014.

[2] G. Fiedler, "UDP vs. TCP," accessed 8-December-2021. [Online]. Available: https://gafferongames.com/post/udp$_v s_t cp$/

[3] "DDSI-RTPS DDS interoperability wire protocol," accessed 26-October-2021. [Online]. Available: https://www.omg.org/spec/DDSI-RTPS/2.3/About-DDSI-RTPS/

[4] *OMG Data Distribution Service (DDS)*, Object Magagement Group, 4 2015, version 1.4.

[5] J. Yang, K. Sandström, T. Nolte, and M. Behnam, "Data distribution service for industrial automation," in *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies Factory Automation (ETFA 2012)*, 2012, pp. 1–8.

[6] "IEEE standard for distributed interactive simulation - application protocols," accessed 26-October-2021. [Online]. Available: https://standards.ieee.org/standard/1278-1-2012.html

[7] "IEEE standard for distributed interactive simulation - communication services and profiles," accessed 26-October-2021. [Online]. Available: https://standards.ieee.org/standard/1278-2-2015.html

[8] D. A. Fullford, "Distributed interactive simulation: its past, present, and future," in *Proceedings of the 28th conference on Winter simulation*, 1996, pp. 179–185.

[9] A. Hakiri, P. Berthou, and T. Gayraud, "Addressing the challenge of distributed interactive simulation with data distribution service," *arXiv preprint arXiv:1008.3759*, 2010.

[10] M. A. Bassiouni, M.-H. Chiu, M. Loper, M. Garnsey, and J. Williams, "Performance and reliability analysis of relevance filtering for scalable distributed interactive simulation," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 7, no. 3, pp. 293–331, 1997.

Author Biographies

NATHANIEL PECK is a Captain in the United States Space Force, currently stationed at Wright Patterson Air Force base. He is a master's student at the Air Force Institute of Technology studying software engineering in modeling and simulation solutions. He is a 2017 graduate of Rochester Institute of Technology, where he studied electrical engineering, robotics. His previous duty station was with AFRL's Aerospace Systems Directorate to develop next generation aircraft power systems.

DOUGLAS D. HODSON is an Associate Professor of Computer Engineering at the Air Force Institute of Technology (AFIT), Wright-Patterson AFB, Ohio USA. He received a B.S. in Physics from Wright State University in 1985, and both an M.S. in Electro-Optics in 1987 and an M.B.A. in 1999 from the University of Dayton. He completed his Ph.D. at the AFIT in 2009. His research interests include computer engineering, software engineering, real-time distributed simulation, and quantum communications. He is also a DAGSI scholar and a member of Tau Beta Pi.

MAJOR RICHARD DILL is an Assistant Professor of Computer Science at the Air Force Institute of Technology (AFIT), Wright-Patterson AFB, Ohio USA. He received a B.S. in Computer Science from the University of Maryland at College Park in 2004, and both an M.S. in Computer Science in 2008 and a PhD in 2018 from AFIT. Major Dill's research interests include computer security, algorithms, and artificial intelligence.

MICHAEL R. GRIMAILA (BSEE 1993; MSEE 1995; PhD 1999, Texas A&M University) is a professor and head of the Department of Systems Engineering and Management at the Air Force Institute of Technology, Wright-Patterson Air Force Base in Ohio, USA. He is a member of Tau Beta Pi, Eta Kappa Nu, and the Association for Computing Machinery, as well as a Senior Member of the IEEE, and a Fellow of the Information System Security Association. He can be contacted via email at michael.grimaila@afit.edu.

VII. Conclusions

Although there are definite pros to adopting DDS as a distribution service for DIS data, there are still cons to be either accepted or addressed. First, DDS is not the only higher layer protocol that would serve as a candidate for distributing DIS data. Second, DDS operation can be further tailored to maximize efficiency of the network resources. Until additional work could confirm or create a most optimal distribution service, DDS makes a suitable candidate for DIS data in the interim. DIS packets are already sized well below common MTUs to avoid fragmentation. The cost of the DDS headers does not sufficiently reduce the payload available to require fragmentation for DIS PDUs.

