IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 23 May 2023, accepted 8 June 2023, date of publication 27 June 2023, date of current version 6 July 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3289922

RESEARCH ARTICLE

Toward the Automatic Network Resource Management of Robot Operating System in Programmable Mobile Networks

GÉZA SZABÓ, (Senior Member, IEEE)
Ericsson Hungary Ltd., 1117 Budapest, Hungary
e-mail: geza.szabo@ericsson.com

ABSTRACT This paper explores the potential for making the use of 3rd Generation Partnership Project (3GPP) Service Enabler Architecture Layer (SEAL) TS 23.434 even easier and more automatic in various industry verticals applying Robot Operating System 2 (ROS2) for their operation. The proposed solution involves implementing a mapping node that converts ROS2 settings to 3GPP SEAL requests and an information collecting proxy node. We evaluate the proposed method in various scenarios and deployments, including a novel deployment option that improves the network's action radius and an aid to overcome some limitations of current network deployments. We cover three SEAL functionalities: group management, network resource management (NRM), network monitoring and provide a solution that automatically maps ROS2 application layer information elements to SEAL requests, requiring the ROS2 application developer need only to tag the ROS2 topics for the special Quality of Service (QoS) handling. We demonstrate the feasibility and light-weight nature of the proposed solution. The code of ROS2 nodes is open-sourced and can be found at: (https://github.com/Ericsson/ros2-3gppSA6-mapper). The demo video of the proposed system in action can be seen at: (https://youtu.be/JXGAHvDSU4o).

INDEX TERMS Robot Operating System 2, 3GPP service enabler architecture layer, network resource management.

I. INTRODUCTION

The 5G networks are designed to support new use cases beyond consumer-oriented mobile broadband: critical Machine Type Communication (MTC), and massive MTC. Critical MTC use cases require very low bounded latency and high reliability, while Massive MTC favors enhanced coverage and long device battery lifetime for sensor type use cases involving massive amounts of devices. A single 5G network can support the diverse service quality requirements of mobile broadband, critical MTC and massive MTC use cases.

The 3GPP [3] Service-Based Architecture (SBA) for 5G core (5GC) specifies a functional architecture and standardized interfaces as 5GC control plane Network Functions (NFs) expose Service Based Interfaces (SBIs). The NFs register their services in the Network Exposure Function (NEF) [4] and services can then be discovered by other NFs. This enables flexible deployment, where every NF allows the other authorized NFs to access the services, which provides possibilities for external third parties to use the services and capabilities provided by 5GC. It is expected that 5GC will be deployed on software-defined infrastructure, however, the choice of implementation rests with the operator or vendor.

The traditional way of configuring cellular networks is via their Operations and Maintenance (O&M) interfaces which are command-line tools or based on NetConf [5]. These interfaces can be used by network operators only, hence the factory needs to issue a customer service request (CSR) to the Communications Service Provider (CSP) for each change. Handling of CSRs typically involves several manual steps and hence it is a time-consuming procedure.

The associate editor coordinating the review of this manuscript and approving it for publication was Bilal Khawaja.

Some simpler configuration steps can be automated by the factory using the NEF Application Programming Interfaces (APIs). These are specified by the 3GPP in Technical Specification (TS) 23.501 [6], 23.502 [7] and 29.522 [4]. While NEF provides a low-level programmability of the network, there was room to improve its user or application developer-friendliness and the common set of capabilities for the 5G verticals was identified and the SEAL 3GPP TS 23.434 [8] is introduced. SEAL exposure interface demonstrated a drastically simplified system integration of industrial 5G devices [9]. The motivation during the development of programmable mobile networks and related standards is to provide the vertical application developers an easy-to-use API set that is an enabler to access common network functions for various industrial applications without deep network knowledge.

A. THE MAIN CONTRIBUTIONS OF THE PAPER

Our motivation is the same in this paper and we take one step further and investigate the feasibility to apply SEAL for the industry verticals in an even easier and more automatic way than it is defined in current 3GPP standards. To reach out to a widespread community we design a solution for the users of the ROS2. We implement our solution for ROS2 and FastDDS and evaluate our method in various scenarios and deployments. Our proposed solution consists of an information collecting proxy node and a node performing the mapping from ROS2 settings to 3GPP SEAL requests. We can deploy these components in three significantly different ways, a mode supporting that 1) the vertical application is the consumer of network exposure, 2) the network as the consumer of application exposure and 3) a tunnel mode. The second deployment option is a novel concept to improve the action radius of the network. The third deployment option is an aid to overcome some limitations of current network deployments with support on the application side.

We cover three SEAL functionalities: group management, network resource management and connection monitoring. We provide a solution to map the ROS2 application layer information elements automatically to these requests. The ROS2 application developer's only job is to tag the ROS2 topics that need special 3GPP QoS handling. We prove that our proposed solution is feasible and lightweight. The code is available as a ROS2 package in [1]. The demo video of the proposed system in action can be seen at [2].

B. THE STRUCTURE OF THE PAPER

The rest of the paper is organized as follows. Section II gives a brief overview on network exposure, NRM and ROS2. Section III discusses the base setup when the vertical application is the consumer of network exposure. Section IV introduces the concept of network as a consumer of application exposure. Section V discusses the deployment of the proposed components in a tunnel mode. Section VI evaluates the proposed components and architectures in detail. Section VII concludes the paper.

II. RELATED WORK

The analysis of Industrial Internet of Things (IIoT) use cases and the definition of the requirements of the Operational Technology (OT) industry on 5G network have been done in the 5G Alliance for Connected Industries and Automation (5G-ACIA), with broad participation from the OT and telecommunication industries. The results are documented in a whitepaper [10], which describes the detailed requirements related to device management use cases, network management use cases, and security aspects. The whitepaper serves as an input to standardization efforts in the 3GPP. The ongoing specification work in 3GPP is targeting a generic approach so that other industrial verticals can also use the exposed capabilities. These include automotive, rail-bound mass transit, electric power distribution, central power generation, health care, and smart cities.

A. SERVICE ENABLER ARCHITECTURE LAYER FOR VERTICALS

To allow factories (and other verticals) to automate the system integration and 5G network configuration tasks, 3GPP introduced the Service Enabler Architecture Layer for Verticals (SEAL) in 3GPP Release 16. 3GPP TS 23.434 [8] specifies APIs for provisioning, connection management, device management, connection monitoring, group management, user profile retrieval, identity and key management, location reporting, events, and network resource management.

One of the first documented prototype is built and discussed in [11] to verify the exposure capabilities of a real private 5G network via a cloud-based digital automation ecosystem. Authors investigate how a commercially available 5G non-public network can be extended with exposure capabilities to automate its operational configuration and customization from an industrial automation system. They demonstrated in practice an interoperable and easy-to-use environment for managing and monitoring 5G connectivity of networked industrial devices.

B. NETWORK RESOURCE MANAGEMENT

Network resource management has three operation modes discussed in the following section.

1) STATIC NRM

The above APIs are used to set up a certain production cell for a normal operation. If the production cell is reconfigured, then the existing communication resources are deleted and re-initiated. The operation does not require reconfiguration of the resources dynamically.

2) DYNAMIC NRM USE CASES

An examination of a Cyber-Physical Production System (CPPS) for a simulated robot with simulated network effects was published by the authors of [12]. The concept of a

wireless resource allocation approach that is Quality of Control (QoC)-aware (QoCa) was presented. The method is based on classifying the robotic arm's movement phases into those that require high and low QoC. The arm movements that do not require high precision are those requiring low QoC. For instance, getting to a position where a robotic arm needs to do a task should not require extreme precision. Contrarily, arm movements necessitating high QoC are those where precise joint movements are necessary in order to successfully complete the necessary tasks. The proposed approach was compared against the locally controlled-based solution from the Agile Robotics for Industrial Automation Competition in a simulation environment (ARIAC). The analysis shows that, on average, 54% of the radio time can be saved without reducing the robot cell's productivity.

3) ARTIFICIAL INTELLIGENCE-BASED ADAPTIVE NRM SETUP

While the previous use case required dynamic NRM setup, it cannot adapt to a changing radio environment. Deploying a complex production cell ends up creating unintended peaks during the radio resource usage. Changing radio environments can be handled with agile NRM in which Artificial Intelligence (AI) is a key component. The purpose of introducing such an architecture in [13] is to enable autonomous network resource management. While preserving the robot cell's production Key Performance Indicators (KPIs), its aim is to use radio resources as little as possible. To accomplish this, we use Reinforcement Learning (RL) in a simulated environment to quickly explore the environment, while the Digital Twin (DT) makes sure that the learnt policy can be used in both the simulated and physical environments. We demonstrated that while preserving the precision of real-world robot control, the requests for Ultra Reliable Low Latency Communication (URLLC) connections may be reduced to roughly 30% of the entire radio time.

Going up in the protocol layer, the Vertical Application Layer (VAL) – the term defined by 3GPP System Aspects (SA) 6 – includes the operation of the ROS.

C. ROBOT OPERATING SYSTEM 2

A collection of software libraries and tools called the Robot Operating System (ROS) are used to create robot applications. ROS contains the open-source resources that are needed for any upcoming robotics project, including drivers, cutting-edge algorithms, and robust development tools. The robotics and ROS communities have undergone significant development since the founding of ROS in 2007. By utilizing what worked well in ROS 1 and enhancing what is not, the ROS 2 [14] project hopes to adjust to these developments.

A wide range of QoS policies are available in ROS 2 and let the developer fine-tune node connectivity. With the correct combination of QoS settings, ROS 2 can have a wide range of potential states, ranging from best-effort User Datagram Protocol (UDP) to Transmission Control Protocol (TCP)-like reliability. ROS2 selected Data Distribution Service (DDS) for transport protocol driven by its maturity, real-time capabilities, scalability, interoperability, and wide adoption in various industries.

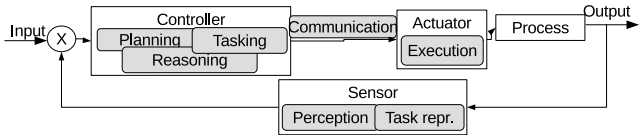


FIGURE 1. The closed-loop control model of the network resource management.

III. VERTICAL APPLICATIONS AS CONSUMER OF NETWORKS EXPOSURE

Exposure consumers are the IIoT applications, which are software entities that use the 5G exposure interfaces' services, whereas exposure producers are the 5G functions that provide the services that are exposed to consumers.

In IEEE P2940 Standard for Measuring Robot Agility, the modeling of the production cell is introduced as a closed loop control (see Fig. 1) with the ten agility aspects involved in the specific components [15]. The industrial process is considered as a system-in-a-system way, thus the network components i.e., the links between the boxes are considered also as a closed-loop system. The components of this closed-loop system are discussed in the following sections.

A. SENSOR – CONNECTION MONITORING

We need to collect information from the ROS2 application to make it feasible to define the network requirements. Fig. 2 shows the various layers in the network stack that can be utilized to collect the necessary information elements for the network exposure requests. The figure shows the part of the 3GPP SA6 architecture discussed in the paper with the ROS2 application stack. (Each box is discussed in detail in the further sections.)

We start from the application layer and discuss the options going down in the protocol stack layer by layer. The network requirements are the consequence of the application-level settings of the application developer and the channel characteristics. The first can be set up and queried directly at both the publisher and subscriber side, while the channel characteristics need to be measured on various layers of the protocol stack.

1) NETWORK REQUIREMENTS SET BY THE APPLICATION DEVELOPER

The application's bandwidth and packet inter arrival time (IAT) is the consequence of the ROS2 topics' message type, packets' size, the publishing rate and sampling interval.

a: NETWORK REQUIREMENTS OF THE APPLICATION

The ROS2 application developer defines the behavior of the publisher and the subscriber. This behavior is only valid on

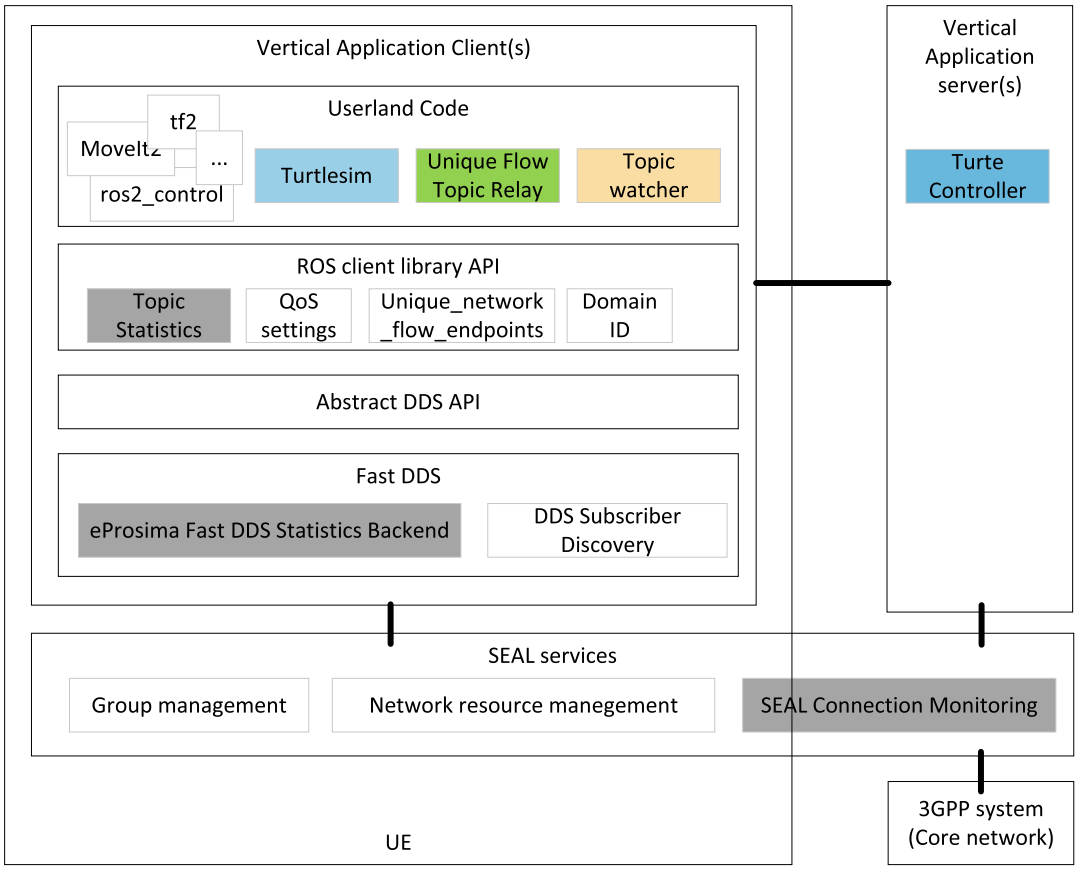


FIGURE 2. The ROS2 protocol stack on top of a 3GPP network from SA6 perspective /gray boxes show the network statistics modules/.

the publisher site. It is important to note that the DDS layer has a feature to support filtering data on a DDS topic [16], though it is not exposed for the ROS2 layer. The issue with this feature is that the filtering is happening at the subscriber. This means all topic data must be transmitted first on the radio channel. After everything arrives successfully, the DDS layer can filter the content of the topic with various string-matching methods and provide to the ROS2 application only the filtered content. This can reduce the load on the ROS2 application – though performance-wise it is also user-space code; thus, it is more convenient for the application developer rather than any performance gain –, but it does not provide any gain on the radio channel.

b: ROS2 QUALITY OF SERVICE SETTINGS

A wide range of QoS policies are available in ROS2 and let the user fine-tune node connectivity. With the correct combination of QoS settings, ROS2 can have a wide range of potential states, ranging from best-effort UDP to TCP-like reliability. ROS2 leverages the flexibility of the DDS transport it utilizes, which proves advantageous in scenarios characterized by unreliable wireless networks, where a "best effort" approach is more suitable. It also proves beneficial in real-time computing setups that demand specific QoS profiles to ensure timely completion of tasks [17].

