Artificial Intelligence in Vehicular Wireless Networks: A Case Study Using ns-3

Matteo Drago
dragomat@dei.unipd.it
Dep. of Information Engineering
University of Padova
Padova, Italy

Tommaso Zugno
tommaso.zugno@huawei.com
Huawei Technologies
Munich Research Center
Munich, Germany

Marco Giordani
giordani@dei.unipd.it
Dep. of Information Engineering
University of Padova
Padova, Italy

Federico Mason
masonfed@dei.unipd.it
Dep. of Information Engineering
University of Padova
Padova, Italy

Mate Boban
mate.boban@huawei.com
Huawei Technologies
Munich Research Center
Munich, Germany

Michele Zorzi
zorzi@dei.unipd.it
Dep. of Information Engineering
University of Padova
Padova, Italy

ABSTRACT

Artificial intelligence (AI) techniques have emerged as a powerful approach to make wireless networks more efficient and adaptable. In this paper we present an ns-3 simulation framework, able to implement AI algorithms for the optimization of wireless networks. Our pipeline consists of: (i) a new geometry-based mobility-dependent channel model for V2X; (ii) all the layers of a 5G-NR-compliant protocol stack, based on the ns3-mmwave module; (iii) a new application to simulate V2X data transmission, and (iv) a new intelligent entity for the control of the network via AI. Thanks to its flexible and modular design, researchers can use this tool to implement, train, and evaluate their own algorithms in a realistic and controlled environment. We test the behavior of our framework in a Predictive Quality of Service (PQoS) scenario, where AI functionalities are implemented using Reinforcement Learning (RL), and demonstrate that it promotes better network optimization compared to baseline solutions that do not implement AI.

1 INTRODUCTION

Artificial intelligence (AI) will be a key component of future 6th generation (6G) wireless networks [9], as a means to achieve autonomous network optimization [16]. In particular, the co-design of communication systems and applications with AI in mind will allow 6G networks to learn, adapt, and support diverse services and requirements, without human intervention.

Among other areas, AI has been recognized as a promising technology in Vehicle-To-Everything (V2X) networks, to enable applications like traffic flow and congestion control, localization, platoon management, and autonomous driving [26]. For these systems to be truly autonomous, intelligent vehicles need to acquire, process, and eventually disseminate massive amounts of data generated by on-board sensors [32]. Notably, AI, in combination with machine learning (ML), can be designed to extract features from input data [10], and support complex V2X tasks like object detection and recognition [24], data compression [22], as well as tracking and trajectory prediction [3].

As far as AI/ML is concerned, the availability of data for training and optimization is essential. In this regard, experiments with real testbeds are impractical due to limitations in the scalability and flexibility of these platforms, as well as the high cost of hardware components. On the other hand, AI-based research should follow an iterative approach where modeling and validation will be cyclically performed. Therefore, computer simulations have emerged as important tools for testing the performance of AI solutions in different conditions and scenarios. Python-based simulators are among the most popular softwares for AI, thanks to desirable features like versatility and readability [21]. Most importantly, many open-source libraries providing base-level ready-to-use coding solutions to develop AI functionalities, like TensorFlow, PyTorch or Keras, are implemented in Python. When it comes to the simulation of wireless networks, however, Python simulators tend to rely on simplified assumptions on the system architecture, and have not proven particularly successful in modeling the protocol stack of complex networks. Instead, discrete-event network simulators, like ns-3 [12], are valid alternatives to analyze the performance of wireless networks in more realistic scenarios. How to integrate the two simulators and build an open software able to support end-to-end
full-stack simulations, as well as network optimization via AI, is still an open challenge. An initial effort in this direction was made in [7], even though the authors focused on OpenAI Gym, a toolkit specific for Reinforcement Learning (RL) research.

In this paper, we fill this gap and propose a simulation pipeline to design and test AI in wireless networks. In particular, we focus on a V2X scenario, in which teleoperated vehicles use a cellular connection to exchange sensor data with a remote driver. Specifically, we extended the ns-3 mmwave module [19], one of the most 5G-oriented frameworks to simulate wireless networks, as follows:

- We implemented a geometry-based channel model for V2X, based on GEMV$^2$ and Simulation of Urban MOBility (SUMO) traces, that is consistent with the actual deployment of buildings and vehicles in the scenario.
- We designed a new ns-3 application to simulate the traffic flow in V2X use cases. Specifically, the ns-3 application involves the exchange of sensor data, which may be pre-emptively compressed and/or segmented to reduce the file size before transmission, modeled based on the Kitti multi-modal dataset [8]. The application is characterized by (i) the size of the input data, and (ii) the time periodicity at which the information is generated and exchanged, and (iii) the level of compression/segmentation of the data.
- We introduced a new entity called “RAN-AI” (the core contribution of this paper) that, connected to the Radio Access Network (RAN) and with Python-based AI algorithms, optimizes network operations.

