Vanetza-NAP: Vehicular Communications and Services in MicroServices Architectures

Rodrigo Rosmaninho*†, Andreia Figueiredo†, Pedro Almeida*, Pedro Rito*, Duarte Raposo*, Susana Sargento*†
*Instituto de Telecomunicações, 3810-193 Aveiro, Portugal
†DETI, University of Aveiro, 3810-193 Aveiro, Portugal

***Abstract*—Vehicle-to-everything represents a major step in the evolution of Intelligent Transportation Systems (ITS), by allowing connected and automated vehicles, and the infrastructure, to share information seamlessly. Vehicular use cases with reduced latency in data processing and transmission require the utilisation of Vehicular Edge Computing (VEC). VEC needs to integrate ITS services comprising communication architectures and message formats that are accessible and efficient, and ensure timing constraints. For this purpose, this paper discusses some of the issues present in current vehicular protocol stack, and proposes Vanetza-NAP, a microservice architecture for ITS-G5 communications. Vanetza-NAP adds new features to Vanetza and support for: (1) new ITS message types, (2) runtime configuration mechanisms that facilitate orchestration, (3) parallelised design to handle multiple messages simultaneously, and (4) integrated messaging technologies to connect with applications (MQTT and DDS). An evaluation is performed to the new architecture, by measuring the delay introduced by different messaging technologies, the impact of the parallelized design, and the overhead of the microservice-oriented approach. The results demonstrate the benefit of the parallelized implementation and priority-based message queuing. The enhanced interoperability and extended capabilities introduced justifies the small overhead in delay to support the microservice-based edge architecture.**

***Index Terms*—ITS-G5, V2X, Vehicular Edge Computing, C-ITS, Data Distribution Service, Multi-access Edge Computing**

I. Introduction

The emergence of Smart City infrastructures and the Multi-access Edge Computing (MEC) paradigm has enabled new 5G verticals, such as connected and automated vehicles with Vehicle-To-Everything (V2X) communication. This has brought Vehicular Edge Computing (VEC) to the forefront of research, since it allows vehicular use cases to leverage proximity in order to achieve reduced latency in data processing and transmission. This is especially relevant in the case of Intelligent Transport Systems (ITS) applications, due to their strict timing constraints and need for consolidation of data from multiple sensors and other sources on the VEC domain.

Historically, Smart City services were developed as closed systems specific to each city's verticals, wherein sensors, networks, and processing devices remained isolated within each service provider's domain. However, recent trends catalysed in part by the emergence of 5G private infrastructures and vehicular networks with VEC capabilities point towards a shift, allowing public infrastructure to be shared among multiple providers and thereby reducing costs and increasing the potential for cross-domain integrations. As such, the notion that the various functions of an ITS Road-side Unit (RSU) are necessarily deployed in bespoke, exclusive, hardware has become antiquated, in favour of deployments at least in part made to a shared MEC infrastructure that is centrally managed through an orchestration system. Therefore, modern ITS solutions should be designed with architectural principles that leverage the distributed computing paradigm, and also create beneficial synergies with the orchestrator itself.

At the same time, ITS use cases have been expanding in scope in a way that broadens the amount and types of information provided by road-side infrastructure, thus reinforcing the collective perception world model, and enabling novel interactions with third-party infrastructures, such as electric vehicle recharging stations. Additionally, there is also an increasing need for the inclusion of ITS information provided by the vehicles into existing data collection pipelines, in order to facilitate traffic pattern prediction, congestion analysis, and other machine learning & data science use cases that inform city planning and other applications. In both cases, it is clear that there is an emerging trend for ever greater integration of ITS services with other VEC and Smart City applications and sensors, which informs the need for communication architectures and message formats that are accessible and efficient, while ensuring timing constraints can still be met.

This work aims to address these issues through Vanetza-NAP, an extension to Vanetza [1] through a microservice based approach, with new ITS message types, runtime configuration mechanisms that facilitate orchestration, parallelised design to handle multiple messages simultaneously, and integrated messaging technologies to connect with applications (MQTT and Data Distribution Service - DDS). The results show the improvement in terms of delays of the different processes and the communication protocol, which makes Vanetza-NAP suitable for time-constrained services. Vanetza-NAP has been very successfully used in both education and research in ITS networks and services.

The remaining of this paper is organised as follows. Section II presents the related work, and Section III explores current monolithic architectures and some of their shortfalls. Our proposed approach is discussed in Section IV, and its mechanisms for a microservice based ITS protocol stack service are presented in Section V, along with their relevant features and design considerations. Section VI, presents and discusses the results, and finally, Section VII presents the conclusions and directions for future work.

II. Related Work

In order to realise the goal of ITS stations running on a shared MEC infrastructure, one of the most important components is the ETSI ITS protocol stack. However, proprietary implementations are generally restricted to specific hardware, or are otherwise ill-suited for this application, as explored in [2], thus favouring the use of Open-Source alternative implementations. Currently there are three projects that have seen some development and use, as identified in [3] and [4]. Namely: Alex Voronov's GeoNetworking stack [5], developed in Java; CCS Labs' OpenC2X experimental and prototyping platform [6], developed in C++; and Technische Hochschule Ingolstadt's Vanetza [1], developed in C++. Unfortunately, the Java stack and OpenC2X have not seen active development in several years, which is problematic for several reasons, most notably in terms of C-ITS message type support, where they are limited mostly to CAM and DENM messages. OpenC2X includes CAN/OBD support, but lacks the Security and GeoNetworking functionalities, as identified in [4].

