Secure integration of extremely resource-constrained nodes on distributed ROS2 applications

Giann Nandi*

David Pereira*

José Proença*

Eduardo Tovar*

Aythami Salvador Rodriguez

*CISTER Research Centre

CISTER-TR-230703

2023/07/14

Secure integration of extremely resource-constrained nodes on distributed ROS2 applications

Giann Nandi*, David Pereira*, José Proença*, Eduardo Tovar*, Aythami Salvador Rodriguez

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP P.Porto)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: giann@isep.ipp.pt, drp@isep.ipp.pt, pro@isep.ipp.pt, emt@isep.ipp.pt

https://www.cister-labs.pt

Abstract

Background: modern robots employ artificial intelligence algorithms in a broad ange of applications. These robots acquire information about their surroundings and use these highly-specialized algorithms to reason about their next actions. Despite their effectiveness, artificial intelligence algorithms are highly susceptible to adversarial attacks. This work focuses on mitigating attacks aimed at tampering with the communication channel between nodes running micro-ROS, which is an adaptation of the Robot Operating System (ROS) for extremely resource-constrained devices (usually assigned to collect information), and more robust nodes running ROS2, typically in charge of executing computationally costly tasks, like processing artificial intelligence algorithms.Methods: we followed the instructions described in the Data Distribution Service for Extremely Resource Constrained Environments (DDS-XRCE) specification on how to secure the communication between micro-ROS and ROS2 nodes and developed a custom communication transport that combines the application programming interface (API) provided by eProsima and the implementation of the Transport Security Layer version 1.3 (TLS 1.3) protocol developed by wolfSSL.Results: first, we present the first open-source transport layer based on TLS 1.3 to secure the communication between micro-ROS and ROS2 nodes, providing initial benchmarks that measure its temporal overhead. Second, we demystify how the DDS-XRCE and DDS Security specifications interact from a cybersecurity point of view.Conclusions: by providing a custom encrypted transport for micro-ROS and ROS2 applications to communicate, extremely resource-constrained devices can now participate in DDS environments without compromising the security, privacy, and authenticity of their message exchanges with ROS2 nodes. Initial benchmarks show that encrypted single-value messages present around 20% time overhead compared to the default non-encrypted micro-ROS transport. Finally, we presented an analysis of how the DDS-XRCE and DDS Security specifications relate to each other, providing insights not present in the literature that are crucial for further investigating the security characteristics of combining these specifications.

RESEARCH ARTICLE

Secure integration of extremely resource-constrained nodes on distributed ROS2 applications [version 1; peer review: awaiting peer review]

Giann Spilere Nandi[1], David Pereira[1], José Proença[1], Eduardo Tovar[1], Antonio Rodriguez[2], Pablo Garrido[2]

[1]Informatics Engineering, CISTER Research Centre/ISEP, Porto, Porto, 4200-135, Portugal
[2]eProsima, Tres Cantos, Madrid, 28760, Spain

v1 **First published:** 14 Jul 2023, **3**:113
https://doi.org/10.12688/openreseurope.16108.1

Latest published: 14 Jul 2023, **3**:113
https://doi.org/10.12688/openreseurope.16108.1

Abstract

Background: modern robots employ artificial intelligence algorithms in a broad ange of applications. These robots acquire information about their surroundings and use these highly-specialized algorithms to reason about their next actions. Despite their effectiveness, artificial intelligence algorithms are highly susceptible to adversarial attacks. This work focuses on mitigating attacks aimed at tampering with the communication channel between nodes running micro-ROS, which is an adaptation of the Robot Operating System (ROS) for extremely resource-constrained devices (usually assigned to collect information), and more robust nodes running ROS2, typically in charge of executing computationally costly tasks, like processing artificial intelligence algorithms.
Methods: we followed the instructions described in the Data Distribution Service for Extremely Resource Constrained Environments (DDS-XRCE) specification on how to secure the communication between micro-ROS and ROS2 nodes and developed a custom communication transport that combines the application programming interface (API) provided by eProsima and the implementation of the Transport Security Layer version 1.3 (TLS 1.3) protocol developed by wolfSSL.
Results: first, we present the first open-source transport layer based on TLS 1.3 to secure the communication between micro-ROS and ROS2 nodes, providing initial benchmarks that measure its temporal overhead. Second, we demystify how the DDS-XRCE and DDS Security specifications interact from a cybersecurity point of view.
Conclusions: by providing a custom encrypted transport for micro-ROS and ROS2 applications to communicate, extremely resource-constrained devices can now participate in DDS environments without compromising the security, privacy, and authenticity of their message exchanges with ROS2 nodes. Initial benchmarks show that encrypted

Open Peer Review

Approval Status *AWAITING PEER REVIEW*

Any reports and responses or comments on the article can be found at the end of the article.

single-value messages present around 20% time overhead compared to the default non-encrypted micro-ROS transport. Finally, we presented an analysis of how the DDS-XRCE and DDS Security specifications relate to each other, providing insights not present in the literature that are crucial for further investigating the security characteristics of combining these specifications.

Keywords
security, ros2, micro-ros, embedded systems, robotics, distributed systems, publish-subscribe

This article is included in the Electrical, Electronic and Information Engineering gateway.

