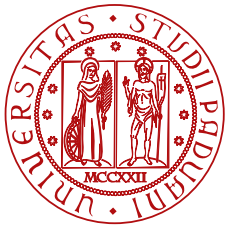


Università degli Studi di Padova

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Department of Information Engineering
Master Degree in ICT for Internet & Multimedia

Design, Implementation and Evaluation of Learning Algorithms for Predictive Quality of Service in Teleoperated Driving Scenarios

Master Candidate

Giacomo Avanzi

Student ID 2088615

Supervisor

Prof. Michele Zorzi

University of Padova

Co-supervisors

Prof. Marco Giordani, Dr. Filippo Bragato

University of Padova

ACADEMIC YEAR 2023/2024
Graduation date 03/09/2024

Ai miei genitori
A mio fratello

Abstract

In the Teleoperated Driving (TD) scenario strict constraints in Quality of Service (QoS) indicators, especially end-to-end (E2E) latency and reliability, of the communication between vehicles and remote drivers must be satisfied. Predictive Quality of Service (PQoS) is a tool to predict network degradation and react accordingly. In this context, Artificial Intelligence (AI) can be used to optimize PQoS operations. However, there are several trade-offs between centralized and decentralized Reinforcement Learning (RL) solutions, which call for additional work in this area. The first goal of this thesis is the introduction of realism in the learning phase with respect to data and metrics (from the 5G protocol stack) gathered by the intelligent agent at the Radio Access Network (RAN) (centralized case) or at the vehicle (decentralized case), including the modelling of communication and computational mechanisms to obtain those metrics in the ns-3 simulator. To further investigate the trade-off between performance and communication overhead, a federated learning framework is evaluated under different strategies of parameters aggregation based on the vehicles status, involving also compression (pruning, quantization and clustering) of the parameters in the agent's model to optimize the channel burden. The second objective is the implementation of a new meta-learning agent that can dynamically choose when to perform the centralized vs. decentralized learning models, depending on the network status. Starting from the global condition of the vehicles described in terms of network metrics, the optimal learning approach is chosen to maximize at the same time QoS and Quality of Experience (QoE). In a scenario with 3 vehicles, the centralized learning approach achieves the best compromise between QoS, with an average E2E delay of nearly 25 ms, and QoE, with an average mean average precision (mAP) of 0.68, considering the best realistic configuration. The distributed approach is able to reduce further the latency by 1 ms, at the cost of poorer mAP (0.67) and more violations of the maximum tolerated E2E delay. The federated approach increases the total delay by 3 ms due to the models sharing, while the QoE shows a significant improvement, exceeding 0.68 with mAP. The meta-learning agent achieves outstanding results, selecting autonomously the centralized approach in good channel conditions and the decentralized approach in a degraded network state.

Sommario

Nello scenario della Guida Teleoperata (GT), ci sono stringenti requisiti di Qualità del Servizio (QS), in particolare la latenza end-to-end (E2E) e l'affidabilità, della comunicazione tra i veicoli e i guidatori remoti. La Predizione della Qualità del Servizio (PQS) è uno strumento per prevedere il degrado della rete e reagire di conseguenza. L'Intelligenza Artificiale (IA) può essere utilizzata per ottimizzare le operazioni di PQS. Tuttavia, esistono diversi compromessi tra l'apprendimento per rinforzo con approccio centralizzato e decentralizzato, che richiedono un ulteriore lavoro in quest'area. Il primo obiettivo della tesi è l'introduzione di realismo nella fase di raccolta dei dati e delle metriche (dello stack protocollare del 5G) da parte dell'agente intelligente durante la fase di apprendimento, alla Radio Access Network (RAN) (caso centralizzato) o al veicolo (caso decentralizzato), compresa la modellazione dei meccanismi di comunicazione e di calcolo per ottenere tali metriche nel simulatore ns-3. Per studiare il compromesso tra prestazioni e costi di comunicazione, si valuta un approccio federato con diverse strategie di aggregazione dei parametri in base allo stato dei veicoli, coinvolgendo anche la compressione (pruning, quantizzazione e clustering) dei parametri del modello dell'agente per ottimizzare il carico sul canale. Il secondo obiettivo è l'implementazione di un nuovo agente di meta-apprendimento che può scegliere dinamicamente quando adottare un modello centralizzato vs. decentralizzato. Partendo dalla condizione dei veicoli, in termini di metriche di rete, il modello di apprendimento ottimale viene scelto per massimizzare allo stesso tempo la QS e la Qualità dell'Esperienza (QE). In uno scenario con 3 veicoli, l'approccio di apprendimento centralizzato raggiunge il miglior compromesso tra QS, con un ritardo medio E2E di quasi 25 ms, e QE, con una precisione media di 0.68, considerando la migliore configurazione realistica. L'approccio distribuito è in grado di ridurre ulteriormente la latenza di 1 ms, al costo di una minore precisione media (0.67) e di maggiori violazioni del ritardo massimo E2E tollerato. L'approccio federato aumenta il ritardo totale di 3 ms a causa della condivisione dei modelli, mentre la QE mostra un miglioramento significativo, con 0.68 di precisione media. L'agente di meta-apprendimento ottiene risultati ottimi, selezionando autonomamente la centralizzazione in buone condizioni di canale e la decentralizzazione in condizioni di rete degradate.

Contents

List of Figures **xi**

List of Tables **xv**

List of Algorithms **xvii**

List of Acronyms **xix**

1 Introduction **1**

1.1 Teleoperated Driving Scenario 1

1.2 Predictive Quality of Service 3

1.2.1 Reinforcement Learning for PQoS in V2X Networks 4

1.3 Thesis contribution and structure 5

2 Centralized and Distributed PQoS **9**

2.1 V2X communication: PQoS for TD scenario 10

2.2 5G NR System Architecture 12

2.2.1 CN 12

2.2.2 RAN 14

2.2.3 Protocol Architecture [14][15] 14

2.3 Reinforcement Learning 16

2.4 PQoS Framework 20

2.4.1 Simulation parameters 20

2.4.2 Ns-3 entities 22

2.5 Centralized Approach 22

2.5.1 RAN-AI 22

2.5.2 RL framework 24

2.5.3 Proposed Solutions and Implementation 26

2.5.4 Performance Evaluation 31
2.6 Distributed Approach 39
2.6.1 UE-AI 39
2.6.2 RL framework 40
2.6.3 Proposed Solutions and Implementation 41
2.6.4 Performance Evaluation 44

3 Federated PQoS **51**
3.1 Problem Formulation 52
3.2 Simulation Framework 53
3.3 Adaptive Federated Optimization Methods 54
3.3.1 Implementation and Results 56
3.4 Model compression 58
3.4.1 Pruning 59
3.4.2 Post-Training Quantization 63
3.4.3 Clustering 68
3.5 Utility Functions 71

4 Meta-learning agent for PQoS **77**
4.1 Centralized vs. Distributed vs. Federated Results 78
4.2 Meta-learning Agent Model 83
4.2.1 State 83
4.2.2 Action 84
4.2.3 Reward 84
4.3 Double Deep Q-Learning Algorithm 84
4.3.1 Implementation and Parameters 86
4.4 Performance Evaluation 87
4.4.1 Good Channel 88
4.4.2 Bad Channel 90

5 Conclusions and Future Works **93**

References **95**

Acknowledgments **101**

List of Figures

2.1 NR DL user-plane protocol stack [14] 13
2.2 RL framework [16] 17
2.3 Portion of the city of Bologna used in the simulations 19
2.4 OFDM symbols usage over the total number of slots inside a simulation with (a) 1 vehicle, (b) 5 vehicles and (c) 10 vehicles. 29
2.5 Results training of centralized RL agent with Proposed State 1 vs. baseline (1 vehicle) 34
2.6 Average E2E Delay from the training with Proposed State 1 vs. baseline (1 vehicle) 34
2.7 Results training of centralized RL agent with Proposed State 2 vs. baseline (1 vehicle) 36
2.8 Average E2E from the training with Proposed State 2 vs. baseline (1 vehicle) 36
2.9 Results training of centralized RL agent with Proposed State 1 vs. baseline (3 vehicles) 37
2.10 Average E2E Delay from the training with Proposed State 1 vs. baseline (3 vehicles) 38
2.11 Results training of distributed RL multi-agent with Proposed State 1 vs. baseline (3 vehicles) 46
2.12 Average E2E Delay from the distributed training with Proposed State 1 vs. baseline (3 vehicles) 47
2.13 Average E2E Delay from the distributed training with Proposed State 1 with 3 vehicles vs. 6 vehicles 48
2.14 Results training of distributed RL multi-agent with Proposed State 2 vs. baseline (3 vehicles) 49
2.15 Action probability of RL agent with Proposed State 2 vs. baseline (3 vehicles) 50

3.1 Results training of federated RL agent with FedAdam vs. FedAvg (baseline) 57
3.2 Average E2E Delay from the federated training with FedAdam vs. FedAdvg (baseline) 57
3.3 Average E2E Delay of the federated training with idealized vs. realistic exchange of federated updates 58
3.4 Results training of federated RL multi-agent with FedAdam and Pruning (0.5) vs. FedAdam (baseline) 61
3.5 Average E2E Delay from the federated training with FedAdam and Pruning (0.5) vs. FedAdam (baseline) 61
3.6 Results training of federated RL multi-agent with FedAdam and Pruning with sparsity ratio 0.5 vs. 0.7 62
3.7 Average E2E Delay from the federated training with FedAdam and Pruning with sparsity ratio 0.5 vs. 0.7 63
3.8 Results training of federated RL multi-agent with FedAdam and quantization vs. FedAdam (baseline) 66
3.9 Action probability of federated RL multi-agent with FedAdam and quantization vs. FedAdam (baseline) 67
3.10 Results training of federated RL multi-agent with FedAdam and clustering vs. FedAdam (baseline) 69
3.11 Action probability of federated RL multi-agent with FedAdam and clustering vs. FedAdam (baseline) 70
3.12 Action probability of federated RL multi-agent with FedAdam and clustering (4 clusters) 70
3.13 Results training of federated RL multi-agent with utility functions (3 vehicles) 73
3.14 Results training of federated RL multi-agent with utility functions (6 vehicles) 74
3.15 Average mAP of the federated RL multi-agent with utility functions (6 vehicles) 75

4.1 Average E2E Delay (QoS) achieved by the proposed configurations in centralized, distributed and federated approaches (3 vehicles) in the last 30 episodes 78

4.2 Average mAP (QoE) achieved by the proposed configurations in centralized, distributed and federated approaches (3 vehicles) in the last 30 episodes 78
4.3 Number of violations of the maximum tolerated E2E delay δ_M during the last 10000 learning steps in centralized, distributed and federated approaches (3 vehicles) 81
4.4 Top 1% values of the E2E delay during the last 10000 learning steps in centralized, distributed and federated approaches (3 vehicles) 82
4.5 Results training of meta-learning agent vs. static approaches (baseline) in good channel conditions 88
4.6 QoS and QoE indicators achieved by meta-learning agent vs. static approaches (baseline) in good channel conditions 88
4.7 Results training of meta-learning agent vs. static approaches (baseline) in bad channel conditions 90
4.8 QoS and QoE indicators achieved by meta-learning agent vs. static approaches (baseline) in bad channel conditions 90

