



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

**QUALITY OF SERVICE AND CYBERSECURITY
COMMUNICATION PROTOCOLS ANALYSIS FOR THE
ROBOT OPERATING SYSTEM 2**

by

Jose M. Fernandez

June 2019

Thesis Advisor:
Second Reader:

Preetha Thulasiraman
Brian S. Bingham

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE June 2019	3. REPORT TYPE AND DATES COVERED Master's thesis	
4. TITLE AND SUBTITLE QUALITY OF SERVICE AND CYBERSECURITY COMMUNICATION PROTOCOLS ANALYSIS FOR THE ROBOT OPERATING SYSTEM 2			5. FUNDING NUMBERS	
6. AUTHOR(S) Jose M. Fernandez				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.			12b. DISTRIBUTION CODE A	
13. ABSTRACT (maximum 200 words) Throughout the Department of Defense, efforts to increase cybersecurity and improve data transfer in unmanned robotic systems (UxS) have been ongoing at warfare centers (NUWC, SPAWAR, etc.) and research facilities (NPS). This thesis explores the performance of the Robot Operating System (ROS) 2, which is built with the Data Distribution Service (DDS) standard as a middleware. Based on how quality of service (QoS) parameters are defined in the robotic middleware interface, it is possible to implement strict delivery requirements to different nodes on a dynamic nodal network with multiple unmanned systems connected. Through this research, different scenarios with varying QoS settings were implemented and compared to baseline values to help illustrate the impact of latency and throughput on data flow. DDS security settings were also enabled to help understand the true cost of overhead and performance when secured data is compared to plaintext baseline values. Our experiments were performed using a basic ROS 2 network consisting of two nodes (publisher and subscriber). Our experiments showed a measurable latency and throughput change between different QoS profiles and security settings. We analyze the trends and tradeoffs associated with varying QoS and security settings. This thesis provides performance data points that can be used to help future researchers and developers make informative choices when using ROS 2 for UxS.				
14. SUBJECT TERMS Robot Operating System 2, ROS 2, cybersecurity, quality of service, QoS, latency, throughput, DDS			15. NUMBER OF PAGES 71	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

**QUALITY OF SERVICE AND CYBERSECURITY COMMUNICATION
PROTOCOLS ANALYSIS FOR THE ROBOT OPERATING SYSTEM 2**

Jose M. Fernandez
Lieutenant Commander, United States Navy
BS, University of Arizona, 2008

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

**NAVAL POSTGRADUATE SCHOOL
June 2019**

Approved by: Preetha Thulasiraman
Advisor

Brian S. Bingham
Second Reader

Douglas J. Fouts
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Throughout the Department of Defense, efforts to increase cybersecurity and improve data transfer in unmanned robotic systems (UxS) have been ongoing at warfare centers (NUWC, SPAWAR, etc.) and research facilities (NPS). This thesis explores the performance of the Robot Operating System (ROS) 2, which is built with the Data Distribution Service (DDS) standard as a middleware. Based on how quality of service (QoS) parameters are defined in the robotic middleware interface, it is possible to implement strict delivery requirements to different nodes on a dynamic nodal network with multiple unmanned systems connected. Through this research, different scenarios with varying QoS settings were implemented and compared to baseline values to help illustrate the impact of latency and throughput on data flow. DDS security settings were also enabled to help understand the true cost of overhead and performance when secured data is compared to plaintext baseline values. Our experiments were performed using a basic ROS 2 network consisting of two nodes (publisher and subscriber). Our experiments showed a measurable latency and throughput change between different QoS profiles and security settings. We analyze the trends and tradeoffs associated with varying QoS and security settings. This thesis provides performance data points that can be used to help future researchers and developers make informative choices when using ROS 2 for UxS.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	BENEFIT TO DEPARTMENT OF DEFENSE.....	1
B.	ROS 2 AND DDS OVERVIEW	2
C.	RESEARCH OBJECTIVES AND CONTRIBUTIONS	3
D.	THESIS ORGANIZATION.....	4
II.	RELATED WORK.....	5
A.	ROS 2 QUALITY OF SERVICE	5
B.	ROS 2 SECURITY.....	7
III.	EXPERIMENT DESIGN AND SETUP.....	11
A.	ROS 2 SYSTEM SETUP	11
B.	QUALITY OF SERVICE SETTINGS.....	12
C.	SECURITY SETTINGS.....	17
D.	PLUGIN VERIFICATION	20
1.	Authentication	20
2.	Access Control.....	21
3.	Cryptography	22
IV.	RESULTS	25
A.	DATA ANALYSIS	25
B.	SIMULATION RESULTS	26
1.	0.25MB File Size.....	28
2.	0.5MB File Size.....	29
3.	1MB File Size.....	31
4.	2MB File Size.....	33
5.	4MB File Size.....	34
C.	SUMMARY OF ANALYSIS	36
V.	CONCLUSION	39
A.	SUMMARY	39
B.	FUTURE WORK.....	40
	APPENDIX A. ACCESS CONTROL YAML CODE	41
	APPENDIX B. PUBLISHER SCRIPT	43

APPENDIX C. SUBSCRIBER SCRIPT	47
LIST OF REFERENCES	51
INITIAL DISTRIBUTION LIST	53

LIST OF FIGURES

Figure 1.	Overview of ROS 2 stack. Source: [8].	2
Figure 2.	Security enabled ROS 2 Overview. Source: [13].	7
Figure 3.	Simple RTPS domain. Source: [9].	11
Figure 4.	Supported ROS 2 RMW Vendors. Source: [6].	12
Figure 5.	Reliability = BEST_EFFORT, History = KEEP_LAST, & Depth = 3	14
Figure 6.	Wireshark screen capture of BEST_EFFORT reliability	15
Figure 7.	Reliability = RELIABLE, History = KEEP_LAST, & Depth = 3	16
Figure 8.	Wireshark screen capture of RELIABLE reliability	17
Figure 9.	Generated public/private keys & certificates for all participants	18
Figure 10.	DDS security architecture. Source: [17].	20
Figure 11.	Authentication plugin verification	21
Figure 12.	Access Control plugin verification	22
Figure 13.	Governance XML file output	23
Figure 14.	Cryptography plugin verification	23
Figure 15.	Publisher and subscriber errors for Case 2e and 4MB file size	35
Figure 16.	Packet Latency vs. File Size plot for all cases	37
Figure 17.	MSG Latency vs. File Size plot for all cases	38
Figure 18.	Throughput vs. File Size for all cases	38

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1.	QoS profiles summary for the publisher/subscriber nodes	13
Table 2.	Access Control plugin scenarios	22
Table 1.	QoS profiles summary for the publisher/subscriber nodes	27
Table 3.	0.25MB overhead results	28
Table 4.	0.25MB performance results.....	29
Table 5.	0.5MB overhead results	30
Table 6.	0.5MB performance results.....	31
Table 7.	1MB overhead results	32
Table 8.	1MB performance results.....	32
Table 9.	2MB overhead results	33
Table 10.	2MB performance results.....	34
Table 11.	4MB overhead results	35
Table 12.	4MB performance results.....	36

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

μs	Microseconds
ACKNACK	Acknowledgement/Negative Acknowledgement
API	Application Programming Interface
CA	Certificate Authority
CPU	Central Processing Unit
DDS	Data Distribution Service
DoD	Department of Defense
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
EDP	Endpoint Discovery Protocol
Gbps	Gigabits per Second
HB	Heartbeat
HDD	Hard Disk Drive
I/O	Input-Output
IA	Information Assurance
IP	Internet Protocol
MB	Megabyte
ms	milliseconds
MSG	Message
NPS	Naval Postgraduate School
OA	Open Architectures
OMG	Object Management Group
OS	Operating System
OSD	Office of the Secretary of Defense
OSRF	Open Source Robotics Foundation
PC	Personal Computer
PDP	Participant Discovery Protocol
PKCS7	Public Key Cryptographic Standard Edition 7
QoS	Quality of Service

rmw	Robot Middleware
ROS	Robot Operating System
RSA	Rivest-Shamir-Adleman
RT	Real Time
RTI	Real Time Innovations
RTPS	Real Time Publish Subscribe
RTT	Round Trip Time
SNR	Signal to Noise Ratio
SPI	Service Plugin Interface
SROS	Secure ROS
SSL	Secure Sockets Layer
TLS	Transport Layer Security
UxS	Unmanned Systems
VM	Virtual Machine
YAML	YAML Ain't Markup Language

ACKNOWLEDGMENTS

To the electrical and computer engineering program, you provided me with the tools and knowledge to succeed at this school. To my cohort, thank you for your friendship and encouragement. To all of my instructors, I heartily thank you for imparting onto me your knowledge and wisdom, which has enabled me to excel in this rigorous program.

To Professor Preetha Thulasiraman and Professor Brian Bingham, thank you for guiding and helping me stay the course during this entire thesis process. The continued direction provided by both of you ensured that I would succeed no matter how difficult it was at times.

To Bruce Allen, thank you for taking the journey of learning ROS 2 with me. Whenever a new and thought-provoking problem occurred, you were always there to help me figure it out. I could not have completed this work without your cooperation.

Above all, my extraordinary spouse, Calandra, and my amazing daughter, Leandra, provided continuous encouragement and support during this entire process. I am lucky to have them by my side.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Unmanned systems (UxS) have been growing in prominence as platforms from which to conduct or support military operations. The increased use of UxS in warfare centric environments makes it increasingly vulnerable to cyber threats. In 2017 the Office of the Secretary Defense (OSD) released their UxS Roadmap in which Data Transport Integration and Cyber Security were identified as two key challenges for UxS in the upcoming decades [1]. Data transport integration is defined as the amount of data collected and transferred by UxS from onboard sensors and onboard computers [1]. Methods to effectively transfer both internal and external collected data, both plaintext and encrypted, has not kept pace with the growth of data generation. This is driving the need to find and create new and innovative methods for data migration in robotic systems.

A. BENEFIT TO DEPARTMENT OF DEFENSE

In order to reduce the costs associated with new software development, the Defense Science Board has pushed the adoption of open architectures (OA) in UxS development [2]. This will allow Department of Defense (DoD) developers to concentrate on domain related problems and less on re-developing middleware or infrastructure software. This approach is deemed cost-effective and within the budget constraints of the DoD.

One such architecture is the Robot Operating System (ROS), “which is an open source meta-operating system for robots” [3]. ROS was developed by Open Source Robotics Foundation (OSRF) in 2008. Using open source sharing sites, such as www.github.com, OSRF has been able to develop a global community of people, universities, research groups, government, and commercial industry in developing ROS and further develop UxS platform tools [3].

The Naval Postgraduate School (NPS) has been performing research in multiple fields using the ROS 1 project and has more recently been studying how cyber security architectures implemented on the ROS 2 project affect overall system performance [4]. ROS 1 is a valuable platform on which to conduct research, but the architecture does not

provide any native security between nodes, which is crucial for mission support. ROS 1 is a proven concept and a stable platform for investigation, prototyping, and testing, but it is still not an appropriate platform for final tactical deployment. ROS 2 provides the user the ability to enable and use security features in order to cyber harden a system. This is imperative within current DoD Information Assurance (IA) requirements [1]. However, one must understand the performance costs (latency, throughput, overhead, etc.) associated with these IA requirements. The tradeoff between system security and system performance must be addressed to ensure timely and effective execution of operational tasks [5].

B. ROS 2 AND DDS OVERVIEW

Following the first release of ROS 2 in August 2015, it has quickly grown and matured with multiple releases leading up to ROS 2 Crystal Clemmys in December 2018 [6]. ROS 2 was created with an emphasis on using end-to-end middleware developed by the Object Management Group (OMG) called Data Distribution Service (DDS). The OMG DDS was leveraged to prevent OSRF from having to build a middleware from scratch to work with ROS 1 [7]. Figure 1 illustrates how DDS is a middleware protocol and Application Programming Interface (API) that lies directly between the application, ROS, and the operating system (Windows, Linux, MacOS, etc.).

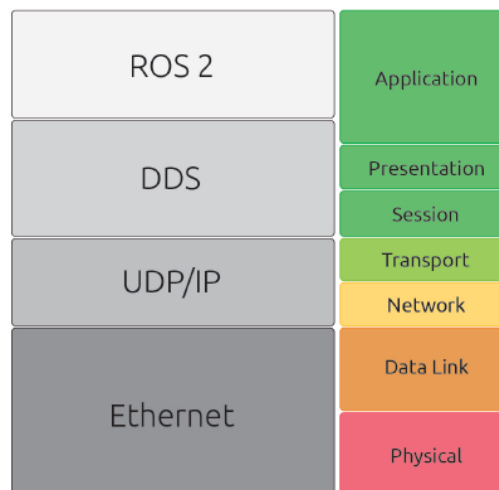


Figure 1. Overview of ROS 2 stack. Source: [8].

ROS 2 with DDS is a from the ground up redesign of the ROS framework that moves ROS away from using in house custom protocols to using middleware with developed communications standards used throughout industry. One key feature of ROS 2 with DDS is the incorporation of the Real-Time-Publish Subscribe (RTPS) communication standard. RTPS is the wire protocol that is designed to implement DDS applications [9] (DDS uses the protocol for data transfer). RTPS provides performance, quality of service (QoS) properties, configurability and scalability. These features translate to improved latency and throughput as seen in the eProsima Fast RTPS performance tests [10] (eProsima is a vendor of DDS middleware.) Another benefit of using DDS it that it also allows the ROS 2 developers to maintain less code.