From the application architecture perspective, there are multiple examples of vehicular network performance studies and ITS cooperative awareness & perception applications that use the aforementioned protocol stacks in a typical monolithic design: [4] uses Vanetza; [7] uses OpenC2X; and [8] uses the Java GeoNetworking stack. Microservice architectures are much less common: in [9], the GeoNetworking Java stack is used to convert incoming CAM and DENM messages into JavaScript Object Notation (JSON) and publish them into Message Queuing Telemetry Transport (MQTT) topics. However, this approach is limited to the incoming direction, and only supports CAM and DENM messages. The work in [10] presents an architecture where a collision avoidance service exchanges messages in JSON format with Vanetza through MQTT in both directions. This proposal is the most similar to the goals of our work, but is limited in terms of message type support (CAM and DENM messages), and in terms of communication strategy (MQTT).

From the network performance perspective, there are several studies that compare the performance of the ITS-G5 and C-V2X technologies, from a theoretical perspective [11], using simulation environments [12] or on real deployments [13]. For instance, the work presented in [11] analyses the evolution of the two protocol stacks on the physical and MAC layers, including a comparison of the access layer technologies of LTE-V2X and NR-V2X in C-V2X. However, studies that compare the performance and architecture of each available protocol stack are missing. Additionally, as presented in [11], there are some studies that analyse some of the performance characteristics and limitations of current publish/subscribe protocols. However, current V2X studies mostly address the time-critical requirements from the network latency and jitter perspective. C-ITS use cases generally operate with strict timing restrictions. One relevant example is the use case of awareness of the presence of a Vulnerable Road User (VRU), in which an alert to the user should be generated within an upper-bound of $100\,\mathrm{ms}$, with a recommended communication latency of $20\,\mathrm{ms}$. Some real measurements of the communication delay on IEEE 802.11p can be found in [14] and [15], where authors show values within the range of a few milliseconds. The work in [16] considers both the communication delay and the processing time of the C-ITS messages, using the aforementioned Java protocol stack. The results present an average delay of $50\,\mathrm{ms}$, highlighting the importance of considering the processing delay of such a system.

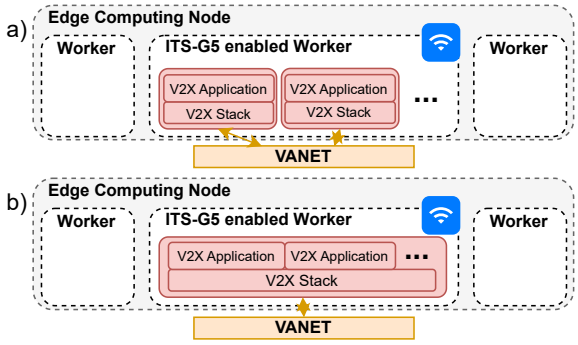

Fig. 1: Typical Monolithic ITS Application Architectures

III. Vehicular Edge Computing Architectures

The most typical architecture used by ITS applications follows a monolithic paradigm where the implementation will, by necessity, include a third-party library or Software Development Kit (SDK) that performs the role of the ITS Protocol Stack. Conceptually, this approach can be accomplished with two separate strategies, which are represented in figure 1.

The first strategy, *a)*, is to implement each ITS application as its own separate service with its own hooks into the protocol stack. This method ensures separation of concerns, but also presents notable drawbacks. Since each application is expected to handle the actual transmission and reception of messages, every application that manages a vehicular network use case is restricted to deployments on worker nodes that support ITS-G5 connectivity capabilities. This limits an orchestrator's ability to balance the resource usage throughout the cluster's worker nodes, and may impose an upper bound limit on the number of ITS applications that each infrastructure node can feasibly support. The use of this architecture in containerised environments raises additional issues such as weakened isolation between running workloads, since each container requires host mode networking capabilities in order to access the raw ITS-G5-enabled wireless network interface. This restriction hinders the overall security of the system by decreasing the level of isolation between running workloads, and may also preclude the applications from accessing advanced network features that are available in namespaced network environments.

Finally, it is important to take into account that multiple ITS applications may need to receive and interpret the same ITS message types in order to perform their functions. As a result, those incoming messages need to be processed and decoded by the protocol stack layer of each of the interested applications.

This is also relevant for unwanted message types, since there is no native filtering mechanism that can be applied at a lower level of the Linux networking stack to automatically select packets by their BTP destination port or C-ITS message type. As such, the only means of ascertaining the message type of an incoming transmission is to at least partially decode it first, which, in practice, results in several redundant processing cycles spent performing the exact same operations multiple times, only for some of the results to be potentially discarded. This is especially relevant in the case of resource-constrained Single-Board Computers (SBCs). As an alternative, vehicular network use cases can be implemented using strategy *b)* identified in figure 1, which combines the logic of every ITS application into a single codebase, thus mitigating some of the aforementioned problems. However, this solution inevitably brings several development and usability issues stemming from the size of the codebase and the inherent difficulties in maintaining it. And, crucially, the proposal still cannot be considered an orchestration or cluster-friendly architecture, since it severely limits the level of control that the orchestrator can exert in order to prioritize one ITS use case over another, and maintain cluster balance. Finally, in both alternatives, the choice of programming language/framework when implementing new ITS applications is restricted by the need to use the protocol stack library.