List of Tables

2.1 Average mAP achieved by the object detector PointPillars on the SELMA dataset, depending on the compression mode 25

List of Algorithms

1 Server's federated update with adaptive optimizers 56
2 Federated update with quantization 65

List of Acronyms

3GPP Third Generation Partnership Project

5G Fifth Generation

5GAA 5G Automotive Association

5G NR Fifth Generation New Radio

6G Sixth Generation

ACK Acknowledgment

AI Artificial Intelligence

ARQ Automatic Repeat Request

BER Bit Error Rate

BLER BLock Error Rate

BSR Buffer Status Report

CN Core Network

CSI Channel State Information

CQI Channel Quality Indicator

DQL Deep Q-Learning

DDQL Double Deep Q-Learning

DL Downlink

E2E End-to-end

ER Experience Reply

FCNN Fully Connected Neural Network

gNB Next Generation Node B

IID Independent and Identically Distributed

IoT Internet of Things

ITS Intelligent Transportation System

KPI Key Performance Indicator

LGC Logical Channel Group

LiDAR Light Detection and Ranging

MAC Medium Access Control

mAP mean Average Precision

MDP Markov Decision Process

MCS Modulation and Coding Scheme

ML Machine Learning

NN Neural Network

ns-3 Network Simulator 3

OFDM Orthogonal Frequency Division Multiplexing

PDCP Packet Data Convergence Protocol

PDU Protocol Data Unit

PQoS Predictive Quality of Service

QoE Quality of Experience

QoS Quality of Service

RAN Radio Access Network

RL Reinforcement Learning

RLC Radio Link Control

RRC Radio Resource Control

SDU Service Data Unit

SE Spectral Efficiency

SGD Stochastic Gradient Descent

SINR Signal-to-Interference-plus-Noise Ratio

TB Transport Block

TBS Transport Block Size

TD Teleoperated Driving

TTI Transmission Time Interval

UE User Equipment

UL Uplink

URLLC Ultra-Realiable Low-Latency Communication

V2X Vehicle-To-Everything

1 Introduction

The advent of Sixth Generation (6G) Networks meets the needs of a society evolving towards a data-centric, data-dependent and automated model. Communication networks will need to transfer massive amounts of data at higher speeds, where the human communication will represent a minimal fraction of the traffic. The Internet of Things (IoT) scenario will be the main source of data, involving not just people devices, but sensors, e-health devices, smart cities and vehicular networks [1].

One of the most revolutionary changes of 6G is the transformation of traditional transportation networks into fully Intelligent Transportation Systems (ITSs). Autonomous and/or teleoperated vehicles with onboard sensors generating high data rates (terabytes per driving hour [2]) will be interconnected ensuring unprecedented levels of reliability and low latency (up to 99.99999% and less than 1 ms, respectively), even in ultra-high mobility scenarios (up to 1000 km/h), thus calling for Ultra-Reliable Low-Latency Communication (URLLC) [1].

Furthermore, the huge development in the areas of Artificial Intelligence (AI) and Machine Learning (ML) enables powerful techniques to intelligently optimize 6G networks, also in the context of teleoperated driving applications [3].

1.1 Teleoperated Driving Scenario

During the past years many advances have been made in the automotive field. However, pure autonomous driving with no human interaction presents

critical technical challenges.

Being able to recognize and interact with many different objects (vehicles, pedestrians, obstacles, traffic signs, etc.) and conditions (weather, traffic, visibility, etc) along public roads is complex for a self-driving car. ML can support the car in automated driving functions, e.g., for environment perception or decision-making, relying on a sufficient amount of training data from different scenarios and sensors. However, it is quite difficult to generalize beyond what the training set statistically represents and the unrealistic completeness of the training samples are hard obstacles to overcome in the fully autonomous driving scenario [4].

Therefore, the research community is focusing on Teleoperated Driving (TD), where a remote driver controls the car exploiting measurements received from the vehicle, generated by onboard sensors, such as high-resolution cameras, depth cameras, and Light Detection and Ranging (LiDAR) sensors.

Initially, the teleoperator will be a human, with the support of a cloud driving architecture, which provides a computing and networking environment for an intelligent teleoperation. As the driving automation level of the cloud system evolves, the tasks of the human operator will be progressively handed over to a cloud driving system software, which will assume complete control of the vehicle [4].

The performance of TD depends strongly on the network conditions where the autonomous vehicles are deployed, since the transmission of high-resolution sensors' data from the vehicle to the remote control center can be source of hundreds of megabits per second data rates. In particular, strict requirements must be satisfied in terms of Quality of Service (QoS):

- low end-to-end (E2E) latency, to not reduce the responsiveness of the driver;
- high reliability, to send accurate perceptions and receive accurate commands.

More precisely, according to the Fifth Generation (5G) specifications of 5G Automotive Association (5GAA), inside an Infrastructure-Based Tele-Operated Driving, the service level latency between vehicle and remote driver is fixed to 50 ms (both in uplink [UL] and downlink [DL]). The service level reliability is fixed to 99% from sensors to the remote host, for the reverse direction to 99.999%, because commands from the remote driver require an higher level of reliability [5].

However, due to the highly dynamic environment in which the vehicles operate, the satisfaction of these strict QoS requirements becomes very challenging. Furthermore, unanticipated degradation of QoS may compromise the adoption of TD, given the catastrophic consequences that a communication failure could have on the reliability of the system. In this context, Predictive QoS (PQoS) [6] is exploited as a mechanism to notify autonomous vehicles in advance about the upcoming QoS changes.

1.2 Predictive Quality of Service

The predictive approach of PQoS allows applications (e.g., TD) to predict unanticipated events and adapt accordingly, in contrast to the reactive mechanisms, where the QoS degradation is only addressed after it has already happened. Therefore, PQoS is fundamental in Vehicle-To-Everything (V2X) networks to preemptively evaluate the risks of QoS deterioration and ensure the satisfaction of requirements, especially latency and reliability, in the driving.

At the beginning, filter-based models (e.g., Kalman filters) and linear regression were explored in order to predict network changes, but they are not feasible in autonomous driving systems since a priori knowledge of the process to learn is required [7]. Moreover, these methods are not able to process large amount of input data or to predict nonlinear relationships between input features and output prediction.

Recently, many studies focused on ML models to predict and optimize wireless networks [8]. Indeed, these tools, starting from samples representing different networks observations (e.g., network metrics, available resources, channel usage, etc.), learn useful relationship to predict a suitable action, aiming to maintain QoS in case of unseen changes. Notably, ML does not require a priori rules, which is the common assumption for PQoS. [9].

In the TD scenario, PQoS can exploit predictions to ensure a high level of safety, by gently entering a safe mode if the QoS parameters become sensibly degraded. In particular, PQoS should predict possible network degradation where data transmissions to/from the remote control center will be slow or unfeasible, e.g., in a congested area or if the link quality is poor: the following principles [4] must be achieved.

1. Awareness: the PQoS mechanism should predict when the network status may lead the driver to lose the control of the vehicle, such that there is time for the remote center to react accordingly;

2. Precautions: before losing the control, the PQoS mechanism should tell the car to adopt safety measures, as reducing speed or pulling over, depending also on the environment;

3. Self-Survival: PQoS mechanism should assist the vehicle in what behavior to maintain until the network is re-established with the remote driver.

1.2.1 Reinforcement Learning for PQoS in V2X Networks

Among all techniques, Reinforcement Learning (RL) has emerged as a powerful tool to support PQoS inside V2X environments, specifically in the TD scenario.

In [9], a RL agent is designed in order to identify appropriate countermeasures if the QoS requirements are not satisfied, starting from a meaningful state for the network status. More precisely, the agent must choose the optimal size of the data (in this case point clouds from LiDAR sensors) generated by the application layer, in the vehicle to optimize data transmission. A suitable compression of sensors data can alleviate the channel usage, but an optimal trade-off between QoS and Quality of Experience (QoE) has to be reached. The reward function is strictly related to QoS, to meet the requirements of TD (latency and reliability), as well as to QoE, to ensure that the quality of transmitted data is good enough to perform TD operations.

The agent is implemented in a centralized approach at the Radio Access Network (RAN), so the following measurements, both from the environment and the RAN itself, can be gathered to represent the state.

- Context information: operational scenario, road elements, network architecture, time of the day, etc.

- Users trajectories

- Traffic information

- Network metrics: measurements from the 5G protocol stack (Physical [PHY], Medium Access Control [MAC], Radio Link Control [RLC], Packet Data Convergence Protocol [PDCP] layers)

- Higher layers metrics: E2E performance indicators (e.g., application latency and throughput)

However, the centralization requires all the vehicles to share local data and measurements (e.g. Signal-to-Interference-plus-Noise Ratio [SINR]) with the Next Generation Node B (gNB) in the RAN; in addition the commands must be sent from the agent to the vehicles. This exchange potentially leads to overhead and congestion, resulting in higher latency in the network. Moreover, it may expose end vehicles to security threads since sensitive raw data must be shared with the network.

Considering the drawbacks of the centralized model, decentralized solutions for PQoS are investigated in [10]. Specifically, the distributed and federated approaches.

In the distributed approach, each vehicle implements an independent and autonomous agent, which learns exploiting only data and measurements gathered locally. Therefore, the communication overhead and delays for transmission with the gNB are reduced, while the convergence becomes slower since less data are available for the training and each vehicle is not aware of global status of the communication scenario.

In order to speed up the convergence to an optimal policy, in the federated approach, vehicles periodically share their own model parameters with a central server (e.g, in the gNB). The aim is training collaboratively a global model aggregating the local models (at the vehicles) in an iterative way. This approach can be considered as an hybrid solution between centralized and distributed solutions, since the training is still mostly local but, at the same time, global model updates are periodically exchanged to the network. As a consequence, the agent's convergence is faster than in distributed learning, even though the exchange of model parameters may have a negative impact in terms of latency.

1.3 Thesis contribution and structure

The objective of this thesis is to design, implement and evaluate optimized learning solutions for PQoS in the TD scenario. In the context of RL, the agent has to perform optimal actions in order to preserve QoS indicators, especially the E2E latency: the more accurate and complete the state description, the more the

agent will have full understanding of network conditions, and optimize driving decisions accordingly. Nevertheless, the availability of measurements the agent can gather depends on the specific learning approach (i.e., centralized or decentralized) which has been implemented in the V2X scenario.