Corresponding author: Giann Spilere Nandi (giann@isep.ipp.pt)

Author roles: Spilere Nandi G: Conceptualization, Investigation, Methodology, Software, Writing – Original Draft Preparation, Writing – Review & Editing; **Pereira D**: Conceptualization, Funding Acquisition, Investigation, Project Administration, Supervision, Writing – Original Draft Preparation, Writing – Review & Editing; **Proença J**: Conceptualization, Investigation, Project Administration, Supervision, Writing – Original Draft Preparation, Writing – Review & Editing; **Tovar E**: Funding Acquisition, Project Administration, Supervision; **Rodriguez A**: Conceptualization, Investigation, Software, Validation; **Garrido P**: Conceptualization, Software, Validation

Competing interests: No competing interests were disclosed.

Grant information: This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No [876852](Verification and Validation of Automated Systems' Safety and Security [VALU3S]). This work was also partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UIDP/UIDB/04234/2020); by project Route 25 (ref. TRB/2022/00061-C645463824-00000063), funded by the EU/Next Generation, within call n.º 02/C05-i01/2022 of the Recovery and Resilience Plan (RRP); it was also supported by grant 2022.13599.BD, financed by FCT through the European Social Fund (ESF) and the Regional Operational Programme (ROP) Norte 2020; Disclaimer: This document reflects only the author's view and the Commission is not responsible for any use that may be made of the information it contains
The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

How to cite this article: Spilere Nandi G, Pereira D, Proença J *et al.* **Secure integration of extremely resource-constrained nodes on distributed ROS2 applications [version 1; peer review: awaiting peer review]** Open Research Europe 2023, **3**:113 https://doi.org/10.12688/openreseurope.16108.1

First published: 14 Jul 2023, **3**:113 https://doi.org/10.12688/openreseurope.16108.1

Introduction

Current robotic applications have enhanced their problem-solving capabilities by using state-of-the-art artificial intelligence (AI) algorithms and distributed computation technologies. Computer vision, for example, allows robots to identify objects around them and assess what to do next[1]. Ultra-reliable low-latency communication technologies, like 5G, allow the offloading of heavy computational tasks to the cloud with little to no noticeable delay on data exchanges[2]. Modern robots, especially those with higher degrees of autonomy, rely on data collected through embedded sensors and external nodes to reason about their subsequent actions[3]. Although this is an intricate aspect of endowing robots with autonomous behavior, current literature shows that numerous AI algorithms are highly susceptible to adversarial attacks[4].

Several methods to disrupt the functioning of AI algorithms have been described in the literature[5]. Among them are those aimed at tampering with the algorithms' input data, which may focus on reducing their performance, obtaining specific outputs, or even halting a system's execution altogether. Because of that, guaranteeing that robots comply with security requirements is a growing concern for the future of robotic applications[6].

With the two current versions of the Robot Operating System (ROS/ROS2) being the current *de-facto* platform for robot development[7], we focus our efforts on mitigating attacks that tamper with the communication channel between ROS2 and micro-ROS nodes. micro-ROS is a stripped-down version of ROS2 that ports core functionalities of the fully-fledged version of ROS2 to extremely resource-constrained nodes[8]. While these devices usually sense their environment and perform simple computations, more robust nodes running ROS2 are responsible for dealing with more complex tasks, like the timely processing of AI algorithms.

More specifically, we place our work in the context of distributed ROS2 applications and develop the necessary tools to guarantee that nodes running micro-ROS applications can securely communicate with nodes running ROS2 applications. We accomplish that by mutually authenticating micro-ROS and ROS2 nodes and encrypting their communication using a custom transport based on the well-established security protocol Transport Layer Security 1.3 (TLS 1.3)[9].

The contribution of this paper is two-fold. First, we present the first open-source custom transport based on TLS 1.3 to secure the communication between micro-ROS and ROS2. The proposed transport is publicly available and allows researchers and practitioners to boost the security guarantees of their projects, requiring little to no modifications at the application level. Initial benchmarks show that our proposed transport, compared to the fastest UDP-based transport of micro-ROS, increases the round-trip delay of messages by around 2 ms, which is remarkably low, considering the additional layer of reliability and security that our transport provides.

Second, we demystify how the Data Distribution Service for Extremely Resource Constrained Environments (DDS-XRCE)[10] and DDS Security[11] specifications, which are the basis of micro-ROS and secure ROS2 applications, interact from a cybersecurity point of view. We do it by presenting a comprehensible model that condenses core details of combining both specifications, serving as a reference for applications aiming at end-to-end secure communication between standard DDS/ROS2 and micro-ROS nodes.

The remainder of this document is organized as follows: Section "Motivating example" presents a use case that will be used as a reference throughout this document and illustrates the use of AI in the context of distributed robotic applications. Section "Background" provides background information to contextualize our contributions. Section "Proposed work and discussion" presents and discusses our contributions. Section "Conclusion and future work" presents our concluding remarks and the envisioned next steps for this project. Finally, section "Data and software availability" points the reader to this work's source-code and benchmark data.

Motivating example

AI has been one of the main drivers of innovation in recent years. Its contributions impacted, and continue to impact, several research fields and applications, including computer vision, robotics, and health[12,13]. Although efforts towards porting AI to extremely resource-constrained devices have been made in recent years[14], a large portion of these devices are still not capable of locally processing complex AI algorithms in an acceptable amount of time. One common approach to overcome this limitation is to minimize the work done on the microcontroller and offload the more resource-intensive tasks to more robust distributed nodes.