IV. MicroService-based Architecture

As an alternative, this work proposes a microservice-oriented approach, Vanetza-NAP, that decouples the protocol stack from the implementation of each vehicular use case. Under this proposal, depicted in figure 2, the protocol stack is deployed as its own standalone service that serves as the system's single point of entry/exit for C-ITS messages, and performs all the respective encoding/decoding operations. In turn, ITS applications are implemented as individual services that interact with the ITS stack through Inter-Process Communication (IPC) mechanisms, in order to receive incoming messages or instruct the stack to send new outgoing ones. This communication can be accomplished using a simpler, universal, representation of ITS message contents (e.g., in JSON), thus avoiding the inclusion of extensive Abstract Syntax Notation One (ASN.1) encoding logic in every application.

This strategy successfully addresses the aforementioned drawbacks of monolithic solutions in regards to their ineffective use of processing power, and greatly simplifies the development of applications that implement vehicular use cases. Furthermore, this separation of the ITS monolithic into a set of individual microservices also presents several advantages when considered in the context of cluster environments. Since each ITS application is subject to a separate worker node scheduling process, the cluster's orchestration system can maintain a balanced cluster state in terms of resource utilisation more effectively. This is especially relevant in cases where the orchestrator determines that one or more ITS applications cannot be feasibly allocated to the same node as the protocol stack service, and must therefore be placed

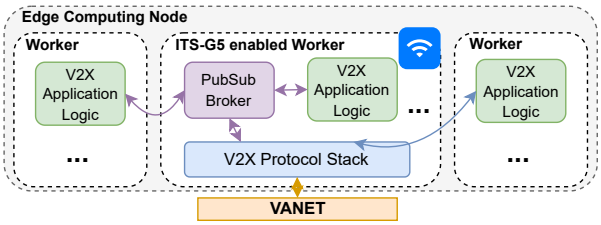

Fig. 2: Microservice-based ITS Application Architecture

in neighbouring nodes to ensure stable performance levels, even if the application incurs a slight latency penalty as a result. Additionally, ITS services can frequently require multiple dependencies with varying levels of importance, which are not necessarily present in the same worker node as the protocol stack. In a microservice architectural paradigm, the orchestrator can act in accordance to these types of constraints for each specific application, without affecting the others. Furthermore, since only the protocol stack service requires access to raw host-level network interfaces, ITS applications no longer require the use of host-mode networking, which increases service isolation, and overall system security.

Under this microservice-oriented architecture, the communication between the protocol stack service and its client ITS applications can be conducted using any available technology. Given the timing restrictions imposed on some ITS applications, the use of direct connections (e.g., UDP/TCP sockets) minimises the amount of traffic hops and overall delay. However, this type of messaging solution can become overly restrictive in this context, since the incoming C-ITS messages are expected to be consumed by multiple different ITS applications and other services. Moreover, the deployment of a new consumer or producer of ITS messages should not require any changes to the protocol stack's implementation.

Instead, this proposal suggests the use of technologies that implement the Publish and Subscribe (PubSub) paradigm, such as the MQTT protocol, which allows for a virtually unlimited number of producers and consumers that can be integrated into the communication architecture in a plug-and-play fashion, without requiring any modifications to the protocol stack's base code. Nevertheless, the centralised nature of some Pub-Sub implementations presents a very significant drawback, since it adds a message broker as a mandatory additional hop, thus incurring higher latency. However, it is important to consider that, at scale, this effect is counterbalanced to some degree by the aforementioned efficiency gains to the processing latency of both the applications and the protocol stack. This issue can be further mitigated by a careful selection of communication technologies that take into account the nature of each application and its particular timing restrictions.

V. Communication, Orchestration and Emulation

As part of this work, we developed the new type of ITS protocol stack named Vanetza-NAP, since it was built upon the open-source V2X Protocol Stack Vanetza [1]. This section describes the features introduced to extend the original Vanetza

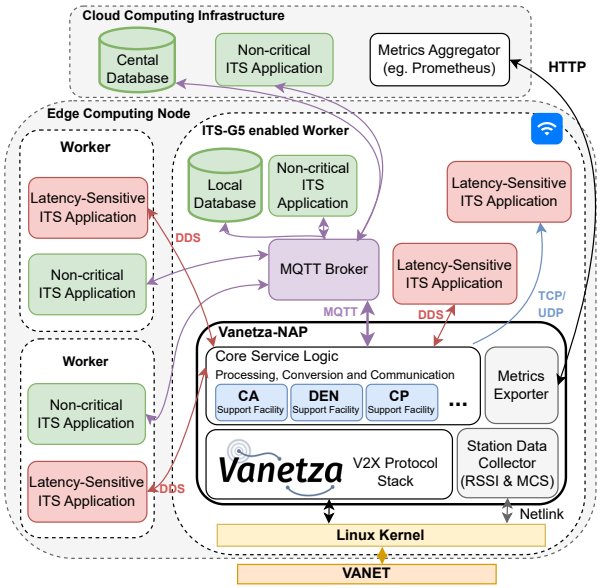

Fig. 3: Vanetza-NAP Architecture and Usage Examples

into a standalone service that supports MEC scenarios at scale, while still supporting time-sensitive applications (figure 3). This library is open-source and publically accessible[1].

A. Communication and encoding strategies

This approach supports an extensive set of ITS message types, which includes the messages supported by the base Vanetza library as well as some new additions, in Table I.