The DDS RTPS protocol is also configurable in a way that allows for secure communications between nodes. There are security plugins at three levels: authentication, access control and encryption of data [9]. Even though DDS has specified standards, third parties or vendors have the freedom to implement the middleware with different degrees of configurability.

Another key service provided by ROS 2 with DDS is the ability to define different QoS profiles. Each profile can be used depending on the type of data that is being transmitted. DDS allows QoS to be achieved in a real-time data environment. In addition, lost data can be retransmitted without having to restart a session. These attributes are key benefits of using QoS in a lossy network. The different QoS settings and the configurable secure communications allows ROS 2 to address the cyber security and data transport challenges mentioned in the OSD report [1].

C. RESEARCH OBJECTIVES AND CONTRIBUTIONS

The objective of this work is to research and quantify the performance of ROS 2 operating in a small two-node, one-topic network while applying different QoS profiles and security settings. We provide an in-depth study on the different QoS profiles available to ROS 2 in the context of network performance. We also investigate the impact of the ROS 2 security plugins on network performance. There has been a significant amount of research on DDS, and it has been proven to be a well defined publish/subscribe

communications standard. However, the integration of ROS 2 with DDS middleware is still in its infancy and has not been extensively studied.

The contributions of this thesis are:

- Analysis and experimentation of varying ROS 2 QoS case profile combinations for plaintext data traffic. Network performance under varying QoS scenarios was measured using the following parameters: 1) packet loss; 2) latency; 3) throughput; and 4) overhead generation.
- Analysis and experimentation of ROS 2 data security and its impact on network performance in terms of: 1) packet loss; 2) latency; 3) throughput; and 4) overhead generation.
- Our goal is to provide a series of performance measurements with different QoS and security combinations such that it can be utilized by and tailored for diverse military use cases.

D. THESIS ORGANIZATION

The rest of this thesis is organized in the following manner. In Chapter II, we discuss the relevant background information on ROS 2 network performance in terms of QoS and security. In Chapter III we describe our experimental setup, including design, implementation, and execution. In Chapter IV, we present our results and analysis from multiple experimental runs. In Chapter V, we conclude this thesis and summarize our findings with recommendations for future work.

II. RELATED WORK

The focus of this thesis is to explore how performance is affected from applying different QoS settings for sending and receiving nodes in the ROS 2 application. This chapter will discuss related work that focuses on individual QoS or security settings and how they affect overall latency and throughput of data transfers.

A. ROS 2 QUALITY OF SERVICE

DDS middleware is capable of applying 22 different parameters that affect the RTPS wire protocol [11]. Of these 22, ROS 2 only natively supports access to 3 different parameters through the robot middleware (rmw) libraries for use by different vendors. For eProsima, History, Reliability, and Durability are the three supported QoS policies. These policies are explained as follows:

Reliability: Two different parameter settings fall under the umbrella of reliability. 1) *Best-Effort*: messages are sent without arrival confirmation from the receiver. This has the fastest delivery but messages can be lost; 2) *Reliable*: the publisher expects arrival confirmation from the receiver. This is a slower method that prevents data loss. [9]

History: This policy refers to message caching. There are two parameter settings for sample/data storage. 1) *Keep-All*: stores all samples/data in memory; 2) *Keep-Last*: stores samples/data up to a maximum queue depth. Queue depth is a configurable option in DDS. [9]

Durability: This policy defines how a node behaves regarding samples/data that existed on a topic before the subscriber joined. Three parameter settings exist. 1) *Volatile*: past samples/data are ignored and the subscriber receives samples/data after the moment it joins; 2) *Transient Local*: when a new subscriber joins, its *History* (queue) is filled with past samples/data that were stored in temporary local cache; 3) *Transient*: when a new subscriber joins its *History* (queue) is filled with past samples/data which are stored in persistent storage. This is located outside of the local storage so that *History* can be recovered if the publisher drops and rejoins the session. [9]

In [12], the authors measured end-to-end latencies and throughput on ROS 1 and ROS 2. The ROS 2 experiments were executed using multiple different rmw implementations (Connect, OpenSplice, and FastRTPS) [12]. Two QoS profile were used

for all ROS 2 experiments that included the configurable parameters of History, Depth, Reliability, and Durability. For their local loopback experiment with small data size (less than 512K), ROS 1 and ROS 2 produced similar latency results but as data size increased to 4MB, the latency of ROS 2 increased by a factor of five [12]. The authors showed that latency differed greatly as data size increased but remained similar at small data values [12]. The researchers concluded that QoS policies and DDS implementations should be chosen based on the best use case [12]. This work concentrated on showing the differences between ROS 1 and ROS 2 performance, while our research will solely look more in depth on ROS 2 performance.

A research group from Spain looked at how ROS 2 QoS affects round trip times (RTT) with 500B payloads using three different DDS vendor implementations [8]. For a baseline, they ran the test for the different DDS settings while the system was idle (apart from system defaults, no other processes were running). For the system under load test, the system was stressed by generating central processing unit (CPU) stress with 8 CPU, 8 virtual machine (VM), 8 input/output (I/O), and 8 hard disk drive (HDD) workers in the personal computer (PC) [8]. The same QoS profile was used for all the experiments where reliability is set to BEST_EFFORT, history is set to KEEP_LAST with a history depth of one, and durability is set to VOLATILE [8]. For the baseline system idle test, low latencies of approximately 3 milliseconds (ms) were observed for all three DDS settings and no missed deadlines were seen. Deadlines in a real time system represent the time for which a task must be completed. For the system under load test, the latencies increased to an average maximum of 26 ms and hundreds of missed deadlines for each DDS setting [8]. The next phase of tests measured latency with real time (RT) settings while the system was idle. The DDS threads were configured using the QoS profile defined previously through extensible markup language (XML) data files [8]. For the idle system, an average maximum latency of 2.3 ms was observed and zero missed deadlines were recorded. For the system under load RT settings, an average maximum of 2.3 ms latency and zero missed deadlines was recorded [8]. These results clearly illustrate that proper utilization of DDS QoS settings with RT configuration can reduce packet latency. Our research will further

compare the difference in performance between different QoS policies implemented on plaintext and secure data.

B. ROS 2 SECURITY

Prior to ROS 2, ROS 1 was primarily used for research and academia with no security capabilities built into the robotic application software. There were attempts to apply security elements into ROS 1 at different communications layers [4], but there were no native security features to turn on or off in ROS 1 itself. With ROS 2, the well-defined DDS middleware has incorporated security implementations that the user can simply enable if they choose. Figure 2 illustrates the basic overview of security enabled ROS 2, where the top represents the user code (that should not be changed) which interfaces with the ROS client library (rcl) API, to the ROS middleware API, and finally to the DDS vendor plugins and DDS security implementations [13].

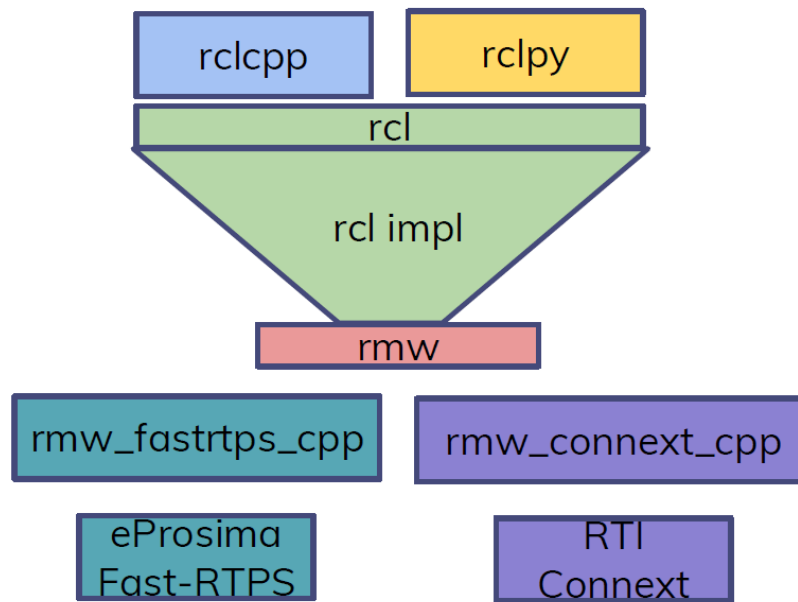


Figure 2. Security enabled ROS 2 Overview. Source: [13].

In [14], the authors conducted a review on ROS 2 and DDS, specifically on the tradeoffs of security, performance, latency, and throughput. The group looked at data sent

as plaintext versus full security enabled data using Rivest-Shamir-Adleman (RSA) 2048 bit and Elliptic Curve Cryptography (ECC) 256 bit. Block sizes of 63KB and approximately 700 packets were transmitted over a time interval of 100 seconds [14]. It was found that, regardless of algorithm or key size, the overhead of security enabled data had an average increase of approximately 137% in latency performance and 132% in the number of packets transmitted. The authors also varied the block size of the plain and encrypted data from 1KB up to 63KB. The results showed that as block size increased so did latency. This resulted in lower throughput and speed (near linear results) [14]. This research was used to select which encryption algorithm was to be used based on performance. Since ECC and RSA had nearly the same performance results, the ECC 256 bit encryption was selected for this thesis.

Another research group analyzed performance metrics on ROS 2 wired and wireless networks, measuring latency and throughput for plaintext versus secured/encrypted data [5]. There are two types of encryption used. One was the encryption provided by DDS middleware vendor eProsima and the other was a Secure Sockets Layer/Transport Layer Security (SSL/TLS) encrypted channel between two nodes established through OpenVPN. The OpenVPN uses AES-128-CBC as the cipher and SHA256 for authentication [5]. The eProsima built in plugin uses AES-GCM-GMAC as the cipher [9]. For both the wired and wireless connections, SSL/TLS outperformed DDS security by a wide margin in both latency and throughput performance. Although the SSL/TLS channel had a better performance for a single channel, DDS security is the more desirable option when used over multiple ROS 2 nodes. Each node in ROS 2 has equal privilege resulting in more resilience and security that cannot be disabled by compromising a single machine such as a VPN server in the SSL/TLS network [5].

At the 2018 ROSCon conference in Madrid, researchers presented the performance impact due to enabling security in DDS, specifically by vendor Real Time Innovations (RTI). They compared latency and throughput from data in four formats that included: (1) plaintext format; (2) secure data that consisted of a signed message; (3) secure data that included a signed message and encrypted data; and (4) secured data that included a signed message, encrypted data and origin authentication [15]. The data size was set to increments

of 32B, 256B, 2KB, 16KB, 128KB, and 1MB. Latency increased significantly from case (1) to (2) and also from case (2) to (3) but there was only a very small increase from case (3) to (4) [15]. It was also observed that from case (1) to (2), there was a large decrease in throughput. From case (2) to (3), throughput also decreased but not by as much as from case (1) to (2). There was only a small decrease in throughput from case (3) to (4) [15]. The authors also looked at how latency and throughput were affected by varying the number of subscriber nodes from 1, 2, and 4 nodes. Very few latency changes were observed as the number of subscribers is increased but the throughput can be seen as steadily decreasing during all three secured data cases [15].

The work performed in this thesis extends on all the work presented in Chapter II. This research goes more in depth in using QoS defined policies and provides a more explicit comparison of the performance between plaintext and secure data.

THIS PAGE INTENTIONALLY LEFT BLANK

III. EXPERIMENT DESIGN AND SETUP

ROS 2 QoS profiles and native security plugins supported via DDS rmw implementations show promise in addressing key concerns in the DoD at both a cybersecurity level and for data transport integration. This chapter discusses the experimental setup, how QoS profiles were defined and used, and what type of security was employed during the multiple simulations. All experiments were executed using a basic simple single topic and participant setup consisting of one publisher and one subscriber node as illustrated in Figure 3.