To exemplify the above scenario and motivate our work, we present a use case inspired by the work of Lourenço[15], illustrated in Figure 1. The illustrated system consists of four entities: `C1`, `R1`, `D1`, and `D2`. `C1` is a wearable device powered by an ESP32 microcontroller responsible for collecting electrocardiogram (ECG) signals and transmitting them to a node `R1` wirelessly. `R1` runs on a Raspberry Pi 4 Model B and is a component of a service robot designed to help the user wearing `C1` on daily tasks, monitor the user's signals for any health anomalies, and inform a set of trusted individuals on the cloud, represented by `D1` and `D2`, about the user's current condition. `R1` comprises AI algorithms that process the signals collected by `C1` to provide:

- **Authentication:** by analyzing the user's ECG signals, `R1` can authenticate and validate the user's identity. This relation between heart-beat signals and one's identity comes from the individual properties associated with every person's heart's unique behavior, analogous to what is done with fingerprints and facial details[15].
- **Health Monitoring:** although unique to every person, ECG signals share common traits that could indicate various body conditions, including heart anomalies, levels of fatigue, and emotional distress[16].

While there are several commercial solutions to secure the communication among `R1` , `D1`, and `D2`, no work in the literature, as far as we are concerned, proposes a secure

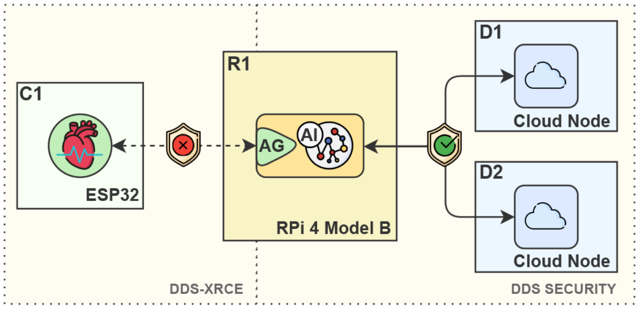

Figure 1. Node C1 collects electrocardiogram signals and transmit them to R1 over a non-secure channel. R1 is responsible for processing it and forwarding the results to a set of trusted nodes in the cloud, represented by D1 and D2. Where RPi = Rasberry Pi and DDS = Data distribution service.

open-source transport for C1 and R1 to communicate. By leaving the channel between C1 and R1 exposed, potential vulnerabilities can be exploited to affect nodes D1 and D2. To address it, we propose a custom transport layer for C1 and R1 to communicate securely.

Background

This section provides the necessary background information for the reader to understand this paper's contribution. It is divided into three subsections, each representing one of the core subjects of this work.

ROS2 and DDS

By providing tools that considerably accelerate the implementation of robotic applications, combined with the immense amount of academic work surrounding it, ROS/ROS2[1] established itself as the *de-facto* framework for robotic software development. Despite the name, ROS is a framework that sits on top of traditional operating systems and provides developers with a set of software libraries and tools for building robot applications.

More than a decade after its initial release, ROS is currently going through a redesign, under the name of ROS2[17], to support the new generation of distributed robotic applications. This section briefly introduces ROS2 and its underlying communication middleware, the data distribution service (DDS). ROS2 follows up on the good facets that made ROS a massive success while also redesigning its core to suit the needs of modern real-time systems and industrial applications. Among the many changes between ROS and ROS2 is the adoption of DDS as the default communication middleware.

DDS is a machine-to-machine communication specification designed and maintained by the Object Management Group (OMG), based on the publish-subscribe communication pattern. Although relatively unknown, DDS is used in several applications[2] and is the foundation of industry standards like the automotive AUTOSAR.[3] DDS differs from other publish-subscribe protocols like MQTT[4] and ZeroMQ[5] by being data-centric and having an extensive quality of service control over its transmitted data. In DDS, participants show interest in data abstracted in the form of topics by subscribing to them (*i.e.,* wanting to receive updates about it) and publishing on them (*i.e.,* sending updates).

Data exchanges in DDS follow a peer-to-peer approach, which avoids the typical central point of failure present in brokered solutions. On top of increasing the system's fault tolerance, peer-to-peer communications exhibit high-performance and low latency, as no intermediate entity is needed for two entities to communicate.

micro-ROS

When considering the importance of microcontrollers in robotics, supporting their integration on ROS2 applications is another crucial step towards simplifying the design of distributed applications. As a result of a European initiative[6], micro-ROS is a framework that provides seamless integration between ROS2 nodes and microcontrollers by porting core functionalities of ROS2 to a lightweight format.

While it is true that ROS2 can run on single-board computers like the Raspberry Pi 4 (Quad-core Cortex-A72 with up to 8GB of RAM), its fully-fledged version certainly cannot be ported to extremely resource-constrained boards[7], like the Espressif ESP32 (ultra-low power dual-core Xtensa LX6 with 520kB of RAM) or the Raspberry Pi Pico (Dual-core Arm Cortex-M0+ with 264kB of RAM). Designed to comply with these constraints, micro-ROS needs no more than a few kilobytes of RAM and can be incorporated in applications based on bare-metal and lightweight real-time operating systems (*e.g.,* NuttX[8], FreeRTOS[9], Zephyr[10]).

Like in ROS2, micro-ROS adopts concepts used in standard DDS communications but adapts them to microcontrollers' harsh working conditions. Most of these adaptations are described in the DDS-XRCE, designed to allow microcontrollers

[1] https://docs.ros.org

[2] https://www.dds-foundation.org

[3] https://www.autosar.org

[4] https://mqtt.org

[5] https://zeromq.org

[6] http://www.ofera.eu

[7] https://docs.ros.org/en/iron/Installation.html

[8] https://nuttx.apache.org

[9] https://www.freertos.org

[10] https://www.zephyrproject.org

to communicate in a DDS domain similarly to how a ROS2 node would.