2) MEASURED CONNECTION RELATED KPIs

Essential information in terms of network statistics and the QoS of the connection are latency, jitter and packet loss. While in the previous section the network flow statistics can be collected on the publisher side to estimate the bandwidth of the application, the influence of the subscriber's setting and the transmission channel needs to be measured.

Fig. 2 shows the ROS2 protocol stack on top of a 3GPP network from SA6 perspective. The gray boxes highlight the possible approaches to gather network statistics in various layers of the system. The information collection method in the specific layers is discussed in the following paragraphs.

a: USERLAND CODE

An option to measure the ROS network statistics can be done in the userland code. This can be done by reading out the settings that the application developer set up for the application. As it is not a generally applicable solution, we aimed to perform the network statistics collection which is transparent for the application developer. We need the ROS topics bandwidth and publishing rate which can be similarly extracted as the command line ROS topic tools, the `hz, bw` operates [18]. The statistics collection is done by summing up the message sizes and their interarrival time in the subscriber.

The tool developed to perform the collection of required information is discussed in Section III-B1 in detail.

b: ROS CLIENT LIBRARY API

The integrated statistics measurement for messages received by any subscription is provided by ROS2 via a statistics topic [19]. Allowing users to get subscription statistics gives them the ability to assess the system's performance or help with problem identification. The received message age and received message period are provided by the measurements.

Topic Statistics measures are disabled by default. Both the received message age and the received message period measures are enabled for that particular subscription once this feature has been activated for a certain node via the subscription configuration options. Data age can be calculated for those subscriptions which message type contains timestamp in the header field, otherwise empty data is published. Only C++ is currently supported for this capability in ROS2 Humble (rclcpp).

The benefit of this approach is that we are in the user space application, and this method measures the same QoS KPIs as the application would perceive. The main issue with this approach is that it is not a transparent solution. It requires the user to modify its code by enabling the subscription feature and forcing them to use time stamped messages.

We tried many designs to use this feature in the Unique Flow Topic Relay (UFTR) (see Section III-B1 in detail). The difficult part is that the subscription node must have a predefined message type during compilation. We checked the examples of the statistical library which made format checking by templates still in the compilation time. We also examined the source code of "RelayField" from ROS2 "topic tools" [20] which showed that the message type is still a rigid requirement just not in compilation time as it is in Python. We were thinking about applying Deep Packet Inspection (DPI) and parsing the headers of the messages as if there were timestamps. We decided that this could give misleading results and dropped the idea. All in all, this solution does not seem to be a viable candidate to include in UFTR as it cannot support general message types.

c: DDS

A C++ library called eProsima Fast DDS Statistics Backend [21] enables users to gather and retrieve performance statistics from a Fast DDS network, such as the topology of the DDS network and the configuration of each DDS entity (i.e., QoS). Additionally, it has a Fast DDS statistics module that enables data collection from the DDS layer and dissemination under certain DDS subjects. This library provides detailed statistics on the DDS level communication: latency, throughput, re-sent packets, submessages, meta-traffic packets, discovery time can be directly collected. The statistics module's built-in DataWriters are used to publish the collected data using DDS across certain topics. As a result, Fast DDS does not compile this module by default, because doing so could slow down the application. To enable it in the ROS2 applications, the Fast DDS needs to be recompiled with a CMake configuration step's `-DFASTDDS STATISTICS=ON` switch. After the successful compilation, the ROS2 abstract DDS API needs to be pointed from the default binary package install to the compiled version.

The advantage of this approach is its detailed KPIs and accuracy. The drawback is the number of steps required to enable this module. Also, full deployment is needed of these modules as during the interaction with other ROS2 nodes, which do not use the statistics enabled DDS version, will not provide any statistical data.

In our experiences, compilation of Fast DDS and modifying the abstract DDS layer does not work with all the ROS packages. Some of them have a dependency on the binary packaged DDS. The whole recompilation of ROS2 is a safe solution, but it takes a long time.

d: TRANSPORT NETWORK

There are three message types in TS 23.434 on retrieving information about the connectivity status. The common way is to subscribe for QoS monitoring and the network pushes down monitoring information periodically. TS 23.434 [8] §14.3.2.22 "Unicast QoS monitoring notification" is the information flow for unicast QoS monitoring notification from the NRM server to the VAL server. QoS monitoring data is an aggregate of QoS measurements obtained from the 5G System (5GS). TS 29.549 [22] §7.4.2.4.2.3 describes the measurement data. It consists of the following attributes: 1) downlink packet delay in milliseconds, 2) uplink packet delay in milliseconds, 3) round-trip packet delay in milliseconds, 4) average packet loss rate, 5) average data rate, 6) maximum data rate, 7) average traffic volume for downlink in bytes, 8) average traffic volume for uplink in bytes.

The following two messages are event notifications from the network. TS 23.434 [8] §14.3.2.15 "QoS downgrade indication" is a message from the NRM client to the NRM server. The report includes the expected or actual QoS or Quality of Experience (QoE) parameters which were downgraded (i.e., latency, throughput, reliability, jitter).

TS 23.434 [8] §14.3.2.16 "Application QoS change notification" is a message from the NRM client to the NRM server. It contains information on the updated or requested QoS parameters for the end-to-end session based on the QoS change on one or both links involved in the network-assisted end-to-end communication.

Note that the way the network retrieves the information is not defined in TS 23.434 [8]. It is possible to involve the Network Data Analytics Function (NWDAF) [23] or make active probing on the established connections.

The advantage of this approach is that it is generally applicable on the 5G network. The drawback is that these measurements refer to the whole connection and not application, ROS2 node specific.

B. PROPOSED ROS2 DATA COLLECTION NODE – IMPLEMENTATION OF THE SENSOR

In this section we discuss the proposed proxy node that is capable of collecting the necessary information from the ROS2 and transport layers for the SEAL request.

1) THE METHOD TO RETRIEVE THE 5-TUPLE IDENTIFIER OF THE ROS2 TOPIC

To fill in the Internet Protocol (IP) address field of the API, or a more detailed 5-tuple identifier (transport protocol, source IP, source port, destination IP, destination port) of the IP flow on which the QoS management request is valid we need to obtain the ROS2 topic flow identifiers. Reference [24] discusses the issue of the default ROS2 topics. The same flow identifiers (5-tuple or 3-tuple) are used by all publishers and subscriptions in communication nodes. This prevents communication nodes from explicitly differentiating network QoS for publishers and subscriptions. Thus, only the same network QoS can be assigned to publishers and subscriptions in communicating nodes.

a: SOURCE IP, PORT

Since ROS2 Galactic, the "Support for unique network flows" is among the features [25]. To enable QoS specifications for these IP streams in network architectures that support such a feature, like 5G networks, applications may now demand that UDP or TCP and IP-based ROS Middleware (RMW) implementations provide unique network flows i.e., unique Differentiated Services Code Points (DSCP) and/or unique IPv6 Flow Labels and/or unique ports in IP packet headers.

A further feature connected with the setting up of unique network flows is a related getter interface for the RMW. ROS2 Galactic provides an API called `get_network_flow_endpoints()` for publishers and subscriptions to understand ports and IP addresses assigned for their messages by the RMW implementation. Note that Network Flow Endpoints (NFEs) established only on the publisher-side are represented by the NFEs supplied by `get_network_flow_endpoints()` for a publisher (local). It does not include details on NFEs for matching subscriptions. Similarly, the NFEs for a subscription returned by the getter function are localized and lack information about matched publishers. This is a design choice for ROS2.

b: DESTINATION IP, PORT

The missing information elements from the 5-tuple identifier can be obtained in a FastDDS specific way. The DDS layer is aware of the destination IP and ports, it is just not exposed towards the ROS2 layer. ROS2 topics consist of 1) DDS topics with the `rt/` namespace, 2) a topic transporting meta traffic information and 3) a topic providing statistical information if it is enabled. The meta traffic topic is used by ROS2 for discovery and configuration purposes. During the DDS subscriber discovery, the `SubscriberEvent` [26] can be queried for the destination IP address and ports. We check the DDS topics and collect the 5-tuple data when the `ReaderDiscoveryInfo` changes regarding those topics which are indicated by the user to special QoS handling.

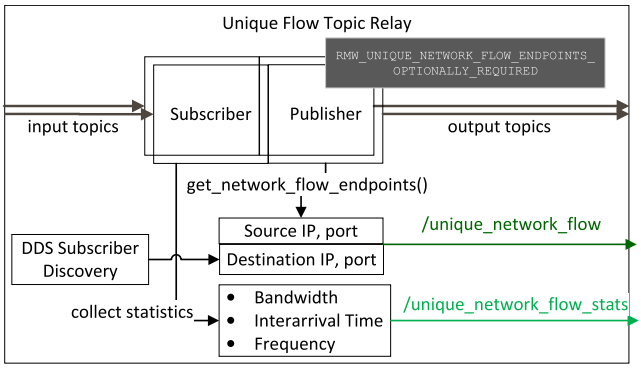


FIGURE 3. Unique Flow Topic Relay (UFTR).

2) UNIQUE FLOW TOPIC RELAY (UFTR)

The required features are collected and realized in our tool called "Unique Flow Topic Relay". Fig. 3 shows the architecture and information flow of the node. This tool is the extension of "ros topic tools" [20]. "ROS topic tools" are tools for meta-level guiding, throttling, choosing, and other manipulation of ROS2 topics. These utilities work on generic binary data utilizing rclcpp's `GenericPublisher` [27] and `GenericSubscription` [28], rather than serializing the streams that are being manipulated. The tools in the package are offered as composable ROS2 component nodes, which enables them to be launched from launch files, initiated from the command line, or spawned into an already running process. We extended the "relay" command line tool. The ROS2 "Relay" node subscribes to a topic and publishes all incoming data to another topic. It is compatible with all message types.

Our extension for the relay node is that the re-published topics are created with the enabled unique flow option, resulting in that these topics are no longer multiplexed on the same 5-tuple as the other and we can request the 5-tuple identifier of these topics.

The intended usage of the node is that the user configures its launch file by tagging those topics that require QoS handling by our mechanism. The user can add a "_QoS" tag to the original topics of the application, and the relay node subscribes for these topics then re-publishes them, removing the tag as well. An option for the application developer is to simply remap the existing topic names with the added "_QoS" tag. In this case there is no need for any modification to the original ROS2 application.

The 5-tuple related information for each QoS handled topic is published on the "unique_network_flow" topic, while the network statistics related information is published on the "unique_network_flow_stats".

The method that provides more detailed network statistics requiring different ROS2 connection architecture is discussed in Section V and VI-A3.b.

C. CONTROLLER – NETWORK RESOURCE MANAGEMENT

We need to map the sensor information from Section III-A to a control process.

TS 23.434 [8] §14.3.2.13 defines the ''end-to-end QoS management request'' from the NRM client to the NRM server. The essential information elements are the ''list of VAL User Equipment (UE)'' for whom the end-to-end QoS management occurs, the ''IP address'' of the VAL UE and the ''end-to-end QoS requirements'' of the application including latency, error rate, etc. for the end-to-end session. Note that the Rel-17 definitions enable the UE-level setup of the QoS parameters. To make it future-proof with further releases, and to follow the recommendations given in [10] §4.2.3 Note 2,3 to support multiple IP flows, we provide UDP/IP-flow level identification of the ROS2 topics.

1) THE 'HELLO WORLD' IN ROS2, TURTLESIM

A simple simulator for learning ROS2 is called Turtlesim [29]. It gives the user an overview of what ROS2 accomplishes at its most fundamental level so that they can subsequently use a real robot or a robot simulation. Fig. 4 shows the basic architecture of Turtlesim with an external controller connected via 5G. The use case is that we spawn a turtle, and it is controlled via closed-loop control by an external control node to move along a circle. In this basic setup there are two unidirectional topics established between the controller and the Turtlesim. Turtlesim publishes the position information; the controller subscribes to this topic (''/turtle2/pose''). Meanwhile the controller publishes the velocity command topic (''/turtle2/cmd_vel'') and the Turtlesim subscribes to it.