As a case study, we validate our AI pipeline for a Predictive Quality of Service (PQoS) application [6], defined as a mechanism to predict Quality of Service (QoS) changes and provide autonomous vehicles with advance notifications to react accordingly. In this scenario, the RAN-AI collects full-stack network metrics from different components of the RAN, and implements an RL framework (first introduced in [17]) that is able to identify the optimal network configuration. We demonstrate that our RAN-AI is able to improve the QoS of V2X applications, compared to other baseline solutions that do not implement AI techniques.

The rest of the paper is structured as follows. In Sec. 2 we discuss how we integrated AI operations in ns-3. In Sec. 3 we validate our ns-3 implementation for PQoS. In Sec. 4 we conclude the paper with suggestions for future research.

2 DESIGN OF INTELLIGENT WIRELESS NETWORKS USING NS-3

Figure 1 provides an overview of the ns-3 simulation framework that we developed to integrate AI functionalities in vehicular networks.

We distinguish four main components, namely (i) the channel and mobility models (Sec. 2.1), (ii) the network model (Sec. 2.2), which simulates the communication network, (iii) the application model (Sec. 2.3), which mimics a real vehicular application, and (iv) the intelligent network controller (Sec. 2.4), which provides AI functionalities to optimize the network configuration. The source code is publicly available at [25].

2.1 Channel Model

A realistic characterization of the wireless channel is of paramount importance to obtain accurate simulation results [14]. The channel model should incorporate the impact of the radio environment on the propagation of wireless signals (e.g., the presence of buildings and other blockers) and user mobility.

The approach adopted in our framework relies on OpenStreetMap (OSM) to obtain a detailed representation of the area of interest. The generation of vehicles’ mobility traces is handled by SUMO [13], a popular open-source tool for the simulation of vehicular traffic. In particular, the OSM representation of the scenario is converted into a SUMO network file using the netconvert utility, and the generation of random routes is carried out using the script randomTrips.py. Finally, the wireless channel is computed using GEMV$^2$, a geometry-based propagation model for V2X scenarios [5], which is open source and publicly available [18]. GEMV$^2$ calculates both geometry-based large- and small-scale fading components of the channel from a map of the environment and the trajectories of the vehicles, taking into account the effect of mobility, buildings outlines and foliage, and outputs the propagation loss for all the possible device pairs at each time step.

To feed the channel traces into ns-3, we created a parser able to read the GEMV$^2$ output files, and eventually compute the received power between a pair of devices based on the current value of the transmit power. The parser is implemented by the class GemvPropagationLossModel, which exploits the modular design of the ns-3 propagation module by extending the PropagationLossModel interface. The new class provides the method DoCalcRxPower(), which reads the traces and retrieves the current simulation time to determine the received power for the desired link(s). For an efficient processing of the Comma Separated Values (CSV) files, we used the CsvReader class available in the ns-3 core module.

2.2 Network Model

For an accurate characterization of the network components, we chose to extend the ns3-mmwave module, an open source an publicly available ns-3 module for the simulation of 5G networks [19].

The ns3-mmwave module implements models for all the layers of the 5G NR protocol stack, for both Next Generation Node Bases (gNBs) and User Equipments (UEs). The custom Physical (PHY) and Medium Access Control (MAC) layers support different NR-compliant frame structures and numerologies, multiple beamforming algorithms and scheduling policies. The Radio Link Control (RLC) and Packet Data Convergence Protocol (PDCP) layers, as well as the core network models, are based on the ns-3 lte module for Long Term Evolution (LTE) networks [23]. It also supports dual connectivity with LTE base stations, which enables the simulation of non-standalone 5G deployments, and Carrier Aggregation (CA) at the MAC layer. Based on these features, ns3-mmwave supports end-to-end full-stack network simulations with a high level of detail, and has been taken as a reference benchmark simulator for 5G scenarios.