7.1 Future Work

Publications in this thesis provide an entry into a much deeper and ever progressing world of data distribution. There are many possible avenues that remain to be explored in this domain. Below are a few examples.

- The work in Chapter VI can be continued as noted within.
- Real simulations involving DIS can be observed for practical use examples and suggestions.
- Other alternatives to DDS can be explored and compared.
- Custom middleware protocols could be based on RTPS to provide more efficient use of network resources. As UDP relies on IP layer information, a custom protocol rely on data already required in DIS PDUs.

Appendix A. Powershell Sockets Module

```
$default_IP = [IPAddress]::Loopback


########################
####  Section: UDP  ####
########################
$(
# Example:
# Listening machine
## £> ipconfig (gather listener IPv4 Address
## £> import this module
## £> Start-Listen (optional) -Port #### # Defaults to 3000
# Sending machine
## £> import this module
## £> Send-UdpDatagram -EndPoint {Address}
##                        -Port [3000](or optional other)
##                        -Message "Hello!"
# Send-UDP -IP 192.168.188.155 -Port 8888 -Send "Are we having fun yet?"
function Send-UDP
{
    Param (
        [ValidateSet("Text","Bytes")]$Mode="Text",
        $Send='Sample Text',$IP=$default_IP,
        $Port=3000,
        $existing_handle,
        [switch]$return_handle
```

```
    )
    if( -not $existing_handle)
    {
        $existing_handle = New-Object psobject
        $endpoint = New-Object System.Net.IPEndPoint ([IPAddress]$IP,$Port)
        $existing_handle | Add-Member "Endpoint" $endpoint
        $client= New-Object System.Net.Sockets.UdpClient
        $existing_handle | Add-Member "Client" $client
    }
    if($Mode -ceq "Text")
    {
        $Send=[Text.Encoding]::ASCII.GetBytes($Send)
    }
    $bytesSent=$existing_handle.Client.Send($Send,
                                            $Send.length,
                                            $existing_handle.Endpoint)
    if( -not $return_handle )
    {
        $existing_handle.Client.Close()
    }else{
        return $existing_handle
    }
}Export-ModuleMember -Function Send-UDP

function Receive-UDP
{
```

```
Param (
    [Parameter(Mandatory=$false, Position=0)]
    [ValidateNotNullOrEmpty()]
    [int] $Port = 3000,
    [Parameter(Mandatory=$true, Position=1)]
    [ValidateSet("Message_Quantity",
                 "Single_Message",
                 "Until_Message_Equals")]
    [string]$Stop_Condition,
    $Value
)
if($($Stop_Condition -cne "Single_Message") -and $( -not $Value ) )
{
    Write-Host "Error! You specified Stop_Condition=",$Stop_Condition,
                " but failed to specify value..."
    $Value = Read-Host -Prompt $("Enter value: "+$Stop_Condition+"=")
}
$endpoint = New-Object System.Net.IPEndPoint ([IPAddress]::Any,$Port)
$client= New-Object System.Net.Sockets.UdpClient $Port
$message_quantity = 0
[bool]$stop_condition_met = $false
while ( -not $stop_condition_met )
{
    $content=$client.Receive([ref]$endpoint)
    $result = [Text.Encoding]::ASCII.GetString($content)
    Write-Host $result
```

```
        if ($Stop_Condition -ceq "Single_Message")
        {
            $stop_condition_met = $true
        }
        if ($Stop_Condition -ceq "Until_Message_Equals")
        {
            if($result -ceq $Value){ $stop_condition_met = $true }
        }
        $message_quantity++
        if ($Stop_Condition -ceq "Message_Quantity")
        {
            if($message_quantity -ge $Value){ $stop_condition_met = $true }
        }
    }
    $client.Close()
    $client.Dispose()
}Export-ModuleMember -Function Receive-UDP