Network simulator 3 (ns-3), an open source discrete-event network simulator, is the main tool used to carry out full stack E2E simulation campaigns in vehicular networks and gather data to train online the RL agent using a Double Deep Q-Learning (DDQL). The benchmark `mmwave` module [11] is extended to incorporate the proposed realistic solutions in the TD scenario. Moreover, the `RAN-AI` [12] and the `UE-AI` [10] entities are modified in order to gather and share with the RL framework the proposed metrics. Simulation results are evaluated in terms of QoS (i.e., E2E delay) and QoE (i.e., mean Average Precision [mAP]), checking if the Key Performance Indicators (KPIs) of TD are satisfied.

In Chapter 2, the goal is to introduce realism in the learning phase with respect to data and metrics available at the RAN infrastructure (centralized case) or at the vehicle (distributed case). This requires a deep analysis of the 5G networks protocol stack to identify which measurements are collected and which are not by the RAN and the users (i.e., the vehicles). Then, in the implementation phase, metrics already provided by the 5G architecture will be exploited, especially scheduling-related control mechanisms at the MAC layer. Moreover, additional traffic to exchange local data between gNB and UE is modelled in order to generate realistic overhead in acquisition of the metrics needed for the learning of PQoS countermeasures.

Specifically, in the centralized case with 3 vehicles, the first proposed state presents a concrete policy of transmission of the SINR perceived by the UE, enlarging consequently the E2E delay of 6% with respect to the idealized state. In this configuration, a new learning evolution slightly improves the QoE. The second proposal employs the Channel Quality Indicator (CQI) reporting mechanism in place of SINR: the delay is comparable with the original state result, the average mAP reaches a value of 0.68, slightly larger than the value obtained in the previous proposal.

In the distributed case, a new state monitoring RLC buffers and involving the transmission of ACKs at the application layer to compute the latency is implemented. The first proposed state generates a very large degradation of the QoS with respect to the original idealized state. This approach does not scale

with a large number of vehicles, for this reason a new state is proposed. It tries to estimate the latency, looking at the transmitted bytes and the Modulation and Coding Scheme (MCS). Very good results are achieved in terms of delay (24 ms), aligned with the idealized configuration, while the mAP is a bit smaller.

In Chapter 3, several approaches are tested to improve the hybrid approach of the federated learning. More precisely, the performance of the RL agent is evaluated following the implementation of three distinct techniques in the `federated_update` function: adaptive federated optimization methods in the parameters aggregation, compression techniques (threshold, quantization and clustering) on the model parameters to share and different utilities of the clients during the federated step.

Best performance is achieved by the FedAdam optimizer with respect to FedAvg, it is able to optimize the overall reward, improving the QoS at the expense of the QoE. Adopting quantization, outstanding results are reached in terms of delay (27 ms) and mAP (more than 0.68), being able to compress efficiently the model updates, without deteriorating the learning convergence. Binary quantization slightly outperforms the ternary one in terms of QoE because smaller packets of 33% are shared. With many vehicles, there are appreciable differences among the different utility definitions. Specifically, the sample-based approach involving the loss value improves the QoS.

In Chapter 4, after a comparison between the results achieved by the proposed solutions in centralized, distributed and federated approaches, a new meta-learning agent that can dynamically choose when to perform centralized vs. decentralized learning models is implemented, jointly optimizing the QoS and the QoE. The DDQL algorithm is exploited and a new RL framework (state, action, reward) is implemented to manage the learning during the simulation campaign in ns-3.

- A centralized approach is more appropriate if the channel is good and the network is not congested, in this way each vehicle is able to perceive the full network state, increasing the accuracy of the decisions. On the other side, the impact of the overhead to communicate with the gNB is significant, and shall be taken into careful consideration..
- A distributed approach is more convenient if the channel is bad and the

network is congested so that the communication overhead for the state aggregation is lower. On the other side, only local metrics are naturally available during the learning.

- The federated approach represents an hybrid solution between centralized and distributed paradigms since it emulates the distributed approach for the inference, but involves federated updates for faster convergence at the cost of higher communication resource usage during the training.

The meta-learning agent shows very promising results, being able to switch autonomously in time between centralized vs. decentralized learning models. More precisely, in good channel conditions, the centralized approach produces the best results, because of low delay and full perception of the network state. Indeed, the meta-agent prefers clearly the centralization (80%) over the decentralization (20%). On the contrary, when the channel becomes more degraded and the probability of outages increases, the decentralized federated approach is preferred (75%) over the centralized case (25%). When the communication with the gNB is not feasible, the centralized agent is not able to gather local metrics related the vehicles to build the state.

2

Centralized and Distributed PQoS

The objective of this chapter is to introduce realism in the collection of metrics to describe the network state to the RL agent. First, a phase of understanding which metrics to collect in the state are available at the centralized and distributed agents is required, looking at the 5G protocol stack. There are many measurements that are available at the RAN and/or the gNB without overhead, there are other metrics that can be collected in the simulation in an idealized way, i.e., these metrics are assumed to be available at the RAN with no additional overhead. In this case, the implementation of realistic communication and computational mechanisms in ns-3 for obtaining those metrics is required to build the network state.

First of all, a description of the V2X scenario, and use cases, requirements and technologies of the PQoS techniques, with an important focus on the TD, is presented. An analysis of the 5G New Radio (5G NR) system architecture is provided, in order to explore protocols and relative mechanisms at different layers of the V2X network stack. Subsequently, new definitions for centralized and distributed learning approaches are proposed, based on realistic knowledge gathered exploiting the following sources:

- local metrics;
- reporting systems;
- additional exchange mechanisms.

Finally, a comparison between different state configurations and their impact on the performance of the agent is presented, with a detailed focus on the QoS and the QoE.

2.1 V2X communication: PQoS for TD scenario

V2X communication is a key component of ITSs, where vehicles are interconnected with other vehicles, infrastructures, pedestrians and networks. Vehicular networks represent an extreme and dynamic environment due to the high mobility of nodes and especially the very strict requirements in QoS, particularly latency and reliability [13]. The recent advent of innovative communication technologies and network architectures, integrating AI and ML functionalities, enables the development of fully autonomous systems capable of meeting QoS demands and optimizing network resource management [1].

In this critical environment, being able to anticipate and notify upcoming network degradation and react accordingly is fundamental to preserve an high QoS in the V2X applications. This mechanism is called PQoS [6].

PQoS has a crucial role in improving the safety and reliability of V2X applications, particularly in scenarios involving high-definition map sharing, TD and high-density platooning. By network predictions, PQoS can anticipate and mitigate potential issues, ensuring that autonomous driving systems satisfy latency and reliability constraints even in challenging conditions.

Focusing on the TD, a remote operator, human or software, is required to control a vehicle. In this scenario, there is the transmission of high-resolution video data from the vehicle's onboard sensors, with data rates reaching hundreds of Mbps [2], to the control center. In addition, ultra-low latency ($\leq$50 ms) and extreme reliability (99% in UL and 99.999% in DL) must be satisfied by the driving commands communication. The former to ensure responsiveness during vehicle control, the latter to ensure the accurate execution of commands and maintain safety.

In this scenario, PQoS chooses countermeasures to deal with network predictions, eventually facing QoS degradation. If this is the case, the system could gently enter in a safe mode or, in the worst case, require the human takeover if QoS is extremely poor and not sufficient to support safely the TD. For instance, if a route becomes congested, the control center can adjust the vehicle's speed and

path to maintain safety with the assist of PQoS, based on the network conditions [6].

In general, PQoS can be implemented at RAN or Core Network (CN) side of cellular infrastructure. Looking at RAN-based solutions, radio map prediction, predictive adaption of radio parameters and predictive scheduling are the main approaches to achieve PQoS [6]. Radio map prediction consists in predicting the radio signal propagation, i.e., the path loss or SINR, starting from measurement collection and then applying ML techniques to analyze and forecast the radio environment in foreseen locations. Predictive adaptation of radio parameters consists in dynamically optimizing network parameters based on QoS estimates. For instance, lower-layer decisions related to the transmission power, system numerology, radio resource allocation ad scheduling can be adapted in real-time to match the foreseen network conditions. Moreover, higher-layer actions can be tuned, e.g., the compression of data generated by the application layer. Predictive scheduling refers to the allocation of network resources in time and frequency, based on predicted mobility patterns and network conditions, before of the network changes. Depending on the coverage, the base station or directly the users can schedule resources in a more efficient way, reducing the latency and improving the service quality.

The most advanced method for generating PQoS predictions is through the neural networks (NNs), which exploit input features related to the network conditions, resource availability, predicted mobility patterns, or other network observations. The output consists in the action/countermeasure to adopt to preserve QoS, especially in case of degradation, without static rules or a-priori knowledge [6]. From the other side, a rich training dataset is required to build a robust predictor, together with computational power and time. Traditional predictors as linear regression or filter-based model are not suitable for large amount of data with non-linear relationships [7].

Recently, RL methods are investigated to achieve PQoS in TD scenario [9][10]. A RL agent, capturing network conditions and QoS indicators, learns how to select the optimal compression mode (i.e., PQoS action) to apply on vehicle's data in order to satisfy TD constraints. In this case, the training is performed online, so a continuous learning is performed by the agent, with a good effect on data distribution variability due to the dynamic environment. In addition, a huge training set is not required.

2.2 5G NR System Architecture

The 5G NR architecture is made up of two main components: the RAN and the CN. The split of functions between RAN and CN is necessary to have the same CN serving different radio-access technologies [14].

The RAN interconnects user terminals (e.g., mobile phones, vehicles, sensors, etc.) to the 5G CN. It mainly manages all the radio resource management function of the whole network: dynamic resource allocation (scheduling), radio bearer and admission control, connection mobility control [15].

The CN manages functions related to access authentication and authorization, mobility management control, Packet Data Unit (PDU) handling and session control. It also provides session management and network slicing functions [15].

2.2.1 CN

The CN offers both control plane and user plane functions. The control plane carries signaling messages to control the connection between user equipments and the network, the user plane carries user traffic.

The user plane involves the User Plane Function: it is in charge of routing and forwarding packets, packet inspection, policy rule enforcement, QoS handling, traffic reporting and verification [15].

The control-plane functions are divided in the following parts [14][15] .