Although this module is meant for the simulation of 5G systems operating at millimeter wave (mmWave) frequencies, we introduced some modifications to make it work at lower frequencies too. In particular, we modified the class MmWaveHelper to accept
the GEMV$^2$-based propagation model presented in Sec. 2.1, and create gNBs and UEs without beamforming capabilities (via the InstallSub6GnbDevice() and InstallSub6UeDevice() methods, respectively).

2.3 Application Model

We focused on a teleoperated driving scenario where a Host Vehicle (HV) is controlled by a remote driver through an ad hoc driving application installed on a remote or edge server. In order to be teleoperated, the HV must be able to disseminate perception data generated from on-board sensors like Light Detection and Rangings (LiDARs) to the remote driver, that will then detect/recognize sensitive entities in the environment (e.g., cars, pedestrians, cyclists, etc.). However, the transmission of sensor data requires a considerable amount of radio resources and could potentially congest the network, preventing a smooth driving experience. To tackle this issue, data compression and segmentation are often applied in order to reduce the size of raw data prior to transmission.

In ns-3, we designed a new application module to simulate this data exchange and generate sensor data as a function of:

1. The size of the input sensor data (in bytes);
2. The time periodicity at which the information is generated and exchanged (typically fixed to 100 ms for LiDAR sensors, according to the device’s data sheets [29]);
3. The level of compression/segmentation for the sensor data.

In this work we consider the sensor data from the Kitti multimodal dataset, collected using a Volkswagen Passat equipped with a Velodyne LiDAR [8]. In particular, we rely on the data compression pipeline proposed in [28]. First, we infer semantic segmentation of point clouds with RangeNet++ [20]. We consider 3 segmentation levels:

- Raw (R): The raw LiDAR acquisition is considered.
- Segmentation Conservative (SC): Data points associated to road elements are removed, thus reducing the file size.
- Segmentation Aggressive (SA): Data points associated to buildings, vegetation, traffic signs, and the background are also removed, thus keeping only the most critical items in the scene (typically pedestrians and vehicles).

Second, we compress the resulting frame using Draco [11], a software whose flexibility allows the support of 15 quantization levels and 11 compression levels. Since there are 3 segmentation levels, 15 quantization levels and 11 compression ratios, overall there would be 495 distinct alternatives; for simplicity, our application implements the 7 most representative ones, in terms of the trade-off between compression accuracy and speed, referred to as “application modes” in the rest of the paper.

To implement the features described above, we started from the application presented in [15], able to generate large data frames and automatically fragment them into bursts of packets that are then re-aggregated at the receiver, whenever possible. Specifically, we extended the TraceFileBurstGenerator, that allows the user to reproduce real world traffic traces, into the KittiTraceBurstGenerator. Using the CsvReader utility, already available in ns-3, we created a method to import sensor traces from the Kitti dataset (after applying compression and segmentation to the data where applicable), and save each frame information into a data structure.

As in a specific time instant of the simulation the application can operate in one specific application mode, we provided KittiTraceBurstGenerator with ad hoc methods to change the mode in real time. In addition, considering that a single traffic trace could incorporate and/or last for multiple scenes, the user can choose what sequence the application is replicating and decide, in case the reader reaches the end of the scene of interest, whether to loop again from the beginning of the same scene or stop the application.

Considering that a LiDAR collects 3D points periodically, we also emulate this behavior by the design of the FramePeriod attribute, that indicates the time interval between a frame and the following one, and is used to subsequently schedule the sending of each burst. The BurstyApplication and BurstSink classes will then take
care of burst fragmentation, transmission and reception of packets, and re-aggregation of the burst.

For easy collection of statistics at the application, the module was integrated with an additional BurstyAppStatsCalculator utility class, which creates an output file to report how many bytes and bursts were received in a specific window of time, and the delay the received bursts have accumulated, on average, for each node.

2.4 Intelligent Network Controller

As highlighted in Sec. 1, the main goal of this work is to develop a framework to integrate AI functionalities in ns-3. In this section, we present a new entity called RAN-AI, installed at the gNB, which interfaces with different components of the RAN, as well as the core network, and incorporates AI capabilities with the purpose of optimizing V2X network operations. In particular, the RAN-AI collects network metrics and takes actions (also referred to as "countermeasures") to control the connected vehicles accordingly.