)

#########################
####  Section: TCP  ####
#########################
$(

function Send-TCP
{
```

```
Param(
    [ValidateSet("Text","Bytes")]$Mode="Text",
    $Send='Sample Text',
    $IP=$default_IP,
    $Port=2500,
    $existing_handle,
    [switch]$return_handle
)
if( -not $existing_handle)
{
    $endpoint = New-Object System.Net.IPEndPoint ([IPAddress]$IP,$Port)
    $existing_handle = New-Object psobject
    $client = New-Object System.Net.Sockets.TcpClient
    $existing_handle | Add-Member "Client" $client
    try{
        $existing_handle.Client.Connect($endpoint)
    }catch{
        Write-Host "Something went wrong connecting to client."
    }
    $stream = $existing_handle.Client.GetStream()
    $existing_handle | Add-Member "stream" $stream
}
if($Mode -ceq "Text")
{
    $Send=[Text.Encoding]::ASCII.GetBytes($Send)
}
```

```
    $existing_handle.stream.Write($Send,0,$Send.length)
    $existing_handle.stream.Flush()
    if( -not $return_handle )
    {
        $existing_handle.stream.Close()
        $existing_handle.Client.Close()
    }else{
        return $existing_handle
    }
}Export-ModuleMember -Function Send-TCP

Function Receive-TCP {
    Param (
        [Parameter(Mandatory=$true, Position=0)]
        [ValidateNotNullOrEmpty()]
        [int] $Port,
        [Parameter(Mandatory=$true, Position=1)]
        [ValidateSet(
            "Message_Quantity",
            "Single_Message",
            "Until_Message_Equals")]
        [string]$Stop_Condition,
        [string]$reactive_command_interpreter_module_path,
        $Value
    )
    Import-Module $reactive_command_interpreter_module_path -Force
```

```
if($($Stop_Condition -cne "Single_Message") -and $( -not $Value ) )
{
    Write-Host "Error! You specified Stop_Condition=",
               $Stop_Condition,
               " but failed to specify value..."
    $Value = Read-Host -Prompt $("Enter value: "+$Stop_Condition+"=")
}
Try{
    # Set up endpoint and start listening
    $endpoint = new-object System.Net.IPEndPoint([ipaddress]::any,$port)
    $listener = new-object System.Net.Sockets.TcpListener $EndPoint
    $listener.start()

    # Wait for an incoming connection, return a TcpClient upon accept.
    $Client = $listener.AcceptTcpClient()

    # Stream setup
    $stream = $Client.GetStream()

    [bool]$stop_condition_met = $false
    [int]$message_quantity = 0
    $bytes = New-Object System.Byte[] 1024

    while ( $( -not $stop_condition_met ) -and
            $( ($i = $stream.Read($bytes, 0, $bytes.Length) ) -ne 0)
          )
```

```
{
    $EncodedText = New-Object System.Text.ASCIIEncoding
    $data = $EncodedText.GetString($bytes,0, $i)
    Write-Host "Message=",$data
    execute-command -command $data
    if ($Stop_Condition -ceq "Single_Message")
    {
        $stop_condition_met = $true
    }
    if ($Stop_Condition -ceq "Until_Message_Equals")
    {
        if($data -ceq $Value){ $stop_condition_met = $true }
    }
    $message_quantity++
    if ($Stop_Condition -ceq "Message_Quantity")
    {
        if($message_quantity -ge $Value){ $stop_condition_met = $true }
    }
}
if( -not $stop_condition_met )
{
    Write-Host "Detected socket closure by sender!"
}

# Close TCP connection and stop listening
Write-Host "Closing connection objects..."
```

```
        $stream.close()
        Write-Host "Successfully closed stream. Closing listener..."
        $listener.stop()
        Write-Host "Successfully closed listener."
        if ($Stop_Condition -ceq "Seconds_Open"){ $stopwatch.Stop() }
    }
    Catch {
        "Receive Message failed with: `n" + $Error[0]
    }
}Export-ModuleMember -Function Receive-TCP
)
```