- The Session Management Function is responsible of session management, IP address allocation for UEs, policy enforcement, and general management of UP function
- The Access and Mobility Management Function controls signalling between the CN and the device and manages data security, reachability of idle UEs, access authentication and authorization, and mobility management control (both intra-system and inter-system)
- The Policy Control Function manages policy rules framework, including QoS, filtering and other decisions
- The Unified Data Management manages authentication credentials and handles access authorization to subscribers' data

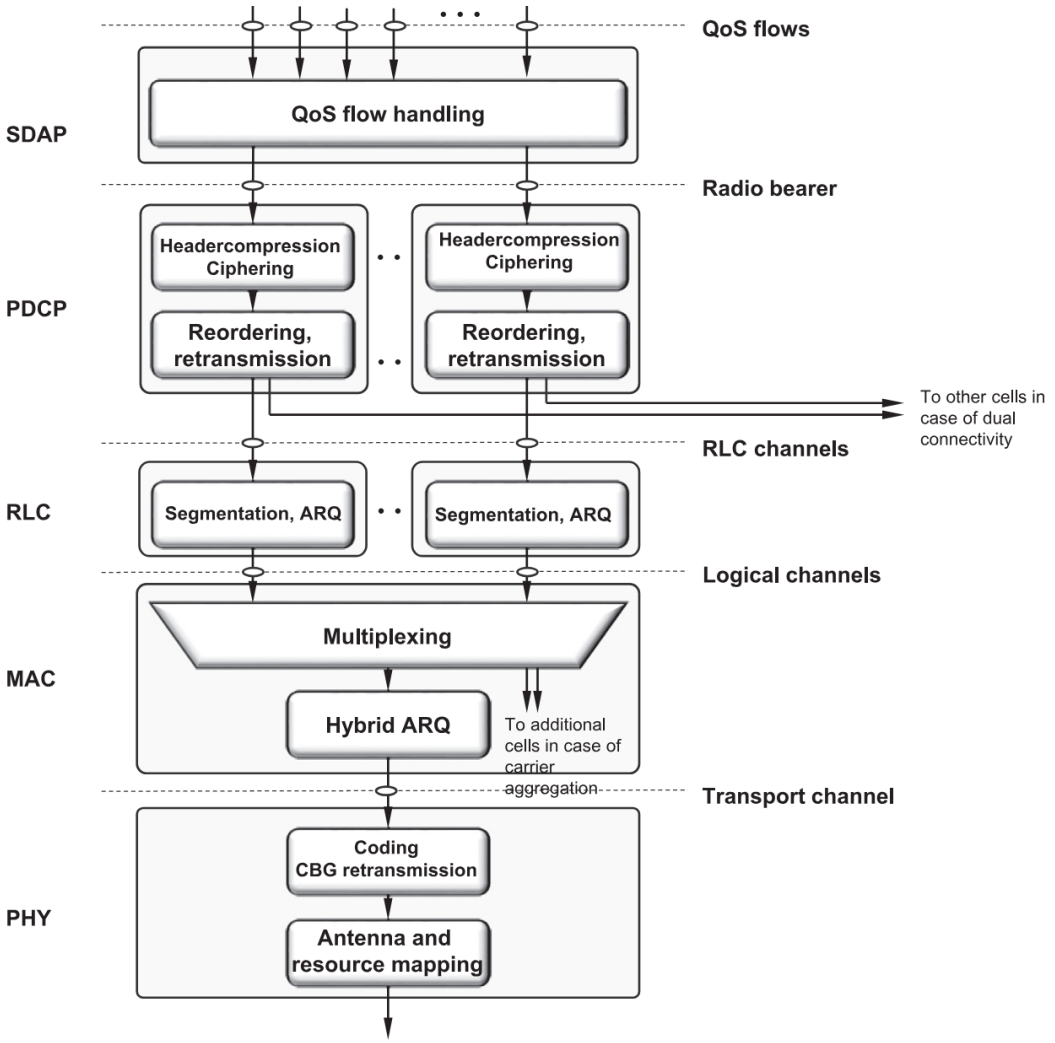


Figure 2.1: NR DL user-plane protocol stack [14]

- The Network Exposure Function exposes network services and capabilities to third-party applications in a secure way
- The NR Repository Function manages a repository of available network functions and relative profiles
- The Authentication Server Function manages the authentication process
- The Network Slice Selection Function selects the appropriate network slice, which is a collection of logical network functions supporting the requirements of a specific use case, to serve a device
- The Unified Data Repository manages subscriber information

- the Application Function controls traffic routing and interacts with the Policy Control Function to request policy control decisions

2.2.2 RAN

The RAN is composed by an arbitrary number of gNBs, interconnecting individual devices (UEs) to the CN. The gNB is in charge of all radio-related functions, as radio resource management, admission control, connection establishment, routing of user/control-plane data and QoS flow management [14].

2.2.3 Protocol Architecture [14][15]

After the brief explanation of the 5G NR system architecture, now the RAN protocol stack is illustrated. The case of interest for this thesis is about the user-plane protocols, since the measurement collected by the agent are linked to the data traffic between vehicle and RAN. This reflects on analyzing how the data is transmitted, received, and managed through the different layers of the user-plane protocol stack.

The RAN user-plane architecture (Fig. 2.1) is made up of the following protocols.

- The Service Data Application Protocol (SDAP) is responsible for mapping QoS flows to data radio bearers according to relative QoS requirements, multiple QoS flows can result in the same data radio bearer. Previously, the UDF function inside the CN mapped IP packets to the proper QoS flow.

- The PDCP is responsible for the transfer of user plane data and the IP header compression and decompression, employing specific protocols. It ensures the encryption and decryption of packets, as well as their integrity protection and verification. Furthermore, the PDCP is in charge of the discarding and reordering of duplicated packets and in-order delivery.

  The number of PDCP entities is equal to the number of radio bearers configured for the device.

- The RLC is responsible for the transfer of PDCP PDUs and the segmentation of RLC SDUs from PDCP layer into smaller RLC PDUs and reassembly. It ensures the retransmission of corrupted received PDUs and the detection and removal of duplicated PDUs.

RLC protocol does not ensure in-sequence delivery and forwards immediately packets to higher layers, in order to reduce the latency due to the reordering mechanisms. If needed, PDCP protocol can provide this service. In addition, to reduce the overall latency, no concatenation is performed by RLC, so RLC PDUs can be reassembled without waiting for the UL scheduling decision.

For each RLC channel, which provides services to the PDCP layer, there is one RLC entity established when a radio bearer is set up.

There are three modes in which a RLC entity can operate: transparent mode, unacknowledged mode, acknowledged mode.

  - In transparent mode, no RLC headers are added to the packet and buffering is done only at transmission. Segmentation and reception feedbacks are not present.
  - In unacknowledged mode, RLC headers are added and buffering is done both in transmission and in reception. Segmentation and duplicate detection is supported, while feedbacks are still not implemented.
  - In acknowledged mode, in addition to the UM features, retransmission of erroneous packets is supported.

- The MAC layer is responsible for the scheduling functions, logical channel multiplexing and hybrid-Automatic Repeat Request (ARQ) error correction. In case of carrier aggregation, it manages the multiplexing and demultiplexing of data across component carriers.

  Scheduling is the most critical task performed by the MAC layer, since many users share resources of the transmission channel in time and frequency. The scheduler at the gNB must decide how to assign, both in UL and DL, the resource blocks (frequency domain) and OFDM symbol slots (time domain) in an efficient way, and this decision will have a strong impact on the E2E latency.

  The are mainly two approaches: semi-static scheduling and dynamic scheduling [15]. In the first approach, referred to as semi-persistent scheduling (SPS), the radio resources are pre-assigned periodically to the UEs, where the periodicity is controlled by the Radio Resource Control

(RRC). As a consequence, the allocation is fixed for a period, reducing control plane overhead but, at the same time, decreasing the flexibility in the scheduler decisions.

In the dynamic scheduling, the scheduler (typically) at every slot decides how to schedule the resources based on the channel state, preferring devices with advantageous conditions (channel-dependent scheduling). More precisely, in DL direction, the device reports its channel quality to the gNB sending frequently a control signal, the Channel State Information (CSI). In the UL direction, the scheduler exploits a sounding reference control signal transmitted from each UE for which the gNB wants to estimate the channel quality.

The MAC provides services to the RLC through logical channels. A logical channel can be classified as a control channel, if controls and configuration information are transmitted, or as a traffic channel, if user data are exchanged.

- The PHY layer is responsible for a large number of functions, including modulation and demodulation, coding and decoding, physical-layer hybrid-ARQ, multiantenna processing and mapping signals to physical time-frequency resources.

  The PHY provides services to the MAC through transport channels. Each transport channel is mapped to a physical channel, which is a set of resources in time and frequency.

2.3 Reinforcement Learning

RL is a ML technique where an agent interacts with an environment to learn how to maximize the cumulative future reward R. Considering $t = 0$ as the initial time step, the (infinite-horizon) discounted R is defined as the following sum over time

$$R = \sum_{t=0}^{\infty} \gamma^t R_t \,, \tag{2.1}$$

where γ determines the present value of future rewards. A $\gamma = 0$ means that only the immediate reward is important, which makes the agent myopic towards the future; instead, the agent takes into account future rewards with γ value

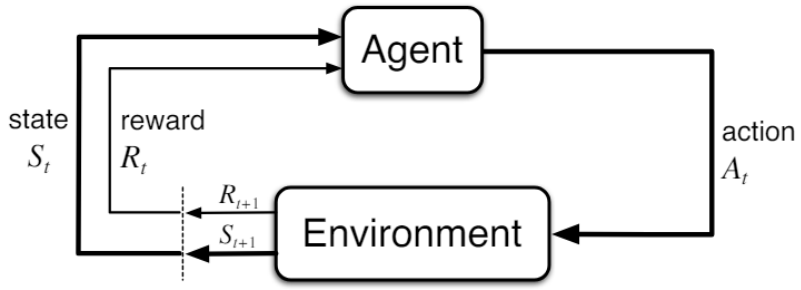


Figure 2.2: RL framework [16]

close to 1.

The RL framework can be formalized mathematically as a Markov Decision Process (MDP). A MDP is defined by the tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$, such that:

- $\mathcal{S}$ is the finite set of states
- $\mathcal{A}$ is the finite set of actions
- $\mathcal{P}$ is the state transition probability matrix, where the elements are

$$\mathcal{P}^{a}_{ss'} = P[S_{t+1} = s' | S_t = s, A_t = a]$$

- $\mathcal{R}$ is the reward function, where

$$\mathcal{R}^{a}_{s} = E[R_{t+1} | S_t = s, A_t = a]$$

- $\gamma \in [0, 1]$ is the discount factor

There are several elements which define the entire RL framework, represented in Fig 2.2.

- Agent: intelligent entity with a task to learn, following a certain policy π
- Environment: element with which the agent interacts, it gives at each time step a reward and moves the agent into a new condition (depending on the state-action pair)
- State: meaningful and representative description of the system (agent and environment) where the agent is located
- Action: decision taken by the agent from a certain state

- Reward: scalar value representing how well the agent is doing

More precisely, at each time step t (discrete time is assumed, the same concepts can be extended to continuous time), the agent interacts with the environment and observes the state S_t, then it decides to take action A_t. The environment, based on (S_t, A_t), provides the agent with reward R_{t+1} and moves the agent in state S_{t+1}.

The agent follows a policy, i.e. a mapping of the state into an action: it can be deterministic, $A_t = \pi(S_t)$, or stochastic, $\pi(a|s) = P[A_t = a, S_t = s]$.