TABLE I: Supported Messages

Acronym	Message Type
CAM	Cooperative Awareness Message
DENM	Decentralized Environmental Notification Message
CPM	Collective Perception Message
VAM	Vulnerable Road User Awareness Message
SPATEM	Signal Phase And Timing Extended Message
MAPEM	MAP (topology) Extended Message
MCM	Maneuver Coordination Message (Draft Spec.)
SSEM	Signal Status Extended Message
SREM	Signal Request Extended Message
RTCMEM	RTCM Extended Message
IVIM	Infrastructure to Vehicle Information Message
EVCSN	Electric Vehicle Charging Spot Notification
EVRSR	Electric Vehicle Recharging Spot Reservation
IMZM	Interference Management Zone Message
TISTPG	Tyre Information System & Tyre Pressure Gauge

For each of these message types, Vanetza-NAP includes bespoke service logic that, conceptually, fulfils the role of the Application Support Domain Facility by "supporting the protocol processing" of the corresponding messages, as defined in [17]. In a broader sense, the Vanetza-NAP service as a whole also implements other Management, Application, and Information Support facilities specified in the standard, such as: ITS-S ID management, Time service, ITS-S positioning service, Data presentation, and Congestion control, among others. Finally, Vanetza-NAP and the underlying library also implement parts of the Transport & Network and Access Technologies layers: ITS Transport, GeoRouting, and ITS-G5, as well as some Security layer features like IEEE 1609.2 Signed Protocol Data Units (SPDU).

In order to exchange information with other services, Vanetza-NAP has concurrent support for two different message formats. The first format is the stream of Unaligned Packed Encoding Rules (UPER) encoded bytes produced using each message type's ASN.1 definition. This approach minimises both the message size, and the time spent on its transmission and marshalling/unmarshalling. However, this efficiency comes at the cost of added complexity on the side of client ITS applications. Alternatively, messages can be marshalled into JSON payloads that follow the exact schema defined in each message type's ASN.1 specification document. Despite incurring an efficiency cost in terms of both payload size and processing latency, we feel that using JSON greatly simplifies the design of client ITS applications and allows for an unmatched level of interoperability with different languages and frameworks, as well as different types of consumer services such as databases and real-time dashboards.

After being prepared, these representations of ITS messages are exchanged between the applications and the Vanetza-NAP protocol stack using a mix of communication technologies, which also depend on the constraints of each individual use case. The first such technology is the MQTT protocol. However, since the existence of a centralized message broker incurs added latency, the protocol is mainly intended to be used by generic, non-critical, applications and use cases.

In order to guarantee support for time-sensitive ITS applications and use cases without violating timing contraints, Vanetza-NAP also supports the use of OMG's Data Distribution Service (DDS), concurrently with the aforementioned MQTT strategy. Crucially, this technology still implements a PubSub paradigm, but through the use of a decentralized, broker-less, architecture with sophisticated peer discovery capabilities. The ITS messages are then exchanged between publishers and subscribers using either multicast transport, direct UDP sockets, or shared memory mechanisms (in cases where the peers are co-located and share an IPC namespace). The lack of the extra network hop typically incurred by a broker, coupled with the extensive list of Quality of Service (QoS) configurations that can be applied, result in significant reductions to the latency experienced by critical applications.

On balance, we feel that this mixed strategy allows Vanetza-NAP to achieve lower latency communications when required, while also retaining the greater interoperability of the MQTT protocol and JSON representation for the remaining use cases. Finally, the service also includes support for exchanging messages using direct TCP or UDP sockets, in order to ensure that timing constraints are still met in the exceptional cases where DDS cannot be integrated.

In conceptual terms, these communication methodologies implement the Facilities/Applications Service Access Point (FA-SAP) defined in [17], which links the ITS Applications (running separately) to the Facilities layer and beyond (implemented by Vanetza-NAP).

[1]https://code.nap.av.it.pt/mobility-networks/vanetza

B. Orchestration facilities and other features

Beyond the aforementioned features, this approach also includes additional considerations that help increase its synergy with any orchestrators managing its deployment.

Firstly, the protocol stack is fully capable of being executed in containerized environments, which is the main deployment mechanism aimed for the service. As such, the project includes Docker images built for both x86-64 and arm64 architectures, which are generated from a multi-stage build process designed to reduce the final image size as much as possible. This support is crucial, since most orchestration solutions rely heavily on the containerization paradigm. However, in the context of a critical service, it is important to consider that a containerised environment does incur a slight performance penalty that can vary depending on the application and the characteristics of the underlying computing node. In our testing [2] performed on resource-constrained edge computing devices, this effect was found to be measurable but not overly significant, especially if successfully mitigated using process isolation mechanisms.

Secondly, Vanetza-NAP includes a robust set of runtime configuration options and mechanisms that enable an orchestrator to deploy multiple instances of the protocol stack based on the same binary or container image, while still being able to fine-tune the behaviour of each individual instance. These configurable parameters include, among others: 1) Information regarding the specific ITS station, such as its ID and Type; 2) Connection details for the respective MQTT Broker; 3) PubSub topic names used for each message type and inbound/outbound flow; 4) Connection details for the Global Positioning System Daemon (GPSD) that provides positioning information; 5) Various feature flags for ease-of-use and debugging functionalities. These parameters can be defined either in Vanetza-NAP's configuration file, which follows the INI key-value pair schema, or as individual environment variables.