Appendix B. Powershell Slave Command Interpreter Module

```
# This module is to be used on a slave machine
# Set the master machine's IP address below for responses from the interpreter.
$master_machine_ip = "192.168.0.1"
$master_machine_listen_port = 2500
function execute-command
{
    param([string]$command)
    Switch ($command)
    {
        "Start Reliable Publisher" {
            $cmd = {
                cmd /c start powershell -Command {
                    $host.ui.RawUI.WindowTitle = "Unique Title"
                    # Call the compiled executable using path on machine.
                }
            }
            Start-Job -ScriptBlock $cmd | Out-Null
            Break
        }
        "Start Unreliable Publisher" {
            $cmd = {
                cmd /c start powershell -Command {
                    $host.ui.RawUI.WindowTitle = "Unique Title"
                    # Call the compiled executable using path on machine.
```

```
        }
        Start-Job -ScriptBlock $cmd | Out-Null
        Break
    }
    "Start Unreliable Subscriber" {
        $cmd = {
            cmd /c start powershell -Command {
                $host.ui.RawUI.WindowTitle = "Unique Title"
                # Call the compiled executable using path on machine.
            }
        }
        Start-Job -ScriptBlock $cmd | Out-Null
        Break
    }
    "Start Reliable Subscriber" {
        $cmd = {
            cmd /c start powershell -Command {
                $host.ui.RawUI.WindowTitle = "Unique Title"
                # Call the compiled executable using path on machine.
            }
        }
        Start-Job -ScriptBlock $cmd | Out-Null
        Break
    }
    "Initiate" {
        C:\Temp\Send_Go.exe
```

```
    Break
}
"Terminate" {
    C:\Temp\Send_Quit.exe
    Break
}
"Start Unreliable Transport" {
    Import-Module {Path To}\Peck_Sockets_v1.0.psm1 -Force
    [int32]$val = 30
    $bytes = [BitConverter]::GetBytes($val)
    $handle = $( Send-UDP
                    -Mode Bytes
                    -Send $bytes
                    -IP $master_machine_ip
                    -Port $master_machine_listen_port
                    -return_handle
                )
    For($i = 0; $i -lt 98; $i++)
    {
        Start-Sleep -Milliseconds 7
        $handle = $( Send-UDP
                        -Mode Bytes
                        -Send $bytes
                        -IP $master_machine_ip
                        -Port $master_machine_listen_port
                        -existing_handle $handle
```

```
                        -return_handle
                    )
    }
    Start-Sleep -Milliseconds 7
    $( Send-Udp
            -Mode Bytes
            -Send $bytes
            -IP $master_machine_ip
            -Port $master_machine_listen_port
            -existing_handle $handle
    )
    Break
}
"Start Reliable Transport" {
    Import-Module {Path To}\Peck_Sockets_v1.0.psm1 -Force
    [int32]$val = 30
    $bytes = [BitConverter]::GetBytes($val)
    $handle = $( Send-TCP
                    -Mode Bytes
                    -Send $bytes
                    -IP $master_machine_ip
                    -Port $master_machine_listen_port
                    -return_handle
                )
    For($i = 0; $i -lt 98; $i++)
    {
```

```
    Start-Sleep -Milliseconds 7
    $handle = $( Send-TCP
                    -Mode Bytes
                    -Send $bytes
                    -IP $master_machine_ip
                    -Port $master_machine_listen_port
                    -existing_handle $handle
                    -return_handle
                )
}
Start-Sleep -Milliseconds 7
$handle = $( Send-TCP
                -Mode Bytes
                -Send $bytes
                -IP $master_machine_ip
                -Port $master_machine_listen_port
                -existing_handle $handle
                -return_handle
            )
Start-Sleep -Seconds 3
$( Send-TCP
    -Mode Text
    -Send "stop"
    -IP $master_machine_ip
    -Port $master_machine_listen_port
    -existing_handle $handle
```

```
            )
            Break
        }
        default {
            Write-Host "Error! ",$command," is not a valid implemented command!"
        }
    }
}Export-ModuleMember -Function execute-command
```

Bibliography

1. "IEEE standard for distributed interactive simulation - application protocols," accessed 26-October-2021. [Online]. Available: https://standards.ieee.org/standard/1278-1-2012.html

2. "IEEE standard for distributed interactive simulation - communication services and profiles," accessed 26-October-2021. [Online]. Available: https://standards.ieee.org/standard/1278-2-2015.html

3. C. Chassot, A. Lozes, F. Garcia, M. Diaz, L. Dairaine, and L. Rojas, "Qos required by a dis application in a new generation internet," in *IEEE Spring Simulation Interoperability Workshop, March*. Citeseer, 1999, pp. 14–19.