Fig. 5 shows the extension of the basic use case with the UFTR nodes that enables the QoS handling of both the position and the velocity topics. The following section discusses the orange box in the figure.

2) THE METHOD TO DEFINE QOS OUT OF THE CONNECTION MONITORING DATA

Until now, we identified the flow identifier of the topic which needs QoS handling. In the following steps we discuss how the QoS handling is achieved.

3GPP TS 23.203 [30] Table 6.1.7 enumerates the standardized QoS Class Identifier (QCI) characteristics in terms of Resource Type (RT), Priority Level (PL), Packet Delay Budget (PDB), Packet Error Loss Rate (PELR), Maximum Data Burst Volume (MDBV) and Data Rate Averaging Window (DRAW) with given example services also. The mapping of a QCI for a certain input parameter set is given by the table. This step is indicated by ''QoS mapper'' in Fig. 5. The open question remains how to set up the input parameters based on the available connection information.

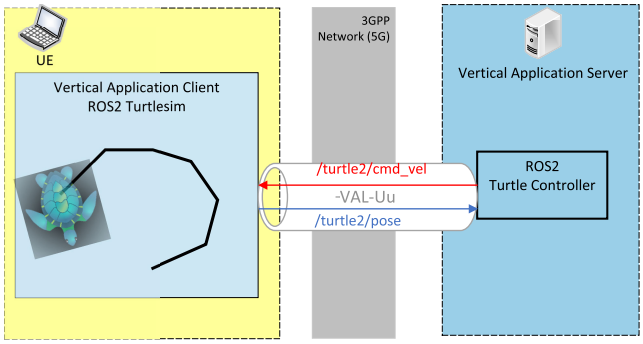


FIGURE 4. ROS2 Turtlesim with an external controller connected via 5G.

```
def __init__(self):
    super().__init__('subscriber')    self.
     create_subscription(UniqueTopicFlow,'
     unique_topic_flow', self.listener_callback, 10)
    self.create_subscription(UniqueTopicFlowStats, '
     unique_topic_flow_stats', self.
     listener_callback_stats, 10)
def listener_callback(self, msg):
    if msg.event_type == 1: #ADDED
        # src, dst, hz, bw
        active_topics[msg.topic] = (msg.source, msg.
     destination, 1, 1)
    if msg.event_type == 2: #REMOVED
        active_topics.pop(msg.topic)
def listener_callback_stats(self, msg):
    for i in msg.stats:
      res = active_topics.get(i.topic)
        if res is not None:
            src = res[0] \n\t dst = res[1]
            current_rate = res[2] \t current_bw = res[3]
    if qos_profile.reliability is QoSReliabilityPolicy.
     RELIABLE: PELR = 1e-5
    elif qos_profile.reliability is QoSReliabilityPolicy.
     BEST_EFFORT: PELR = 1e-2
    PDB = 1. / i.window_rate * 10e3
    if PDB > qos_profile.lifespan.nanoseconds / 10e9:
        print("WARN: ROS2 lifespan setting is too low")
    if abs(current_rate/i.window_rate-1)<0.1: RT = "GBR"
    else: RT = "Non-GBR"
    if i.window_duration_secs < 2:
        print("WARNING: DRAW is not enough")
        continue
    MDBV = i.num_bytes/i.num_messages
    if RT == "GBR" and PDB <= 100 and PELR <= 10e-2:
        QCI = 1
        ExampleServices = "Conversational Voice"
        hit = hit + 1
    ...
    active_topics[i.topic] = (res[0], res[1], i.
     window_rate, i.window_bandwidth)
```

LISTING 1. QoS mapping code.

Listing 1 shows the important part of the QoS mapping code that we discuss in detail.

''Topic watcher'' subscribes to both the ''unique_flow'' and ''unique_flow_stat'' topics (see Fig. 5). The topic ''unique_flow'' provides information in case a new topic becomes active or if it has ceased to exist i.e., there are no more publishers or subscribers for the topic. In case these events occur, an update is published with the 5-tuple identifier of the topic's IP flow. A dictionary stores the list of active topics. The ''unique_flow_stat'' topic receives network statistics data on the active topics in every 10 secs. The 5-tuple for the active topic and the statistic from the previous window is retrieved from the dictionary.

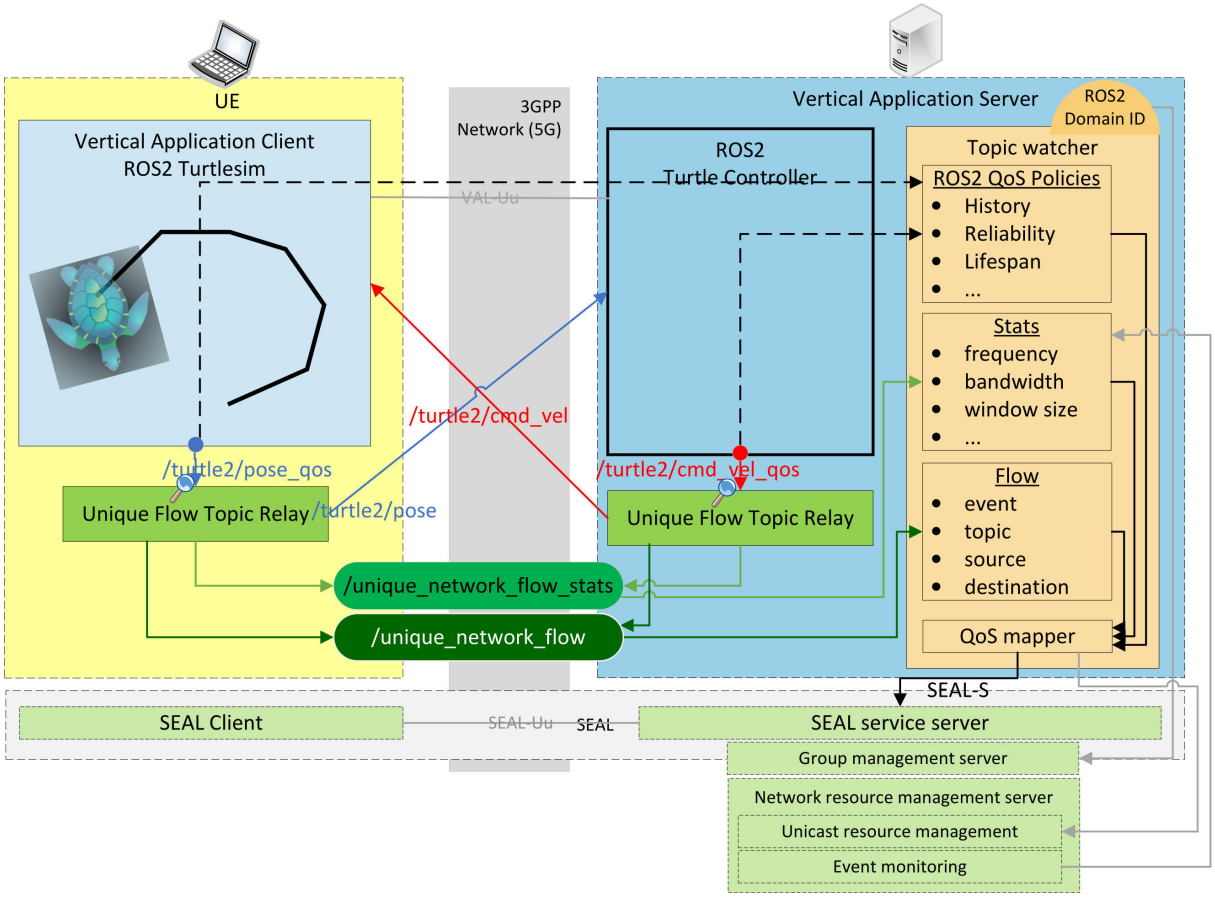


FIGURE 5. ROS2 Turtlesim with UFTR deployment.

The PELR is mapped from the ROS2 QoS reliability. ROS2 have two QoS reliability settings:

- Best effort: make an effort to deliver samples, but if the network is not reliable, they could be lost.
- Reliable: provides a delivery assurance and several retries.

3GPP TS 23.203 [30] Table 6.1.7-B has QCI values for automation in which the most reliable values are in 10^{-5}, which is given for the reliable ROS2 topics and 10^{-2} is considered for the best effort ROS2 topics.

PDP is set directly by the measured packet generating frequency of the ROS2 application. The lifespan duration set by ROS2 QoS policy is the longest period that can pass from the time a message is published and when it is received without the message being considered stale or expired (expired messages are silently dropped and are effectively never received). We give a warning if the measured PDP is higher than the lifespan duration as it means that the ROS2 application considers them as expired.

We compare the data sending rate of the ROS2 topic in the current and previous measurement window. If their difference is within 10% we consider the topic as Guaranteed Bitrate (GBR) otherwise as a Non-GBR topic which is set to the Resource Type (RT) for the QCI mapping.

The DRAW has to be at least 2000 ms according to 3GPP TS 23.203 [30] Table 6.1.7-B. If the measurement window is smaller than that we pop up a warning.

The MDBV is estimated from the average packet size in the measurement window. Extending the statistics message could directly provide this information. 3GPP TS 23.203 [30] Table 6.1.7-B use it rarely. Basically, two types of MDBV packets are considered: one for small packets below 255 bytes, and one with Maximum Transferable Unit (MTU) with 1354 bytes.

An optional information that can be used is the lease duration of the ROS2 topics to provide indication to the publisher that it is alive before the system considers it to have lost liveliness. Losing liveliness could be an indication of a failure. A callback is triggered by the ROS2 system in case the maximum time of the lease duration is reached (see [31] as an example). The 3GPP system can utilize this to remove the flow related requests of the ROS2 topic e.g., NRM requests, network monitoring subscriptions explicitly. It is done automatically by the 3GPP system, but in large deployments, it can reduce the load on the 3GPP nodes.

The rest of the algorithm goes through the 3GPP TS 23.203 [30] Table 6.1.7 as a set of conditional switches. Note that the rest of the ROS2 QoS parameters are DDS

specific or ROS2 application specific (e.g., deadline) and have no relevance in the radio transport network setup. The following ROS2 QoS policies are not considered: 1) history: the strategy how the samples are stored, 2) depth: the number of samples to be stored, 3) durability: check if the publisher is the responsible for persisting samples for ''late-joining'' subscriptions, 4) deadline: the maximum time that should pass between posting new messages to a topic. Finally, the stats are updated in the dictionary.

D. THE CONFIGURATION STEPS OF THE 5G NETWORK AND ROS NODES

In this section we describe the required network configuration steps during the ROS2 nodes operational phases. We assume the following network setup. Wireless connectivity is provided by a 5G Non-Standalone (NSA) Mobile Private Network that can be deployed on the factory premises, also termed as Non-Public Network (NPN). This private 5G network comes with an integrated edge cloud, which the enterprise can use to run IoT application instances on. In our setup, the ''topic watcher'' and the controllers of the turtles run in a persistent Virtual Machine (VM) in the edge cloud.

The following subsections describe the steps of setting the ROS2 environment into work and the associated network configuration tasks.

1) DEVICE PROVISIONING AND ONBOARDING

To let the network know about the existence of the modular devices, their identities and credentials have to be provisioned into the subscriber database of the 5G network. The device is pre-configured with a vendor certificate (or other type of credential) that allows successful identification and authentication of the device by the 5G network. Our devices are pre-configured with a Subscriber Identity Module (SIM) card and an NPN identity. The 5G NPN is configured with subscription profiles for numerous services, including the 3GPP network access authentication. The UE plugged into the execution environment of the ROS2 application instances is equipped with a SIM card, that holds the identities and the authentication keys needed to authenticate the device in the 5G NPN.

As part of this provisioning step, 5G NPN needs to be provided with the Generic Public Subscription Identifier (GPSI) of the UEs among other information, so that it can authenticate these UEs and devices.

The 5G NPN provides a default logical network (typically used for application-level device onboarding purposes), which provides best-effort connectivity, and for security reasons is isolated from the other logical networks used for production. The logical networks are also known as device groups, according to the SEAL terminologies.

When the device is turned on, it successfully connects to the default logical network (bootstrap device group). The 5G network has to let the IoT application know about the newly connected device by sending an event notification. The IP address is also provided so that the IoT application can address the device.

2) SETUP ROS2 NETWORK

The first step is the launching of TurtleSim, the main process that requires control. Note that the launch phase in ROS2 does not guarantee the exact order of the execution of the commands that are defined in the launch file. This setup phase should be considered transient, while all the nodes start and the topics are published, subscribed.

Networks where multicast traffic has issues, or with large numbers of participants, even the discovery of the new nodes can take considerable time. In addition to the 5G Radio Access Network (RAN) characteristics, the 5G system (5GS) offers solutions for Ethernet networking and URLLC in the core network (CN). Native Ethernet protocol data unit (PDU) sessions are supported by the 5G CN [7]. In such cases the discovery server is not needed. The above feature requires such UEs as well. To relax this requirement, we applied the discovery server-based setup [32]. The Client-Server Architecture offered by the Fast DDS Discovery Server enables nodes to connect with one another through a middle server. Every node performs the role of a discovery client, sending information to one or more discovery servers and receiving information back from them. This minimizes network bandwidth associated with discovery and does not need multicasting capabilities.

After TurtleSim has spawned, its related ''Unique Flow Topic Relay for TurtleSim'' node subscribes for the ''/turtle1/pos_qos'' topic that is published by TurtleSim. Note that the topic names contain the ''_qos'' tag, which will be removed later. The ''Turtle 1 Controller'' is spawned to provide control for the TurtleSim. The controller subscribes to the commands published by the controller, intended for the TurtleSim. The ''Unique Flow Topic Relay for Turtle 1 Controller'' is launched, and it subscribes to the ''/turtle1/velocity_qos'' topic. ''Turtle 1 Controller'' requires the position information of the ''turtle1'' via ''/turtle1/pose''. This topic is published by ''UFTR for TurtleSim''. TurtleSim requires the velocity commands from the ''Turtle 1 Controller'', which is published by the ''UFTR Turtle 1 Controller'' via the ''/turtle1/velocity'' topic. Note that the ''_qos'' tag is removed from this topic.