Finally, each instance of Vanetza-NAP continuously exposes a set of metrics in the background relating to its current state and performance level. An orchestrator can then monitor for changes in performance and react accordingly by increasing resource limits or evicting lower-priority deployments to maintain stable levels. To ensure consistency and interoperability, Vanetza-NAP exposes metrics using the Prometheus format. These exposed values include the number of messages that have been transmitted and received for each of the ITS message types, and, most importantly, the average processing latency observed (during the decoding and JSON generation tasks or vice-versa) for each message type and direction.

Vanetza-NAP also includes other note-worthy architectural elements, aimed at increasing the level of determinism to the processing time of the various message types. One such element is the use of an Extended Berkeley Packet Filter (eBPF) that is applied to the raw network socket to more efficiently filter incoming packets by the GeoNetworking Ethertype, thus avoiding extraneous CPU cycles spent on unrelated traffic.

The parallelised design of Vanetza-NAP's message processing pipeline ensures that the service is capable of handling

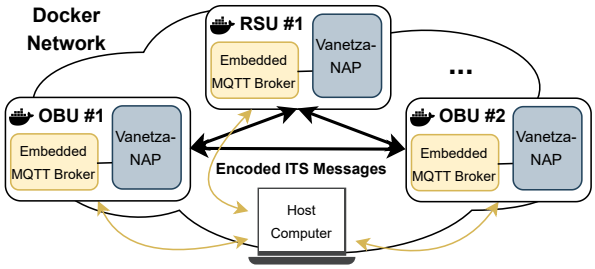

Fig. 4: Emulation use case example topology using Docker

multiple messages simultaneously. This is accomplished using a system of thread pools and priority queues, where arriving packets are sorted into a traffic class according to the criticality of the ITS message type (following the priority mappings set in [18]) and placed in a queue. Then, a collection of worker threads (whose number depends on the core count of the host CPU) continuously consume them and perform the required processing operations. This approach greatly reduces the risk of messages being delayed due to previous messages that are still being processed, as would happen in a more sequential pipeline design. Moreover, it ensures that lower-priority, often larger, message types do not starve out the processing bandwidth of more critical messages with restrictive timing constraints (eg. DENM). This parallelised architecture applies to incoming encoded ITS messages from the selected network interface, as well as message transmission requests sent by other microservices (also sorted by the communication mechanism; DDS peers with more priority than MQTT traffic).

Vanetza-NAP also includes other features that aim to provide additional useful metadata to ITS applications. One such functionality is the timestamping of key events during the lifecycle of an ITS message, described in detail in section VI. Using this information, both ITS applications and offline tools that analyse persisted data can determine the total time elapsed while performing the decoding, parsing, and JSON generation tasks (or vice-versa), which we define as the processing latency of the Vanetza-NAP service. Additionally, Vanetza-NAP also continuously communicates with the Linux kernel using a Netlink socket, to obtain the Received Signal Strength Indicator (RSSI) value for each ITS message received by another ITS station, as well as the current Modulation Coding Scheme (MCS) index that the host has selected for transmissions to that station. This information can be useful to some ITS applications in real-time, since they can be used to rank neighbouring stations by signal strength and throughput instead of just by geographical distance.

C. Emulation and educational uses

Since Vanetza-NAP can be executed in containerized environments, we can use it to very easily create an emulation environment with two or more Vanetza-NAP instances as ITS stations. The containers can use the underlying Docker network, in place of the wireless medium, where they can exchange encoded ITS messages, while simultaneously ex-

changing JSON and UPER representations of messages via MQTT and DDS, with ITS applications running as other Docker containers or on the host machine, as depicted in figure 4. To simplify this specific scenario, the Vanetza-NAP container image also includes its own built-in MQTT broker.

This ability to easily deploy different ITS scenarios with several stations in one PC, coupled with the ease of use of the JSON schema and MQTT protocol, enables Vanetza-NAP to be an effective educational tool, which has been included in the Autonomous Networks and Systems course of the Master's degree in Computer and Telematics Engineering at the University of Aveiro, since the academic year of 2021/2022[2].

VI. Performance Evaluation

This section details the evaluation of Vanetza-NAP, quantifying several metrics that are critical to understand its feasibility, namely: 1) the performance characteristics of each supported IPC mechanism and payload format; 2) the processing delay incurred by different ITS message types, as complexity and payload size are increased; 3) the impact of the parallelised pipeline design on the determinism of the results, when compared to a sequential approach; and 4) the overhead incurred by our microservice-based proposal relative to a more traditional implementation, in isolated conditions.

The experimental setup, depicted in figure 5, is composed of an On-board Unit (OBU) and an RSU powered by a resource-constrained PCEngines APU2 Linux SBC[3], used in real MEC deployments. The units share an IEEE 802.11p channel, and both contain a running instance of Vanetza-NAP, as well as an instance of the Mosquitto MQTT Broker and a C++ application responsible for performing the various tests and collecting results, all deployed within the Docker container runtime. For most of the test runs, the ITS producer app generates 5000 message requests at a rate of 10 Hz, and the results comprise the mean and 95% confidence intervals.