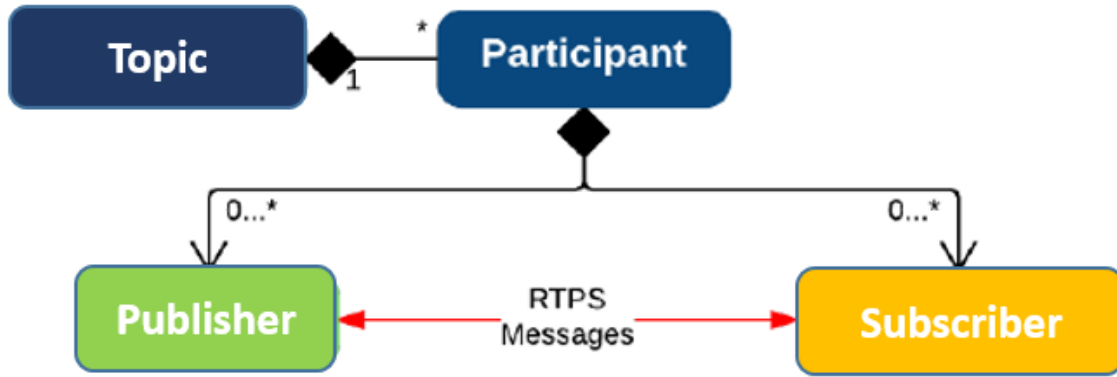


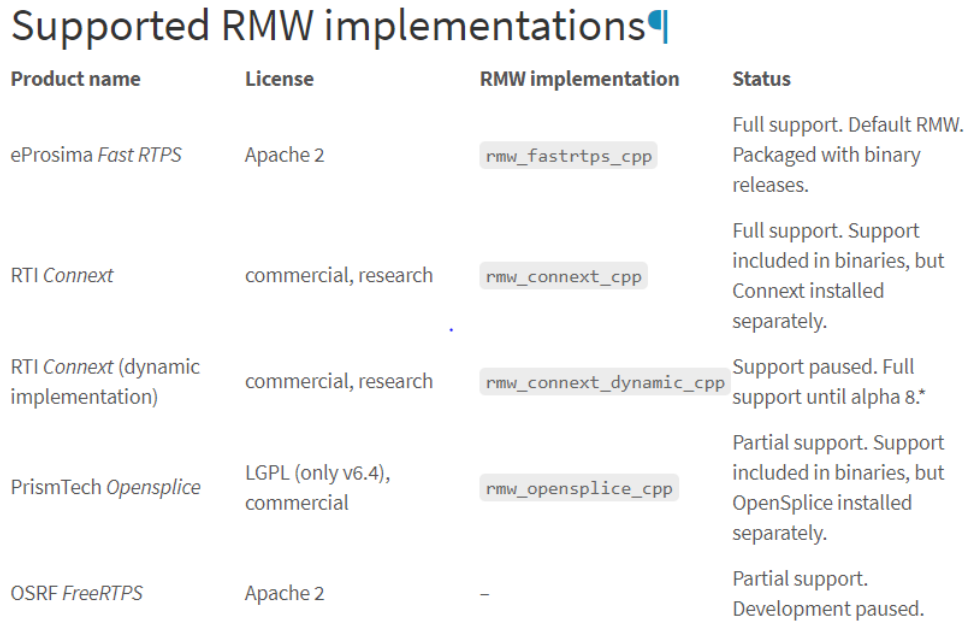
Figure 3. Simple RTPS domain. Source: [9].

A. ROS 2 SYSTEM SETUP

Experiments in this thesis were performed on a SYSTEM76 Wild Dog Pro desktop with a 4.6 GHz i7-8700 (6 cores, 12 threads) processor, 32 GB of DDR4 memory, and Intel Ethernet connection I219-V 1000Base-T network interface. The PC operating system (OS) was the Ubuntu 18.04 LTS (Bionic Beaver) with ROS 2 binaries Crystal Clemmys patch release 2 (February 2019). Wireshark version 2.6.6 was used to capture and analyze all one-way network traffic on the loopback internet protocol (IP) address 127.0.0.1.

After installing ROS 2, per the installation instructions given in ROS Index [6], the ROS 2 source workspace was set as an underlay and a separate workspace for all research

was set as the overlay. This underlay/overlay relationship is highly recommended due to the numerous errors that may be generated when working and modifying files in the source workspace. Lastly, the source workspace needs to have the rmw implementation vendor set. The vendor implementation is the user's choice. The default installed rmw implementation, eProsima Fast RTPS, was used throughout this research. Figure 4 displays all currently supported rmw implementations in ROS 2.



Product name	License	RMW implementation	Status
eProsima <i>Fast RTPS</i>	Apache 2	<code>rmw_fastrtps_cpp</code>	Full support. Default RMW. Packaged with binary releases.
RTI <i>Connext</i>	commercial, research	<code>rmw_connext_cpp</code>	Full support. Support included in binaries, but Connext installed separately.
RTI <i>Connext</i> (dynamic implementation)	commercial, research	<code>rmw_connext_dynamic_cpp</code>	Support paused. Full support until alpha 8.*
PrismTech <i>OpenSplice</i>	LGPL (only v6.4), commercial	<code>rmw_opensplice_cpp</code>	Partial support. Support included in binaries, but OpenSplice installed separately.
OSRF <i>FreeRTPS</i>	Apache 2	–	Partial support. Development paused.

Figure 4. Supported ROS 2 RMW Vendors. Source: [6].

B. QUALITY OF SERVICE SETTINGS

ROS 2 defines the following standard QoS profiles: Default, Services, Parameters, System Default. Initial simulations were conducted using these provided profiles with defined static constants that calls each individual QoS profile. The default provided profiles did not present a clear picture of how parameter settings affected performance, five separate profiles were generated to distinctly measure performance.

This thesis tested five different QoS profiles for plaintext and secure data. The parameters of each profile are defined in Table 1. These parameters were applied to both node participants in order to avoid compatibility issues between the two participants.

Table 1. QoS profiles summary for the publisher/subscriber nodes

Case	Participant	History	Depth	Reliability	Durability
a	All	KEEP_LAST	5	BEST_EFFORT	VOLATILE
b	All	KEEP_ALL	N/A	BEST_EFFORT	TRANSIENT_LOCAL
c	All	KEEP_LAST	5	RELIABLE	VOLATILE
d	All	KEEP_LAST	1000	RELIABLE	VOLATILE
e	All	KEEP_ALL	N/A	RELIABLE	TRANSIENT_LOCAL

The first two cases both use BEST_EFFORT reliability but with differing history and durability settings. These cases were designed to determine the impact of the history parameter on performance metrics. For Case (a), depth was set to five and durability was set to VOLATILE. This means that the amount of data saved in the history cache is set to five packets and the VOLATILE setting requires that no old data be sent to a new subscriber participant that joins in the middle of transmissions. The history cache is saved in a round robin type environment where data past the limit (oldest) will be overwritten with the newest RTPS message data. For Case (b), both participants attempt to maintain a complete history in the cache up to a specific limit (KEEP_ALL). This limit can be set by the user (Depth value) [9]. The TRANSIENT_LOCAL setting was selected as this allows a subscriber to join the topic late and receive previously sent messages up to the limits set in depth. This parameter setting takes more resources to implement resulting in lower performance. In summary, Cases (a) and (b) are set to opposite ends of the performance spectrum (Case (a) results in fast data transmission while Case (b) results in slow data transmissions) under BEST_EFFORT reliability.

Figure 5 illustrates the BEST_EFFORT setup with History set to KEEP_LAST and Depth set to three. Initially, a series of heartbeat (HB) messages and acknowledgements (ACKNACK) messages are sent between the publisher and subscriber. The HB is a submessage sent from the publisher to the subscriber that describes the information that is available to the publisher [11]. The ACKNACK can be used as a positive (ACK) or negative (NACK) acknowledgement from the subscriber. The ACKNACK notifies the

publisher of which packet sequence numbers the subscriber has received and which packet sequence numbers remain missing [11]. Once initial discovery occurs via the Participant Discovery Protocol (PDP), information is exchanged on the endpoints using an Endpoint Discovery Protocol (EDP) via a series of HBs and ACKNACKs that are passed back and forth [11]. After this initial discovery process is completed, data begins to be transferred non-stop from publisher to subscriber filling the history cache in the process. Since Figure 5 illustrates BEST_EFFORT reliability, ACKNACKs for the data packets are never sent between subscriber and publisher. In Figure 6, we display a Wireshark capture of a session where 1MB files were transmitted with reliability selected as BEST_EFFORT. Sixteen packets were transmitted for one message without any ACKNACK or HB being passed back and forth.

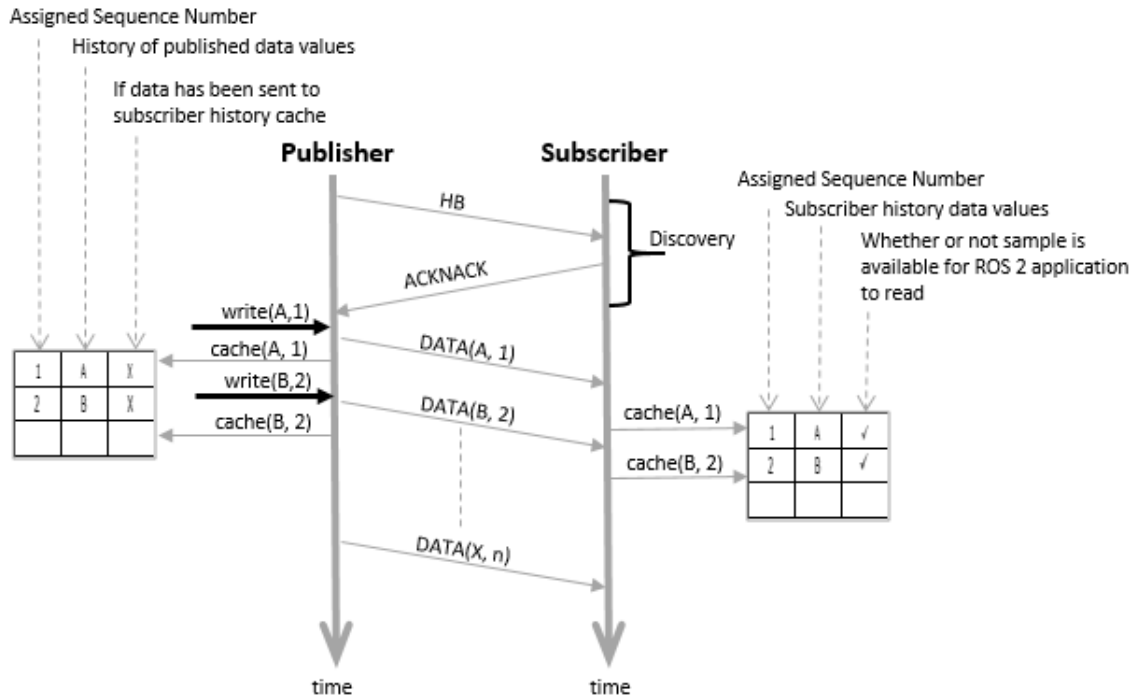


Figure 5. Reliability = BEST_EFFORT, History = KEEP_LAST, & Depth = 3

122	0.011272228	0.000010721	33806	INFO_DST, INFO_TS, DATA_FRAG
123	0.011437477	0.000165249	65542	INFO_DST, INFO_TS, DATA_FRAG
124	0.011456022	0.000018545	65542	INFO_DST, INFO_TS, DATA_FRAG
125	0.011474860	0.000018838	65542	INFO_DST, INFO_TS, DATA_FRAG
126	0.011492597	0.000017737	65542	INFO_DST, INFO_TS, DATA_FRAG
127	0.011510343	0.000017746	65542	INFO_DST, INFO_TS, DATA_FRAG
128	0.011529023	0.000018680	65542	INFO_DST, INFO_TS, DATA_FRAG
129	0.011546706	0.000017683	65542	INFO_DST, INFO_TS, DATA_FRAG
130	0.011563683	0.000016977	65542	INFO_DST, INFO_TS, DATA_FRAG
131	0.011580923	0.000017240	65542	INFO_DST, INFO_TS, DATA_FRAG
132	0.011597890	0.000016967	65542	INFO_DST, INFO_TS, DATA_FRAG
133	0.011614989	0.000017099	65542	INFO_DST, INFO_TS, DATA_FRAG
134	0.011632637	0.000017648	65542	INFO_DST, INFO_TS, DATA_FRAG
135	0.011651175	0.000018538	65542	INFO_DST, INFO_TS, DATA_FRAG
136	0.011668465	0.000017290	65542	INFO_DST, INFO_TS, DATA_FRAG
137	0.011683919	0.000015454	65542	INFO_DST, INFO_TS, DATA_FRAG
138	0.011695067	0.000011148	33806	INFO_DST, INFO_TS, DATA_FRAG
139	0.011706020	0.000017453	65542	INFO_DST, INFO_TS, DATA_FRAG
> Frame 122: 33806 bytes on wire (270448 bits), 33806 bytes captured (270448 bits) on interface 0				
> Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)				
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1				
> User Datagram Protocol, Src Port: 49053, Dst Port: 7433				
> Real-Time Publish-Subscribe Wire Protocol				

Figure 6. Wireshark screen capture of BEST_EFFORT reliability

Cases (c), (d), and (e) were setup up in a similar manner to Case (a) and (b) in terms of anticipated performance. For these three cases, the reliability parameter was set to RELIABLE. In these cases, HBs and ACKNACKs are expected from the publisher and subscriber throughout the session and any unacknowledged data samples will result in re-transmissions from the subscriber. It is still possible to have lost data samples if the history depth is not large enough to allow for re-transmissions. Case (c) is the most resource costly setup, but it ensures all data reaches the subscriber even if they join the domain after RTPS messages have already begun to be streamed.

Figure 7 illustrates the acknowledgement and retransmissions process when Reliability is set to RELIABLE, History is set to KEEP_LAST, and Depth is set to three. As shown in Figure 7, the data caches on the publisher side are updated once an acknowledgement is received. Also illustrated is the key benefit in the RELIABLE parameter, which is the retransmission process. In this case, DATA(A, 1) and HB(1) are transmitted but never received by the subscriber. When the subscriber receives DATA(B, 2) and HB(1-2), the history cache is updated and ACKNACK(1) is sent, indicating the subscriber is ready to receive DATA(A, 1). This ACKNACK(1) translates

to the publisher as needing to resend DATA(A, 1) prior to sending the next packet in the queue. Once the subscriber receives DATA(A, 1), the third column is marked with a “✓” indicating that those packets are releasable from the cache to the application (ROS 2 is the application). On the other side of this exchange, once the publisher receives an ACKNACK for a previously sent HB, the third column in the publisher will change to a “✓” indicating the packet has been delivered to the subscriber cache. This whole process takes more time than BEST_EFFORT and increases overall latency in the session. Figure 8 displays the exchange between participants in Wireshark when the reliability parameter is set to RELIABLE. The HB message can be sent as individual packets or can be piggybacked with a message fragment. ACKNACKs usually include the acknowledgement of multiple received fragments.