The DDS-XRCE protocol relies on two core entities: the XRCE-Client and the XRCE-Agent[18]. The XRCE-Client lives in the microcontroller and, through an XRCE-Agent, can participate in DDS networks by publishing on and subscribing to topics. The XRCE-Agent lives on a ROS2 node and acts on the XRCE-Client's behalf. Through message exchanges, XRCE-Clients request XRCE-Agents to perform operations, and XRCE-Agents reply with the result of the said operation. We may refer to XRCE-Clients as Clients and XRCE-Agents as Agents for brevity reasons.

Clients have a pre-defined number of operations they can request an agent to perform. Among these are the operations designed to create DDS entities, which are at the center of this work's contribution. A DDS Entity is the base class for many of the objects that are part of the publish-subscribe scheme of DDS. Below, we succinctly describe some of the entities a client can request an agent to create[11]:

- **Topic:** the main data abstraction used in publish-subscribe communication schemes. DomainParticipants can publish on a topic or subscribe to a topic. Every time a DomainParticipant publishes data on a topic, all Domain-Participants who subscribe to it should receive it;
- **Publisher:** the Entity that holds a set of DataWriters, which is the entity responsible for publishing data on a topic (publishes data on multiple topics);
- **Subscriber:** the entity that holds a set of DataReaders, which is the entity responsible for reading data published on a topic (reads data from multiple topics);
- **DomainParticipant:** the entity responsible for encapsulating a set of Publishers and Subscribers in a DDS domain. A domain is a conceptual representation of the system. Entities in a Domain can only communicate with other entities in the same domain.

By requesting the creation of a DomainParticipant, a client can participate on a DDS domain similarly to how other ROS2 nodes do. The key difference is that while a client depends on an agent to host its DomainParticipant, a standard ROS2 node can do it by itself.

Cyber threats and artificial intelligence algorithms

Despite its effectiveness in problem-solving, AI algorithms have shown to be highly susceptible to numerous sorts of adversarial attacks. These attacks range from tampering with the source of information to altering the algorithm itself[5]. These attacks can impact a targeted system in different ways, including confidence decline, intended misclassification, or even completely halting computation altogether. We use this section to motivate the use of security mechanisms in distributed robotic applications by exemplifying the possible damage caused by successful cyber attacks.

Let us point back to the use case presented in "Motivating example". Figure 2 illustrates a scenario where a malicious participant `M` controls the communication channel used by `C1` and `R1`. If `C1` were to transmit heartbeat signals in the form of plain text in this insecure channel, `M` could directly breach security properties like secrecy, privacy, and data integrity.

Starting with secrecy, by knowing that `R1` uses personal ECG signals to validate the identity of its users, `M` could perform an identity theft attack by playing back to `R1` signals it eavesdropped from previous message exchanges. This attack would lead to `R1` incorrectly authenticating `M` as a legitimate individual, allowing `M` to perform actions under a user's identity and potentially further impact the functioning of `R1`.

Regarding data integrity, `M` could tamper with the data transmitted to `R1` and disrupt the expected behavior of its AI algorithms. For instance, `M` could manipulate the ECG signals and misclassify a healthy heart behavior as abnormal or lead a genuine heart attack to be misclassified as a healthy heart behavior. Data could also be tampered with to reduce the accuracy of an algorithm's output, leading `R1` to behave unpredictably, reducing the trust levels of those using it. When it comes to privacy, `M` could eavesdrop on sensitive health information, leading to the disclosing of a user's health condition. Such data disclosures could be especially troublesome for users who depend on regular medicine ingestion, as ill-intentioned individuals could threaten one's safety.

Proposed work and discussion

To address the above issues, we propose an open-source custom transport communication layer that secures the communication between micro-ROS and ROS2 nodes and complies with the DDS-XRCE specification.

Implementing the custom transport

Our work consists of integrating an encryption protocol into a custom communication transport for XRCE-Clients and XRCE-Agents, providing guarantees of secrecy, privacy, and integrity for their message exchanges while still complying with the DDS-XRCE specification[10]. More specifically, our work combines the lightweight TLS 1.3 library developed and maintained by wolfSSL[12] and the application programming interface (API)[13] provided by eProsima for implementing of custom transports for micro-ROS nodes. Although TLS 1.3 is still not as popular as its predecessor, it offers superior security guarantees and significant performance improvements compared to previous versions[19].

[11] Please refer to the DDS-XRCE specification for more details[10]

[12] https://www.wolfssl.com

[13] https://micro.ros.org/docs/tutorials/advanced/create_custom_transports/

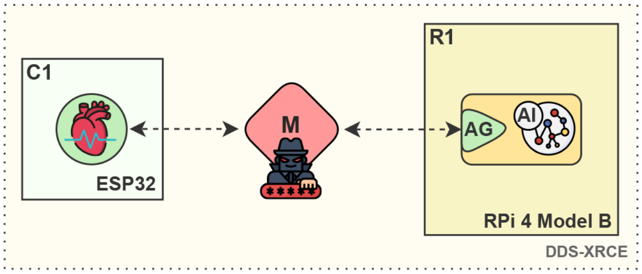

Figure 2. A malicious participant `M` intercepting plain text messages exchanged between `C1` and `R1`.