In our framework, the RAN-AI is responsible for:

1. Collecting metrics from the gNB (i.e., cell-related information) and the end users.
2. Running AI algorithms using as inputs the collected metrics. We highlight that our implementation is AI-agnostic, in the sense that it supports different AI models able to solve heterogeneous problems.
3. Determining the actions to take in order to maximize the network performance.
4. Communicating the actions to the relevant entities so that they can tune their behavior accordingly. In particular, in this work we allow the RAN-AI to control the end users, even though our framework does not prevent the collection of countermeasures to be considered.

With respect to the code structure, RAN-AI functionalities are implemented by the RanAI class, that is in charge of integrating the features of na3-ai [31], an ad hoc module to provide efficient and high-speed data exchange between Python-based AI algorithms and ns-3. Specifically, this module uses a shared memory implementation for interprocess communications, and provides a high-level interface in both Python and C++ for different algorithms, which is the reason why we integrated it in our framework.

To enable this new entity, an instance of the RanAI class has to be installed on each gNB in the simulation environment. First, we included in MmWaveEnbNetDevice the attribute m_ranAI representing the RAN-AI instance, which is initialized by calling the class method InstallRanAI(). Specifically, InstallRanAI() (i) initializes instances that gather full-stack network statistics through the classes MmWaveBearerStatsCalculator and BurstyAppStatsCalculator, and (ii) schedules SendStatusUpdate(), a routine that is executed every m_statusUpdate seconds. In particular, SendStatusUpdate() allows:

1. The gNB to collect measurements at the PHY, MAC, RLC, PDCP, and application layers, pertaining to the end users.
2. The gNB to organize this information to be compatible with ReportMeasures() input requirements and, through this method, provide it to the RAN-AI.
3. The RAN-AI to process the dataset via the AI framework.

When it comes to delivering the agent’s decision (i.e., the optimal action) to an end user, the RAN-AI checks whether that user is already configured to operate as specified by the action. If not, the action must be communicated to the end user, and in our ns-3 framework we devise two possibilities:

- Ideal notification: We directly trigger a callback function to apply the agent’s decision. No packet is transmitted, thus we do not model the impact of the transmission delays and/or communication errors/failure when changing the action.
- Real notification: The notification involves the transmission of a real packet towards the intended end user. Specifically, each notification packet contains (i) the agent’s decision, (ii) the International Mobile Subscriber Identity (IMSI) of the user, and (iii) the Radio Network Temporary Identifier (RNTI) of the user in the cell. As such, both transmission delays and communication errors are involved in the process (i.e., the packet could be lost, leading the end user to operate sub-optimally).

With respect to the AI framework running on top of the RAN-AI, we deployed multiple Markov Decision Processes (MDPs), each of which represents the behavior of a distinct end user. Specifically, we developed a novel Python module, named CentralizedAgent, which allows an agent to interact with multiple learning environments, and to test multiple learning configurations, thereby evaluating the performance of different AI algorithms applied to the same scenario. The CentralizedAgent module implements two main methods, namely get_action() and update().

- get_action() takes as an input the states of the $N_u$ end users in the target scenario, and computes $N_u$ actions. Notably, the state of an end user is a vector that includes all the input parameters that the RAN-AI entity can collect for the given vehicle. Depending on how state and action spaces are explored, the method will return different actions.
- update() takes as an input a list of $N_u$ learning transitions, i.e., sequences $[s_t, a_t, s_{t+1}, r_t]$ associated with each of the $N_u$ end users in the target scenario during slot $t$. Specifically, a learning transition consists of (i) the state $s_t$ of the end user at the beginning of time slot $t$, (ii) the action $a_t$ performed by the end user during slot $t$, (iii) the state $s_{t+1}$ at the beginning of the next slot $t+1$ and, (iv) the reward $r_t$ that the end users receives at the end of slot $t$. In general, the quality of the agent’s decisions increases as more data are gathered by the update() method.

The reward function is specific to the target application. As a case study, in this paper we will validate our AI framework for a PQoS application, as described in Sec. 3.1. Nevertheless, our RAN-AI implementation is transparent to the ns-3 scenario, the only dependence being how the input data is structured.