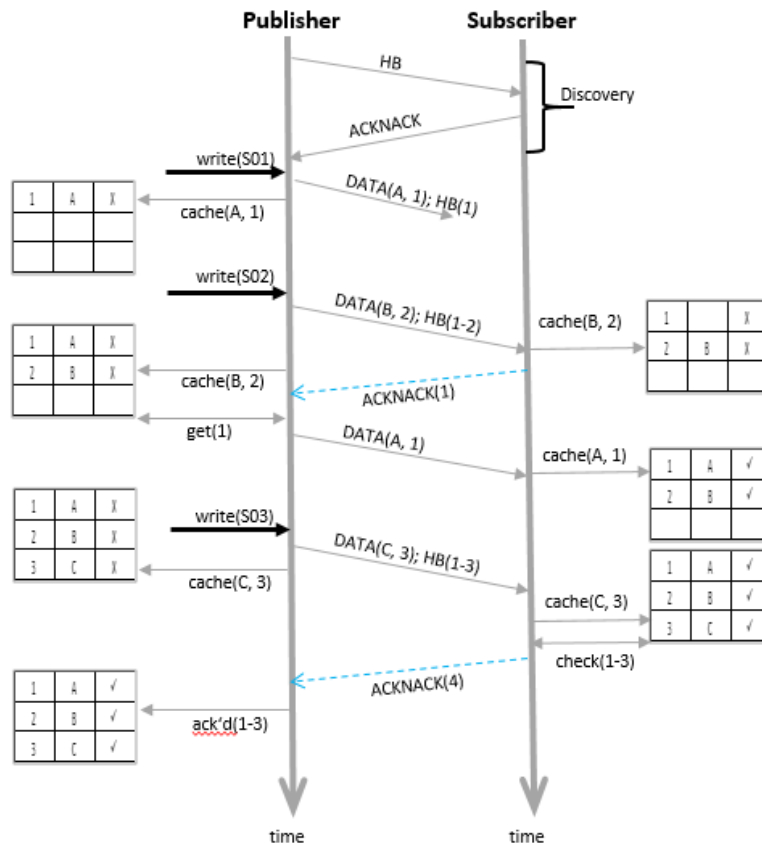


Figure 7. Reliability = RELIABLE, History = KEEP_LAST, & Depth = 3

```

324 2.124513351 -0.000000176 358 INFO_DST, ACKNACK, NACK_FRAG, NACK_FRAG, NACK_FRAG, NACK_FRAG, NACK_FRAG, NACK_FRAG, NACK_FRAG, NACK_FRAG
325 2.124525325 0.000011974 65542 INFO_DST, INFO_TS, DATA_FRAG
326 2.124535108 0.000009783 110 INFO_DST, HEARTBEAT
327 2.124548684 0.000013576 65542 INFO_DST, INFO_TS, DATA_FRAG
328 2.124559841 0.000011157 65542 INFO_DST, INFO_TS, DATA_FRAG
329 2.124576330 0.000016489 33838 INFO_DST, HEARTBEAT, INFO_TS, DATA_FRAG
330 2.124947633 0.000371303 65542 INFO_DST, INFO_TS, DATA_FRAG
331 2.124962836 0.000015203 65542 INFO_DST, INFO_TS, DATA_FRAG
332 2.124973822 0.000010986 110 INFO_DST, HEARTBEAT
333 2.124987998 0.000014176 65542 INFO_DST, INFO_TS, DATA_FRAG
334 2.125001634 0.000013636 65542 INFO_DST, INFO_TS, DATA_FRAG
335 2.125011975 0.000010341 110 INFO_DST, HEARTBEAT
336 2.125026025 0.000014050 65542 INFO_DST, INFO_TS, DATA_FRAG
337 2.125039516 0.000013491 65542 INFO_DST, INFO_TS, DATA_FRAG
338 2.125049814 0.000010298 110 INFO_DST, HEARTBEAT
339 2.125064878 0.000015064 65542 INFO_DST, INFO_TS, DATA_FRAG
340 2.125077832 0.000012954 65542 INFO_DST, INFO_TS, DATA_FRAG
341 2.125087955 0.000010123 110 INFO_DST, HEARTBEAT
<
> Frame 1: 110 bytes on wire (880 bits), 110 bytes captured (880 bits) on interface 0
> Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 55962, Dst Port: 7441
> Real-Time Publish-Subscribe Wire Protocol

```

Figure 8. Wireshark screen capture of RELIABLE reliability

C. SECURITY SETTINGS

As discussed before, it is expected that the performance for security enabled data will be much lower than that for plaintext. As seen in [14], both the ECC 256 bit and RSA 2048 bit algorithms have very similar latency and throughput results when compared side by side against plaintext performance. Based on these results, the ECC 256 bit algorithm was used in this thesis. OpenSSL software library was needed in order to generate the ECC certificates and keys necessary for the (1) authentication, (2) access control, and (3) cryptographic plugins to work. OpenSSL commands are used to generate all necessary public and private keys and certificates for both the publisher and subscriber. In addition, for the eProxima vendor, all packages must be compiled by adding “-DESURITY=ON” during the “colcon build” ROS 2 package build process. Figure 9 displays the ROS 2 tree layout of all security items generated, including public and private keys and certificates for all participants.

```
ros-2@nps-528a:~/sros2$ tree
.
├── keys
│   ├── 1000.pem
│   ├── 1001.pem
│   ├── ca.cert.pem
│   ├── ca_conf.cnf
│   ├── ca.key.pem
│   ├── ecdsaparam
│   ├── governance.p7s
│   ├── governance.xml
│   ├── index.txt
│   ├── index.txt.attr
│   ├── index.txt.attr.old
│   ├── index.txt.old
│   ├── listener
│   │   ├── cert.pem
│   │   ├── ecdsaparam
│   │   ├── governance.p7s
│   │   ├── identity_ca.cert.pem
│   │   ├── key.pem
│   │   ├── permissions_ca.cert.pem
│   │   ├── permissions.p7s
│   │   ├── permissions.xml
│   │   ├── req.pem
│   │   └── request.cnf
│   ├── serial
│   └── serial.old
└── talker
    ├── cert.pem
    ├── ecdsaparam
    ├── governance.p7s
    ├── identity_ca.cert.pem
    ├── key.pem
    ├── permissions_ca.cert.pem
    ├── permissions.p7s
    ├── permissions.xml
    ├── req.pem
    └── request.cnf
```

Figure 9. Generated public/private keys & certificates for all participants

eProsima defines the three security plugins as follows:

1. Authentication: This built-in plugin provides authentication between discovered participants. Authentication is achieved with a trusted Certificate Authority (CA) and implements Elliptic Curve Digital Signature Algorithm (ECDSA) to perform the mutual authentication. It also establishes a shared secret key using Elliptic Curve Diffie-Hellman (ECDH) Key Agreement Methods. When a remote participant is detected, Fast RTPS tries to authenticate using the activated authentication plugin. If the authentication process finishes successfully, then both participants

match and the discovery protocol continues. If authentication fails, the remote participant is rejected. [9]

2. Access Control: Provides validation of entity permissions and access control using a permissions document signed by a shared CA. After a remote participant is authenticated, its permissions need to be validated and enforced. It is configured with three documents: `governance.xml`, `permissions.xml`, and `permissions_ca`. [9]
3. Cryptographic: Provides encryption support applied over three different levels of the RTPS protocol: 1) encryption over the whole RTPS messages; 2) encryption of RTPS submessages of a particular entity (publisher or subscriber); or 3) encryption of the payload (user data) of a particular publisher. Fast RTPS provides a built-in cryptographic plugin. The cryptographic plugin is configured by the Access control plugin. [9]

The ROS2/DDS security architecture is shown in Figure 10. Only the identified three plugins are necessary for DDS compliance. The logging and tagging plugins shown in Figure 10 are identified as optional by OMG [17]. The environmental variables must be defined in this process to enable secure plugins and point to the location of all keys and certificates. This is done by setting `ROS_SECURITY_ENABLE = "true,"` then the `rcl` and then the `rmw` (following Figure 2) will incorporate the DDS security plugins. `ROS_SECURITY_STRATEGY` must be set to "Enforce" in order for a node to use all three plugins. Otherwise, a permissive node will be created that is unsecure. Once security is enabled and strategy is enforced, a node that uses all three security artifacts, shown in Figure 10, will be generated and authentication and cryptography will be enforced for all participants in that topic. By default, an access control strategy will be set up but no access restrictions will be set. The user defines the restrictions for the topic and participant.

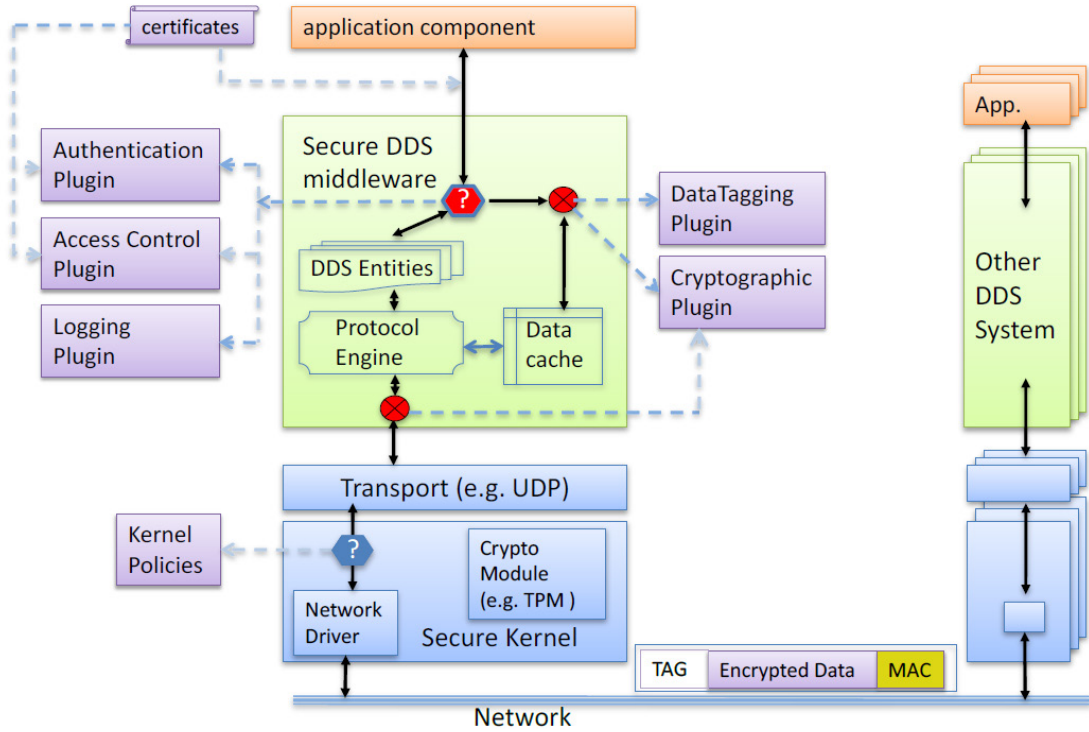


Figure 10. DDS security architecture. Source: [17].

D. PLUGIN VERIFICATION

1. Authentication

To verify that the authentication plugin denies access to unauthorized participants, two different scenarios were tested. The publisher participant was first established with a node name of “talker1” vice “talker.” The first error is a failure to initialize the node due to no keys or certificates being found for “talker1,” therefore the unauthorized node was rejected. The second case modified a single character in the publisher certificate resulting in security errors. Specifically, the Public Key Cryptographic Standard 7th edition (PKCS7) projected an error from the single character change in the certificate. This can be seen in the bottom half of Figure 11. In addition, private keys were also modified. No errors were generated by modifying the private key, but the node was unable to be established.

```

ros-2@nps-528a:~$ ros2 run nodes talker chatter
terminate called after throwing an instance of 'rclcpp::exceptions::RCLError'
  what(): failed to initialize rcl node: SECURITY ERROR: unable to find a folder
  matching the node name in the "ROS_SECURITY_NODE_DIRECTORY" or "ROS_SECURITY_P
  ROOT_DIRECTORY" directories while the requested security strategy requires it, at
  /home/ros-2/ros2c_ws/src/ros2/rcl/rcl/src/rcl/node.c:390
ros-2@nps-528a:~$ ros2 run nodes talker chatter
2019-05-14 10:48:27.121 [SECURITY Error] Error validating the local participant
  permissions. (Cannot read as PKCS7 the file file:///home/ros-2/sros2/keys/talker
  /governance.p7s (/home/ros-2/ros2c_ws/src/eProsima/Fast-RTPS/src/cpp/security/ac
  cesscontrol/Permissions.cpp:403)) -> Function init
2019-05-14 10:48:27.122 [RTPS_PARTICIPANT Error] Cannot create participant due t
  o security initialization error -> Function createParticipant
2019-05-14 10:48:27.123 [terminate called after throwing an instance of 'PARTICI
  PANTrclcpp::exceptions::RCLError'
  Error] Problem creating RTPSParticipant -> Function createParticipant
  what(): failed to initialize rcl node: create_node() could not create partici
  pant, at /home/ros-2/ros2c_ws/src/ros2/rmw_fastrtps/rmw_fastrtps_shared_cpp/src/
  rmw_node.cpp:95, at /home/ros-2/ros2c_ws/src/ros2/rcl/rcl/src/rcl/node.c:401

```

Figure 11. Authentication plugin verification

2. Access Control

Access control configurations were setup using a YAML Ain't Markup Language (YAML) file to manually define the allowed node names, topic names, and participants allowed actions as seen in Appendix A. ROS 2 then uses a shortcut command, “create_permission,” to read in the YAML file and convert it to a DDS readable permissions.xml file for specified participants. Once defined, nodes will only have access to topics listed and publish or subscribe privileges are limited to those listed in the XML file. “P” is used for publish, “S” is used for subscribe, and “PS” is used to annotate that a participant can both publish and subscribe in the topic. Table 2 lists six different cases to verify that the access control plugin rejects nodes from being established if they do not follow the rules. Chatter is the approved topic name. The function of the publisher script is only to publish data and the subscriber script only subscribes.