A. Communication protocol and data format

In the first scenario, the requests use different combinations of IPC mechanisms and payload formats: 1) MQTT & JSON; 2) DDS & JSON; 3) DDS & ASN.1 UPER. Each of these variations is tested once for each of the following ITS message types: CAM, DENM, CPM with 4 detected objects (CPM *SMALL*), CPM with 12 detected objects (CPM *BIG*), and MAPEM. Each message request is received by the OBU's instance of Vanetza-NAP and subsequently transmitted to the RSU's instance, where it is processed and sent to the ITS consumer application. The DDS participants on each ITS station are able to leverage the use of Shared Memory as the underlying IPC mechanism. The applications used by both the producer and the consumer record their respective transmission and reception instances. Vanetza-NAP also registers several relevant timestamps at different stages of the processing pipeline, which are sent after-the-fact to a testing-specific MQTT topic. The timestamps include: ❶

[2]https://www.ua.pt/en/uc/15279

[3]https://www.pcengines.ch/apu2.htm

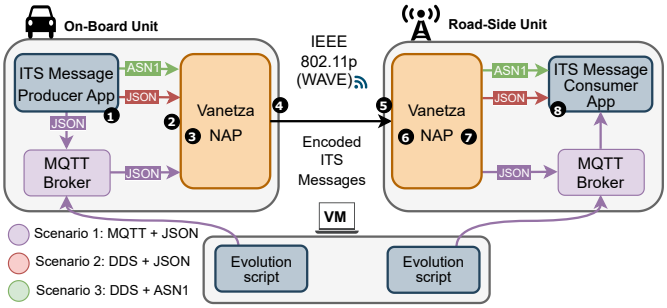

Fig. 5: Experimental setup for the Performance Evaluation

❷ reception of IPC request; ❸ consumption from queue by worker thread; ❹ end of encoded message transmission; ❺ reception of encoded message; ❻ consumption from queue by worker thread; ❼ generation of payload; ❽ end of IPC transmission. On the generation side, the duration of the MQTT/DDS transmission can be calculated from step ❶ to step ❸, and Vanetza-NAP's encoding time can be determined from step ❸ to step ❹. On the reception side, Vanetza-NAP's processing time while performing the decoding and representation generation tasks can be calculated from step ❺ to step ❼. Finally, the duration of the MQTT/DDS transmission can be determined from step ❻ to step ❼.

The results obtained for the three different combinations (MQTT & JSON; DDS & JSON; DDS & ASN.1 UPER) are shown in figures 6 and 7. From this data, the analysis indicates that DDS consistently transmits messages at a faster rate than MQTT, with approximately half of the mean delay value. Further analysis of the standard deviation values shows that DDS has a more stable timing profile, while MQTT's results show more fluctuations. These results demonstrate the efficiency and stability advantages of DDS over MQTT, which we attribute to its broker-less design and use of Shared Memory, among others. Additionally, the results also show that, increasing the message size and complexity has an adverse effect on both the IPC transmission time, and Vanetza-NAP processing delay of the messages. This effect is most pronounced when using the MQTT protocol and the JSON format, due to its inefficient use of payload size. Conversely, the effect is least pronounced when using the DDS protocol and the ASN.1 UPER format. We can draw a comparison between the performance of using JSON and ASN.1 as message formats. The analysis indicates that using ASN.1 results in substantially lower mean delay values for both message generation and reception when compared to JSON, since Vanetza-NAP's processing pipeline is much shorter when using this format. The IPC transmission time is slightly improved as well, due to the smaller payload.

These results confirm that the use of DDS and/or the ASN.1 UPER message format in this context can mitigate the overhead of a microservice-based ITS architecture and ensure lower and more deterministic end-to-end latency values when compared to alternative IPC solutions. If critical applications employ these technologies, this architecture is not in danger of becoming a bottleneck that jeopardises timing restrictions, especially when combined with the techniques described in

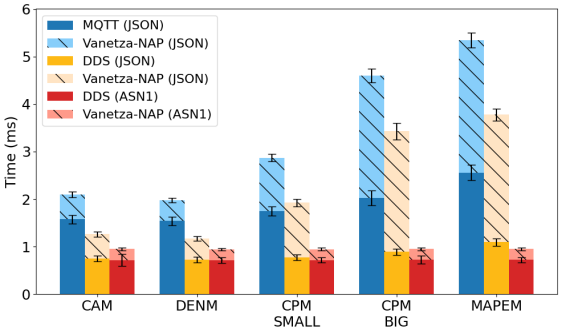

Fig. 6: Communication protocol and data format for Generation

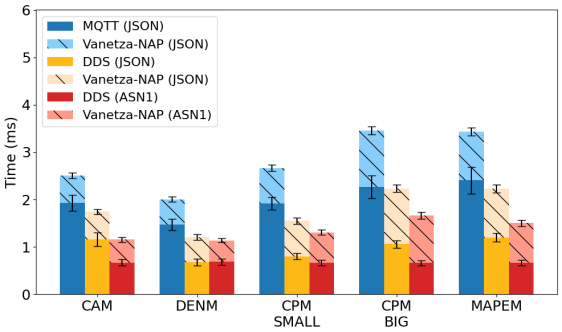

Fig. 7: Communication protocol and data format for Reception

[2]. Despite this, as mentioned in section V, general purpose applications will still benefit from the use of the MQTT protocol and JSON format, which are able to greatly simplify their design, while maintaining acceptable performance levels.