3) ROS2 NODES ARE OPERATIONAL

When the ROS2 nodes become operational, the ''topic watcher'' node is spawned, and it sends a message to the SEAL server on group creation with the DDS domain identifier (ID). The DDS domain ID can be queried from a ROS2 node via `rclpy.get_default_context().get_domain_id()` function call. The ROS2 nodes belonging to the same domain ID can be isolated in their own logical network in the 5G NPN. These logical networks have to be created in the form of device groups with the following attributes: name, communication type

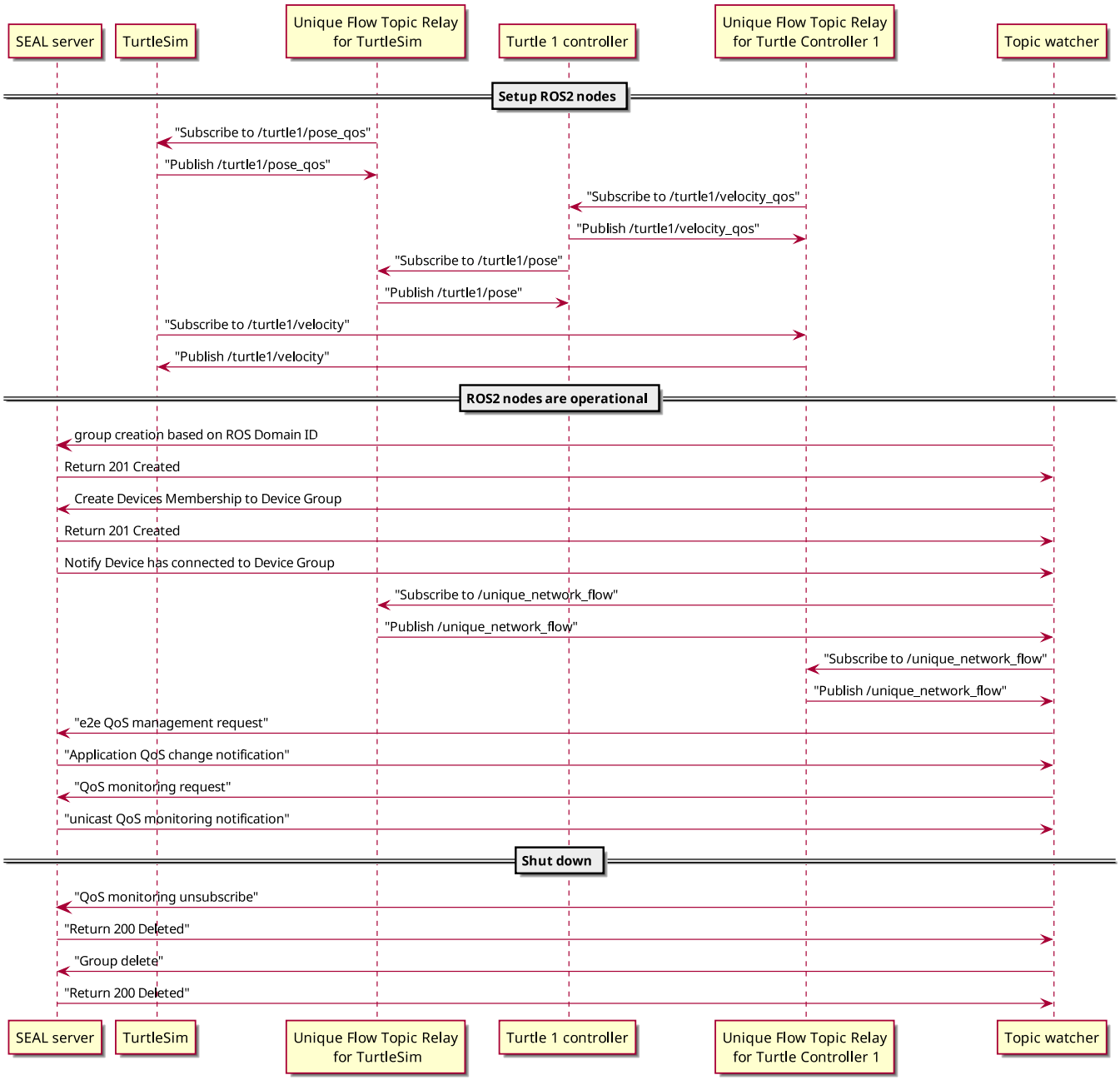


FIGURE 6. The communication steps between the ROS2 and SEAL nodes.

(IP or Ethernet), VLAN ID (Virtual Local Area Network Identifier), and default QoS parameters for prospective device connections to this group (see TS 23.434 [8] §10.3.2.1 Group creation request). The device group management service provides the 5G Local Area Network (5GLAN) group data to the 5G system and replies with an external group identifier for the 5GLAN group. Once the device group is ready, the execution environments of the ROS2 nodes – each identified with a GPSI – require membership in the newly created 5GLAN device group identified with the external group identifier. As a result, logical networks (device groups) are created for each execution environment of the ROS2 nodes, and the UEs are entitled to establish user plane connections to their device group.

"Topic watcher" subscribes for both "unique_network _flow" topics published by the "UFTR for Turtlesim" and the "UFTR for Turtle 1 Controller". Beside the flow data, flow statistics are published on the "unique_network_flow _stats" topic, which is not drawn on Fig. 6 to make the sequence diagram more comprehensible. "Topic watcher" calculates the required QoS and sends "e2e QoS Management requests" to the SEAL server. The SEAL server can

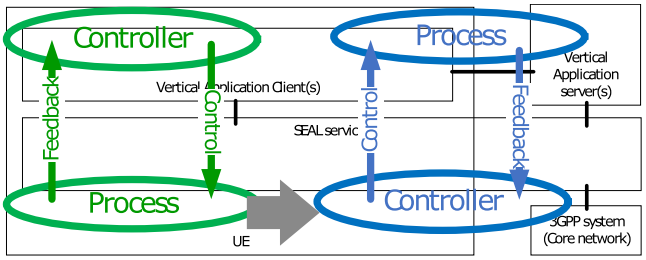


FIGURE 7. The control loops in the "vertical application as consumer of network exposure" (Section III) vs "the network as consumer of application exposure" (Section IV).

respond with an ''Application QoS change notification'' if the request can be fulfilled. It is possible to request on demand or subscribe to connectivity monitoring via the ''QoS monitoring request''. The SEAL server can send a ''unicast QoS monitoring notification'' periodically for the subscription.

4) SHUT DOWN

After the ROS2 tasks are executed successfully the network and the edge cloud are requested to release the resources and go back into the initial phase. It consists of procedures such as unsubscribing the ''topic watcher'' from QoS monitoring and the deletion of the group.

IV. THE NETWORK AS CONSUMER OF APPLICATION EXPOSURE

The current design of 3GPP SEAL provides both information and control on the network towards the vertical application, but eventually to be able to provide smart network solutions, the other way of information and control is a desired state in the future. In this way the exposure consumer is the 5G network, whereas exposure producers are the vertical application functions that provide the services that are exposed to consumers (see Fig. 7).

In another interpretation, the previous section discussed rigid VAL applications and adaptive networks, while this section extends the capabilities of VAL applications with adaptive behavior in cooperation with the network.

The NRM requested by the VAL application worked in an open-loop manner so far. We had no sensor to check the effects of the NRM request on the VAL application performance. In this section our goal is to close this control-loop.

In this section we introduce the Vertical Application Service Enabling Architecture Layer (VASEAL) that exposes the vertical application layer towards the network. Fig. 8 shows the concept. Similarly, as in SEAL, VASEAL has four main components: VASEAL Service Server and Client and their respective network side nodes, the Network Layer Service Server and Client. VASEAL Service Server runs next to the Vertical Application Server hosting the Resource Management Server and Event Monitoring for the production cell or other VAL application. VASEAL Service Client runs next to the Vertical Application client running the GUI for the data collection of the event monitoring. The Network Layer Service Client runs next to the SEAL Client. The Network Layer Service Server runs next to the SEAL Service Server.

In connection with P2940 the VAL process, like the Turtlesim, is considered as a closed-loop control process in which the certain elements are discussed in the following sections.

A. SENSOR – ROS2 PROCESS RELATED KPIS

The main idea to reduce the needed infrastructure can be the assumption that if the product does not have significant quality impact due to the looser network requirements (see mentioned examples), then significant cost savings can be achieved by relaxing some QoS parameters of the network.

The strict communication requirements demanded by the manufacturing processes are to meet a mostly binary defined satisfactory level: acceptable or unacceptable of the final product. ARIAC [33] proves that, during the evaluation of an agile production cell with pick-and-place tasks for on-demand orders, the performance metric of the production cell can be a non-binary metric, resulting in a fine-grained score to make it possible to rank competitors.

Telecommunications introduced a fine-grained Mean Opinion Score (MOS) in the 2000s [34] to measure the human-judged overall quality of an event or experience. In typical telecommunication services, MOS is a ranking of the quality of voice and video sessions. Most often judged on a scale of 1 (bad) to 5 (excellent), MOS is the average of several individual human-scored parameters. Although MOS originally was derived from surveys of expert observers, today MOS is often produced by an objective measurement method approximating a human ranking, called Estimated MOS (EMOS). QoE KPIs such as video buffering time, or missing video frames are directly correlated with MOS. QoE is affected by end-to-end QoS settings such as average bandwidth or packet drop on a certain network link.

Our intention in this section is to pursue the track that has been used for decades already in Mobile BroadBand (MBB) to evaluate the perceived quality of a service of the user, extend this to the industrial and manufacturing area, and analyze a yet unexplored case how network QoS affects the quality of robotic sanding. This principle can be applied in other industrial areas where precision and process-speed requirements allow a broader range. These fields could be for example, painting, spraying, enameling, coating, iron casting, bonding, and sealing, etc. To achieve this, we introduce an interface, a ''MOS'' topic which can be populated in the following two ways.

1) MEAN OPINION SCORE PROVIDED BY THE USER

We intend to demonstrate in this paper that industrial processes can be examined one-by-one based on their network performance requirements, and their performance can be evaluated by fast expert opinion leaving out the

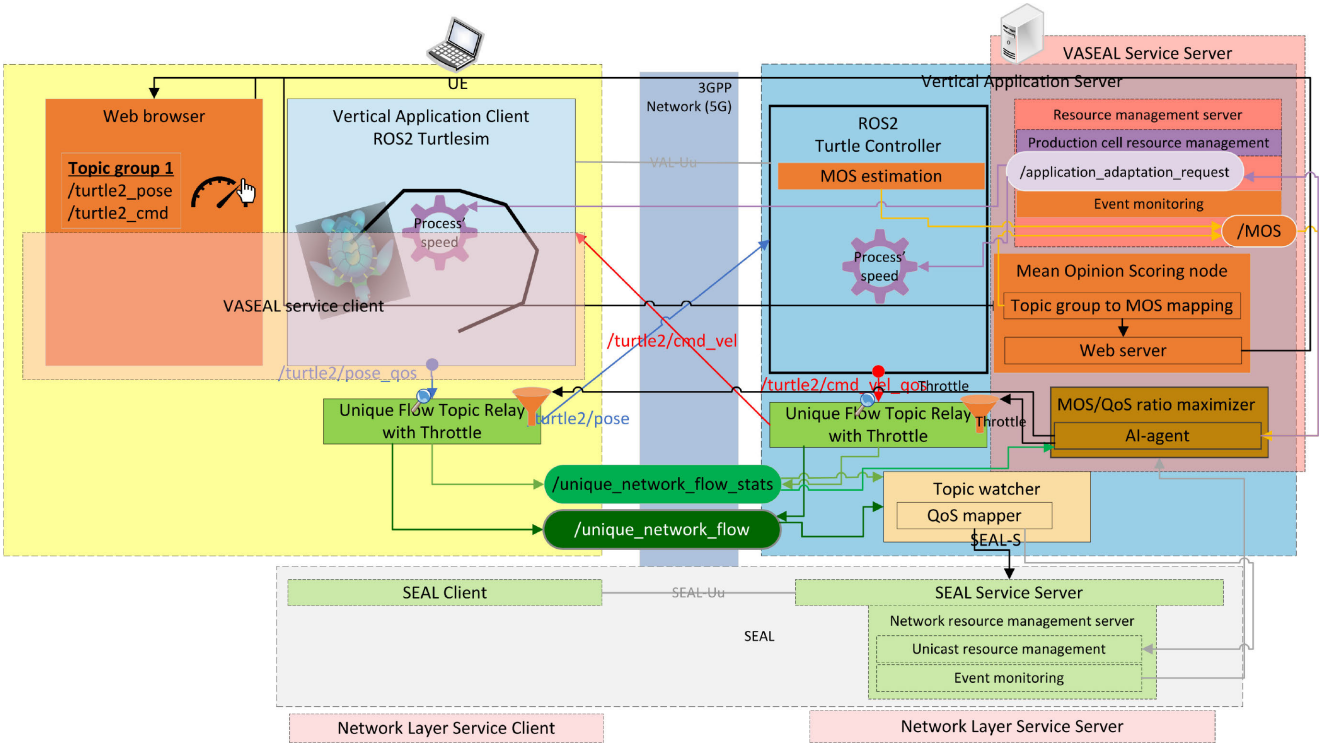


FIGURE 8. The architecture of the ROS2 Turtlesim with UFTR with throttle and MOS deployment.

tedious low-level process specific KPI measurements. This option involves a human expert in the evaluation. The "Mean Opinion Scoring" node hosts a web server which communicates with a browser via websocket. The user is provided with an interface on which all the topics relayed by the UFTR in the system are enumerated. The user can group the topics by a self-defined ID which represents that a set of topics is responsible for a certain perceived MOS. After the grouping of the topics, the user can set a MOS value for the process on a scale 1-5. The websocket sends back the selected value associated with the list of topics and the "Mean Opinion Scoring" node publishes it on the "MOS" topic.