It should be mentioned that training an AI algorithm may require a huge computational effort. In our case, this challenge is exacerbated by the fact that the algorithm’s input is directly taken from the ns-3 simulation running under the hood, to provide information on how the system evolves and reacts to the decisions of the agent. At every step, in fact, the agent needs to wait for ns-3 to  

1We recall that our framework is based on the na3-mmwave module, that we extended to support communication at both sub-6 GHz and mmWave frequencies.
compute and collect a set of metrics, that are then used for training. In this sense, the proposed framework supports transfer learning, where initial learning can be performed in a simplified (and faster) environment, with further training in a more realistic environment (in our case, in ns-3).

3 A CASE STUDY

In this section, we validate our AI simulation pipeline for a PQoS use case, and demonstrate that it represents a valid tool to evaluate and optimize wireless networks. In Sec. 3.1 we present our AI learning setup, in Sec. 3.2 we describe our simulation parameters, and in Sec. 3.3 we show initial numerical results.

3.1 AI Algorithm

Overview. As a case study, we focus on PQoS, a paradigm to provide autonomous systems with advanced notifications in case of upcoming QoS changes [6]. To address this problem, the RAN-AI introduced in Sec. 2.4 collects network statistics at the RAN level and, based on them, defines and applies network countermeasures in case QoS requirements are not satisfied. Specifically, we designed the RAN-AI to implement an AI agent, based on RL, able to identify the optimal application mode (see Sec. 2.3) for the end users when transmitting sensor data. The rationale behind this choice is that end users will be encouraged to select more aggressive application modes (e.g., those that apply compression/segmentation) to reduce the size of the packets to send and promote faster transmissions.

Learning agent. Based on our previous work [17], our AI/RL agent is trained according to the Double Q-learning (DQL) algorithm described in [27], which is an extended version of the classical Q-learning [30]. Hence, whenever the update() function is called, CentralizedAgent follows the DQL procedure to perform a new training step. Our framework approximates the agent’s policy by means of a Neural Network (NN), which makes it possible to handle continuous state spaces and overcome the curse of dimensionality phenomenon [4]. We consider a Feed-Forward NN, with S inputs and A output neurons, and implement the Rectified Linear Unit (ReLU) activation function across the different layers [2].

We highlight that the input size of the NN coincides with the dimension of the system’s state, i.e., the number of input parameters of the RAN-AI entity. Instead, the output size of the NN corresponds to the number of possible actions for the agent, i.e., in our case the different application modes.

Reward function. In our target scenario, the performance of the system depends on two different aspects:

- The Quality of Service (QoS): The vehicles should satisfy QoS requirements, especially in terms of maximum end-to-end delay $\delta_M$ and minimum Packet Reception Ratio (PRR) $\text{PRR}_{\text{min}}$.
- The Quality of Experience (QoE): The transmitted data should be accurate enough to perform driving operations. For our case study, the Quality of Experience (QoE) depends on the symmetric point-to-point Chamfer Distance $\text{CD}_{\text{sym}}$, which is inversely proportional to the quality of the received data [28].

To incorporate both these factors, the agent reward is designed to balance between QoS and QoE via a tuning parameter $\alpha \in [0, 1]$.

Let $\text{PRR}_t$ and $\hat{\delta}_t$ be the PRR and average delay of the vehicle at time $t$, respectively. If the QoS requirements are not met, i.e., $\delta_t \geq \delta_M$ and $\text{PRR}_t < \text{PRR}_{\text{min}}$, the agent reward $R(t)$ is 0, otherwise it is given by

$$R(t) = (1 - \alpha) \frac{\delta_M - \hat{\delta}_t}{\delta_M} + \alpha \frac{\text{CD}_{\text{sym},m} - \text{CD}_{\text{sym},t}}{\text{CD}_{\text{sym},m}},$$

where $\text{CD}_{\text{sym},t}$ and $\text{CD}_{\text{sym},m}$ are the Chamfer Distance at time $t$ and the maximum Chamfer Distance that can be tolerated, respectively.