The agent goal is to find the optimal policy π^* that maximizes the expected return G_t, defined as the discounted R from time t

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \, . \tag{2.2}$$

The (state-)value function of a state s under a policy π is the expected returnSS from state s and following π. In a MDP, it is defined as follows.

$$v_\pi(s) = E_\pi[G_t|S_t = s] \tag{2.3}$$

The action-value function of taking action a from state s under a policy π is the expected return from state s, taking action a and following π. In a MDP, it is defined as follows.

$$q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a] \tag{2.4}$$

Knowing the optimal action-value function, it is possible to derive directly the optimal policy selecting the action which maximizes this function.

A MDP underlies every RL problem, however, in typical real-world scenarios (as the TD), the model of the environment is unknown a priori. As a consequence, the MDP can't be fully defined because there is no knowledge about $\mathcal{P}$ (the state transition probabilities) and $\mathcal{R}$ (the reward function). Dynamic programming methods, which rely on Bellman equations, cannot be exploited since a complete and accurate model of the environment is required.

To address this problem, model-free approaches (e.g., Monte Carlo methods, temporal-difference learning, etc.) can be explored in order to solve the RL

problem. Differently from the model-based case, these methods learn from experience by sampling sequences of states, actions and reward, so real world data are gathered interacting with the environment, without any requirement on its model.

In this thesis, model-free value-based algorithms are explored, meaning that the agent tries to estimate the optimal action-value function and learn a policy based on that. Q-Learning [17], SARSA [18] and Deep Q-Learning [**DQN**] are examples of value-based methods. The counterpart approaches are base on a policy-based strategy, where the agent tries to learn directly the optimal policy, without taking into account the value function.

However, in the TD scenario, the state space becomes continuous and enormous: finding the optimal value function for each state (and the policy) is not feasible. The main problem is related to the lack of generalization of tabular methods, and, as a consequence, it is impossible to require the system to experience an infinite number of states. In the absence of generalization, the agent cannot exploit knowledge learned from one state to make decisions in similar states. Thus, tabular methods are not the best choice in this context [16].

The new objective is to find a good approximate solution, being able to generalize from previously visited states that are similar to the current one. A function approximator is required in order to take samples of the value function and attempt to generalize, building an approximation of the entire function [16]. In practical terms, the value function depends on a vector of parameters, accord-

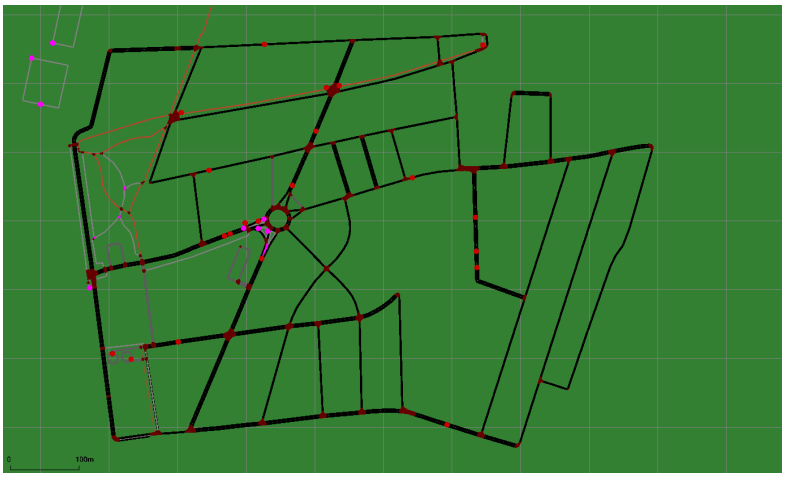


Figure 2.3: Portion of the city of Bologna used in the simulations

ing to a definition based on the approximation approach. The state of the art for function approximation is represented by the powerful Deep Neural Networks, but there are also approaches based on linear combinations of feature, decision tree, multivariate regression, etc.

2.4 PQoS Framework

The PQoS framework is applied to the TD service, involving a vehicular scenario set in a portion of the city of Bologna (Fig. 2.3). The parameters of the simulation are described, beginning with the network topology and its parameters, continuing with the channel model and the application layer description. Subsequently, a focus on the ns-3 entities is presented. These parameters are fixed in every PQoS learning approach.

2.4.1 Simulation parameters

Network

In the simulated TD scenario, the centralized PQoS framework involves a V2X network composed by:

- a gNB
- a remote host (teleoperator/driving application)
- n vehicles (i.e., UEs).

A wired channel connects the remote host to the gNB, with a propagation delay τ of 10 ms and a transfer data rate R of 100 Gbps. The gNB communicates with the vehicles through the 5G NR technology. The system operates at a carrier frequency f_c of 3.5 GHz (NR band n78) and with an available bandwidth B of 50 MHz. The numerology adopted is the 3rd. The gNB has a transmission power of 30 dBm, while the UEs have a tranmission power of 23 dBm.

Channel

The mobility of the vehicles is simulated by Simulation of Urban Mobility (SUMO) [19], in an area of the city of Bologna. The channel propagation loss

model is directly computed by GEMV [20], a geometry-based propagation simulator. GEMV traces describe the received powers of 50 different vehicles, for a total time of 90 s and with an interval of 100 ms between consecutive acquisitions. As a consequence, the path loss is not calculated using ns-3 methods, but it is directly acquired by reading the GEMV traces.

Vehicle data

Each vehicle is equipped with on-board sensors, as RGB cameras and LiDARs, to track the city environment and share it with the remote teleoperator. In particular, each LiDAR perception generates a 3D representation of the surroundings in the form of a point cloud. If the resolution and the sampling frequency are very high, the average data rate to exchange these point clouds can reach hundreds of Mbps. For this simulation, the application installed at the UE produces point clouds at a frequency of 30 Hz.

Compression

The frequent transmission of huge point clouds could be challenging for the network infrastructure, especially if high standards relative to E2E latency must be achieved in TD. Thus, compression techniques can be applied to the point clouds in order to reduce their size, decreasing the overall data rate and consequently the burden on the transmission channel.

Draco [21] is exploited to achieve a real-time compression of LiDAR perceptions. Draco is an open-source library, developed by Google, for compressing and decompressing 3D geometric data, especially meshes and point clouds. The compression can be tuned setting two parameters:

- the number of quantization bits $q \in [1, 31]$,
- the number of compression levels $c \in [0, 10]$.

Operating on q and c, a trade-off between compression ratio and decoding performance can be achieved. More precisely, q determines the amount of bits used to encode each coordinate of the point cloud, affecting in a strong way the size and the quality of the compression. The other main parameter c controls the operations performed during the compression, affecting heavily the time required for the point cloud compression and decompression.

2.4.2 Ns-3 entities

A full-stack E2E simulation is provided exploiting the network simulator ns-3.

`mmwave` module

The open-source `mmwave` module [11] is exploited for simulating 5G NR networks, supporting communication both in Frequency Range 1 and in Frequency Range 2 (millimeter waves). This module is integrated with RAN-AI and UE-AI classes and functionalities to enable PQoS during the simulation.

Applications

At the vehicle an application to model the transmission in UL of LiDAR data to the remote driving application installed at the remote host. The traffic model is managed by the `SelmaTraceBurstGenerator`, which controls at runtime the type of DRACO compression applied to the real point cloud, extracted from the SELMA dataset [22]. Then, the `BurstyApplication` and `BurstSink` are in charge of transmitting the compressed point cloud, through fragments of smaller size (1200 byte), and receiving all the packets doing an aggregation at the destination, respectively.

In the DL direction, the `UdpClient` at the transmitter and the `PacketSink` at the receiver are in charge of exchanging the remote driver commands to the vehicles. A data rate of 0.32 Mbps per vehicle is generated.

In both directions, the applications run based on the connectionless UDP transport, to reduce the overhead of flow control and congestion control. Error correction features and reliability are ensured by lower layers (RLC, MAC and PHY) mechanisms.

2.5 Centralized Approach

2.5.1 RAN-AI

In the centralized approach, the RAN-AI entity [12] can be defined as an intelligent network controller, installed in the gNB and connected to all the RAN and CN components. The role of the RAN-AI is to collect networks metrics from

different sources in order to take decisions (i.e., the compression mode) ensuring PQoS, to optimize the V2X network status in a predictive way. This task is computed exploiting AI functionalities, i.e. the RL agent incorporated in the learning framework.

More precisely, the RAN-AI operations are performed through the following steps.

1. Collection of the network measurements from the RAN, related to each vehicles
2. Creation of the states representations from the gathered metrics for the RL agent
3. Estimation of the PQoS countermeasures, based on the RL agent
4. Delivering of the actions to the vehicles

The class `RanAI` implements the RAN-AI functionalities, so an instance of that class must be included in the gNB class (`MmWaveEnbNetDevice`). The following key methods of the gNB enables the RAN-AI to operate.

1. `InstallRanAI()` is in charge of initializing the RAN-AI instance and relative calculators at RLC and PDCP layers and APP layer. In addition, it schedules the collection of the measurements by the RAN-AI entity.
2. `SendStatusUpdate()` is in charge of gathering and assembling the full-stack network statistics for each vehicle to share with the RL agent.
3. `ReportMeasures()` is in charge of transferring the metrics to the agent and letting it to compute the optimal action(s) for the given input state(s). Then, the decision(s) is (are) read from the same shared memory and set in the traffic generator at the vehicle(s).
4. `NotifyActionIdeal()` is in charge of propagating the RL agent's decisions to the application on each vehicle, where the compression mode has to be configured accordingly.

The communication between ns-3 simulations and RL Python algorithms is integrated inside the `ns3-ai` module [23].

2.5.2 RL FRAMEWORK

For the centralized scenario, the RL framework involves the following definitions of state space $\mathcal{S}$, action space $\mathcal{A}$ and reward function $\mathcal{R}$.

Original State The discrete state space contains vectors $\mathbf{s} \in R^{18}$ of measurements from the entire protocol stack of a user (i.e., vehicle), related to the UL direction. For completeness, they are reported in the underlying list.

PHY layer

- SINR perceived at the gNB;
- MCS of the transmission;
- Number of Orthogonal Frequency-Division Multiplexing (OFDM) symbols used in a slot.

RLC and PDCP layers

- Average delay of PDUs;
- Standard deviation of delay of PDUs;
- Minimum delay of PDUs;
- Maximum delay of PDUs;
- Packet Reception Rate (PRR), computed from the number of transmitted PDUs and the number of received PDUs.

APP layer

- Average delay of bursts;
- Standard deviation of delay of bursts;
- Minimum delay of bursts;
- Maximum delay of bursts;
- PRR, computed from the number of transmitted burst and the number of received burst.