B. Parallel vs Sequential processing pipeline

In the second scenario, the producer app purposefully sends a sequence of messages, where a MAPEM is immediately followed by a DENM. This forces the occurrence of a probable edge case where a processing-intensive lower-priority message may delay a more critical message that arrives shortly after. The test is performed with both parallelised and sequential versions of the Vanetza-NAP pipeline, with the message being sent in a JSON format through DDS. The Vanetza-NAP processing delay is split into two measurements: the message queue wait time, which is calculated from step ❷ to step ❸; and the message processing time, which is calculated from step ❸ to step ❹. The results are showcased in figure 8.

A similar test is performed for the reverse flow, where a packet capture containing a MAPEM packet immediately followed by a DENM packet is synthetically injected onto the network interface at the same rate of 10 Hz. Similarly to the previous test, two measurements are performed: the message queue wait time, which is calculated from step ❺ to step ❻; and the message processing time, which is calculated from step ❻ to step ❼. The results are showcased in figure 9.

In both the generation and the reception processes, there is a significant increase in the wait time for a DENM message when parallelism is not employed, since its processing is delayed while the service finishes processing the MAPEM.

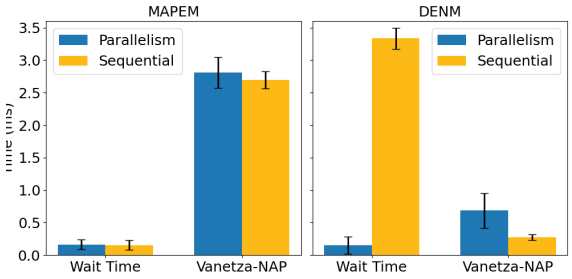

Fig. 8: Parallelism & Sequential results for Generation

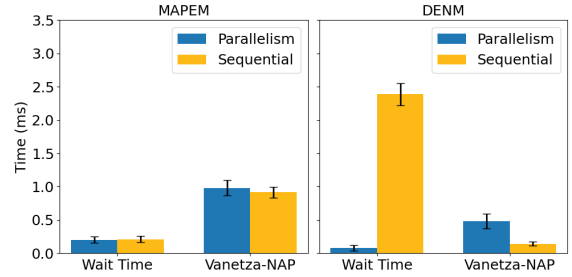

Fig. 9: Parallelism & Sequential results for Reception

This reinforces the importance of the parallelised pipeline architecture, so that higher priority messages can experience greater determinism in terms of overall processing delay.

From these results, we conclude that the base wait time is around 150 μs which, after further testing, we are able to attribute to Linux CPU scheduling overhead when waking up suspended worker threads upon the reception of messages (in this particular hardware configuration). Another interesting finding is that the DENM processing delay, excluding queue wait time, is improved when it is done sequentially, which is counter-intuitive. However, in a scenario where the same test is run without sending the MAPEM messages, this "sequential" DENM time immediately rose to the same value as the parallelised results. The fact that the thread processed a MAPEM immediately before the DENM, makes the latter's processing faster. This may happen due to the occurrence of cache and/or branch misses after the thread has been waiting for 95+ milliseconds, which does not happen in the sequential test case since the processing thread does not suspend between processing the MAPEM and DENM messages.

C. Comparison with monolithic application (Base Vanetza)

We now compare the Vanetza-NAP with the original *socktap* example application included in the base Vanetza library, which transmits CAMs at a rate of 10 Hz. This application represents a rudimentary example of a monolithic ITS application, which is fully self-contained. Slight modifications were made to the application to register timestamps at the start of message creation, and end of message transmission/reception.

Based on the results presented in figure 10, the Vanetza-NAP implementation exhibits slightly higher mean latency values for both message generation and reception, compared to Vanetza's *socktap* example. However, the total delay reported for Vanetza-NAP also includes the communication time

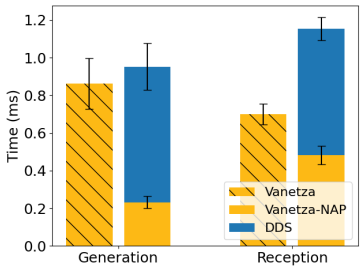

Fig. 10: Comparison between base Vanetza Library and Vanetza-NAP

required for transmitting or receiving information to interact with external services. This additional layer, which involves DDS coupled with the ASN.1 UPER format, contributes to the slight increase in the total time observed in the Vanetza-NAP. In summary, although our microservice-based proposal may exhibit some unavoidable overhead compared to the typical monolithic approaches, this effect is mitigated to a certain degree through efficient architectural and design decisions, choosing the right technologies for each use case, and other optimizations described in this work. On balance, we feel that the benefits of enhanced interoperability and extended capabilities justify this slight delay increase, making Vanetza-NAP a valuable addition for ITS scenarios and deployments.

VII. Conclusions and Future Work

The Vanetza-NAP ITS protocol stack embraces microservice-based architectural principles and represents a paradigm shift in the way that certain VEC use cases can be designed, and the efficiency with which they can be deployed and orchestrated within MEC clusters at scale. This allows for a much more accessible and observable ITS application ecosystem, while still keeping within timing restrictions and offering several mechanisms to minimise latency. Moreover, although the proposed approach has been tested with ITS-G5, it is available for other technologies, such as 5G and WiFi.

This service is currently in use as an effective educational tool in an MSc course, and is also deployed in the production environment of the Aveiro Tech City Living Lab (ATCLL) [9], where it delivers C-ITS to the public transportation of the city, enabling several cooperative awareness, notification, and perception applications where the city-wide roadside infrastructure exchanges information from radars and cameras with vehicles, through WiFi, ITS-G5 and 5G. The project has also been used in other V2X research works [19], [20], [21], including integrations with autonomous vehicles [22], [23].