3.2 Simulation Parameters

The simulation parameters are shown in Table 1.

| Parameter | Description | Value |
|-----------|-------------|-------|
| $f_c$     | Carrier frequency | 3.5 GHz |
| $B$       | Total bandwidth  | 50 MHz |
| $P_{TX}$  | Transmission power | 23 dBm |
| $T$       | RAN-AI update periodicity | 100 ms |
| $\tau_s$  | Simulation time  | 80 s |
| $N_v$     | Number of vehicles | [1, 5] |
| $\lambda$ | Discount factor  | 0.95 |
| $\epsilon$ | Learning rate | $10^{-4}$ |
| $c$       | Weight decay  | $10^{-3}$ |
| $\alpha$  | QoS/QoE weight | 1 |
| $\text{PRR}_{\text{min}}$ | Max. tolerated delay | 50 ms |
| $\text{PRR}_{\text{max}}$ | Min. tolerated PRR | 1 |
| $\text{CD}_{\text{sym},m}$ | Max. tolerated Chamfer Distance | 45 |
| Layer size (inputs $\times$ outputs) | $8 \times 12 \rightarrow 12 \times 6 \rightarrow 6 \times 3$ |

Table 1: Simulation parameters.
Figure 2: Our simulation scenario, corresponding to a portion of the city of Bologna (Italy). The red star represents the position of the gNB.

Table 2: Average QoE ∈ [0, 1] performance of different PQoS policies, as a function of the RAN-AI implementation for notifications (real or ideal) and $N_u$.

| PQoS policy | RAN-AI notification ($N_u = 1$) | Number of vehicles $N_u$ (Real Notifications) |
|-------------|---------------------------------|---------------------------------------------|
|             | Real                            | Ideal                                       |
| C-R         | 1                               | 1                                           |
| C-SC        | 0.88                            | 0.88                                        |
| C-SA        | 0.22                            | 0.22                                        |
| DQL         | 0.98                            | 0.98                                        |

based on 5GAA specifications [1], so we have $\delta_M = 50$ ms and PRR$_m = 1$. Finally, $\text{CD}_{sym, m}$ is set to 45, while $\text{CD}_{sym, t}$ depends on the application mode and increases as considering more aggressive compression/segmentation. Simulation results as a function of $\alpha$ can be found in [17], where we proved that decreasing $\alpha$ has the benefit to further improve the QoS, even beyond the QoS requirements under consideration, at the expense of some QoE degradation.

3.3 Results

To validate our framework, we compared the following policies:
- **DQL** (proposed), where at each step the agent implements our proposed RAN-AI framework.
- **Constant** (benchmark), where at the beginning of the simulation the end user maintains one application mode $\in \mathcal{A}$ for the whole simulation.

The two policies have been tested separately, and will be compared in terms of the user’s (i) QoS, expressed in terms of delay and PRR at the application layer, and (ii) QoE, which depends on the Chamfer Distance. We investigate the impact of the RAN-AI implementation for delivering the agent’s decisions to the end users (real or ideal) and the number of vehicles $N_u$.

In Table 2 we report the QoE for different PQoS policies. We observe that all the baseline solutions show constant values of the QoE, regardless of the RAN-AI implementation and $N_u$, as the compression level remains always fixed, while DQL shows different QoE performance depending on the adopted policy. The best QoE is achieved when transmitting raw sensor data (C-R), given that data segmentation privileges efficiency over accuracy and eventually distorts the LiDAR data before transmission. Moreover, from Table 2 we can see that introducing ideal notifications leads to QoE degradation, on average from 0.98 to 0.94. Although this behavior may sound counterintuitive, we found that, when using ideal notifications, DQL was encouraged to adopt a more aggressive behavior, i.e., the agent was encouraged to change the application mode more frequently throughout the learning process. This has the benefit to improve the QoS (as we will show in Fig. 4), at the cost of a lower QoE. This is an indication that QoS and QoE should be studied together to really understand the performance of the system, as we will discuss in the following results.

In Figures 3a and 3b we illustrate the distributions of the delay and PRR, respectively, experienced at the application layer vs. $N_u$. Precisely, Figure 3b represents a split violin plot, which depicts the estimated probability density function of the PRR for different values of $N_u$, for each application mode.

Figure 3: Performance of different PQoS policies, considering real notifications at the RAN-AI and the impact of the number of users.

First, as expected, the median and the percentiles of the delay (PRR) are always higher (lower) when $N_u = 5$, due to the fact that increasing the number of vehicles may congest the communication
channel, thus degrading the overall system performance. This result validates the accuracy and realism of our ns-3 framework.