Custom transports for micro-ROS consist of developing a set of four functions for the micro-ROS client (developed in C[20]) and four functions for the micro-ROS agent (developed in C++[21]). For the sake of better readability, we omit the function parameters and return types of the functions. On the client's side, we developed the following functions:

- `ct_tls_open`: this function is the starting point of the custom transport. It establishes a Transmission Control Protocol (TCP) connection with the agent and executes the handshake protocol described in the TLS 1.3 specification. Although unusual for the majority of TLS 1.3 applications, we enforce that the client authenticates the agent, and that the agent authenticates the client, further strengthening the security guarantees. To do so, it uses a set of keys and certificates that should be provided by the user and stored in the transport's assigned folder[14];
- `ct_tls_write`: once the handshake has been performed and a mutual encryption key for the agent and the client has been established, this function is responsible writing messages in the transport. In the context of this work, writing means encrypting messages using the shared secret derived from the handshake and sending it to the appropriate recipient;
- `ct_tls_read`: similarly to the write function, the read function uses the agreed encryption key to decipher the messages received in the transport. It is important to clarify that our transport encrypts not only the publishing and subscribing messages, but all data communicated between the client and the agent after the successful handshake;
- `ct_tls_close`: finally, this funtion* terminates the transport communication by closing the socket previously created for the transport.

Analogously, the agent also implements the following set of functions:

- `init_function`: this functions accepts connections from clients and performs the handshake protocol described in the TLS 1.3 specification;
- `send_msg_function`: just like in the client's side, this function encrypts data using the common shared key derived from the handshake process and sends data on the communication channel;
- `recv_msg_function`: similarly to the client's read function, this function deciphers messages coming from clients using the respective agreed key;
- `fini_function`: closes the connection and frees any allocated memory used during the message exchanges.

Although not necessarily related to the transport itself, it is important to highlight the roles that clients and agents play in the communication process. While the client runs an application responsible for subscribing to topics and publishing on topics, the agent's main job is to forward such messages between the various participants of a DDS domain. We point the reader to the "Software availability" section[22] of this document for the complete source code for both the client and the agent transports.

Benchmarks

Enhancing systems with security mechanisms inevitably incurs overheads of some kind. Quantifying such overheads is especially relevant in the case of microcontrollers, where computational resources are extremely scarce. As an initial effort to evaluate the impact of our custom transport, we benchmarked a ping-pong application to calculate a message's round-trip delay (RTD).

In the context of micro-ROS, a ping-pong application means having a micro-ROS node publishing messages on a topic it also subscribes to. Pointing back to our use case and considering that `C1` publishes on a topic `T1`, our experiment consists of: `C1` sending a message msg_i , related to topic `T1`, to agent AG; AG processing msg_i and disseminating it to all nodes subscribing to `T1`; `C1` receiving msg_i back as a result of being subscribed to `T1`. The source-code for the ping-pong application is available in the "Data and software availability" section[22] of this document.

To obtain an estimate of the average and minimum RTD achieved by our custom transport, we evaluated the RTD of 4500 sequential messages containing a single incremental value, equally split into 10 runs of the same experiment. More specifically, in each of the 10 experiments, messages containing the message's index were sent, *i.e.,* 1 for the first message, 450 for the last message. We then compared these measurements to the results obtained by performing the same 10 runs of the experiment using the fastest standard transport of micro-ROS (best-effort UDP), which comes by default with every micro-ROS application[15].

[14] https://bitbucket.org/mars-language/micro_ros_tls13/src/master/certs

[15] https://micro.ros.org/docs/tutorials/advanced/microxrcedds_rmw_configuration/

Figure 3 illustrates the well-controlled setup used to obtain the 9000 RTD values resulting from the above-mentioned experiments (4500 UDP + 4500 custom transport). This setup comprises three entities, a micro-ROS node `C1` running on an ESP32, a router that intermediates the communication between `C1` and `R1`, and a ROS2 node `R1` hosting a micro-ROS agent running on a Ubuntu desktop[16]. While `C1` communicates wirelessly with the router, `R1` and the router exchange messages using ethernet. Setup I illustrates the UDP transport setup for micro-ROS[15], and Setup II represents our custom transport, which has an underlying TCP protocol powering the TLS 1.3 security protocol and follows the encryption steps explained in the subsection above.

Figure 4 represents our analysis of the collected data. To calculate the average RTD of each transport, we averaged their 10 samples of the 450 messages sent in each run. Similarly, we calculated the transports' minimum RTD by picking the minimum value among the 10 samples of each of the 450 messages published by `C1`.

Our results show that we can obtain RTDs as low as 10 ms using our custom transport, which is only 2 ms longer than the minimum RTD value obtained with the best-effort UDP transport. This difference is remarkably low, especially when considering that our transport provides data delivery guarantees (given by TCP) and needs to perform four encryption/decryption operations (illustrated in Figure 3): i) `C1` encrypts and sends the mesage; ii) `R1` receives and deciphers the message; iii) `R1` encrypts and sends the message back; iv) `C1` receives and deciphers the returning message. The averaged RTD values followed a similar trend, with the UDP values being mainly in the range of 10 to 12 ms, while our custom transport obtained RTDs predominantly between 12 and 14 ms[23].

Combining the DDS-XRCE and DDS security specifications

The DDS-XRCE and DDS security specifications extensively describe how applications implementing them should behave in their respective contexts. Contrastingly, their combination is not clearly documented in the literature and leaves room for questions only answerable by those with extensive knowledge of their specification and implementation levels. From a cybersecurity point of view, this lack of material addressing the relationship between both specifications makes it challenging to assess how systems built around them are affected by cyber threats.

This section demystifies how these specifications relate by explaining their interaction and presenting a conceptual model that abstracts and condenses practical aspects of their implementation. To achieve that, we further detail the behavior of XRCE-Clients and XRCE-Agents in a DDS domain that supports the Authentication and Access Control plugins described in the DDS security specification[11].