2) APPLICATION WITH BUILT-IN MOS ESTIMATION

Beside the human focused mean opinion scoring, it is also common to estimate the MOS automatically to provide high level feedback to the content provider based on measured network QoS related KPIs e.g., [35], [36]. As a demonstration we included an automatic estimation for the controlled turtles in the Turtlesim. We extended Turtlesim with a function to compare the Proportional-Integral-Derivative (PID) error or movement resolution of the two controlled turtles. After every 10 Hz difference in the control loop of the two turtles, the MOS score of the slower control loop is reduced by 1 unit. This is calculated in the background and published on the "MOS" topic.

In case both the human expert mean opinion scoring and automatic MOS estimation publishes on the MOS topic, the human-based is considered only. The "MOS" topic publisher has an identifier field to signal whether it comes from the human scoring system or from the automatic one.

B. CONTROLLER – APPLICATION BEHAVIOR INFLUENCING INTERFACE

As we got inspired by telecommunications to describe the perceived quality of transmitted voice and video by a simple MOS score, we can take one step further in terms of the control or the influence of the quality of the audio and video. 3GPP TS 26.247 [37] describes the working mechanism of Progressive Download and Dynamic Adaptive Streaming over Hypertext Transfer Protocol (HTTP) (3GP-DASH). §11.2.4.1 tells that the DASH client keeps consuming the media content after the presentation has begun by repeatedly requesting Media Segments or parts of Media Segments and playing content in accordance with the media presentation timeline. With latest information from its environment, such as a change in observed throughput, the client may alter Representations. In a simple implementation, the client may change to a different Representation with any request for a Media Segment that begins with a stream access point. One Adaptation Set's Representations represent the same media content elements, hence all the media streams it contains are thought to be perceptually comparable. An Adaptation set contains various attributes out of which the following three types influence the QoE: 1) min and max bandwidth, 2) min and max Width and Height of the frames, and 3) min and

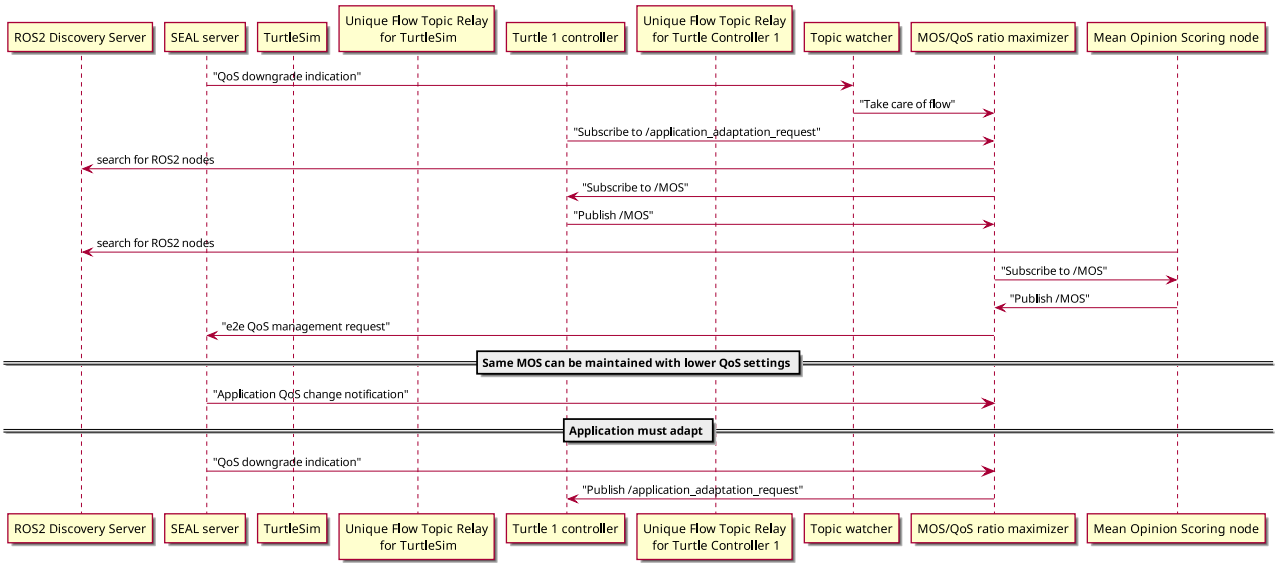


FIGURE 9. The communication steps between the ROS2 and SEAL nodes when communication resources are depleted.

max frame rate. From an application's i.e., a robot controller's perspective, the bandwidth is a consequence of the control packet size. While in audio and video encoding there are various codecs and compression rates which significantly alter the required bandwidth, in terms of control messages e.g., ROS2 standard twist message there are two vectors only which is difficult to compress further. The resolution can refer directly to the control frequency e.g., a haptic device has higher control frequency or resolution than an industrial heavy duty robotic arm. The frame rate in video codecs requires various playback speeds for the specific frame rate. In case there is inadequate network bandwidth to download the 50 Frames Per Second (FPS) video in real-time then playing out with 25 FPS frame rate makes more room for the network to buffer with the doubled play-out time. We can interpret this from the VAL applications point of view as slowing down the application till there is no significant degradation of production KPIs, e.g., cycle time can relax the network requirements. Assuming the VAL application can work similarly to the DASH client, it would mean that the VAL application adapts to the network characteristics. This behavior is only required if the network's resources are depleted. To achieve this the network requires clean interfaces on the VAL application for which the VASEAL architecture provides an approach.

C. NETWORK RESOURCES ARE DEPLETED

TS 23.434 [8] §14.3.2.16 ''Application QoS change notification'' types of messages can arrive from the NRM client to the NRM server.

1) ROS2 IMPLEMENTATION

Fig. 9 shows the communication steps between the ROS2 and SEAL nodes when notification events occur from the network. In case the network resources get depleted, the SEAL server sends ''QoS downgrade notification'' to the ''topic watcher''. The ''topic watcher'' notifies the ''MOS/QoS ratio maximizer'' node to spur into action which opens a topic ''/application_adaptation_request'' for which the ''Turtle 1 Controller'' subscribes. The ''MOS/QoS ratio maximizer'' node subscribes to the ''/MOS'' topic on which the ''Turtle 1 Controller'' publishes estimated MOS values automatically without human interaction. The ''Mean Opinion Scoring node'' is launched which also publishes on the ''/MOS'' topic for which the ''MOS/QoS ratio maximizer'' node subscribes. These MOS scores are set up by human experts. The ''MOS/QoS ratio maximizer'' calculates actions on keeping the MOS acceptable but reducing the requested QoS and the ''e2e QoS management requests'' are sent towards the SEAL server.

There are two outcomes of these requests. One, the SEAL server can fulfill the requested QoS and responds with ''Application QoS change notification''. In this case we reached a new operation point. Or the second case, when the QoS requests still cannot be fulfilled and a further ''QoS downgrade notification'' arrives from the SEAL server. In this case the ''MOS/QoS ratio maximizer'' node publishes an adaptation request towards the ''Turtle 1 Controller'' via the ''/application_adaptation_request'' topic to slow down and lower the frequency of the control loop.

2) THE MOS/QOS RATIO MAXIMIZER IN THE VASEAL ARCHITECTURE

The right middle brown box represents the MOS/QoS ratio maximizer in the VASEAL architecture in Fig. 8. It is hosted in the VAL server, but it is also possible to implement as a NF. The node has access to the event monitoring and resource management features of the SEAL and VASEAL servers.

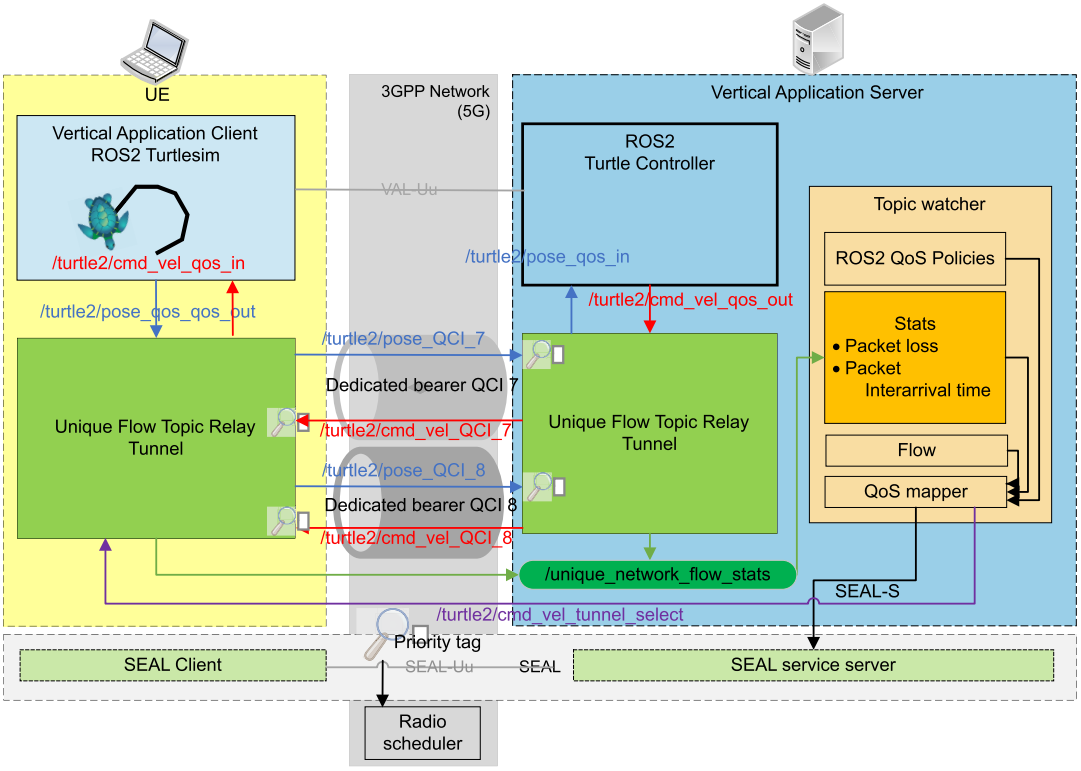


FIGURE 10. ROS2 architecture with tunnel.

In this way it can collect information and have control of both the VAL and SEAL servers. An implementation of the MOS/QoS ratio maximizer is discussed in Section VI-A3.

V. TUNNEL MODE

In this section we describe a third architectural concept and a further operation mode of the UFTR. The two main features why we introduced this architecture are 1) to support the detailed network KPI measurements in the userland code and 2) to provide enhanced dynamic QoS handling for the ROS2 applications. We need to measure latency, jitter and packet drop on any general message type. We examined the code of ROSBag [38] as its recorded data would make it possible for an offline evaluation in terms of the QoS KPIs of the topics' traffic. The command line utility "ros2 bag" is used to record data published on subjects in the ROS2 system. It compiles the information shared on a variety of subjects and stores it in a database by extending the topic messages with a header containing a timestamp. After that, the user can replay the data to get the same outcomes as they test and experiment.

A. UNIQUE FLOW TOPIC RELAY WITH TUNNEL

Our design choice was to extend and modify the original functionality of UFTR. We connect the UFTRs' of the ROS2 application instances then the UFTRs publish the relayed topics to another UFTR instance which subscribes to it creating a tunnel between the two (see Fig. 10). This design choice allows us to extend the transferred messages with

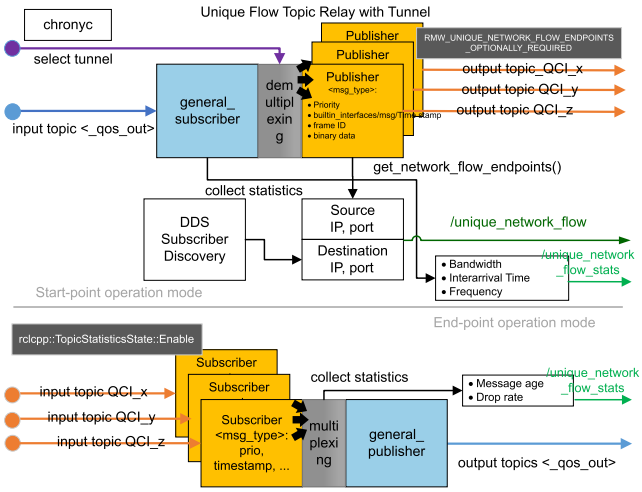


FIGURE 11. Unique Flow Topic Relay with Tunnel.

custom headers, as if we make an encapsulation of the original topics with ROS2 message headers.

Fig. 11 shows the extended architecture of the UFTR. There are two modes of operation of the subscriber and publisher based on the function of the ROS topic. The function of the ROS topic is indicated with an additional tag (beside the `_qos` tag that identifies the flows needed for the unique flow relay features). The additional tag represents the user intended direction of the topic. First, the

application publishes a topic tagged with `_qos_out`. The UFTR subscribes to it with a general subscriber meaning that no message type is needed to be set, as it is not parsed by this node. Second, compared to the way rosbag works – in which the compiled message type library is dynamically loaded to parse the message content –, a publisher is initiated with a custom message type including a timestamp, a message counter, and a binary field. The timestamp is filled with the clock, the message counter is an ever-increasing integer, and the binary data is the whole message received on the general subscriber topic. Note that there is no parsing happening in this case, thus it is message type agnostic, an encapsulation occurred only. In this operation mode of the publisher, the topic is transmitted via QoS handling, thus the unique network flow endpoint option is set.

