Masters Student Internship Proposal
ROS 2 using Fast DDS middleware

Chair of Communication Networks,
Department of Electrical and Computer Engineering
Technical University of Munich

Supervisor: Yash Deshpande, M.Sc. – LKN Department
Student: Sara Rustemi, 03750820, LRZ ID: ge26hec
Place, Date: Munich, 17/06/2023

Reason for the internship

ROS is an open-source, meta-operating system for building robot applications. It provides the services to develop a complex system using existing solutions for small problems. It also provides services like hardware abstraction, message-passing between processes and package management. The Robot Operating System (ROS) is a framework and set of tools and libraries that provide functionality across multiple computers. The key feature of ROS is that it allows you to design complex software without knowing for certain how does the hardware work. The way that the software runs and communicates, makes it very easy for users. ROS is a very important and necessary topic because it provides functionality for device drivers, hardware abstraction, tools for testing and visualization and communication between processes over multiple machines. It connects a network of processes(nodes), which run on multiple devices, with a central hub.

Learning objectives

Through my internship I want to learn about ROS2, why it is important to know and use it, what are the areas that ROS covers and Fast DDS, which is a middleware for ROS2. What is DDS and its conceptual model. Get to know more about Fast DDS and its architecture in order to use it better when applying to ROS2. The main reason for doing this topic is to find the latency between writer and receiver, end to end communication. Also need to find the latency in each layer which are DDS layer, Transport layer and RTPS layer. To achieve this goal, will set up the writer and receiver part, modify the code and record the latency in the parts that we are interested in.
**Internship assignment**

- Read and learn about Fast DDS
- Its layer model application and the connection between Fast DDS and ROS2.
- Have an insight in the languages that both ROS2 and DDS use, which mainly are C++ and Python.
- Will work on writing codes in order to achieve our goals.
- After having a good knowledge and enough information on Fast DDS, will start working with ROS2 using the middleware.
- Will connect some ROS application at the end in order to find the latency in different types of ROS communication.
- How do different QoS message types translate over different layers of ROS middleware, Fast DDS.

**Previous work and research**

The Robot Operating System (ROS) has been growing since 2007 and is still ongoing with the development of ROS2. Since ROS is an open-source framework that helps in building and reusing code between robotics applications, there are a lot of research done and still continue to. There is a global community that work and research for ROS. As far as I have read until now, there are some research and work done in latency analysis of ROS2. There is also research in latency and throughput in ROS2, performance of ROS2, evaluation of the latency of ROS2 with different node configurations, different DDS implementations and different QoS policies. My internship work will focus on finding the latency in different types of ROS communication, end to end latency and latency in each layer.

ROS 2 offers a variety of QoS policies in order to make it possible for multiple nodes to communicate with each other. With this set of Quality of Service policies, ROS 2 can be as reliable as TCP and as best-effort as UDP, unlike ROS 1 which only supported TCP. A set of QoS policies combine to form a QoS profile. QoS profiles can be specified for publishers, subscriptions, clients and service servers. Some important policies are:

- **History**:
  - Keep last: only store up to N samples, configurable via queue depth option.
  - Keep all: store all samples, subject to the configured recourse limits of the DDS vendor.

- **Depth**:
  - Size of the queue: only honored if used together with “keep last”.

- **Reliability**:
  - Best effort: attempt to deliver samples, but may lose them if the network is not robust.
  - Reliable: guarantee that samples are delivered, may retry multiple times.
Durability:

- Transient local: the publisher becomes responsible for persisting samples for “late-joining” subscriptions.
- Volatile: no attempt is made to persist samples.

RAPLET and CARET are two tools that have helped a lot in measuring latency, mainly for publish/subscribe latency. RAPLET, ROS-Aware Publish/Subscribe Latency Evaluation Tool, is designed to measure and visualize the details of the publish/subscribe latency in ROS. It consists of the LD_PRELOAD scheme that inserts function hooks in user-land and the extended Berkeley Packet Filter (eBPF) scheme that monitors the run-queue level and the network states in kernel-land. [1] In the paper “RAPLET: Demystifying Publish/Subscribe Latency for ROS Applications”, they used RAPLET to demystify the source of the latency of the given ROS applications. Also applied RAPLET to the simple Talker application and the real-world Autoware application. They run into the conclusion that the impact of the message size, CPU/network activities, and real-time classes provided by the LINUX kernel on ROS applications can be successfully quantified. However, RAPLET supports only ROS, not ROS 2. On the other hand, CARET is a tool that evaluates the latency of ROS 2 applications. It is capable of measuring the callback latency, node latency, communication time between nodes, and end-to-end latency. CARET can measure the end-to-end latency by combining information on each latency and the message flow. [2] In the paper, “CARET: Chain-Aware ROS 2 Evaluation Tool”, the authors evaluated Autoware.Univers, which is ROS 2-based self-driving software, using CARET. After the measurement, results indicate that the end-to-end latency of Autoware.Univers is approximately 300 ms. There is still a lot of research and development to do in this area.

References