4. H. W. Holbrook, S. K. Singhal, and D. R. Cheriton, "Log-based receiver-reliable multicast for distributed interactive simulation," in *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, 1995, pp. 328–341.

5. G. Fiedler, "UDP vs. TCP," accessed 8-December-2021. [Online]. Available: https://gafferongames.com/post/udp_vs_tcp/

6. M. A. Bassiouni, M.-H. Chiu, M. Loper, M. Garnsey, and J. Williams, "Performance and reliability analysis of relevance filtering for scalable distributed interactive simulation," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 7, no. 3, pp. 293–331, 1997.

7. D. A. Fullford, "Distributed interactive simulation: its past, present, and future," in *Proceedings of the 28th conference on Winter simulation*, 1996, pp. 179–185.

8. *OMG Data Distribution Service (DDS)*, Object Magagement Group, 4 2015, version 1.4.

9. "DDSI-RTPS DDS interoperability wire protocol," accessed 26-October-2021. [Online]. Available: https://www.omg.org/spec/DDSI-RTPS/2.3/About-DDSI-RTPS/

10. J. Yang, K. Sandström, T. Nolte, and M. Behnam, "Data distribution service for industrial automation," in *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies Factory Automation (ETFA 2012)*, 2012, pp. 1–8.

11. B. Almadani, M. N. Bajwa, S.-H. Yang, and A.-W. A. Saif, "Performance evaluation of dds-based middleware over wireless channel for reconfigurable manufacturing systems," *International Journal of Distributed Sensor Networks*, vol. 11, no. 7, p. 863123, 2015.

12. S. Profanter, A. Tekat, K. Dorofeev, M. Rickert, and A. Knoll, "Opc ua versus ros, dds, and mqtt: performance evaluation of industry 4.0 protocols," in *2019 IEEE International Conference on Industrial Technology (ICIT)*. IEEE, 2019, pp. 955–962.

13. A. Hakiri, P. Berthou, and T. Gayraud, "Addressing the challenge of distributed interactive simulation with data distribution service," *arXiv preprint arXiv:1008.3759*, 2010.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
25–02–2022	Master's Thesis	Sept 2020 — Mar 2022

4. TITLE AND SUBTITLE

Considering DDS in the Domain of DIS - Pros and Cons

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Nathaniel R. Peck

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-053

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

AFRL/RQQD
Building 145
WPAFB OH 45433-7765
DSN 798-6556, COMM 937-904-6556
Email: James.Zeh@us.af.mil

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

DIS is a legacy IEEE standard for defining and structuring PDUs in large scale distributed wargames. Although the standard specifies various QoS appropriate for certain PDUs, a one-size-fits-all transport strategy is traditionally employed via UDP. Since the inception of DIS, the OMG has produced a standard for a DDS which has been implemented by several middleware vendors. DDS middleware offers an abstraction for network communications that allows applications and developers to easily employ configurable QoS by topic. Adoption and use of these QoS in DIS applications may introduce greater compliance with the IEEE standard and enrich the service features available to distributed wargames and their developers. In this thesis, current use cases of DIS and DDS are examined. The cost, network burden, and performance of DDS is measured and analyzed through experimentation and support DDS's eligibility to promote greater compliance with the IEEE standard for DIS.

15. SUBJECT TERMS

DIS,DDS,Reliability

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Douglas D. Hodson, AFIT/ENG
U	U	U	UU	96	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636, ext 4719