The same UFTR has the second operation mode, in which it subscribes to the encapsulated topic. As this topic has a timestamped header, the `TopicStatistics` (see Section III-A2.b) feature can be enabled. The header is parsed, and the binary data stream is forwarded through a general publisher. This step is indicated in the topics naming with a `_qos_in` tag that this topic is decapsulated and can be forwarded towards the application.

Time synchronization is crucial in the estimation of message age. We used chrony [39], which is a Network Time Protocol implementation (NTP). Chrony provides features to make the system clock be synchronized using NTP servers, reference clocks, like a Global Positioning System (GPS) receiver, or just manually entered time using a wristwatch and a keyboard. In order to offer a time service to other computers on the network, it can also function as an NTPv4 (RFC 5905 [40]) server and peer. It is built to function well under a variety of circumstances, including erratic network connections, overloaded networks, changing temperatures (regular computer clocks are sensitive to temperature), and systems that do not run continuously or on virtual machines.

B. QOS HANDLING SUPPORT

QoS handling can be done with packet marking or in separate tunnels. Section VI-A2 discusses in detail why this mode of operation is required to support a fully dynamic NRM setup.

1) TUNNEL WITH PRIORITY BITS

Creating the tunnels and encapsulating the original topic makes it possible to add extra headers that can ease the QoS handling.

One such approach is to set up the diffserv [41] values of the packets. Differentiated Services (DS) is a QoS design that is frequently used in IP networks. In the 6-bit DS Code Point (DSCP) sub-field of the 8-bit DS field in the IP packet header, the application specifies the necessary DS-based QoS. For instance, the DSCP setting of $0 \times 2E$ says that the desired QoS is expedited forwarding. Real-time data like audio and video is forwarded through expedited methods.

a: IT IS POSSIBLE TO UTILIZE DIFFSERV IN THE RADIO NETWORK AS WE DID AS WELL IN AN EARLIER WORK IN [42]

The core operation of the idea is that a packet filter is assigned to a dedicated bearer based on a DSCP value. It makes it possible to assign the packet forwarding strategy per packet based on DSCP marking of the application data. This method has short convergence time, but the bearer switching can result in temporary packet reordering. In the radio scheduler each bearer has its own queue and at switching to dedicated bearer the packet queued by default bearer can be taken over by newly arrived packets. The packet reordering can result in out-of-order sensor and control data, which may cause glitches in robot operations. Packet reordering can be handled easily by the UFTR itself by applying a simple and effective packet handling strategy that drops all older packets compared to the packet with the highest sequence number which has arrived already. The second strategy is to apply a hold-and-buffer strategy as in the Time Sensitive Network (TSN) handling in 3GPP TS 23.501 [6] and make reordering during the buffering period. Also, packet reordering can be avoided by advanced queue management. The queue management in the base station can do packet forwarding between the queues of the bearers to keep in-sequence delivery of packets.

Another option is to apply the diffserv values directly as QCI parameters. In order to reconcile the marking recommendations made by the 3GPP and the IETF and maintain a consistent QoS treatment between cellular networks and the Internet, [43] specifies a set of 3GPP QCI and 5G QoS Identifiers (5QIs) to DSCP mappings. Note that this option provides limited benefits as QCI update is a slower process compared to the fast reaction time of a packet router altering its packet queuing on-the-fly based on the diffserv fields.

b: APPLICATION OF DIFFSERV IN ROS2

For diffserv-based QoS for publishers and subscriptions, ROS2 does not provide an API. There are long discussions on this topic in various ROS2 forums e.g., [44]. The current approach of the ROS2 middleware working group is that diffserv is a transport protocol specific feature and if they expose it to the ROS2 middleware then several transport layer implementations may lack this feature. Currently, the DDS traffic generated by ROS2 Humble has DSCP value set to 0×00 in their flows. Independently from ROS2, FastDDS itself has no API to set up diffserv values. This means that the network must provide default QoS treatment for these flows. A further issue apart from setting the diffserv value is that intermediary routers can reset or change the DSCP value within flows making it risky to use DS-based QoS. The careful configuration of intermediary routers to preserve DSCP indications from incoming to outgoing traffic is one approach.

A tailor-made priority field in the custom ROS2 tunnel header is a feasible solution for the setup phase. It is only an issue during the parsing phase to prepare e.g., the base station to parse into the ROS2 topic header to look for the headers.

2) MULTIPLE TUNNELS

The UFTR can be extended with one more feature as it is used in topic tools ''mux'' [20]. First, the application publishes a topic tagged with `_qos_out`. The UFTR subscribes to it with a general subscriber (see Fig. 11). Second, several publishers are initiated functioning as the tunnel start-point. The topic messages are encapsulated in a new message header. In this operation mode of each publisher the topics are transmitted via QoS handling, thus the unique network flow endpoint option is set. The published topic tunnels are tagged with the priority level or QCI value ''_QCI_8''. The UFTR receives from the publisher either via an external control plane topic when to enter certain traffic priority handling modes and the UFTR sends the incoming topic messages to the specific outgoing topic with the unique flow id and QCI tagging. The QoS mapper establishes dedicated bearers with NRM requests to the known 5-tuples, thus proper QoS handling is ensured for the QCI tagged topics (see Fig. 10).

The same UFTR has the second operation mode, in which it subscribes to all the encapsulated topics with the various ''_QCI'' tags. The header is parsed, and the binary data stream is forwarded through a general publisher. This step is indicated in the topics naming with a `_qos_in` tag that this topic is decapsulated and can be forwarded towards the application.

VI. EVALUATION

As described in Section III-C1 the use case on which we tested the proposed method is a simple simulator for learning ROS2 is called Turtlesim [29].

A. MAIN OPERATION MODES

There are three operation mode of the proposed method that we can evaluate:

- static NRM: The ROS2 nodes initialize, the topic watcher node performs the NRM request towards the 3GPP right at the beginning. The QoS of the channel is intact until the end of the session.
- dynamic NRM: There is a periodic or event triggered recurrent remapping of the ROS2 QoS to the 3GPP QCI. (Both static and dynamic NRM are discussed in Section III).
- AI-enhanced dynamic NRM: The QoS requested by the application is optimized by an AI-agent to relax the 3GPP QCI requests even further while maintaining the application-level performance KPIs of the ROS2 application. (See Section IV.)

1) STATIC NRM

The static NRM case is well supported by both the ROS2 applications and both 3GPP NRM. It is a common practice to over-provision the network resource requests to ensure that the traffic of the application fits in the requested resources in case of high network load. Also, it is easier from the application developer's perspective to over-provision the requested network resources than to search for the minimal required resources.

2) DYNAMIC NRM

a: THE DESIGN OF ROS2

As ROS1 was not designed with QoS in mind, the QoS policies of ROS2 were a huge step toward feasible setup of more complex network settings. As the ROS2 design article summarizes, there was no common method of overriding QoS settings while constructing a node prior to ROS2 Foxy [45]. Reusing nodes that other people have created is popular in the ROS2 ecosystem, and in many use cases allowing QoS profiles to be altered makes sense. However, there are major issues with QoS reconfiguration during run-time. The main one is that only when the publisher and the subscriber QoS profiles are compatible can a link between a publisher and a subscription be established. A ''Request vs. Offered'' model is used to determine whether two QoS profiles are compatible. Publishers supply a QoS profile that is the ''maximum quality'' they can provide, while subscriptions want a QoS profile that is the ''minimum quality'' they are prepared to accept. Connections are only established if no requested QoS profile policy is stricter than any offered QoS profile policy. Publishers and subscribers need to have compatible QoS settings otherwise they either cannot publish or cannot subscribe. Overriding QoS settings run-time can end up losing the connection. In ROS2 Galactic, developers agreed on a way to support QoS overriding which is done via the `QoSOverridingOptions` [46] class that can be setup during initialization, run-time and have a callback function if it is running. There are a few well-known examples in the ROS2 community using QoS overriding. The most common one is ros2bag [38], but there can be more upcoming as the latest ROS2 releases become more widespread in the community.

The other requirement of a feasible dynamic NRM use case on ROS2 is the possibility of creating executors [47] with varying wake up frequency. ROS2 is designed to initialize executors with given frequency through the `rclcpp::Rate` object, but it is certainly possible to overcome the limitations by e.g., specifying multiple rate objects for the same executor and switching between them.

b: THE 5G NETWORK

A common method to implement a QoS for a certain flow for the NRM request in 5G is to establish a dedicated bearer with the requested QCI between the UE and the base station then make a routing rule to route a certain flow through the established bearer. In this concept, updating the QCI triggers a new bearer establishment and a new routing rule to route the flow through the newly established bearer. Establishing a new bearer takes time in a magnitude of a few seconds. Rerouting happens within 1 second. Updating a bearer means a tear down and a new bearer establishment. Though it is important to note that some default bearers are always present

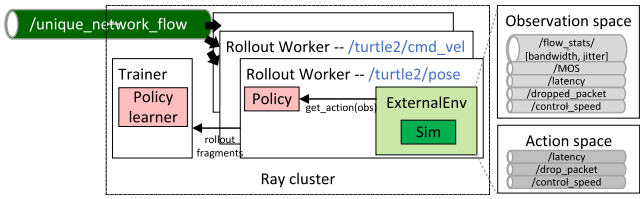


FIGURE 12. The architecture of the MOS/QoS ratio maximizer.

while connectivity is provided to the UE, connection outage does not happen during the bearer switching.

This means that NRM updates can be dynamic in a few seconds granularity. However, as current design patterns for both ROS2 and 3GPP SEAL are formed to make the NRM at the connection initialization and do not modify it during the lifetime of the connection, this is a low granularity NRM (even without in-band signaling).

c: THE PROPOSED METHOD TRANSPARENTLY SUPPORTS THIS TYPE OF OPERATION

The main point is to have a ROS2 application that has a topic for a sensor e.g., camera, lidar, or controller that supports dynamic data sending rate of its content.

In this case the application itself may still over-provision the required network resources if the automatic NRM process estimates the required network resource based on the published raw data of the application. E.g., the raw data from the sensor is transmitted through the network, though the speed of perception or the processing of the raw sensor data is slower than the raw data rate. Another common issue is that the actuator has magnitudes of lower control frequency than the sensor or the other way around, thus some part of the control-loop is over-represented in the network traffic and irrelevant in the final performance of the use case.

3) AI-ENHANCED DYNAMIC NRM

The case of over-provisioning by the application developer (see Section VI-A1) or by the application (see Section VI-A2), but it is not considered how the use case performs from the user's point of view in these cases. To search for the minimal required network resources that can still work well for the user, the MOS/QoS ratio maximizer is a key component (see Section IV-C2 for the concept).

a: MOS/QOS RATIO MAXIMIZER

The implementation of the MOS/QoS ratio maximizer is done as a ROS2 node.

Fig. 12 shows the architecture of the proposed system. The proposed solution is a reinforcement learning based solution. We implemented our solution using Ray 2.2.0 with RLlib [48]. Ray is a general-purpose and universal distributed compute framework to flexibly run any compute-intensive Python workload including distributed training, hyperparameter tuning or deep reinforcement learning. RLlib is an open-source library for reinforcement learning supporting highly distributed RL workloads. Within RLlib we used a multi-agent environment. The policy graph optimizer used the Proximal Policy Optimization (PPO) algorithm. The output of the system is optimized policy graphs for the ROS2 topic controlling agents.

RLlib supports environments for external agents and applications [49]. It is frequently unnecessary for an environment to be "stepped" by RLlib. For instance, it makes more sense for an agent to query a service that serves policy decisions and for that service to grow via experience over time if a policy is to be employed in a web serving system. This situation also occurs with external simulators that operate freely outside of RLlib's control but may still need to use RLlib for training. The ExternalEnv class from RLlib serves this function. ExternalEnv has its own control thread, in contrast to other environments. Agents on that thread are always able to query the current policy for decisions via `self.get_action()` and report rewards, done-dictionaries, and other information elements via `self.log_returns()`. This is also possible for multiple concurrent episodes.

Every topic that is handled by the UFTR has an associated AI-agent, a rollout worker in RLlib terms. The external *environment* is the ROS2 setup. The *observation space* is collected by subscribing to various ROS2 topics: bandwidth and jitter values of the flow statistics, MOS, latency, dropped packet and control speed.

The *action space* of the agents consists of the increasing or decreasing the latency, dropped packet, control speed with one minor step in the specific value set. The decided action of the agent is published through the above ROS2 topics.

Besides AI, many other types of controllers and heuristics could be used. Our aim is to prove the viability of the concept and give a universal solution that can be fine-tuned easily in an application specific way. With this RL-agent it is as simple as providing the application-specific MOS-scores and fine-tuning the weights in the reward function.

In the observation space, the *reward* is the function that enforces the learning algorithm to optimize onto it. The reward should contain information elements on those parts that can be influenced from the action space, everything else should go into the observation space. The weights on the reward elements influence the learning rate speed. If we set up the weights biased, then the learning rate could slow down so much that we do not see the convergence. Thus, the learning rate influences the success of learning in the end. The reward function for each topic is the following:

$$reward_{topic} = w_1 * mos + latency_{topic} - drop_{topic}, \quad (1)$$

where

- w_1 is selected to be 200 in our measurements,
- *mos* is score is defined as the square error between the desired position and current position of the turtle scaled with the process speed,
- $latency_{topic}$ is considered as the number of times the "UFTR with Throttle" delays the packet. Its effect depends on the topic update rate.

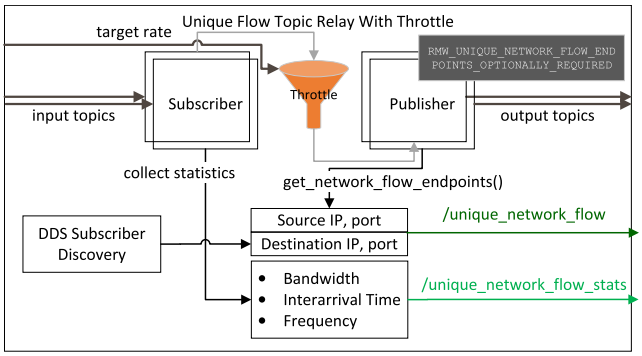


FIGURE 13. Architecture and information flow diagram of the Unique Flow Topic Relay with Throttle function.