Authentication

Aiming at protecting DDS domains from malicious participants, the DDS security specification requires every

[16] https://ubuntu.com/download/desktop

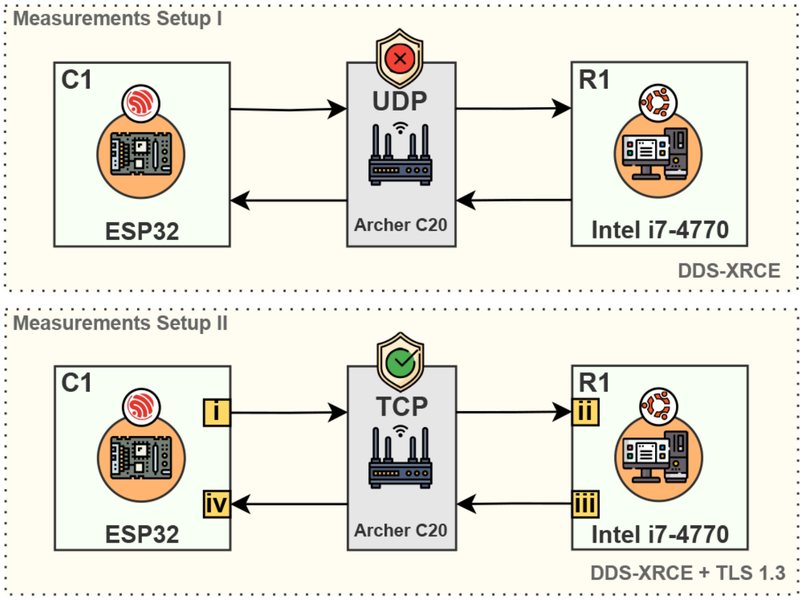

Figure 3. Two benchmark setups evaluate the round-trip delay of a message published on Topic `T1` to return to node `C1`, which is subscribing to `T1`. Measurement Setup I uses an unprotected channel to communicate over User Datagram Protocol (UDP). Measurement Setup II uses our custom transport based on the Transport Security Layer 1.3 (TLS 1.3) over Transmission Control Protocol (TCP) and performs four additional security-related actions.

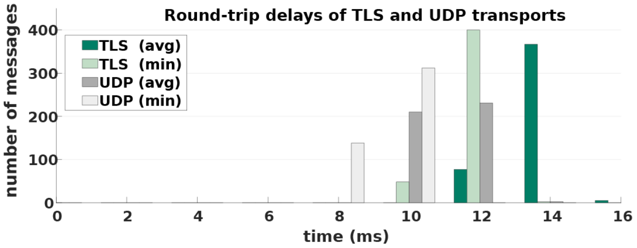

Figure 4. Histogram presenting benchmarks obtained from a best-effort User Datagram Protocol (UDP) transport and a Transport Layer Security version 1.3 (TLS 1.3) transport. Values for average and minimum round-trip delays are presented.

DomainParticipant to go through an authentication process[11]. Successful authentications depend on a set of certificates and public/private keys.[17]

Although these files are mandatory for every DomainParticipant authenticating on a DDS domain, clients are not required to possess them to request the creation of DomainParticipants on their behalf. In fact, the DDS-XRCE specification refrains from describing any security-related functions and configurations of XRCE-Clients. Nonetheless, not having these files does not exempt DomainParticipants associated with XRCE-Clients from authenticating on DDS domains.

Like other computationally costly tasks, the authentication of DomainParticipants is delegated to XRCE-Agents. To understand how this process happens, let us take a step back and first address the creation of a DomainParticipant by an agent. As described in the “Background” section, DomainParticipants encapsulate the means to send and receive data related to a topic. To create a DomainParticipant, a client must send a message to an agent describing the exact profile of the DomainParticipant that will act on his behalf. If this profile is compatible with what the agent can create, the agent creates the DomainParticipant and informs the client about it. Among the settings on this profile is a reference to the set of certificates and keys that the client would like the DomainParticipant to use for authentication purposes.

One of the critical points to be understood here is that clients can only reference files that, in theory, already exist on the agent’s side. Clients do not, in general, know the content of these files, nor can they access it through the XRCE protocol. In other words, clients are agnostic to the content of the certificates and keys used to authenticate their DomainParticipants on a given DDS domain, but they can reference these files during the creation of the requested DomainParticipant.

Deriving models that comply with the above mentioned constraints is possible and encouraged. This section presents one compatible configuration, illustrated in Figure 2, that developers could directly adopt in their projects. To illustrate how this setup works, we expand our use case of Section II by adding another micro-ROS node named C2.

Similar to C1, C2 is responsible for monitoring vital signals (blood-related data in this case) and sending them to our ROS2 node R1. To secure their communications, C1 and C2 perform TLS 1.3 handshakes with R1, resulting in communication channels encrypted by the shared secrets KC1AG and KC2AG, respectively. Once clients and agent establish a secure channel, C1 and C2 can request agent AG to create the DomainParticipants P1 and P2 to act on their behalf.

In this generic configuration, AG contains only one set of certificates and keys that all the DomainParticipants it hosts should share. In practice, that means P1 and P2 will use the same certificates and keys to authenticate on the DDS Domain and establish shared secrets with other DomainParticipants. Despite sharing the security files provided by AG, P1 and P2 authenticate themselves individually to other DomainParticipants. To illustrate this behavior, our generic model represents the P1 and D1 mutually authenticating each other and P2 mutually authenticating D1 and D2.