(q, c)	Average mAP
(8, 0)	0.257
(8, 5)	0.257
(8, 10)	0.257
(9, 0)	0.580
(9, 5)	0.572
(9, 10)	0.574
(10, 0)	0.686
(10, 5)	0.683
(10, 10)	0.683

Table 2.1: Average mAP achieved by the object detector PointPillars on the SELMA dataset, depending on the compression mode

Actions The discrete action space involves 9 possible choices for the PQoS countermeasure to adopt, i.e. the compression mode. They represent all the combinations of the 2 main DRACO parameters, the number of quantization bits and the number of compression levels, restricted to $q \in \{8, 9, 10\}$ and $c \in \{0, 5, 10\}$, since they are the most representative values for the purposes of this thesis.

Reward The reward function at time t, $R_t(s, a)$, depends on two contributions:

- The QoS to ensure the satisfaction of TD requirements in terms of Key Performance Indicators (KPIs); in this work, only the E2E delay δ_t is considered. δ_M is the maximum tolerated E2E delay, fixed to 50 ms from the 5GAA.
- The QoE to ensure that the transmitted data is accurate enough to perform remote driving operations, as the object detection. The mean average precision (mAP) m_a achieved by the object detector PointPillars [24] on the SELMA dataset, with compression mode a is considered as metric to evaluate the accuracy of the object detection. The average values achieved are reported in Tab. 2.1.

The formula of $R_t(s, a)$ is defined as:

$$R_t(s, a) = \begin{cases} m_a & \text{if } \delta_t \leq \delta_M \\ -\frac{\delta_t}{100} & \text{otherwise} \end{cases} \tag{2.5}$$

2.5.3 Proposed Solutions and Implementation

The original state, described in Sec. 2.5.2, presents several limitations and assumptions that are too strong for a realistic centralized scenario.

PHY layer The state does not capture the BLock Error Rate (BLER) and the Transport Block Size (TBS), two important metrics related to the Transport Block (TB), i.e. the packet exchanged between the MAC and PHY layers. The BLER, defined as the ratio between the number of TB received with errors and the total number of received TB, can help the agent to evaluate the physical layer performance of the vehicle, together with PRR values at the RLC, PDCP and APP layers, already present in the state. As far as the TBS size is concerned, it is directly linked to the amount of data that can be transmitted at PHY layer over a Transmission Time Interval (TTI) slot. A large TBS indicates that the channel can support higher data rates, due to an higher MCS, while a small TBS reflects poor channel conditions requiring more robustness [25]. As far as the implementation is concerned, inside the `mmwave` module, the `RxPacketTraceEnbCallback()` method in the `MmWaveEnbNetDevice` class is modified to store over time (between two consecutive state acquisition) TBS and BLER values every time a TB is received by the PHY layer at the gNB.

Furthermore, the SINR value present in the state is directly computed at the gNB, without any mechanism to evaluate it involving an exchange of control signals with the user. There are scheduling-related control mechanisms at the MAC layer, specified by the Third Generation Partnership Project (3GPP) standards, that can be exploited to get the SINR perceived at the UE, but just in a quantized value. For example, via the Channel Quality Indicator (CQI).

The implementation of a new additional control message is provided in order to have a fine-grained value of the SINR perceived by the UE.

RLC and PDCP layers The state is not able to describe the situation of the transmission buffer at the user and the overhead to transmit such information must be avoided. Therefore, another scheduling-related control mechanism of the MAC allows the gNB to make an estimation of the vehicle buffer, without explicit interaction with the UE. The Buffer Status Report (BSR) provides frequently the gNB with UL data volume of RLC and PDCP layers.

CQI

The CQI [25] is part of the CSI, which contains other measurements essential for an efficient and adaptive scheduling of the available resources, such as the Rank Indicator (RI) and the Precoding Matrix Indicator (PMI).

In particular, the CQI reflects the quality of the data communication channel between the UE and the base station (i.e., the gNB) and it is computed starting from the SINR measured at the UE. At the scheduler, the CQI impacts also on the type of MCS adopted for the transmission to the UE in order to achieve the required BLER.

More precisely, the CQI notification process goes through the following steps.

1. CQI computation

   Knowing the SINR value at the UE, the spectral efficiency (SE) achieved with the target Bit Error Rate (BER) is computed. Exploiting the tables mapping CQI indexes to SE (plus code rate and modulation) reported in 3GPP standards [25], the highest CQI index which ensures a SE not greater than the SE achievable is chosen. CQI indexes are integer values in [0, 15].

   The following formula is exploited inside the `mmwave` module to compute the SE from the SINR and the BER.

   $$\mathrm{SE} = \log_2\left(1 + \frac{\mathrm{SINR}}{-\frac{\ln(5\cdot\mathrm{BER})}{1.5}}\right) \tag{2.6}$$

2. CQI reporting
   The time scheduling of the reporting mechanism is specified by the higher layer parameter *reportConfigType* of Radio Resource Control (RRC) [25] and it can be:

   - aperiodic;
   - semi-persistent on the physical uplink control channel;
   - semi-persistent on the physical uplink shared channel;
   - periodic.

   Inside the `mmwave` module, after that data packets are received, the PHY layer at the UE generates the CQI and reports it to the gNB in control slots

[11]. Anyway, it is always enabled the periodic approach, where the periodicity is fixed in number of slots. The available values are 4, 5, 8, 10, 16, 20, 40, 80, 160, 320. More precisely, the attribute `CqiReportPeriod` in the `MmWaveUePhy` class allows to tune the CQI periodic reporting.

In the simulations, the period is set to 10 slots, which corresponds to a time interval of 1.25 ms in numerology 3. In this way, a sufficient granularity in the updates to the gNB, then acquired by the RL agent, is ensured.

3. Adaptive Modulation and Coding

The gNB uses the reported CQI to set the transmission parameters, in particular the MCS, based on real-time channel conditions.

- High CQI: higher-order modulations and higher code rates are employed.
- Low CQI: lower-order modulations and lower code rates are employed.

These steps creates a feedback loop where the UE frequently shares the channel condition with the gNB, so that the gNB can adapt its transmission strategy accordingly.

Proposed SINR control message

The CQI value could be not sufficient to fully describe the channel status at the vehicle, so it is reasonable to transmit to the gNB the original value of the SINR perceived by the UE. The exchange of this additional metric is not present in the simulator, so a new control message `MmWaveDlSinrMessage` (extending the `MmWaveControlMessage` class) and relative overhead have been modelled.

To model this exchange, an additional symbol with respect to the one already dedicated to the UL control is reserved. As a consequence, among the 14 OFDM symbols inside a slot, 11 remain for data transmission and the other 3 symbols are used for the control (1 in DL, 2 in UL). The addition of a control symbol, which corresponds to the removal of a data symbol, makes sense if all the data symbols have been previously employed for the transmission of application data, otherwise this choice has no impact on the network performance. As far as the implementation is concerned, inside the `mmwave` module, when the slots allocation is initialized in `InitializeSlotAllocation()` of the `MmWaveUePhy`

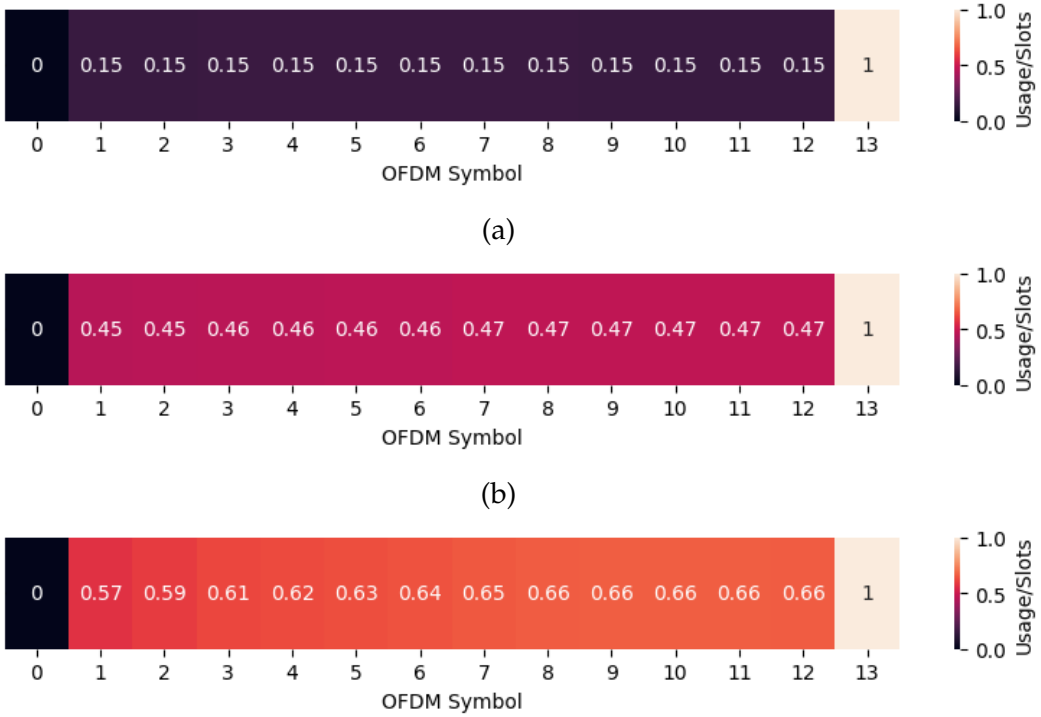


Figure 2.4: OFDM symbols usage over the total number of slots inside a simulation with (a) 1 vehicle, (b) 5 vehicles and (c) 10 vehicles.

class, two symbols are reserved for UL control by `SetSlotCtrlStructure()` (`MmWavePhy`) and `SetUlCtrlSymbols()` (`MmWavePhyMacCommon`).

Since the `BurstyApplication` generates burst transmissions of point clouds, all symbols are utilized immediately in the same way, independently by the number of vehicles. As a consequence, the additional symbol reserved to the control represents an important overhead for the communication. However, the congestion of the network due to many vehicles impacts for sure the evolution in time of the channel resources: the usage of data symbol increases, resulting in fewer slots reserved solely for control during the simulation. The usage of OFDM symbols (over the total number of slots) with a different number of vehicles is shown in Fig. 2.4.

The easiest approach is sending the control message every time a packet (with data or control) is sent in DL. A more realistic strategy can be exploited following the policy described below.

1. If the SINR is below a certain threshold, this means that the channel is too

degraded. An outage occurs and the gNB does not receive the SINR report from the UE.

2. If the SINR is similar to the previously sent value according to a certain threshold, the UE does not transmit the SINR report to the gNB. As a consequence, the gNB reuses the old SINR value.

3. If the SINR is similar to the previously transmitted value but an update timer has expired, the new value is sent to the gNB anyway (unless an outage is occured). A minimum of synchronization with the gNB must be ensured.

The generation of the control message with the continuos value of the SINR, according to the described policy, is implemented by the `GenerateDlSinrReport()` method in the `MmWaveUePhy` class.

BSR

The BSR is a reporting procedure to inform the gNB about the volume of UL data in the MAC entity. The amount of data should consider both the RLC and the PDCP buffers [26].