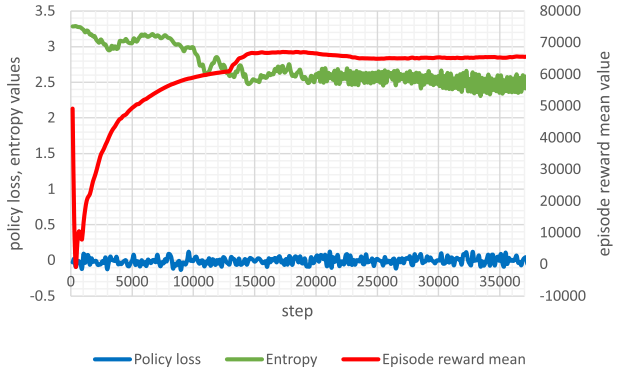


FIGURE 14. The training process.

	bandwidth	jitter	MOS	latency	dropped packet	control speed
/turtle1/pose	1484	0.0163	4.9405	0	248	
/turtle1/cmd_vel	525	0.0999	4.9405	1	254	0.5
/turtle2/pose	1500	0.0161	4.6904	4	252	
/turtle2/cmd_vel	131	0.4	4.6904	0	184	2

FIGURE 15. The final setup for each topic after learning.

- $drop_{topic}$ means that "UFTR with Throttle" drops every n^{th} packet of the topic. A higher value results in less drop.

Note that the reward function does not contain the VAL process speed. It is included in the MOS score provided by the custom turtle controller.

b: UFTR WITH THROTTLE

An option to collect the network statistics in userland code if the relay node itself adds the network effects to the topic. We aimed at extending UFTR to provide two new features: 1) introduce drop in the topic as it is done in topic tools "throttle" and "drop" [20] and 2) introduce network delay. With these features the relay node is aware of the introduced drop and delay, thus it is possible to know the degradation of the network statistics compared to the existing connection features. Obviously, this approach cannot enhance the network connection by any means.

Fig. 13 shows the architecture and information flow diagram of the UFTR with Throttle function. The throttle function has an external interface to control the drop rate and delay length. Note that as much as the UFTR is a sensor to collect network statistics, it also becomes a controller with the introduced throttle mechanism. This capability is exploited by the AI-agent to influence the traffic characteristics of the topic.

c: LEARNING PHASE

The PPO is configured in the following way. We set the `no_done_at_end` to true to keep the same simulation environment running and do not reset or exit. This induced the `soft_horizon` setting to true as well while the horizon size is 128. Detailed description of these parameters can be found in [50].

Fig. 14 shows the training process. The episode reward mean shows the achieved average reward of the policy across all the agents. In the first few hundred steps as the agents start to explore it drops significantly but soon performs better. At about 15k steps it achieves its maximum value during the training and at 20k steps the entropy shows that no further information can be learnt from the available action space. The policy loss function correlates to how much the policy (process for deciding actions) is changing. These values oscillate during training, but they usually fall below 1.0. The entropy shows how random the decisions of the model are. It should slowly decrease during a successful training process. The values in the figure show successful learning. Note that though each topic has a different rollout worker, the policy for all the agents is common. Therefore, it is the performance of the common policy that can be tracked on Fig. 14.

d: INFERENCE PHASE

Fig. 15 shows the final setup for each topic that the agents decided to apply after learning. Note that the two controllers of the turtles ended up in different strategies.

Turtle 1 halved its original speed (default speed value is represented with 1, the control speed values should be considered relative to this value) and decided to take only minor packet loss and minor latency achieving higher MOS score than Turtle 2 which took bigger packet loss and latency but doubled the speed of the turtle. These two turtles represent different use cases.

B. NUMBER OF LINES OF CODE (LOF) FOR CONFIGURATION

In Section III-C we discussed what are the required information elements and how to collect them. The estimated LOF per SEAL function can be seen in Fig. 16 with the corresponding LOF ratio. TS 23.434 [8] §14.3.2.13 defines the "end-to-end QoS management request" from the NRM client to the NRM server. The request contains mandatory and optional information elements. Though to create a valid API call, only the mandatory elements are required, but as

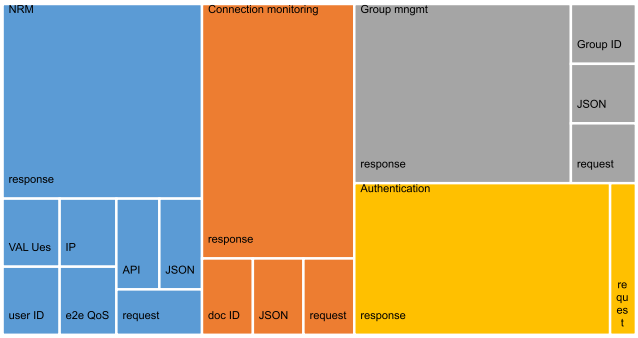


FIGURE 16. Estimated Lines of Code for the SEAL functions.

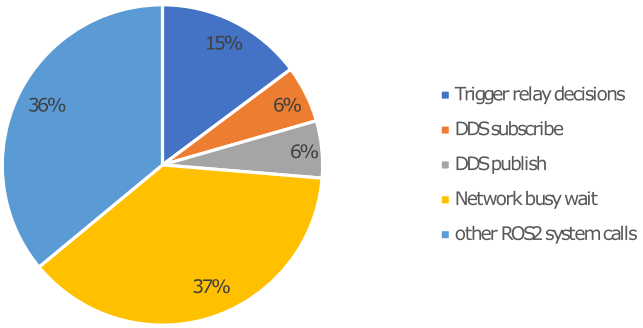


FIGURE 17. CPU cycles on certain functions in the Unique Flow Topic Relay (UFTR).

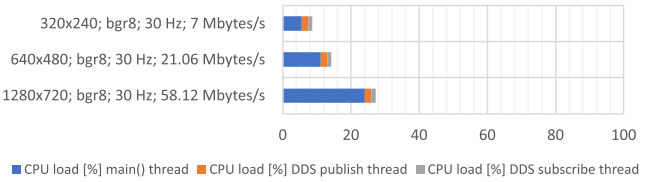


FIGURE 18. CPU load on the system of the Unique Flow Topic Relay (UFTR) node with various input topics.

the end-to-end QoS requirement field defines the requested QoS and it is optional, a meaningful NRM request needs an optional field as well. The following 4 information elements need to be set: list of VAL UEs, user ID, IP address, end-to-end QoS requirements. We can assume that each information element requires at least one LOF to collect. An HTTP handling library needs to be imported to make the API call. The POST messages need to be set up in proper JavaScript Object Notation (JSON) format. Also, the responses need to be parsed or event-based callbacks need to be implemented.

If connection monitoring and group management are important for a certain use case, they require at least one API call per function towards the SEAL server whose response needs to be parsed. Usually, authentication is also required towards these servers, which is at least one more API call.

The proposed method's QoS mapper node executes the mapping results via an abstract class which makes it possible to connect various API's by implementing various API calls for the same request, thus making SEAL API calls exchangeable for e.g., NEF APIs calls (TS 23.501 [6], 23.502 [7] and 29.522 [4]) easily. The SEAL API handler for the above functions is implemented in approx. 200 LOF. The proposed method provides the described functionalities with a few extra characters onto the existing code at the topic names, or a few lines to remap the topic names in a launch file or command line.

C. PROCESSING OVERHEAD OF THE ADDED NODES ON THE SYSTEM

We discuss three main components with their performance analysis. Other components not discussed here are more like a few lines of code without any considerable performance impact.

a: UNIQUE FLOW TOPIC RELAY

The relay nodes are responsible for handling the majority of information gathering tasks. These nodes can be affected by the trespassing traffic, which requires specific quality of service (QoS) management.

We compiled our node with debug symbols to make a detailed call stack analysis with the `perf` [51] tool. Fig. 17 shows the ratio of Central Processing Unit (CPU) cycles spent in various functions of the UFTR. The decisions connected to new DDS topic events and the network statistics calculations cost 14.8% of the total CPU cycles. The subscription on the DDS topic and the republishing of the received topic messages cost 5.76% CPU cycles each. The node spends a 37.7% of CPU cycles in network busy wait, while and other 36% of the CPU cycles are spent on other ROS2 related system calls.

We did performance load tests on the UFTR node. We repeated the tests with the streamed camera video as it is done in the ROS2 documentation for lossy network traffic test [52]. The camera was set up in various resolutions to cause various loads on the UFTR node. Fig. 18 shows the CPU load on the system of the UFTR node with various input topics. Note that ROS2 nodes running locally on the same machine can utilize various optimizations of FastDDS. FastDDS employs fast intra-process communication solutions [53] like shared memory (SHM) [54], data sharing delivery [55] or zero-copy communication [56] in a local machine. The two most beneficial features of SHM for our use case are the following. First, it reduces the number of memory copies by using SHM transport, which allows all target endpoints to share a single memory buffer when transmitting the same message to many endpoints. Other protocols demand that each endpoint receive a single copy of the message. Second, it causes less operating system overhead: compared to the other protocols, shared memory transfers require significantly fewer system calls when initial setup is finished. As a result, employing SHM improves performance and time consumption.

We switched off the optimized message communications of FastDDS to check for the worst-case scenario. The results

shown in Fig. 18 like 9%, 14% and 27% CPU load (Intel i7 9700k) for 320 × 240, 640 × 480 Standard Definition (SD) and 1280 × 720 High Definition (HD) camera resolutions can be lower if the specific ROS2 setup can employ any of the FastDDS optimizations. For performance test results, how these solutions optimize CPU load see the latency [57] and throughput [58] performance tests.

We argue that the relay node is not a significant overhead to utilize. Instead of the original direction of the topic towards the network stack, the topics are routed through the relay nodes, leveraging optimized inter-process communication within the local machine.

b: QOS MAPPER

QoS mapper runs in case new network statistics data is received on the ''unique_network_flow_stats'' topic. The period of the network data calculation can be set up by the user. The related calculations for determining the required QCI are not significant. Defining the HTTP requests with the proper JSON parameters requires several string operations. The number of callbacks is in the function of the newly created and destroyed topics. In current robotics use cases the topic is created mostly during the system initialization, and they are static for the rest of the lifetime. We expect that the QoS mapper does not create a significant load on the system.

c: MOS/QOS RATIO MAXIMIZER

The MOS related scoring and throttling of applications have one component only that can have significant computation load. That is the MOS/QoS ratio maximizer node with its AI agent utilizing RLlib [59]. After collecting some observations, it starts to optimize a policy which can utilize some CPU. This CPU consumption can be relaxed by utilizing the Graphical Processing Unit (GPU) instead, making the policy optimization off-line or on an external machine. Once the policy is considered ready, the inference phase of the AI-component does not cause any significant system load. Further, the AI-component is an example for the demonstration of the concept and can be exchanged with a more use case optimized algorithm.

D. EXECUTION SPEED OF THE NETWORK EXPOSURE REQUESTS

To evaluate the execution speed of the network exposure requests we need to discuss how the bearer management occurs based on TS 23.501 [6] and TS 23.434 [8].

Initially, a default bearer with a certain QCI, e.g., QCI 83 is created. The terminal and the base station become connected. When a new NRM request is performed with a different QoS requirement, the network establishes a new dedicated bearer towards the UE. Update of an existing unicast resource is not supported by Rel-17 SEAL, there are create and delete procedures defined only. To overcome this, the UFTR tunnel mode can be used to set up all the possible bearers during the lifetime of the ROS2 application and the tunnel mode emulates the QoS updates by switching between the various tunnels.

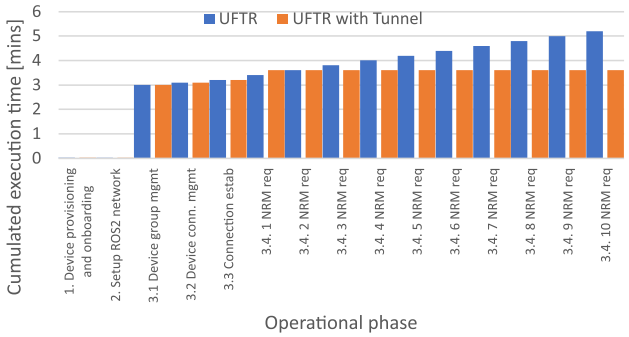


FIGURE 19. The cumulative execution time of the operational phases.

Fig. 19 shows the comparison of the cumulative execution time of the operational phases when the basic UFTR mode and when UFTR with tunnel mode is applied. We performed the measurements in a private 5G NSA network without any background traffic or load on the system apart from the tested UE. The NRM update is performed by a unicast resource deletion followed by a unicast resource creation procedure. Section III-D describes the required steps in detail. Fig. 19 shows that in the initial setup phase the two modes require the same time to execute a NRM update, but after the UFTR with tunnel creating all the required bearers during the initial setup, it does not perform any further bearer or IP routing reconfiguration. The QoS updates are managed by the UFTR itself with in-app routing in the ROS2 layer. The required time for the NRM switch is sub-second level while the bearer setup requires several seconds. Note that the exact values are terminal and network dependent, the magnitude of duration is the most important factor that we should consider.

VII. CONCLUSION AND FURTHER WORK

This paper proposes a solution to make the use of SEAL even easier and more automatic for industry verticals by introducing a mapping node and an information collecting proxy node. We demonstrate the feasibility and light-weight nature of the proposed solution, which covers three SEAL functionalities and can automatically map ROS2 application layer information elements to 3GPP SEAL requests. This solution offers novel deployment options like the VASEAL and tunneling mode that improves the network's action radius and provides an aid to overcome some limitations of current network deployments. Overall, this solution offers a convenient way to apply SEAL in various scenarios and deployments, with minimal input required from the ROS2 application developer.