As illustrated in Figure 5, the above behavior results in P1 and D1 encrypting their communication with a shared secret KD1P2, P2 and D1 encrypting their communication with a shared secret KD1P2, and finally, P2 encrypting its communication with D2 with a shared secret KD2P2. It is important to stress that the shared secrets KD1P1 and KD1P2 do not necessarily need to be the same, as the output of handshake protocols depends on more than just the public and private keys used during the process.

One key aspect that needs to be clear about this model is that two types of authentication are happening. The first one takes place between clients and agents through our custom transport. Although unusual to most TLS 1.3 applications, authenticating clients to agents is crucial to increase the trust in all the DomainParticipants who enter a DDS domain. The second authentication happens between the DomainParticipants

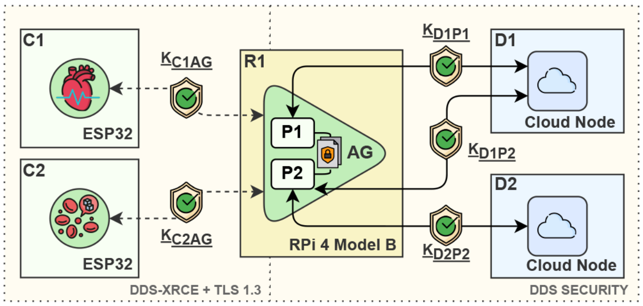

Figure 5. Generic model illustrating how eXtremely Resource Constrained Environments Clients (XRCE-Clients) C1 and C2 have their respective DomainParticipants authenticated by standard Data Distribution Service (DDS) nodes D1 and D2.

[17] Generating and managing these certificates and keys is a problem of its own and is out of this document’s scope.

hosted by agents and other DomainParticipants, like nodes D1 and D2.

The main difference between these two authentication processes is that while clients have individual keys and certificates to authenticate themselves to an agent, the DomainParticipants hosted by that agent share a set of keys and certificates to authenticate themselves to other DomainParticipants. It is essential to mention that, regardless of P1 and P2 sharing, or not, keys and certificates, AG would still need to know the content of these files to act on behalf of clients C1 and C2. As previously mentioned, the above model is just one possible abstraction of the specifications. For instance, one could create another model where *AG* provides unique keys and certificates to authenticate its hosted DomainParticipants, limiting the number of clients allowed to connect to the Agent to only those knowing what to reference in the first place. Analyzing the pros and cons of various possible models is something we wish to address in future works but is out of the scope of this publication.

Access control

While the Authentication plugin aims to guarantee that only trusted DomainParticipants enter a DDS domain, the Access Control plugin aims to enforce that trusted DomainParticipants perform only the set of actions they are allowed to perform.

Similar to the authentication of DomainParticipants, the access control permissions of XRCE-Clients on a DDS Domain are also dependent on their respective XRCE-Agents. From a cyber security point of view, the critical point to be clarified here is the boundaries of what a client can request an Agent to do on its behalf. The DDS-XRCE specification currently limits the client's requests to permissions associated with the agent it connects.

In practice, this means that the permissions of a client in a Domain are a subset of the permissions of the agent it connects. That is, a client can have as many permissions as the host of its DomainParticipants. More accurately, the client's permissions on a given domain are the permissions associated with the security settings referenced during the creation of its DomainParticipant.

Conclusion and future Work

This work briefly reviewed the impact of cyber attacks in distributed robotic applications and presented a custom transport for mitigating cyberattacks that target the communication channel between micro-ROS and ROS2 nodes. Our transport uses TLS 1.3 to authenticate and encrypt its messages and presents a remarkably low temporal overhead.

On top of that, we explained how the DDS-XRCE and DDS Security specifications interact from a cyber security point of view, providing researchers and developers with a comprehensible model to serve as a reference.

In future iterations of this work, we plan on i) optimizing the current code for better performance; ii) deriving a set of possible DDS-XRCE + DDS Security models; iii) performing formal verifications on these models to understand their pros and cons.

Ethics and consent

Ethical approval and consent were not required.

Data and software availability

This section points the reader to the data collected during the benchmark experiments and the source code needed to replicate the experiments and apply the custom transport in their own micro-ROS/ROS2 applications.

Underlying data

HarvardDataverse: Repository: Round-trip delays for micro-ROS transports. https://doi.org/10.7910/DVN/ZXDBX0[23].

This project contains the following underlying data:

- ***rtd.xlsx:*** spreadsheet file containing a total of 20 round-trip delay benchmark experiments. The *"UDP"* sheet stores 10 experiments, showing the timestamp of when the message left, and the timestamp of when the message returned. The *"TLS"* sheet follows this same structure. Finally, the *"SUMMARY"* sheet presents a frequency table, which was used to plot the histogram of Figure 4.

Data are available under the terms of the Creative Commons Zero "No rights reserved" data waiver (CC0 1.0 Public domain dedication).