In terms of QoS, C-SA outperforms any other solution, since in this configuration the data are extremely compressed and segmented before transmission, thus reducing the size of the packets to send, at the expense of a very low QoE (0.22). In turn, C-R maximizes the QoE, but results in a QoS degradation (up to 2.3× higher latency compared to C-SA). It appears clear that, in our DQL implementation, the RAN-AI tries to adapt the compression level to the conditions of the scenario via AI/RL, and achieves the best trade-off between QoE and QoS. In particular, with $N_u = 1$, DQL is able to guarantee an average QoE of 0.98, while ensuring an average end-to-end delay lower than the maximum tolerated value for teleoperated applications (i.e., 50 ms). Furthermore, DQL reduces the variability of both the delay and the PRR compared to its competitors, a critical requirement in V2X.

In Figure 4 we demonstrate that the impact of the additional overhead introduced when control notifications are transmitted from the RAN-AI to the end users is not negligible. Specifically, with ideal notification settings, both the delay and the PRR improve: the median delay and percentiles take lower values, while the probability of packet loss decreases, given that the transmission of real notification packets may incur additional delays and communication errors.

4 CONCLUSIONS

In recent years, V2X networks are incorporating AI as a method to analyze large volumes of data and self-optimize. While simulators like ns-3 have been popular tools to analyze the performance of wireless networks, how to simulate AI techniques and their impact on the communication stack is still an open question. In this paper we address this challenge, and propose a novel framework in ns-3 able to simulate AI algorithms. To this aim, we implemented a new geometry-based channel model and application for V2X, and a new intelligent entity (called RAN-AI) for optimizing wireless networks. We demonstrate the accuracy and technical soundness of our framework in a test scenario where the network performance is controlled via PQoS. We show from ns-3 simulations that V2X performance requirements in terms of QoS and QoE can be satisfied when the RAN-AI implements an RL algorithm for optimization. We provide the source code of the simulator at [25], in the hope that it will be useful to the broader community when implementing and evaluating AI techniques to improve wireless networks performance.

REFERENCES

[1] 5GAA. 2020. C-V2X Use Cases Volume II: Examples and Service Level Requirements. White Paper (Oct. 2020).
[2] Abien Fred Agarap. 2018. Deep learning using rectified linear units (ReLus). arXiv preprint arXiv:1803.08375 (2018).
[3] Minjin Baek, Donggi Jeong, Dongho Choi, and Sangsun Lee. 2020. Vehicle trajectory prediction and collision warning via fusion of multisensors and wireless vehicular communications. Sensors 20, 1 (Jan. 2020), pp. 288.
[4] Richard Bellman. 1966. Dynamic programming. Science 153, 3731 (Jul. 1966), pp. 34–37.
[5] M. Boban, J. Barros, and O. K. Tonguz. 2014. Geometry-Based Vehicle-to-Vehicle Channel Modeling for Large-Scale Simulation. IEEE Transactions on Vehicular Technology 63, 9 (November 2014).
[6] Mate Boban, Marco Giordani, and Michele Zorzi. 2021. Predictive Quality of Service: The Next Frontier for Fully Autonomous Systems. IEEE Network 35, 6 (November/December 2021), pp. 104–110.
[7] Piotr Gawlowicz and Anatolij Zubow. 2019. Ns-3 Meets OpenAI Gym: The Playground for Machine Learning in Networking Research (MSWM ’19). Association for Computing Machinery, New York, NY, USA, pp. 113–120.
[8] A. Geiger, P. Lenz, and R. Urtasun. 2012. Are we ready for autonomous driving? The KITTI vision benchmark suite. In IEEE Conference on Computer Vision and Pattern Recognition.
[9] Marco Giordani, Michele Polese, Marco Mezzavilla, Sundreep Rangan, and Michele Zorzi. 2020. Toward 6G networks: Use cases and technologies. IEEE Communications Magazine 58, 3 (Mar. 2020), pp. 55–61.
[10] Marco Giordani, Andrea Zanella, Takamasa Higuchi, Omar Altintas, and Michele Zorzi. 2019. Investigating Value of Information in Future Vehicular Communications. In IEEE 2nd Connected and Automated Vehicles Symposium (CAVS).
[11] Google. 2017. Draco 3D Data Compression. (2017). https://github.com/google/draco
[12] Thomas R Henderson, Matharu Lacaige, George F Riley, Craig Dowell, and Joseph Kopena. 2008. Network Simulations with the ns-3 Simulator, SIGCOMM demonstration 14, 14 (2008), pp. 527.
[13] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker. 2012. Recent Development and Applications of SUMO – Simulation of Urban Mobility. International Journal On Advances in Systems and Measurements 5, 3&4 (Dec. 2012), pp. 128–138.
[14] Mattia Lecchi, Paolo Testolina, Michele Polese, Marco Giordani, and Micheile Zorzi. 2021. Accuracy Versus Complexity for mmWave Ray-Tracing: A Full Stack
[15] Mattia Lecci, Andrea Zanella, and Michele Zorzi. 2021. An ns-3 Implementation of a Bursty Traffic Framework for Virtual Reality Sources. In ACM Workshop on NS3 (WNS3).