Table 2. Access Control plugin scenarios

Case	Publisher Topic	Subscriber Topic	Publisher Allow	Subscriber Allow	Connection Established
1	chatter	chatter	P	S	Yes
2	chatter	not chatter	P	S	No
3	not chatter	chatter	P	S	No
4	chatter	chatter	S	S	No
5	chatter	chatter	P	P	No
6	chatter	chatter	PS	PS	Yes

Figure 12 displays the error output when a participant tries to establish a node in an unapproved topic. The “chatter” topic was manually changed to “not_chatter” when testing the access control plugin for unauthorized topics. The YAML script was used to change the allowed “P,” “S,” or “PS” for participants when testing the other listed cases.

```

ros-2@nps-528a:~$ ros2 run nodes talker chatter:=not_chatter
2019-05-13 19:18:06.363 [SECURITY Error] Error checking creation of local writer
9a.dd.34.40.e7.49.21.46.b7.ff.27.0|0.0.1b.3 (rt/not_chatter topic not found in
allow rule. (/home/ros-2/ros2c_ws/src/eProxima/Fast-RTPS/src/cpp/security/access
control/Permissions.cpp:111))
-> Function register_local_writer
2019-05-13 19:18:06.363 [PARTICIPANT Error] Problem creating associated Writer -
> Function createPublisher
terminate called after throwing an instance of 'rclcpp::exceptions::RCLError'
what(): could not create publisher: create_publisher() could not create publi
sher, at /home/ros-2/ros2c_ws/src/ros2/rmw_fastrtps/rmw_fastrtps_cpp/src/rmw_pub
lisher.cpp:151, at /home/ros-2/ros2c_ws/src/ros2/rcl/rcl/src/rcl/publisher.c:173

```

Figure 12. Access Control plugin verification

3. Cryptography

The access control plugin generates a domain governance XML file that defines how the domain should be encrypted [9]. Some key elements managed in the XML file includes both discovery and RTPS data. Discovery data includes data related to the EDP and PDP, this is data that is involved in the initial node handshake phase. RTPS data includes all payload data and metadata (RTPS submessages from a participant). Figure 13

displays the contents of the XML file are set to ENCRYPT, but NONE can be entered to pass along unencrypted data. Figure 13 displays the message traffic where all payload, submessages (HBs and ACKNACKs), and discovery data are encrypted.

```
<allow_unauthenticated_participants>false</allow_unauthenticated_participants>
<enable_join_access_control>true</enable_join_access_control>
<discovery_protection_kind>ENCRYPT</discovery_protection_kind>
<liveliness_protection_kind>ENCRYPT</liveliness_protection_kind>
<rtps_protection_kind>SIGN</rtps_protection_kind>
<topic_access_rules>
  <topic_rule>
    <topic_expression>*</topic_expression>
    <enable_discovery_protection>true</enable_discovery_protection>
    <enable_liveliness_protection>true</enable_liveliness_protection>
    <enable_read_access_control>true</enable_read_access_control>
    <enable_write_access_control>true</enable_write_access_control>
    <metadata_protection_kind>ENCRYPT</metadata_protection_kind>
    <data_protection_kind>ENCRYPT</data_protection_kind>
```

Figure 13. Governance XML file output

313 0.105118381	0.000145515	46018	SRTPS_PREFIX, INFO_SRC, INFO_DST, SEC_PREFIX, SEC_BODY, SEC_POSTFIX, SEC_PREFIX, SEC_BODY, SEC_POSTFIX,
314 0.106291074	0.001172693	234	SRTPS_PREFIX, INFO_SRC, INFO_DST, SEC_PREFIX, SEC_BODY, SEC_POSTFIX, SRTPS_POSTFIX
315 0.110348486	0.004057412	64978	SRTPS_PREFIX, INFO_SRC, INFO_DST, SEC_PREFIX, SEC_BODY, SEC_POSTFIX, SEC_PREFIX, SEC_BODY, SEC_POSTFIX,
316 0.110708111	0.000359625	65066	SRTPS_PREFIX, INFO_SRC, INFO_DST, SEC_PREFIX, SEC_BODY, SEC_POSTFIX, SEC_PREFIX, SEC_BODY, SEC_POSTFIX,
317 0.111026965	0.000318854	64978	SRTPS_PREFIX, INFO_SRC, INFO_DST, SEC_PREFIX, SEC_BODY, SEC_POSTFIX, SEC_PREFIX, SEC_BODY, SEC_POSTFIX,
318 0.111394071	0.000367106	65066	SRTPS_PREFIX, INFO_SRC, INFO_DST, SEC_PREFIX, SEC_BODY, SEC_POSTFIX, SEC_PREFIX, SEC_BODY, SEC_POSTFIX,
319 0.111933677	0.000539606	64978	SRTPS_PREFIX, INFO_SRC, INFO_DST, SEC_PREFIX, SEC_BODY, SEC_POSTFIX, SEC_PREFIX, SEC_BODY, SEC_POSTFIX,
320 0.112200037	0.000367360	65066	SRTPS_PREFIX, INFO_SRC, INFO_DST, SEC_PREFIX, SEC_BODY, SEC_POSTFIX, SEC_PREFIX, SEC_BODY, SEC_POSTFIX,

```
submessageId: SRTPS_PREFIX (0x33)
submessageId: INFO_SRC (0x0c)
submessageId: INFO_DST (0x0e)
submessageId: SEC_PREFIX (0x31)
submessageId: SEC_BODY (0x30)
submessageId: SEC_POSTFIX (0x32)
submessageId: SEC_PREFIX (0x31)
submessageId: SEC_BODY (0x30)
```

7c 7a 79 37 c8 29 16 bd	fe 76 0d 4a 06 66 85 c1	zy7-)···v·J·f··
94 98 56 ac 4f 85 50 f2	5c 18 53 a2 19 a9 32 21	·V·O·P·\·S···2!
09 52 aa 82 e3 8a 2d 34	4b 2e 06 54 91 0e 47 55	·R·····4 K·T··GU
d8 9a 71 4d 4a c9 cf 1f	c5 ae 64 08 6f 74 d9 64	·qMj····d·ot·d
9e 58 ba c9 2f f8 0c 44	58 47 7f ce 97 eb ea 52	·X·/··D XG····R
cc 26 4b 8c 75 83 e9 db	de 2f 5b e1 c5 9c 56 13	·&K·u····/[···V·
4b 70 bf 92 91 34 7d 01	16 de c4 d9 75 d2 3c 75	Kp···4}····u·<u
ec 89 e8 26 49 a9 fd a8	ec 97 c6 2b 90 50 8e 31	···&I·····+·P·1
74 bb 8b 78 eb 71 0b 33	a8 45 89 9b bf 96 74 ef	t··x·q·3 ·E····t·
4a 35 78 f8 f9 87 f4 68	e5 8a 83 25 85 88 35 3a	J5x····h ···%···5:

Figure 14. Cryptography plugin verification

THIS PAGE INTENTIONALLY LEFT BLANK

IV. RESULTS

A series of test runs were conducted for the five QoS cases listed in Table 1 for plaintext data and secure data in which the three security plugins are enabled (authentication, access control, and encryption). This chapter discusses the results of the experimental runs based on the setup given in Chapter III.

A. DATA ANALYSIS

The simulations consist of two separate nodes transferring messages of varying sizes in both plain and encrypted text format. The message traffic is analyzed by looking at the amount of time it takes to transmit individual packets (message fragments, ACKNACKs, HBs, etc.) and the corresponding whole messages from the publishing node only. The transmission latency time was chosen as the measurement parameter for latency because traditional latency calculations can vary greatly depending on transfer medium properties (distance, traffic congestion, connection types, etc.). Transmission latency times include the time the robotic middleware takes to process a message, encrypt the data, serialize the data and finally send it to a buffer cache to be transmitted. All latency and throughput values will be compared against Case 1a values in a percentage format per Equation (1) and (2). Case 1a was chosen since it has the best performance for all message sizes in plaintext data.

$$latency\Delta\% = \frac{latency_{new} - latency_{1a}}{latency_{1a}} * 100\% \quad (1)$$

$$throughput\Delta\% = \frac{throughput_{new} - throughput_{1a}}{throughput_{1a}} * 100\% \quad (2)$$

While modifying the robotic middleware QoS and security settings, different overhead values due to computation time, excess protocol data generation, and packet retransmission can be observed. We define overhead as the amount of excess data packets that are sent in addition to actual message fragments. These excess data packets primarily consist of metadata that is not appended to RTPS message fragments (metadata consists of

HBs and ACKNACKs). Understanding the overhead tradeoffs between the different QoS and security settings is a key objective of this research.

B. SIMULATION RESULTS

The QoS profile for Case 1a was used to establish the baseline for both latency and throughput measurements. Case 1a had the lowest latency for both fragmented packets and overall message latency as well as the highest message throughput for each tested file size. The following definitions are how each column of data was recorded or calculated.

- **Total Packets:** Packets counted in Wireshark from the first transmitted RTPS message fragment (after discovery protocol handshake) until the last transmitted message fragment (prior to session termination procedure). Includes all metadata (HBs and ACKNACKs) and discovery protocol messages transmissions after fragment one.
- **Message (MSG) Fragment Packets:** Packets counted in Wireshark that only include RTPS message fragments.
- **Overhead Packets (%):** The ratio of overhead packet messages divided by the total packets. This gives a percentage value that quantifies the amount of overhead packets that were transmitted outside of RTPS message fragments.
- **MSGs Lost:** Number of messages that were not received by the subscribing node. This could be due to a lost fragment, a lost message, data collision, etc.
- **MSG Fragment Latency (μ s):** For each simulation, 1000 messages were transmitted at the identified message size. The latency for each transmitted RTPS fragment was calculated by subtracting the timestamp from the current message fragment from the previously transmitted message fragment. This is measured in microseconds (μ s). This column represents the average latency of message fragments only.

- **MSG Latency (μs):** RTPS message fragments were added up to determine the total latency for transmitting one message. This value was then averaged across the other 999 messages transmitted, including some retransmitted fragments and other messages with incomplete fragments. This is measured in microseconds.
- **MSG Throughput (Gbps):** This was calculated using the size of the message divided by the MSG latency value. The size of the message is equal to the total size of the message fragments, added together. Throughput was measured in Giga bits per second (Gbps). The throughput calculation is shown in Equation (3).

$$Throughput(Gbps) = \frac{MSGsize(bits)}{latency(\mu s)} \quad (3)$$

- $\Delta\%$: Each $\Delta\%$ is a comparison for either the average MSG fragment latency, average MSG latency, or average throughput compared against Case 1a by using Equations (1) or (2).

To facilitate understanding of the data tables, the five cases that we test are shown again in Table 1

Table 1. QoS profiles summary for the publisher/subscriber nodes

Case	Participant	history	depth	Reliability	Durability
a	All	KEEP_LAST	5	BEST_EFFORT	VOLATILE
b	All	KEEP_ALL	N/A	BEST_EFFORT	TRANSIENT_LOCAL
c	All	KEEP_LAST	5	RELIABLE	VOLATILE
d	All	KEEP_LAST	1000	RELIABLE	VOLATILE
e	All	KEEP_ALL	N/A	RELIABLE	TRANSIENT_LOCAL

1. 0.25MB File Size

A 0.25 megabyte (MB) character string was produced to transmit a continuous payload of 1000 messages from a publishing node to a subscriber node. Table 3 displays the overhead data for plaintext and secure data as well as messages lost during transmission. Table 4 displays all latencies and throughputs for the ten 0.25MB experimental runs.

In Table 3, for Cases 1a and 1b, little overhead was generated when compared to Cases 1c, 1d, and 1e. This was because metadata was not appended to the RTPS message fragments (for Cases 1c, 1d, and 1e), but was transmitted separately. Cases 2c, 2d, and 2e had a majority of its metadata appended to RTPS fragments, which resulted in similar overhead values to cases 1a and 1b. For the RELABLE Cases (c, d, and e), it was expected that zero messages would be lost as long as a sufficient history depth is set. For Case 1d and 1e, we see zero messages lost and Case 1c has some messages lost, as expected, since depth is set to five. For the secure cases (Cases 2a-e), there is an unexpected high number of lost messages. The tests for the secure RELIABLE cases were run multiple times to ensure the results were accurate. The number of lost messages stayed the same during each experimental trial (greater than 10%).