The proposed method is evaluated in terms of LOF for configuration, the processing overhead of the information collecting node, the mapper node and the MOS/QoS ratio maximizer node. The required extra configuration efforts of the ROS2 user are as minimal as adding a few extra characters to the topic to be treated with QoS on the radio. The UFTR

node adds an extra 27% load on the CPU with a HD camera stream, but significantly lower for any control or feedback topic. We also proposed an AI-agent that can influence the application layer controller and application execution speed and resolution to maintain the quality of experience of the process for the user but reducing the required QoS of ROS2 topics. We also evaluated the NRM in terms of execution time and showed the tunnel mode of the UFTR can speed up the network resource adaptation in case of frequent requests for change.

The proposed method can be extended to further SEAL functionalities not used in this work, including provisioning, connection management, device management, user profile retrieval, identity and key management, location reporting. It is also possible to integrate other VAL services and connect with other 3GPP SA6 standards like V2X, EdgeApp, Uncrewed Aerial Systems. In this paper we utilized the unicast flow management of SEAL. The TSN and multicast features of the SEAL NRM functions are good candidates for further work as well.

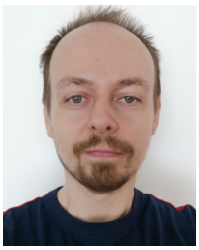
ACKNOWLEDGMENT

The author would like to extend his sincere gratitude to his colleagues József Petö, Áron Szabó, and Gergely Seres for their invaluable support and insightful discussions throughout this project.

REFERENCES

[1] (2023). *Implementation of the Proposed System*. [Online]. Available: https://github.com/Ericsson/ros2-3gppSA6-mapper
[2] (2023). *Demo Video of the Proposed System in Action*. [Online]. Available: https://youtu.be/JXGAHvDSU4o
[3] (2021). *The 3rd Generation Partnership Project*. [Online]. Available: https://www.3gpp.org/
[4] *5G System; Network Exposure Function Northbound APIs; Stage 3*, document 3GPP TS 29.522, Jun. 2021. [Online]. Available: https://www.3gpp.org/ftp/Specs/archive/29_series/29.522/29522-h20.zip
[5] (Jun. 2011). *Network Configuration Protocol (NETCONF)*. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6241
[6] *System Architecture for the 5G System (5GS)*, document 3GPP TS 23.501, Jun. 2021. [Online]. Available: https://www.3gpp.org/ftp/Specs/archive/23_series/23.501/23501-h11.zip
[7] *Procedures for the 5G System (5GS)*, document 3GPP TS 23.502, Jun. 2021. [Online]. Available: https://www.3gpp.org/ftp/Specs/archive/23_series/23.502/23502-h10.zip
[8] *Service Enabler Architecture Layer for Verticals (SEAL); Functional Architecture and Information Flows*, 3GPP TS 23.434, Jun. 2021. [Online]. Available: https://www.3gpp.org/ftp/Specs/archive/23_series/23.434/23434-h20.zip
[9] G. Szabó, G. Seres, M. L. Mikecz, M. Hortobágyi, S. Rácz, J. Peto, and T. Cinkler, "Assessment of the efficiency of 5G network exposure for the industrial Internet of Things," in *Proc. IEEE Conf. Standards Commun. Netw. (CSCN)*, Dec. 2021, pp. 52–58.
[10] 5G-ACIA. (2021). *Exposure of 5G Capabilities for Connected Industries and Automation Applications*. [Online]. Available: https://5g-acia.org/whitepapers/exposure-of-5g-capabilities-for-connected-industries-and-automation-applications/
[11] A. Karaagac, O. Dobrijevic, D. Schulz, G. Seres, A. Nazari, H. Przybysz, and J. Sachs, "Managing 5G non-public networks from industrial automation systems," in *Proc. IEEE 19th Int. Conf. Factory Commun. Syst. (WFCS)*, Apr. 2023, pp. 1–8.
[12] G. Szabó, S. Rácz, N. Reider, J. Pető, and R. R. Aschoff, "Quality of control-aware resource allocation in 5G wireless access networks," in *Proc. IEEE 19th Int. Symp. World Wireless, Mobile Multimedia Netw. (WoWMoM)*, Jun. 2018, pp. 1–6.
[13] G. Szabo, J. Peto, L. Nemeth, and A. Vidacs, "Information gain regulation in reinforcement learning with the digital twins' level of realism," in *Proc. IEEE 31st Annu. Int. Symp. Pers., Indoor Mobile Radio Commun.*, Aug./Sep. 2020, pp. 1–7.
[14] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Sci. Robot.*, vol. 7, no. 66, May 2022, Art. no. eabm6074. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074
[15] (2021). *Ten Agility Aspects in a Closed Loop Control Modelling*. [Online]. Available: https://ieee-sa.imeetcentral.com/p/aQAAAAAEz9vp
[16] (May 2023). *FastDDS Filtering Data on a Topic*. [Online]. Available: https://fast-dds.docs.eprosima.com/en/latest/fastdds/dds_layer/topic/contentFilteredTopic/createContentFilteredTopic.html
[17] (May 2021). *ROS 2 Design, About Quality of Service settings*. [Online]. Available: https://docs.ros.org/en/rolling/Concepts/About-Quality-of-Service-Settings.html
[18] (May 2022). *ROS 2 Topic Hz*. [Online]. Available: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html#ros2-topic-hz
[19] (May 2022). *ROS2, About Topic Statistics*. [Online]. Available: https://docs.ros.org/en/humble/Concepts/About-Topic-Statistics.html
[20] (Oct. 2022). *ROS2 Topic Tools*. [Online]. Available: https://github.com/ros-tooling/topic_tools
[21] (Nov. 2022). *eProsima Fast DDS Statistics Backend*. [Online]. Available: https://www.eprosima.com/index.php/products-all/tools/eprosima-fast-dds-statistics-backend
[22] *Service Enabler Architecture Layer for Verticals (SEAL); Application Programming Interface (API) Specification; Stage 3*, 3GPP TS 29.549, Sep. 2022. [Online]. Available: https://www.3gpp.org/ftp/Specs/archive/29_series/29.549/29549-h60.zip
[23] *5G Security Assurance Specification (SCAS);Network Data Analytics Function (NWDAF)*, 3GPP TS 33.521, Jun. 2022. [Online]. Available: https://www.3gpp.org/ftp/Specs/archive/33_series/33.521/33521-h20.zip
[24] (May 2021). *Ananya Muddukrishna: ROS2 Design, Unique Network Flows*. [Online]. Available: https://design.ros2.org/articles/unique_network_flows.html
[25] (May 2021). *Support for Unique Network Flows*. [Online]. Available: https://docs.ros.org/en/galactic/Releases/Release-Galactic-Geochelone.html#support-for-unique-network-flows
[26] (Nov. 2022). *FastDDS DomainParticipantListener*. [Online]. Available: https://fast-dds.docs.eprosima.com/en/latest/fastdds/dds_layer/domain/domainParticipantListener/domainParticipantListener.html
[27] (May 2023). *rclcpp: GenericPublisher Class Reference*. [Online]. Available: https://docs.ros2.org/galactic/api/rclcpp/classrclcpp_1_1GenericPublisher.html
[28] (May 2023). *rclcpp: GenericSubscription Class Reference*. [Online]. Available: https://docs.ros2.org/galactic/api/rclcpp/classrclcpp_1_1GenericSubscription.html
[29] (May 2022). *Turtlesim Package From Ros_Tutorials Repo*. [Online]. Available: https://index.ros.org/p/turtlesim/github-ros-ros_tutorials/
[30] *Policy and Charging Control Architecture*, 3GPP TS 23.203, Dec. 2021. [Online]. Available: https://www.3gpp.org/ftp/Specs/archive/23_series/23.203/23203-h20.zip
[31] (Jan. 2023). *ROS2 Quality of Service Demo*. [Online]. Available: https://github.com/ros2/demos/blob/master/quality_of_service_demo/rclpy/quality_of_service_demo_py/liveliness.py
[32] (Oct. 2022). *Using Fast DDS Discovery Server as Discovery Protocol*. [Online]. Available: https://docs.ros.org/en/humble/Tutorials/Advanced/Discovery-Server/Discovery-Server.html
[33] A. Downs, Z. Kootbally, W. Harrison, P. Pilliptchak, B. Antonishek, M. Aksu, C. Schlenoff, and S. K. Gupta, "Assessing industrial robot agility through international competitions," *Robot. Comput.-Integr. Manuf.*, vol. 70, Aug. 2021, Art. no. 102113.
[34] *P.800.1: Mean Opinion Score (MOS) Terminology*. Accessed: Dec. 2022. [Online]. Available: https://www.itu.int/rec/T-REC-P.800.1-200303-S/en
[35] D. Minovski, C. Åhlund, K. Mitra, and P. Johansson, "Analysis and estimation of video QoE in wireless cellular networks using machine learning," in *Proc. 11th Int. Conf. Quality Multimedia Exp. (QoMEX)*, Jun. 2019, pp. 1–6.
[36] M. Eckert, T. M. Knoll, and F. Schlegel, "Advanced MOS calculation for network based QoE estimation of TCP streamed video services," in *Proc. 7th Int. Conf. Signal Process. Commun. Syst. (ICSPCS)*, Dec. 2013, pp. 1–9.

[37] *Transparent End-to-End Packet-Switched Streaming Service (PSS); Progressive Download and Dynamic Adaptive Streaming Over HTTP (3GP-DASH)*, document 3GPP TS 26.247, Jun. 2022. [Online]. Available: https://www.3gpp.org/ftp/Specs/archive/26_series/26.247/26247-h10.zip
[38] (May 2022). *ROS2, Recording and Playing Back Data*. [Online]. Available: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Recording-And-Playing-Back-Data/Recording-And-Playing-Back-Data.html
[39] (Nov. 2022). *Chronyc*. [Online]. Available: https://chrony.tuxfamily.org/index.html
[40] *Network Time Protocol Version 4: Protocol and Algorithms Specification*, document RFC 5905, Jun. 2010. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5905
[41] *New Terminology and Clarifications for Diffserv*, RFC 3260, Apr. 2002. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc3260
[42] G. Szabo, S. Racz, S. Malomsoky, and A. Bolle, "Potential gains of reactive video QoE enhancement by app agnostic QoE deduction," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1–7.
[43] (Oct. 2020). *Diffserv to QCI Mapping, Expired Internet-Draft (Individual)*. [Online]. Available: https://datatracker.ietf.org/doc/draft-henry-tsvwg-diffserv-to-qci/
[44] (Aug. 2020). *Transport Priority QoS Policy to Solve IP Flow Ambiguity While Requesting 5G Network QoS*. [Online]. Available: https://discourse.ros.org/t/transport-priority-qos-policy-to-solve-ip-flow-ambiguity-while-requesting-5g-network-qos/15332
[45] (Nov. 2020). *External Configurability of QoS Policies*. [Online]. Available: http://design.ros2.org/articles/qos_configurability.html
[46] (Nov. 2022). *QoS Overriding Options in Rclpy*. [Online]. Available: https://github.com/ros2/rclpy/blob/rolling/rclpy/rclpy/qos_overriding_options.py
[47] (Jan. 2023). *ROS2Executors*. [Online]. Available: https://docs.ros.org/en/rolling/Concepts/About-Executors.html
[48] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, J. Gonzalez, K. Goldberg, and I. Stoica, "RLLib: A composable and scalable reinforcement learning library," 2017, *arXiv:1712.09381*.
[49] (Dec. 2022). *External Agents and Applications*. [Online]. Available: https://docs.ray.io/en/latest/rllib/rllib-env.html#external-agents-and-applications
[50] (Dec. 2022). *Specifying Rollout Workers*. [Online]. Available: https://docs.ray.io/en/latest/rllib/rllib-training.html
[51] (Nov. 2022). *Perf: Linux Profiling With Performance Counters*. [Online]. Available: https://perf.wiki.kernel.org/index.php/Main_Page
[52] (Nov. 2022). *Using Quality-of-Service Settings for Lossy Networks*. [Online]. Available: https://docs.ros.org/en/humble/Tutorials/Demos/Quality-of-Service.html
[53] (Feb. 2021). *eProsima Fast DDS: From Shared Memory to ZERO COPY*. [Online]. Available: https://discourse.ros.org/t/eprosima-fast-dds-from-shared-memory-to-zero-copy/18877
[54] (Oct. 2022). *FastDDS Shared Memory Transport*. [Online]. Available: https://fast-dds.docs.eprosima.com/en/latest/fastdds/transport/shared_memory/shared_memory.html
[55] (Nov. 2022). *FastDDS Data-Sharing Delivery*. [Online]. Available: https://fast-dds.docs.eprosima.com/en/latest/fastdds/transport/datasharing.html
[56] (Nov. 2022). *FastDDS Zero-Copy Communication*. [Online]. Available: https://fast-dds.docs.eprosima.com/en/latest/fastdds/use_cases/zero_copy/zero_copy.html#
[57] (Feb. 2021). *Fast DDS V2.2.0 Latency Performance*. [Online]. Available: https://discourse.ros.org/t/fast-dds-v2-2-0-latency-performance/18989
[58] (Mar. 2021). *Fast DDS V2.2.0 Throughput Performance*. [Online]. Available: https://discourse.ros.org/t/fast-dds-v2-2-0-throughput-performance/19218/1
[59] (Nov. 2022). *RLlib: Industry-Grade Reinforcement Learning With TF and Torch*. [Online]. Available: https://github.com/ray-project/ray/tree/master/rllib



GÉZA SZABÓ (Senior Member, IEEE) joined Ericsson Research as an undergraduate student in 2005. He wrote the M.Sc. thesis about comparing various application traffic classification methods in 2006. He holds a Ph.D. degree in informatics since 2011. Since 2017, he works in the field of Industrial 4.0 and evolves robot cells into cyber-physical production systems via the help of 5G and AI. He was a delegate in 3GPP SA6 in 2021-2022 and the vice-chair of IEEE P2940 standardization group.