To replicate the experiments and compare its results to the RTD we obtained, start of by arranging the Measurement Setup II illustrated in Figure 3. Next, run the agent application included with the provided agent's docker image. Third, configure the WiFi parameters and the agent's IP on the ping-pong client application before flashing it to the required ESP32 board. Finally, connect the ESP32 *via* USB to the machine used to flash the application and run the command to read the serial output printed by the board. No further coding or configuration is needed, as the client is pre-configured to continuously individually publish integer values to a topic it also subscribes to. Details on how to perform each of these actions are provided in the link to the Docker repository in the "Software availability" section

Software availability

Software available from: https://hub.docker.com/repository/docker/gcncister/mars/general

Client's source code available from: https://bitbucket.org/mars-language/micro_ros_client_tls13

Agent's source code available from: https://bitbucket.org/mars-language/micro_ros_agent_tls13

Archived source code at the time of publication: https://doi.org/10.5281/zenodo.8072419[22]

License: GPL-3.0

As described in previous sections, this work presents a custom transport developed for micro-ROS and ROS2 applications to communicate securely. To do so, two network participants are required: the Client and the Agent. We point the reader to the work's Docker repository (https://hub.docker.com/repository/docker/gcncister/mars/general), which contains pre-configured Docker images for those who wish to test them and adapt it to their own applications. Instructions on how to compile and execute can be found on the given repository, together with the source code for the custom transport developed for both the Client and the Agent and the ping-pong application used for benchmarking.

Acknowledgments

All figures in this document were designed using icons made by Smashicons, Freepik, monkik, and Pixel perfect from www.flaticon.com.

References

1. Iscimen B, Atasoy H, Kutlu Y, *et al.*: **Smart robot arm motion using computer vision.** *Elektronika ir Elektrotechnika.* 2015; **21**(6): 3–7.
   Publisher Full Text
2. Voigtlander F, Ramadan A, Eichinger J, *et al.*: **5g for robotics: Ultra-low latency control of distributed robotic systems.** In: *2017 International Symposium on Computer Science and Intelligent Controls (ISCSIC).* IEEE, 2017.
   Publisher Full Text
3. Alatise MB, Hancke GP: **A review on challenges of autonomous mobile robot and sensor fusion methods.** IEEE Access, 2020; **8**: 39830–39846.
   Publisher Full Text
4. Akhtar N, Mian A: **Threat of adversarial attacks on deep learning in computer vision: A survey.** *IEEE Access*. 2018; **6**: 14410–14430.
   Publisher Full Text
5. Qiu S, Liu Q, Zhou S, *et al.*: **Review of artificial intelligence adversarial attack and defense technologies.** *Appl Sci.* 2019; **9**(5): 909.
   Publisher Full Text
6. Fosch-Villaronga E, Mahler T: **Cybersecurity, safety and robots: Strengthening the link between cybersecurity and safety in the context of care robots.** *Comput Law Secur Rev.* 2021; **41**: 105528.
   Publisher Full Text
7. Albonico M, Đorđević M, Hamer E, *et al.*: **Software engineering research on the robot operating system: A systematic mapping study.** *J Syst Softw.* 2023; **197**: 111574.
   Publisher Full Text
8. Staschulat J, Lange R, Dasari DN: **Budget-based real-time executor for micro-ros.** *arXiv.* preprint arXiv: 2105.05590, 2021.
   Publisher Full Text
9. Dowling B, Fischlin M, Günther F, *et al.*: **A cryptographic analysis of the TLS 1.3 handshake protocol.** *J Cryptol.* 2021; **34**(4): 37.
   Publisher Full Text
10. Object Management Group: **DDS For Extremely Resource Constrained Environments 1.0.** 2020.
    Reference Source
11. Object Management Group: **DDS Security Specification Version 1.1**.
    Reference Source
12. Shinde PP, Shah S: **A review of machine learning and deep learning applications.** In: *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA).* IEEE, 2018.
    Publisher Full Text
13. Sünderhauf N, Brock O, Scheirer W, *et al.*: **The limits and potentials of deep learning for robotics.** *Int J Robot Res.* 2018; **37**(4–5): 405–420.
    Publisher Full Text
14. Alongi F, Ghielmetti N, Pau D, *et al.*: **Tiny neural networks for environmental predictions: An integrated approach with miosix.** In: *2020 IEEE International Conference on Smart Computing (SMARTCOMP).* IEEE, 2020.
    Publisher Full Text
15. Lourenço A, Silva H, Fred A: **Unveiling the biometric potential of finger-based ECG signals.** *Comput Intell Neurosci.* 2011; **2011**: 1–8.
    PubMed Abstract | **Publisher Full Text** | **Free Full Text**
16. Lourenço A, Alves AP, Carreiras C, *et al.*: **CardioWheel: ECG biometrics on the steering wheel.** In: *Machine Learning and Knowledge Discovery in Databases,* Springer International Publishing, 2015; 267–270.
    Publisher Full Text
17. Macenski S, Foote T, Gerkey B, *et al.*: **Robot operating system 2: Design, architecture, and uses in the wild.** *Sci Robot.* 2022; **7**(66): eabm6074.
    PubMed Abstract | **Publisher Full Text**
18. Solpan S, Kucuk K: **DDS-XRCE standard performance evaluation of different communication scenarios in IoT technologies.** *EAI Endorsed Trans IoT.* 2022; **8**(4): e1.
    Publisher Full Text
19. Kobeissi N: **Formal verification for real-world cryptographic protocols and implementations.** Theses, Université Paris sciences et lettres, 2018.
    Reference Source
20. Kernighan BW, Ritchie DM: **The c programming language.** 2002.
21. Josuttis NM: **The c++ standard library: a tutorial and reference.** 2012.
    Reference Source
22. Spilere Nandi G, Pereira D, Proença J, *et al.*: **Custom Transport for micro-ROS Based on TLS 1.3.** [Source code], Zenodo.2023.
    http://www.doi.org/10.5281/zenodo.8072420
23. Spilere Nandi G: **Round-trip delays for micro-ROS transports.** [Data], Harvard Dataverse, V1, 2023.
    http://www.doi.org/10.7910/DVN/ZXDBX0