Table 3. 0.25MB overhead results

Case	File Size (MB)	Total Packets	MSG Frag Packets	Overhead Packets (%)	MSGs Lost
1a	0.25	3672	3627	1.23	2
1b		4048	3999	1.21	12
1c		5989	3979	33.56	4
1d		5023	3949	21.38	0
1e		4843	3782	21.91	0
2a	0.25	3561	3512	1.38	122
2b		3847	3800	1.21	50
2c		3705	3601	2.81	103
2d		3646	3537	2.99	121
2e		3621	3516	2.90	129

Table 4 displays performance metrics related to latency and throughput. Case 1a has no data displayed in the $\Delta\%$ columns since this case was considered the baseline results for the 0.25MB file size runs. All other data runs will follow suit and will maintain Case 1a as the baseline case for comparisons. Case 1b and 2b have the worst performance metrics when compared to the baseline. This is when QoS settings were set to BEST_EFFORT, KEEP_ALL, and TRANSIENT_LOCAL. Another trend seen here is as history increases in each reliability subset case, throughput performance decreases.

Table 4. 0.25MB performance results

Case	File Size (MB)	MSG Frag Latency (μs)	Δ%	MSG Latency (μs)	Δ%	MSG Throughput (Gbps)	Δ%
1a	0.25	25.8	-	105.0	-	2.477	-
1b		40.4	56.6	162	54.3	1.590	-35.8
1c		29.5	14.3	118	12.4	2.130	-14.0
1d		32.3	25.2	129.6	23.4	1.979	-20.1
1e		30.3	17.4	128.1	22.0	2.127	-14.1
2a	0.25	62.7	143.0	250.7	138.8	0.988	-60.1
2b		84.9	229.1	345.6	229.1	0.730	-70.5
2c		70.7	174.0	282.2	168.8	0.876	-64.6
2d		74.7	189.5	297.7	183.5	0.850	-65.7
2e		72.3	180.2	287.9	174.2	0.861	-65.2

2. 0.5MB File Size

The file size was doubled to 0.5MB character sting size to see if previous trends continued or if new behaviors presented themselves. All other setup parameters remained the same from the previous file size.

In Table 5, the trend of the overhead results are very similar to the 0.25MB results with the exception for Cases 1c and 2c. It appears that Case c, for both plaintext and secure data, produced a large amount of RTPS message fragment retransmission attempts. For Case 1c, the retransmissions included a large number of metadata and for Case 2c, the retransmitted metadata was appended to the message fragment resulting in a larger amount

of MSG fragment packets. Case 1d and 1e continue to have 100% message delivery but Case 2d and Case 2e continue to have losses. As the message sizes approximately doubled in size, the losses for all secured transmitted data have approximately decreased by the half. There were no changes in the way the messages were transmitted to explain why losses decreased by half for all secure cases.

Table 5. 0.5MB overhead results

Case	File Size (MB)	Total Packets	MSG Frag Packets	Overhead Packets (%)	MSGs Lost
1a	0.50	7094	7046	0.68	7
1b		8040	7994	0.57	19
1c		13531	7831	42.13	3
1d		10633	7614	28.39	0
1e		11119	8004	28.02	0
2a	0.50	7609	7560	0.64	55
2b		7851	7807	0.56	25
2c		16828	16521	1.82	49
2d		7846	7682	2.09	58
2e		7814	7650	2.10	39

Table 6 expressed very similar trends from Table 4, including Case 1b and 2b continuing to display the worst performance metrics compared to Case 1a. Latency and throughput stayed nearly constant as history depth increased in the RELIABLE cases. This is slightly different from the 0.25MB cases.

Table 6. 0.5MB performance results

Case	File Size (MB)	MSG Frag Latency (μs)	Δ%	MSG Latency (μs)	Δ%	MSG Throughput (Gbps)	Δ%
1a	0.50	26.3	-	242.6	-	2.459	-
1b		40.8	55.1	335	38.1	1.554	-36.8
1c		30.4	15.6	242	0.2	2.085	-15.2
1d		32.2	22.4	260.5	7.4	2.024	-17.6
1e		30.3	15.2	243.4	0.3	2.125	-13.6
2a	0.50	65.8	150.2	522.2	115.3	0.959	-61.0
2b		90.5	244.1	730.8	201.2	0.700	-71.5
2c		72.8	157.2	1233	177.8	0.876	-61.3
2d		71.1	170.3	563.1	132.1	0.889	-63.8
2e		71.5	171.9	577.8	138.2	0.884	-64.1

3. 1MB File Size

The message string was again doubled to yield a 1MB output file for the next ten cases. Case setup parameters continue to remain constant.

In Table 7, Case 1c continues to have a larger amount of retransmitted metadata. This was expected due to the REIABILITY setting and history depth set to five. The subscriber continually informs the publisher of the missing data, but since history is so small, the publisher is unable to retransmit from the temporary cache. This was described in Figure 7 in Ch. III. The plaintext message losses trend the same for Case 1a and 1b but are higher for Case 1c. Overall, for the secure data, the message losses are continuing to decrease as the message size increases. For Case 2d and 2e, as the message size doubles, the messages lost decreases by a half. This follows the trend that was seen for the 0.5MB file size. All other trends appear relatively the same.

Table 7. 1MB overhead results

Case	File Size (MB)	Total Packets	MSG Frag Packets	Overhead Packets (%)	MSGs Lost
1a	1	15527	15487	0.26	10
1b		15393	15357	0.23	17
1c		27716	15552	43.89	28
1d		23315	16077	31.04	0
1e		23892	16491	30.98	0
2a	1	15871	15824	0.30	41
2b		15787	15744	0.27	19
2c		16828	16521	1.82	49
2d		16724	16419	2.82	16
2e		16740	16306	2.59	19

From Table 8, we can see that Cases 1b and 2b perform slightly better when compared the 0.25MB and 0.5MB results. However, they remain at the bottom of the 1MB test run in terms of performance. RELIABILITY maintains performance results for fragment latency and overall throughput with metrics appearing to start converging towards each other as file size increases.

Table 8. 1MB performance results

Case	File Size (MB)	MSG Frag Latency (μs)	Δ%	MSG Latency (μs)	Δ%	MSG Throughput (Gbps)	Δ%
1a	1	28.3	-	443.8	-	2.261	-
1b		45.2	59.7	722.2	62.7	1.451	-35.8
1c		33.2	17.3	529.9	19.4	1.938	-14.3
1d		36.1	27.6	608.7	37.2	1.869	-17.3
1e		33.1	17.0	558.5	25.8	1.986	-12.2
2a	1	68.3	141.3	1065	140.0	0.935	-58.6
2b		84.2	197.5	1347	203.5	0.759	-66.4
2c		72.8	157.2	1233	177.8	0.876	-61.3
2d		75.1	165.4	1249	181.4	0.864	-61.8
2e		74.0	161.5	1240	179.4	0.868	-61.6

4. 2MB File Size

Again, the file size is doubled and the QoS profiles and security plugin settings for each case are maintained constant. In Table 9, the overhead metrics maintain the same trend but now message losses are increasing by a significant amount compared to the 1MB file size and smaller cases. For the secure data cases, this is the first time all messages were delivered as expected for the Case 2d and Case 2e data runs. Experiments for these two cases were run multiple times to ensure these were accurate results. The tradeoff of achieving 100% message delivery was the very high increase of message fragment retransmissions required to achieve this. From here, file size was manually adjusted to determine when 100% reliability could be achieved for this experimental setup. At an approximate 1.25MB file size, a continuous 100% message delivery is achieved for Case 2d and 2e.

Table 9. 2MB overhead results

Case	File Size (MB)	Total Packets	MSG Frag Packets	Overhead Packets (%)	MSGs Lost
1a	2	30127	30089	0.13	77
1b		29024	28986	0.13	24
1c		55504	32062	42.23	99
1d		50167	33817	32.59	0
1e		49952	33700	32.54	0
2a	2	31867	31824	0.13	17
2b		32043	32000	0.13	42
2c		33908	33332	1.70	22
2d		56632	56228	0.71	0
2e		43930	43383	1.25	0

In Table 10, previous discussed trends remain with the exception of Case 2d and Case 2e. The throughput decreases and is approaching Case 2b results. This was expected since there is a cost associated with all of the retransmissions in these cases in order to have 100% message delivery. Another overall expected trend is that as file size increased, all latencies gradually increased and throughput decreased.

Table 10. 2MB performance results

Case	File Size (MB)	MSG Frag Latency (μs)	Δ%	MSG Latency (μs)	Δ%	MSG Throughput (Gbps)	Δ%
1a	2	34.8	-	1029	-	1.868	-
1b		48.6	39.7	1471	43.0	1.369	-26.7
1c		37.4	7.5	1266	23.0	1.767	-5.4
1d		38.1	9.5	1317	28.0	1.720	-7.9
1e		36.9	6.0	1313	27.6	1.806	-3.3
2a	2	75.2	116.1	2139	107.9	0.846	-54.7
2b		90.5	244.1	730.8	201.2	0.700	-71.5
2c		77.8	123.6	2591	151.8	0.830	-55.5
2d		81.3	133.6	3497	239.8	0.751	-59.8
2e		79.7	129.0	3147	205.8	0.799	-57.2

5. 4MB File Size

Table 11 displays the overhead results for the last file size, which was increased to 4MB. The overhead for Case 1c decreased from a trend of approximately 43% to more closely match Case 1d and 1e in the low 30% range. Overall, for Case 1d and 1e, as file size increased the ratio of produced overhead from metadata also increased. It increased from 21% message packets up to approximately 33% message packets as file size increased. The Case 2d and Case 2e runs could not be completed since errors kept occurring at this file size, as seen in Figure 15. These errors seemed to be primarily related to not having enough memory to complete the data run. In Figure 15, the right side of the image is from the publisher and the left side is from the subscriber. These errors disappeared at a file size of approximately 3MB. Another key note is that if pauses/delays are added in between message transmissions, then Cases 2d and 2e can achieve 100% message delivery. The cost here is the large increase in latency (2000 μ s for each message) for all file sizes. Case 2c also had a large increase in lost messages. The assumption is that this is caused by the large file size and unknown memory allocation issues.

Table 11. 4MB overhead results

Case	File Size (MB)	Total Packets	MSG Frag Packets	Overhead Packets (%)	MSGs Lost
1a	4	58853	58815	0.06	30
1b		53107	53062	0.08	42
1c		95416	61506	35.54	31
1d		92246	61560	33.27	0
1e		90064	60388	32.95	0
2a	4	60590	60536	0.09	30
2b		60674	60620	0.09	46
2c		64588	63426	1.80	263
2d		**	**	**	**
2e		**	**	**	**

**No results due to errors received during the experiment run

```

[INFO] [listener]: Received: 60
2019-04-26 15:19:18.407 [SECURITY_CRYPT Error] Not enough memory to serialize S
ecureDataTag -> Function encode_rtps_message
2019-04-26 15:19:18.407 [RTPS_WRITER Error] Error encoding rtps message. -> Func
tion send
2019-04-26 15:19:18.415 [SECURITY_CRYPT Error] Not enough memory to serialize S
ecureDataTag -> Function encode_rtps_message
2019-04-26 15:19:18.415 [RTPS_WRITER Error] Error encoding rtps message. -> Func
tion send
2019-04-26 15:19:18.423 [SECURITY_CRYPT Error] Not enough memory to serialize S
ecureDataTag -> Function encode_rtps_message
2019-04-26 15:19:18.423 [RTPS_WRITER Error] Error encoding rtps message. -> Func
tion send
2019-04-26 15:19:18.431 [SECURITY_CRYPT Error] Not enough memory to serialize S
ecureDataTag -> Function encode_rtps_message
2019-04-26 15:19:18.432 [RTPS_WRITER Error] Error encoding rtps message. -> Func
tion send
2019-04-26 15:19:18.440 [SECURITY_CRYPT Error] Not enough memory to serialize S
ecureDataTag -> Function encode_rtps_message
2019-04-26 15:19:18.440 [RTPS_WRITER Error] Error encoding rtps message. -> Func
tion send
2019-04-26 15:19:18.448 [SECURITY_CRYPT Error] Not enough memory to serialize S
ecureDataTag -> Function encode_rtps_message
2019-04-26 15:19:18.448 [RTPS_WRITER Error] Error encoding rtps message. -> Func
tion send
[INFO] [talker]:
Sending: '1000'
Done
2019-04-26 15:17:58.727 [SECURITY Error] Cannot find local writer 9a.dd.34.40.e7
.49.bf.e8.8.64.a8.87|0.0.1b.3 -> Function encode_writer_submessage
2019-04-26 15:17:58.727 [RTPS_WRITER Error] Cannot encrypt HEARTBEAT submessage
for writer 9a.dd.34.40.e7.49.bf.e8.8.64.a8.87|0.0.1b.3 -> Function add_heartbeat
2019-04-26 15:17:58.727 [SECURITY Error] Cannot find local writer 9a.dd.34.40.e7
.49.bf.e8.8.64.a8.87|0.0.1b.3 -> Function encode_writer_submessage
2019-04-26 15:17:58.727 [RTPS_WRITER Error] Cannot encrypt DATA submessage for w
riter 9a.dd.34.40.e7.49.bf.e8.8.64.a8.87|0.0.1b.3 -> Function add_info_ts_in_buf
fer
2019-04-26 15:17:58.727 [SECURITY Error] Cannot find local writer 9a.dd.34.40.e7
.49.bf.e8.8.64.a8.87|0.0.1b.3 -> Function encode_serialized_payload
2019-04-26 15:17:58.727 [RTPS_WRITER Error] Error encoding change 613 -> Funcio
n add_data_frag
2019-04-26 15:17:58.727 [RTPS_WRITER Error] Error sending fragment (613, 6) -> F
unction send_any_unsent_changes
2019-04-26 15:17:58.727 [SECURITY Error] Cannot find local writer 9a.dd.34.40.e7
.49.bf.e8.8.64.a8.87|0.0.1b.3 -> Function encode_writer_submessage
2019-04-26 15:17:58.727 [RTPS_WRITER Error] Cannot encrypt DATA submessage for w
riter 9a.dd.34.40.e7.49.bf.e8.8.64.a8.87|0.0.1b.3 -> Function add_info_ts_in_buf
fer
2019-04-26 15:17:58.727 [SECURITY Error] Cannot find local writer 9a.dd.34.40.e7
.49.bf.e8.8.64.a8.87|0.0.1b.3 -> Function encode_writer_submessage

```

Figure 15. Publisher and subscriber errors for Case 2e and 4MB file size

In Table 12, Cases 1b and 2b maintain the same trend of having the worst performance metrics compared to the other eight cases in the 4MB test run. Throughput continues to approach a single value for the secure data with the exception of Case 2d and Case 2e since these two cases could not be fully analyzed due to errors.