[16] Khaled B Letaief, Wei Chen, Yuanming Shi, Jun Zhang, and Ying-Jun Angela Zhang. 2019. The roadmap to 6G: AI empowered wireless networks. IEEE Communications Magazine 57, 8 (August 2019), pp. 84–90.

[17] Federico Mason, Matteo Drago, Tommaso Zagni, Marco Giordani, Mate Boban, and Michele Zorzi. 2022. A Reinforcement Learning Framework for PQoS in a Teleoperated Driving Scenario. IEEE Wireless Communications and Networking Conference Workshops (WCNC WKSHPS) (2022).

[18] Mate Boban. 2014. GEMV2: Geometry Based Efficient Propagation model for V2V communication. (2014). http://vehicle2x.net/

[19] M. Mezzavilla, M. Zhang, M. Polese, R. Ford, S. Dutta, S. Rangan, and M. Zorzi. 2018. End-to-End Simulation of 5G mmWave Networks. IEEE Communications Surveys and Tutorials 20, 3 (Apr. 2018), pp. 2237–2263.

[20] Andres Milioto, Ignacio Vizzo, Jens Behley, and Cyrill Stachniss. 2019. RangeNet++: Fast and accurate LiDAR semantic segmentation. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).

[21] Ablinav Nagpal and Goldie Gabrani. 2019. Python for data analytics, scientific and technical applications. In Amity international conference on artificial intelligence (AIAC). IEEE.

[22] F. Nardo, D. Peressoni, P. Testolina, M. Giordani, and A. Zanella. 2022. Point Cloud Compression for Autonomous Driving: A Performance Comparison. In IEEE Wireless Communications and Networking Conference (WCNC).

[23] Giuseppe Pusino, Nicola Baldo, and Marco Miozzo. 2011. An LTE module for the ns-3 network simulator. In 4th International ICST Conference on Simulation Tools and Techniques.

[24] Valentina Rosi, Paolo Testolina, Marco Giordani, and Michele Zorzi. 2021. On the Role of Sensor Fusion for Object Detection in Future Vehicular Networks. In Joint European Conference on Networks and Communications 6G Summit (EuCNC/6GS).

[25] SGNET University of Padova. 2022. RAN-AI ns-3 source code. (2022). Retrieved February 17, 2022 from https://github.com/signetlabdei/ns3-ran-ai

[26] Wang Tong, Azhar Hussain, Wang Xi Bo, and Sabita Maharjan. 2019. Artificial intelligence for vehicle-to-everything: A survey. IEEE Access 7 (January 2019), pp. 10823–10843.

[27] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double Q-learning. In Proceedings of the AAAI conference on artificial intelligence, Vol. 30.

[28] Andrea Varischio, Francesco Mandrizzato, Marcello Bullo, Marco Giordani, Paolo Testolina, and Michele Zorzi. 2021. Hybrid Point Cloud Semantic Compression for Automotive Sensors: A Performance Evaluation. IEEE International Conference on Communications (ICC) (2021).

[29] VeloView. 2022. Velodyne LiDAR Dataset. (2022). Retrieved February 17, 2022 from https://data.kitware.com/#collection/5b7f46f98d77f706857cb206

[30] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. Machine learning 8, 3–4 (May 1992), pp. 279–292.

[31] Hao Yin, Pengyu Liu, Keshu Liu, Liu Cao, Lytianyang Zhang, Yayu Gao, and Xiaojun Hei. 2020. Ns3-Ai: Fostering Artificial Intelligence Algorithms for Networking Research. In Proceedings of the 2020 Workshop on Ns-3. pp. 57–64.

[32] Shan Zhang, Jiayin Chen, Feng Lyu, Nan Cheng, Weisen Shi, and Xuemin Shen. 2018. Vehicular communication networks in the automated driving era. IEEE Communications Magazine 56, 9 (Sept. 2018), pp. 26–32.