The RLC data volume [27] involves the following contributions:

- RLC SDUs and RLC SDU segments not included in an RLC data PDU;
- RLC data PDUs pending for initial transmission;
- RLC data PDUs pending for retransmission (RLC AM mode).

The PDCP data volume [28] involves the following contributions:

- PDCP SDUs not included in an PDCP Data PDUs;
- PDCP Data PDUs not submitted to lower layers;
- PDCP Control PDUs;
- PDCP SDUs to be retransmitted in particular cases (RLC AM mode).

The buffer status is reported for each Logical Channel Group (LCG), i.e., a group of logical channels. Each LCG has an identifier called LCG ID.

These pieces of information are packed inside the BSR Control Element, which can be in a short or long format. The short format is used only when

one LCG has data to transmit when the MAC PDU with the BSR is to be built, otherwise the long format is exploited in presence of multiple LCGs with data ready for transmission. There is also a truncated version of these formats that must be used when the number of padding bits is larger than the short BSR but smaller than the long BSR, and there are more LCGs with data available [26].

The fields inside the BSR MAC Control Element are the LCG ID and the buffer size, which is quantized according to tables provided by the 3GPP [26]. The LCG ID requires 3 bits, the buffer size requires 5 bits in the short format or 8 bits in the long format. In addition, the long format contains for each LCG a field to indicate the presence of the buffer size for that channel group.

The BSR reporting mechanism [26] is regulated by 3 timers, unless a triggering event occurs:

- periodic timer, to define the periodicity of the BSR message;
- retransmission timer, to define the maximum interval to wait before the retransmission of the BSR;
- logical channel scheduling request timer (optional), to define the delay the UE must wait before sending a new scheduling request after it has data to transmit.

In the simulations, the periodicity of BSR is set to 20 ms by the attribute `ReportBufferStatusTimer` in the `LteRlcAm` class, unless no data are available on the LCGs. In addition, every time a RLC PDU is transmitted, the BSR is reported. In this way, a sufficient granularity in the updates to the gNB, then acquired by the RL agent, is ensured.

2.5.4 Performance Evaluation

The performances of the proposed realistic solutions are evaluated though a simulation campaign in ns-3, where the centralized RL agent learns dynamically how to maximize the reward from the metrics perceived at the RAN. The reward takes into account both the QoS and the QoE. The analysis starts from the results achieved by the different state definitions in terms of (average) reward, E2E maximum delay and actions probability. In RL a dataset for training and then testing the performance of the algorithm is not available. So, the agent with the original state is considered as the baseline for the following evaluation of the results.

Simulation campaign structure The simulation campaign is made up of 800 episodes, where each episode is an independent simulation of nearly 25 s in ns-3. Inside every episode, 500 acquisitions of the RAN state are performed by the agent every 50 ms, exactly the maximum tolerated E2E delay in the TD scenario. As a consequence, the agent has to decide 500 actions about the compression mode to adopt.

RL algorithm and parameters The DDQL algorithm is implemented to solve the RL problem. An extensive description of this RL algorithm is provided in Ch. 4.

Since the centralized approach is considered, all the state observations related to the vehicle(s) are given in input to a single agent installed in the RAN. The discount factor λ is set to 0.95 so that the agent takes into account the previous history. DDQL stores the learning transitions (state, action, reward, next state) in an Experience Replay (ER) buffer with a capacity B = 80000, when it is full, the learning phase can start. In the DDQL algorithm, the weights of the Target Network are replaced every 8000 steps with the weights of the Q Network. During the training of the agent, a batch-approach is exploited, with a batch size set to 32.

A Fully Connected Neural Network (FCNN) is employed to approximate the Q-function. The input layer contains a number of neurons equal to the state dimension, the input is normalized using a batch normalization layer, then it goes through the hidden part, reaching the output layer that contains a number of neurons equal to the possible actions, i.e., the 9 compression modes. The structure of the hidden layers is the following:

1. fully connected layer with 64 neurons and ReLu activation function, followed by a batch normalization layer;
2. fully connected layer with 128 neurons and ReLu activation function, followed by a batch normalization layer;
3. fully connected layer with 64 neurons and ReLu activation function, followed by a batch normalization layer;
4. fully connected layer with 16 neurons and linear activation function.

The weights of neural network are updated using the Adam algorithm, with a learning rate $\alpha = 10^{-4}$.

An ϵ-greedy strategy with a linear decay factor along the learning steps is adopted to deal with the exploitation-exploration dilemma. Specifically, the epsilon factor decays linearly from 1.0 to 0.01 in 400 episodes.

Results Two scenarios are considered: the first involves a single vehicle, while second scenario involves a larger number of vehicles, set to 3. Increasing the number of vehicles in the simulations, the congestion of the network becomes higher, this makes more challenging the satisfaction of the KPIs.

1 VEHICLE

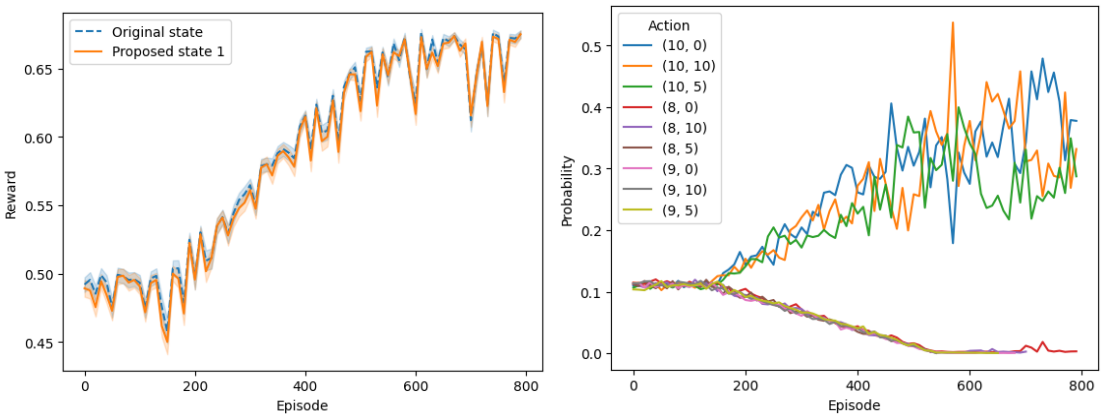
Considering only 1 vehicle, the congestion of the network is very low. In this way, a fair comparison of the training performance can be presented on the basis of the proposed state definitions. First of all, the learning of the RL agent involving a realistic collection of the SINR of the vehicle is described. Then, the state is modified replacing the SINR value with the CQI value.

Proposed State 1 In the Proposed State 1, the idealized exchange of the SINR is removed. Specifically, the realistic exchange and the reporting policy are implemented. As far as the policy is concerned, an outage threshold under which the SINR is not transmitted to the gNB is fixed to 0 dB (in this case, the agent collects a default value, e.g., the same outage threshold), together with a similarity threshold of 2 dB to not exchange too frequently the SINR value and an update timer of 300 ms to support a maximum of 6 state acquisitions by the agent without a SINR update. For the overhead related to the mechanism of exchanging the SINR, 1 OFDM symbol in UL inside the transmission slot is reserved. In addition, the TBS and the BLER are gathered as physical layer parameters.

Furthermore, all features (delay and PRR) related to the RLC and PDCP layers are removed. Only the buffer status reporting mechanism is exploited.

For completeness, the following metrics are extracted and then gathered by the `SendStatusUpdate()` method in the `MmWaveEnbNetDevice` class to define the state.

- SINR
- MCS



(a) Average Reward (b) Action probability

Figure 2.5: Results training of centralized RL agent with Proposed State 1 vs. baseline (1 vehicle)

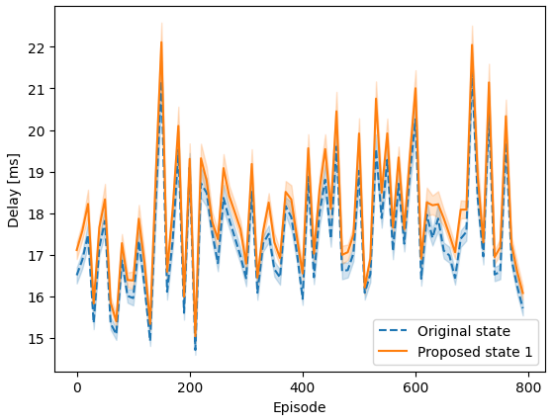


Figure 2.6: Average E2E Delay from the training with Proposed State 1 vs. baseline (1 vehicle)

- OFDM symbols
- TBS
- BLER
- BSR
- APP PRR
- Average, standard deviation, minimum and maximum of the APP delay

The results achieved by the RL agent during the training phase (Fig. 2.5) are very impressive: the reward with this proposed state is very similar to the

reward of the baseline solution in which the state metrics are assumed to be available at the RAN with no additional overhead. The new reward is slightly smaller because of two main reasons: the new realistic exchange mechanism of the SINR and the quantized estimate of the UL volume of data reported by the BSR. In this context, the realistic policy leads to some moments in which the SINR value is not up-to-date, or even not reported in the case of an outage.

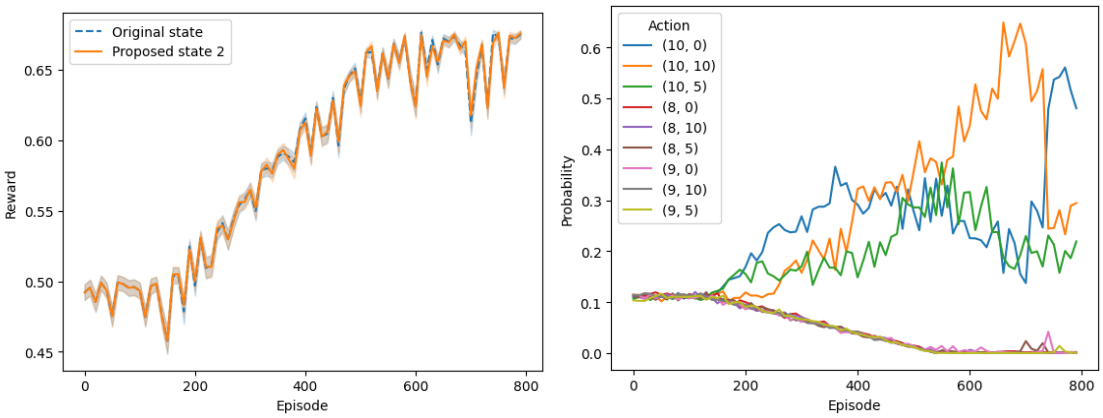
At the end of the training phase, when ϵ tends to zero, the average reward converges to a value of 0.65. As far as concern the action probability (Fig. 2.5b), after an initial exploration, the agent prefers to choose the more conservative compression modes, whose reward is the highest. They are less aggressive and offer better performance in terms of mAP (i.e., QoE) at the remote driver. This behavior happens because the agent learns that the network scenario is not congested and so it can transmit without a too aggressive compression, satisfying the TD requirements anyway.