Table 12. 4MB performance results

Case	File Size (MB)	MSG Frag Latency (μs)	Δ%	MSG Latency (μs)	Δ%	MSG Throughput (Gbps)	Δ%
1a	4	41.0	-	2417	-	1.586	-
1b		56.8	38.5	3043	25.9	1.176	-25.7
1c		43.7	6.6	2818	16.6	1.501	-5.4
1d		45.2	10.2	2710	12.1	1.422	-10.3
1e		44.9	9.5	2857	18.2	1.478	-6.8
2a	4	77.0	87.8	4490	85.8	0.817	-48.5
2b		90.9	121.7	5720	136.7	0.739	-53.4
2c		80.6	96.6	5161	113.5	0.816	-48.5
2d		**	**	**	**	**	**
2e		**	**	**	**	**	**

**No results due to errors received during the experiment run

C. SUMMARY OF ANALYSIS

Cases 1b and 2b had a performance poorer than all cases, especially Cases 1e and 2e, even though the only difference between all the Cases b and e were the Reliability settings set to BEST_EFFORT and RELIABLE. This was unexpected since BEST_EFFORT was anticipated to outperform RELIABLE cases for History and Depth settings. Another key takeaway is that for the secure Cases 1c-1e, message losses occurred until the message size reached approximately 1.25MB, then retransmissions started happening. The only way to achieve secure message retransmissions when RELIABLE is set, was to add a small pause during individual message transmissions. This is highly undesirable since this adds to overall message latency and lowers throughput.

To obtain a clearer picture of how the performance metrics compared to one another, plots were generated to include all ten cases and all file sizes. Figure 16, 17, and 18 were generated from data obtained from all five performance table results for latencies and throughput. From Figure 16, it can be seen that Case 1b and Case 2b continually had the worst performance for latency in packet fragment transmissions. Looking at Figure 17, Case 1b continues this trend for performance but the Case 2b poor performance is

surpassed by Cases 2d and 2e. The performance of Cases 2d and 2e dropped due to the large amount of message retransmissions to obtain 100% message delivery at 2MB.

As message size increases so does the cost in terms of latency. The cost seems to be consistent with plaintext data and varies slightly with secure data. With plaintext data, it appears that a smaller message size will yield better performance metrics. With secure data, after the 2MB file size, the cost appears to be the same, but the user is more limited on the capabilities of the hardware (memory size) as was seen from the errors received at the 4MB file size in Figure 15.

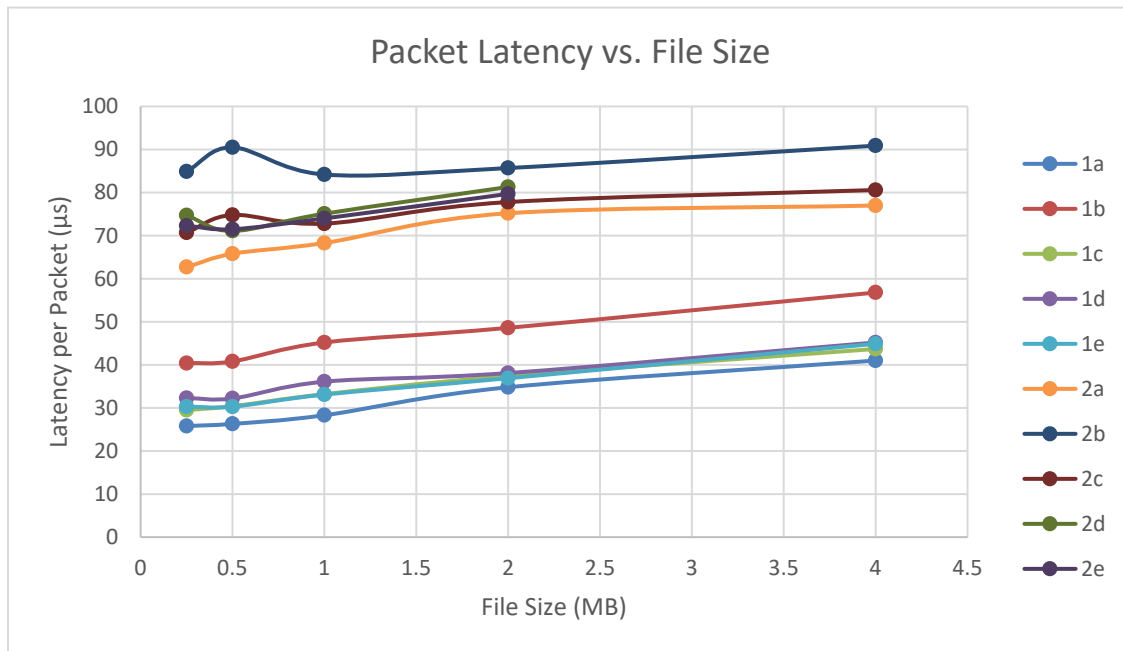


Figure 16. Packet Latency vs. File Size plot for all cases

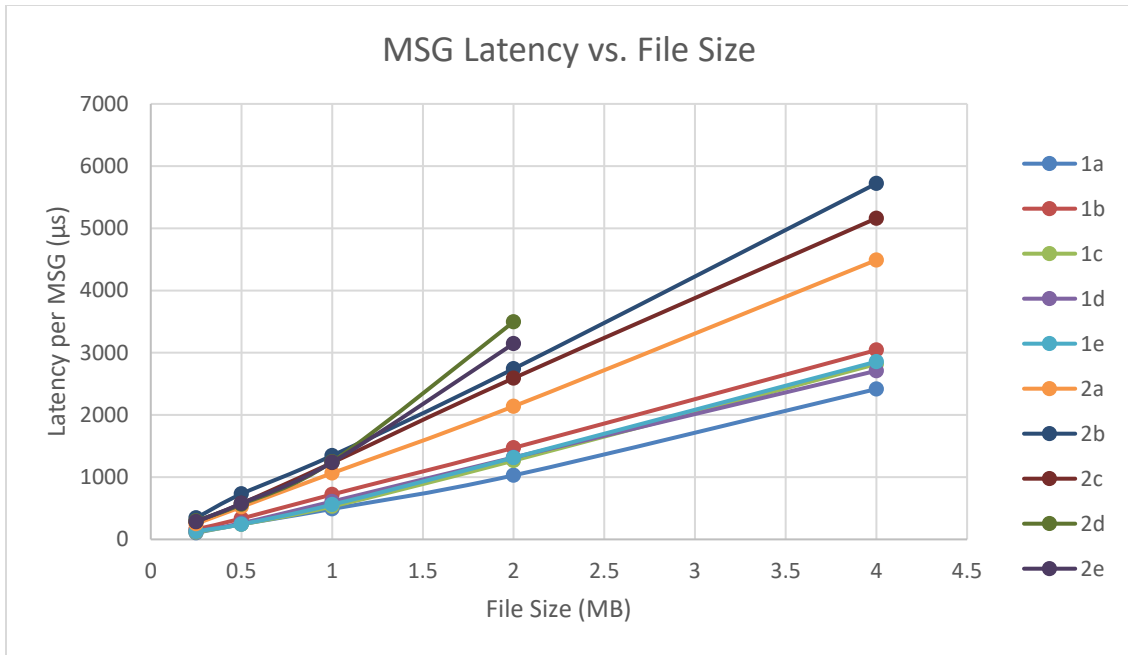


Figure 17. MSG Latency vs. File Size plot for all cases

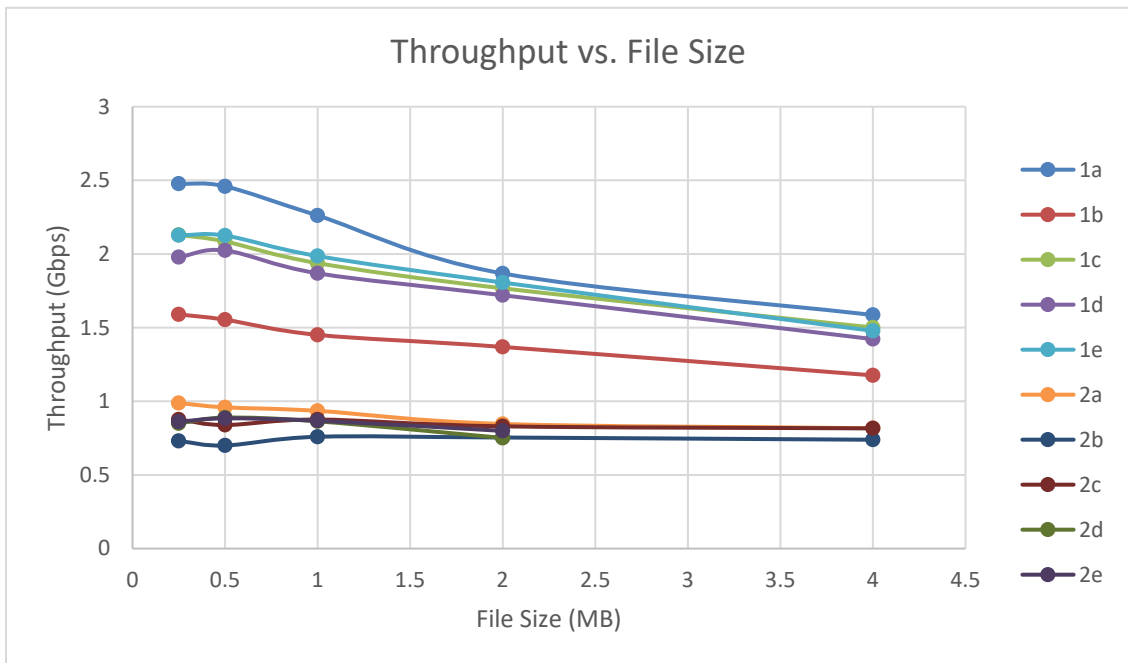


Figure 18. Throughput vs. File Size for all cases

V. CONCLUSION

This thesis does not provide specific recommendations of how to utilize ROS 2 but instead provides numerical comparisons of different cases relative to a baseline case. The results will help developers and users choose appropriate settings, based on their needs, so that they can maintain effective and efficient communications between nodes. Looking at ROS 2 performance as it relates to latency, throughput, and overhead has provided detailed insight on what tradeoffs are sacrificed as data is secured or different QoS profiles are used.

A. SUMMARY

The use of UxS will continue to grow as a platform from which to conduct or support military operations. The increased usage of UxS in warfare centric environments will enhance the necessity to continue to find methods to accurately cyber harden these systems. The growth in the complexity and the amount of data that is either sent or received by these systems will be an ongoing problem now and into the future, requiring new and innovative solutions. In this thesis, we demonstrated the performance and capabilities of using ROS 2 in a Linux based environment. The ability of ROS 2 to use well defined security plugin protocols and apply QoS profiles to data transfers was verified.

Through our experimental setup, the capabilities and performance of ROS 2 were demonstrated through 50 different scenarios. The versatility of how the DDS security plugins can be applied is promising for use in actual warfare environments. The introduction of QoS profiles and the ease of using these profiles helps address some of the data integration and transport issues discussed in [1]. Costs (latency, throughput, and overhead) displayed in the results, will help future researchers to better model/setup their experiments to generate more desirable results. In the future, ROS 2 is expected to include many more capabilities than it currently has. Therefore, continued researched on the effectiveness of ROS 2 with DDS in DoD UxS is required.

B. FUTURE WORK

(1) Lossy Network

Throughout the simulations, there were no simulated losses. For example, the signal to noise ratio (SNR) can be varied for a wireless channel. The experimental runs in this thesis can be repeated on a lossy network. This could provide further data points on how well the QoS profiles or parameters perform.

(2) Network Size

The experiments in this thesis can be conducted on a more complex network that have multiple publishing and/or subscribing nodes. Multiple topics can also be used to continue to explore the performance of ROS 2 with DDS.

(3) Testing with Threat Vectors

During our experiments, no outside factors were introduced to test the effectiveness of the security plugins. There are multiple network and security vulnerabilities that can be explored in order to determine if the ROS 2 DDS security features are sufficient to use. Testing threat vectors can also help substantiate how performance is affected in a network under attack.