As future work, we aim to: 1) expand the ITS message support, adding new types of messages and extending the current support to updated versions of their specifications, that have since been released; 2) explore alternative packet queuing & scheduling algorithms, and fine-tune different QoS policies for DDS topics according to the message type. Linking both of these prioritization mechanisms would enable a more consistent QoS experience over the entire message lifecycle.

Acknowledgment

This work is supported by the European Union/Next Generation EU, through Programa de Recuperação e Resiliência (PRR) Project Nr. 29: Route 25 (02/C05-i01.01/2022.PC645463824-00000063).

References

[1] R. Riebl, C. Obermaier, S. Neumeier *et al.*, "Vanetza: Boosting research on inter-vehicle communication," *Proceedings of the 5th GI/ITG KuVS Fachgespräch Inter-Vehicle Communication (FG-IVC 2017)*, 2017.

[2] R. Rosmaninho, D. Raposo, P. Rito *et al.*, "Time constraints on vehicular edge computing: A performance analysis," in *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, 2023.

[3] M. Klapez, C. A. Grazia, and M. Casoni, "Minimization of ieee 802.11p packet collision interference through transmission time shifting," *Journal of Sensor and Actuator Networks*, vol. 9, no. 2, 2020.

[4] A. Abunei, C. Comsa, and I. Bogdan, "Implementation of etsi its-g5 based inter-vehicle communication embedded system," 07 2017.

[5] A. Voronov, J. De Jongh, D. Heuven *et al.*, "Implementation of etsi its g5 geonetworking stack," *Java: CAM-DENM/ASN*, 2016.

[6] S. Laux, G. S. Pannu, S. Schneider *et al.*, "Demo: Openc2x — an open source experimental and prototyping platform supporting etsi its-g5," in *2016 IEEE Vehicular Networking Conference (VNC)*, 2016.

[7] F. Klingler, G. Pannu, C. Sommer *et al.*, "Poster: Field testing vehicular networks using openc2x," in *MobiSys 2027*, 06 2017.

[8] V. Dolk, J. d. Ouden, S. Steeghs *et al.*, "Cooperative automated driving for various traffic scenarios: Experimental validation in the gcdc 2016," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, 2018.

[9] P. Rito, A. Almeida, A. Figueiredo *et al.*, "Aveiro tech city living lab: A communication, sensing, and computing platform for city environments," *IEEE Internet of Things Journal*, vol. 10, no. 15, 2023.

[10] R. Parada, F. Vázquez-Gallego, R. Sedar *et al.*, "An inter-operable and multi-protocol v2x collision avoidance service based on edge computing," in *2022 IEEE 95th Vehicular Technology Conference: (VTC2022)*.

[11] Z. Kang, R. Canady, A. Dubey *et al.*, "A study of publish/subscribe middleware under different iot traffic conditions," in *Int. Workshop on Middleware and Applications for the Internet of Things*, 2021.

[12] B. Cheng, A. Rostami, and M. Gruteser, "Experience: Accurate simulation of dense scenarios with hundreds of vehicular transmitters," in *22nd Intern. Conference on Mobile Computing and Networking*, 2016.

[13] M. N. Tahir and M. Katz, "Performance evaluation of ieee 802.11p, lte and 5g in connected vehicles for cooperative awareness," *Engineering Reports*, vol. 4, no. 4, 2022.

[14] F. Pereira, A. S. Oliveira, N. Borges Carvalho *et al.*, "When Backscatter Communication Meets Vehicular Networks: Boosting Crosswalk Awareness," *IEEE Access*, vol. 8, 2020.

[15] M. Jutila, J. Scholliers, M. Valta *et al.*, "ITS-G5 performance improvement and evaluation for vulnerable road user safety services," *IET Intelligent Transport Systems*, vol. 11, no. 3, 2017.

[16] A. Figueiredo, P. Rito *et al.*, "Mobility sensing and v2x communication for emergency services," *Springer Mobile Networks and Applications*, Nov 2022.

[17] ETSI, "Technical specification 102 894–1 v1.1.1 (2013–08)," *European Telecommunications Standards Institute*, 2013.

[18] ——, "Technical specification 102 636-4-2 v1.2.1 (2020-04)," *European Telecommunications Standards Institute*, 2020.

[19] P. Almeida, A. Figueiredo, P. Rito *et al.*, "On the real evaluation of a collective perception service," in *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, 2023.

[20] J. Oliveira, P. Teixeira, P. Rito *et al.*, "Microservices in edge and cloud computing for safety in intelligent transportation systems," in *NOMS 2024 IEEE/IFIP Network Operations and Management Symposium*.

[21] J. Perpétua, P. Rito, S. Sargento *et al.*, "Communication and sensing for two-wheelers' safety in an autonomous shuttle scenario," in *21st Medit. Communication and Computer Networking Conference*, 2023.

[22] J. Amaral, J. Viegas, B. Lemos *et al.*, "Autonomous shuttle integrated in a communication and sensing city infrastructure," in *IEEE International Conference on Mobility, Operations, Services and Technologies*, 2023.

[23] J. Ramos, A. Figueiredo, P. Almeida *et al.*, "Enhancing autonomous vehicles control: Distributed microservices with v2x integration and perception modules," in *IEEE International Conference on Mobility, Operations, Services and Technologies*, 2024.