(4) Adaption to a Real-Life Environment

This research should be tested using a real-life environment with the use of UxS. As shown in Chapter III, section A, the desktop used to conduct all simulations is a high performance machine. The typical drone that would be used would not have the same computational, memory, or power performance. Expanding this research to UxS to verify the results obtained would be advantageous to all parties.

APPENDIX A. ACCESS CONTROL YAML CODE

The following YAML script is used to generate access controls for the two participant setup in the experiment.

```
nodes:           # list of allowed nodes
  listener:      # name of subscriber node
    topics:      # list of allowed topics for listener
      chatter:   # name of topic
        allow: s  # only subscribe allowed
  talker:
    topics:
      chatter:
        allow: p  #only publish allowed
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. PUBLISHER SCRIPT

The following publisher code is used to generate a publishing node named “talker” and generates and sends 1000 messages to a topic named “chatter”. The message sizes adjustable by changing the size of the “s” string data. This code was modified from the ROS 2 talker/listener example.

```
-----  
#include <chrono>  
#include <cstdio>  
#include <memory>  
#include <string>  
#include <stdlib.h>  
#include <time.h>  
#include <unistd.h>  
#include <stdio.h>  
#include <iostream>  
#include "rclcpp/rclcpp.hpp"  
#include "rcutils/cmdline_parser.h"  
#include "std_msgs/msg/string.hpp"  
  
using namespace std::chrono_literals;  
void print_usage()  
{  
    printf("Usage for talker app:\n");  
    printf("talker [-t topic_name] [-h]\n");  
    printf("options:\n");  
    printf("-h : Print this help function.\n");  
    printf("-t topic_name : Specify the topic on which to publish. Defaults to chatter.\n");  
}  
//Defining QoS profiles to be used throughout experiment  
static const rmw_qos_profile_t qos_profile_1 =  
{  
    RMW_QOS_POLICY_HISTORY_KEEP_LAST,  
    5,  
    RMW_QOS_POLICY_RELIABILITY_BEST_EFFORT,  
    RMW_QOS_POLICY_DURABILITY_VOLATILE,  
    false  
};  
  
static const rmw_qos_profile_t qos_profile_2 =  
{
```

```

    RMW_QOS_POLICY_HISTORY_KEEP_ALL,
    1,    //Ignore, cannot have a bank field
    RMW_QOS_POLICY_RELIABILITY_BEST_EFFORT,
    RMW_QOS_POLICY_DURABILITY_TRANSIENT_LOCAL,
    false
};

static const rmw_qos_profile_t qos_profile_3 =
{
    RMW_QOS_POLICY_HISTORY_KEEP_LAST,
    5,
    RMW_QOS_POLICY_RELIABILITY_RELIABLE,
    RMW_QOS_POLICY_DURABILITY_VOLATILE,
    false
};

static const rmw_qos_profile_t qos_profile_4 =
{
    RMW_QOS_POLICY_HISTORY_KEEP_LAST,
    1000,
    RMW_QOS_POLICY_RELIABILITY_RELIABLE,
    RMW_QOS_POLICY_DURABILITY_VOLATILE,
    false
};

static const rmw_qos_profile_t qos_profile_5 =
{
    RMW_QOS_POLICY_HISTORY_KEEP_ALL,
    1,    //ignore, cannot have a bank field
    RMW_QOS_POLICY_RELIABILITY_RELIABLE,
    RMW_QOS_POLICY_DURABILITY_TRANSIENT_LOCAL,
    false
};

// Create a Talker class that subclasses the generic rclcpp::Node base class.
// The main function below will instantiate the class as a ROS node.
class Talker : public rclcpp::Node
{
public:
    explicit Talker(const std::string & chatter)
    : Node("talker")
    {
        msg_ = std::make_shared<std_msgs::msg::String>();
        for (int i=0; i<=14250; ++i)
        {

```

```

s<<"Test Packets "<< i;
}
s<<"End";

packetdata = s.str();

// Create a function for when messages are to be sent.
auto publish_message =
[this]() -> void
{
    auto string_msg = std::make_shared<std_msgs::msg::String>();
    string_msg->data = std::to_string(count_);

    count_++;
    msg_->data = packetdata;

    RCLCPP_INFO(this->get_logger(), "");

    // Put the message into a queue to be processed by the middleware.
    // This call is non-blocking.
    pub_->publish(msg_);

    printf("Sending: '%s'\n", string_msg->data.c_str());

    if (count_ == 1001)
    {
        for (int c=0; c<= 10000; ++c)
        {
            for(int d=1; d<= 10000; ++d)
            {}
        }
        printf("Done\n");
        //add short delay to give time for buffer to finish transmitting
        //prior to node shutdown
        usleep(2500000);
        rclcpp::shutdown();
    }
};

// Create a publisher with a custom Quality of Service profile.
pub_ = this->create_publisher<std_msgs::msg::String>(
    chatter, qos_profile_5);

// Use a timer to schedule periodic message publishing.
timer_ = this->create_wall_timer(0s, publish_message);

```

```

}

private:
    size_t count_ = 1;
    std::shared_ptr<std_msgs::msg::String> msg_;
    rclcpp::Publisher<std_msgs::msg::String>::SharedPtr pub_;
    rclcpp::TimerBase::SharedPtr timer_;
    std::stringstream s;
    std::string packetdata;
};

int main(int argc, char * argv[])
{
    // Force flush of the stdout buffer.
    // This ensures a correct sync of all prints
    // even when executed simultaneously within the launch file.
    setvbuf(stdout, NULL, _IONBF, BUFSIZ);

    if (rcutils_cli_option_exist(argv, argv + argc, "-h")) {
        print_usage();
        return 0;
    }

    // Initialize any global resources needed by the middleware and the client library.
    // You must call this before using any other part of the ROS system.
    // This should be called once per process.
    rclcpp::init(argc, argv);

    // Parse the command line options.
    auto topic = std::string("chatter");
    char * cli_option = rcutils_cli_get_option(argv, argv + argc, "-t");
    if (nullptr != cli_option) {
        topic = std::string(cli_option);
    }

    // Create a node.
    auto node = std::make_shared<Talker>(topic);

    // spin will block until work comes in, execute work as it becomes available, and keep
    blocking.
    // It will only be interrupted by Ctrl-C.
    rclcpp::spin(node);
    rclcpp::shutdown();
    return 0;
}

```

APPENDIX C. SUBSCRIBER SCRIPT

The following subscriber code was used to generate a node named “listener” that subscribes to a topic named “chatter”. A counter is added to determine how many messages are received on the subscribing end. This code was modified from the ROS 2 talker/listener example.

```
-----  
#include <cstdio>  
#include <memory>  
#include "rclcpp/rclcpp.hpp"  
#include "std_msgs/msg/string.hpp"  
  
using namespace std::chrono_literals;  
static const rmw_qos_profile_t qos_profile_1 =  
{  
    RMW_QOS_POLICY_HISTORY_KEEP_LAST,  
    5,  
    RMW_QOS_POLICY_RELIABILITY_BEST_EFFORT,  
    RMW_QOS_POLICY_DURABILITY_VOLATILE,  
    false  
};  
  
static const rmw_qos_profile_t qos_profile_2 =  
{  
    RMW_QOS_POLICY_HISTORY_KEEP_ALL,  
    1,    //Ignore, cannot be left blank for KEEP_ALL  
    RMW_QOS_POLICY_RELIABILITY_BEST_EFFORT,  
    RMW_QOS_POLICY_DURABILITY_TRANSIENT_LOCAL,  
    false  
};  
  
static const rmw_qos_profile_t qos_profile_3 =  
{  
    RMW_QOS_POLICY_HISTORY_KEEP_LAST,  
    5,  
    RMW_QOS_POLICY_RELIABILITY_RELIABLE,  
    RMW_QOS_POLICY_DURABILITY_VOLATILE,  
    false  
};  
  
static const rmw_qos_profile_t qos_profile_4 =
```

```

{
    RMW_QOS_POLICY_HISTORY_KEEP_LAST,
    1000,
    RMW_QOS_POLICY_RELIABILITY_RELIABLE,
    RMW_QOS_POLICY_DURABILITY_VOLATILE,
    false
};

static const rmw_qos_profile_t qos_profile_5 =
{
    RMW_QOS_POLICY_HISTORY_KEEP_ALL,
    1, //Ignore, cannot be left bank for KEEP_ALL
    RMW_QOS_POLICY_RELIABILITY_RELIABLE,
    RMW_QOS_POLICY_DURABILITY_TRANSIENT_LOCAL,
    false
};

class ListenerBestEffort : public rclcpp::Node
{
public:
    ListenerBestEffort()
    : Node("listener")
    {
        auto callback =
            [this](const typename std_msgs::msg::String::SharedPtr msg) -> void
            {
                RCLCPP_INFO(this->get_logger(), "Received: " + std::to_string(count_++));
            };

        // Create a publisher with a custom Quality of Service profile.
        sub_ = create_subscription<std_msgs::msg::String>(
            "chatter", callback, qos_profile_5);
    }

private:
    rclcpp::Subscription<std_msgs::msg::String>::SharedPtr sub_;
    size_t count_ = 1;
};

int main(int argc, char * argv[])
{
    // Force flush of the stdout buffer.
    setvbuf(stdout, NULL, _IONBF, BUFSIZ);

    rclcpp::init(argc, argv);

```

```
auto node = std::make_shared<ListenerBestEffort>();  
rclcpp::spin(node);  
rclcpp::shutdown();  
return 0;  
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

- [1] Office of the Secretary of Defense, “Unmanned systems integrated roadmap FY 2017–2042,” Washington, DC, USA. [Online]. Available: https://www.defensedaily.com/wp-content/uploads/post_attachment/206477.pdf
- [2] Defense Science Board, “Task force report: The role of autonomy in DoD systems,” Washington, DC, USA [Online]. Available: <https://fas.org/irp/agency/dod/dsb/autonomy.pdf>
- [3] Open Robotics, “Our services.” Accessed May 1, 2019. [Online]. Available: <https://www.openrobotics.org/>
- [4] S. Sandoval, “Cyber security testing of the robot operating system in unmanned aerial systems,” M.S. thesis, Dept. of Elec. Eng., NPS, Monterey, CA, USA, 2018. [Online]. Available: <http://hdl.handle.net/10945/60458>
- [5] J. Kim, J.M. Smeraka, C. Cheung, S. Nepal, M. Grobler, “Security and performance considerations in ROS 2: A balancing act,” Sep. 24 2018. [Online]. Available: [arXiv:1809.09566v1](https://arxiv.org/abs/1809.09566v1) [cs.CR]
- [6] ROS Index, “ROS2 overview.” Accessed Apr. 29, 2019. [Online]. Available: <https://index.ros.org/doc/ros2>
- [7] ROS 2 Design, “ROS on DDS.” Accessed Apr. 29, 2019. [Online]. Available: <https://design.ros2.org>
- [8] C.S.V Gutierrez, L.U. San Juan, I.Z. Ugarte, V.M. Vilches, “Towards a distributed and real-time framework for Robots: Evaluation for ROS 2.0 communications for real-time robotic applications,” Sep. 7, 2019. [Online]. Available: [arXiv:1809.02595v1](https://arxiv.org/abs/1809.02595v1) [cs.RO]
- [9] EProsima, *FastRTPS Documentation*, Release 1.7.2, 2019. [Online]. Available: <https://readthedocs.org/projects/eprosima-fast-rtps/downloads/>
- [10] EProsima The Middleware Experts, “eProsima Fast RTPS Performance,” Accessed Apr. 20, 2019. [Online]. Available: <https://www.eprosima.com/index.php/resources-all/performance/40-eprosima-fast-rtps-performance>
- [11] Object Management Group. “The real-time publish-subscribe protocol (RTPS) DDS interoperability wire protocol specification Version 2.2,” Sep. 2014. [Online]. Available: <https://www.omg.org/spec/DDSI-RTPS/2.2>

- [12] Y. Maruyama, S. Kato, T. Azumi, “Exploring the performance of ROS2,” *EMSOFT ‘16 Proceedings of the 13th International Conference on Embedded Software Article No. 5*, 10 pages. Oct. 2016. [Online]. DOI: <http://dx.doi.org/10.1145/2968478.2968502>
- [13] M. Arguedas, “SROS 2,” presented at IROS, Madrid, Mar. 2018. [Online]. Available: https://ruffsl.github.io/IROS2018_SROS2_Tutorial/content/slides/SROS2_Basics.pdf
- [14] V. DiLuoffo, W. R. Michalson, B. Sunar, “Robot operating system 2: The need for a holistic security approach to robotic architectures,” *International Journal of Adv. Robotics Sys.* May 3, 2018. [Online]. DOI: 10.1177/1729881418770011
- [15] G. Pardo, R. White, “Leveraging DDS security in ROS2,” presented at ROSCon, Madrid, Sep. 29, 2018. [Online]. Available: https://roscon.ros.org/2018/presentations/ROSCon2018_DDS_Security_in_ROS2.pdf
- [16] Real Time Innovations (RTI), 2018. RTI_Perftest, 2.4. [Online]. Available: <https://github.com/rticommunity/rtiperftest>
- [17] Object Management Group. “DDS security version 1.1,” Jul. 2018. [Online]. Available: <https://www.omg.org/spec/DDS-SECURITY/1.1>
- [18] ROS2, “SROS2.” Accessed May 10, 2019. [Online]. Available: <https://github.com/ros2/sros2>

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California