This is a great result since the proposed state contains a smaller number of metrics related to the network status, especially at the PDCP and RLC layers, but the performance achieved is almost identical to the original configuration. In this context, the idealization of the SINR exchange is removed.

The realism and frequency of the updates of the SINR at the gNB, then acquired by the agent, are ensured by the proposed reporting policy. Furthermore, an impact in the UL communication channel is present due to the overhead introduced by the additional control symbols required to report the SINR at the RAN. The effect can be noticed in Fig. 2.6, where the E2E delay of the networks is slightly higher than the one obtained with the original state, reaching a maximum increment of 0.5 ms.

Proposed State 2 In this proposal, the state is identical to the Proposed State 1, unless the replacement of the SINR value with the CQI. The aim is to understand how the learning performs using a preexisting mechanism in the 5G protocol stack. There is no additional overhead in terms of control since the CQI is already provided by the CSI mechanism, but, on the other side, the CQI is a discrete value mapped from the SINR. For completeness, the following proposal of state definition is evaluated.

- CQI
- MCS



(a) Average Reward (b) Action probability

Figure 2.7: Results training of centralized RL agent with Proposed State 2 vs. baseline (1 vehicle)

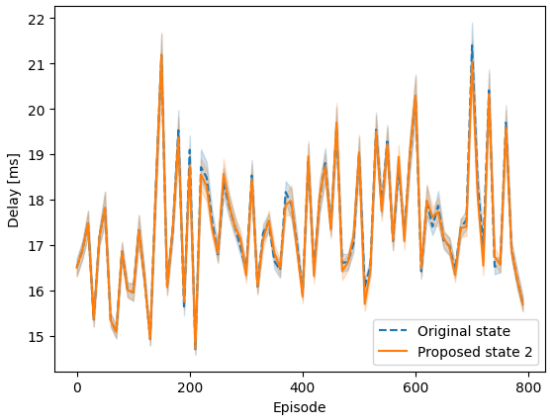


Figure 2.8: Average E2E from the training with Proposed State 2 vs. baseline (1 vehicle)

- OFDM symbols
- TBS
- BLER
- BSR
- APP PRR
- Average, standard deviation, minimum and maximum of the APP delay

The results achieved with the Proposed State 2 (Fig. 2.7) are basically aligned with the previous results. The reward is comparable with the reward of the base-

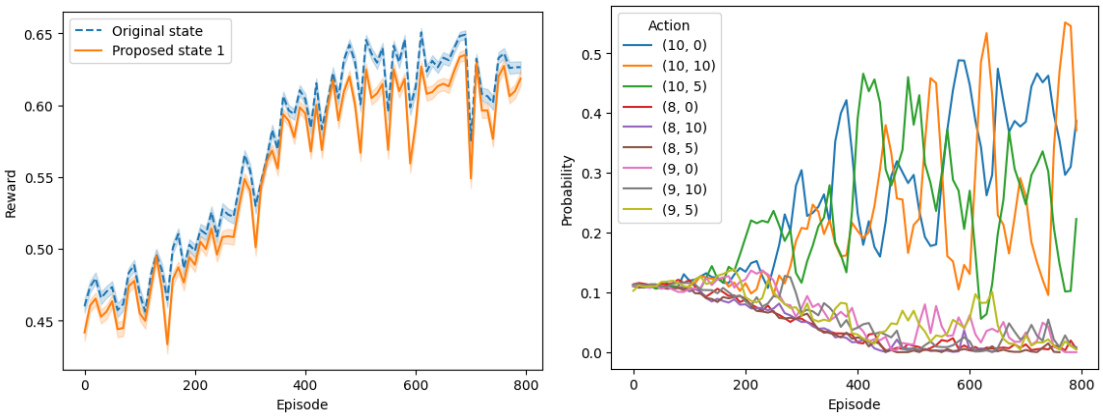
line. In general, the reward gained with the CQI is lower than the reward considering the idealized SINR in the state. This happens because CQI is a quantized and less granular representation of the channel quality, leading to slightly less precise optimization. However, considering the delay obtained with the previous state, the new delay with CQI improves since it avoids the overhead associated with direct SINR reporting mechanism. By using existing CQI reports, the learning system reduces the need for additional signaling, so the performance in terms of delay are improved. This can be seen comparing Figures 2.8-2.6.

As far as concern the action probability (Fig. 2.7b), the situation at the end of the learning is identical to the previous state: the actions keeping more information in the point clouds are preferred. This means that the replacement of SINR with CQI has no impact on the learning of the RL agent and relative action preference.

3 VEHICLES

Considering 3 vehicles, the congestion of the network increases. As a consequence, different choices in the compression mode selection can be explored in order to keep the E2E delay below the strict constraints of the TD.

Only the results achieved by the Proposed State 1 are presented, since the results of Proposed State 2 are very similar.



(a) Average Reward

(b) Action probability

Figure 2.9: Results training of centralized RL agent with Proposed State 1 vs. baseline (3 vehicles)

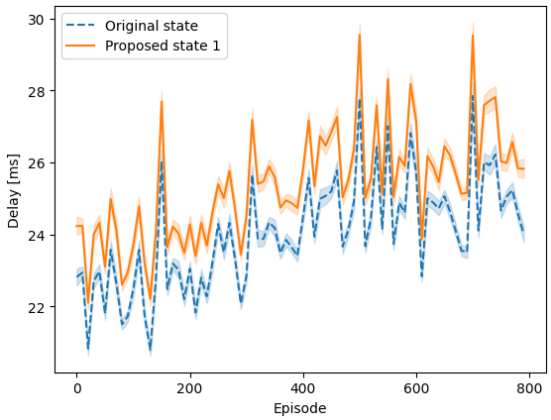


Figure 2.10: Average E2E Delay from the training with Proposed State 1 vs. baseline (3 vehicles)

First of all, the reward (Fig. 2.9a) of both the baseline and the proposed solution with 3 vehicles is smaller than the case with just 1 vehicle. This is reasonable because the congestion of the network is higher, as a consequence the E2E delay goes above the maximum tolerated latency in some steps during the learning. Therefore, in this cases, the reward mechanism is defined to penalize the agent, such that actions with a more aggressive compression mode are less frequently adopted. Fewer data is transmitted by the vehicles if the network status is too overloaded.

The effects of the network congestion can be noticed also in the plot of the delay in Fig. 2.10, where the average value of the metric is above the value obtained with 1 vehicle. In addition, the action probability plot (Fig. 2.9b) shows some peaks corresponding to most aggressive compression modes during the learning. These are the moments in which the network is not able to satisfy the TD requirements. However, in general, the predominant actions remain the more conservative ones, since the network is not extremely congested with only 3 vehicles.

As far as the comparison with the baseline configuration is concerned, the behavior of the agent is similar to the case of 1 vehicle. The realistic exchange of the SINR and the generated overhead contribute to decrease the reward and to increase the overall delay, as illustrated in Fig. 2.9. In this case, the delay is up to 2-3 ms higher than in the ideal baseline scheme since more data is transmitted by the vehicles, so the impact of the additional control symbol is much greater than the previous scenario.

2.6 Distributed Approach

The simulated vehicular scenario considered in the distributed learning approach is identical to the centralized case: the network topology, the channel and the applications are unchanged. The new aspect is the replacement of the centralized RAN-AI with the distributed UE-AI [10] entity, which is the key element enabling the decentralized (distributed and federated) PQoS .

2.6.1 UE-AI

The UE-AI entity can be defined as a distributed controller of the network. There is a UE-AI entity on each vehicle, which is able to collect network metrics gathered only from a single vehicle, i.e, the one where it is installed. The objective of the UE-AI is to take optimal decisions to ensure PQoS, optimizing the V2X network status. In order to compute this task, the RL framework is exploited, as in the centralized case.

The major difference with respect to the centralized framework is the presence of a multi-agent RL problem: a RL agent for each UE-AI, so for each vehicle, is present in this learning scenario and all the agents cooperate to provide PQoS. This means that each RL agent deals only with metrics coming from the local vehicle, with a limited perception of the overall network status. The learning problem becomes more challenging, also for the availability of metrics in a distributed context (e.g., E2E delay is not directly computable at the vehicle).

The flow of the operations for each UE-AI is articulated in following steps.

1. Collection of the network measurements from the UE, related to the local vehicle
2. Creation of the state representation from the gathered metrics for the RL agent
3. Estimation of PQoS countermeasure by the agent
4. Sharing of the actions to the local vehicle

The class `UeAI` implements the UE-AI functionalities, so an instance of that class must be included in the UE class (`MmWaveUeNetDevice`). Each UE entity has a UE-AI instance. The following methods of the UE enables the UE-AI to operate.

1. `InstallUeAI()`
2. `SendStatusUpdate()`
3. `ReportMeasures()`
4. `NotifyActionIdeal()`

The relative descriptions are avoided because they are very similar to RAN-AI entity. The main difference is that each UE employs these methods to take distributed PQoS actions, instead of the gNB.

2.6.2 RL FRAMEWORK

The RL framework is identical to the centralized RL framework described in Sec. 2.5.2. A brief recap of the state, action and reward definitions is reported below.

- Original State

  The state is a vector $\mathbf{s} \in R^{18}$ of following measurements from the entire protocol stack of a user (i.e., vehicle), related to the UL direction.

  PHY layer

  - SINR;
  - MCS;
  - OFDM symbols used.

  RLC and PDCP layers

  - Average, standard deviation, minimum and maximum delay of PDUs;
  - PRR.

APP layer

- Average, standard deviation, minimum and maximum delay of bursts;
- PRR.

- Action

  9 possible choices for the PQoS countermeasure to adopt, i.e., the compression mode. All the combinations of DRACO parameters $q \in \{8, 9, 10\}$ and $c \in \{0, 5, 10\}$.

- Reward

  $R_t(s, a)$ is defined as:

$$R_t(s, a) = \begin{cases} m_a & \text{if } \delta_t \leq \delta_M \\ -\frac{\delta_t}{100} & \text{otherwise} \end{cases} \tag{2.7}$$

2.6.3 Proposed Solutions and Implementation

The original state presents several limitations and assumptions too strong in a realistic distributed scenario.

PHY layer The state does not capture the TBS, an important metric to express the physical channel condition together with SINR, MCS and OFDM symbols used. It represents the amount of bytes that can be transmitted over a TTI in the channel. It is also linked to the modulation scheme exploited for the transmission, depending on the channel conditions. As far as the implementation is concerned, inside the `mmwave` module, the `RxPacketTraceUeCallback()` method in the `MmWaveUeNetDevice` class is modified to store over time (between two consecutive state acquisition) TBS value every time a TB is received by the PHY layer at the UE.

RLC, PDCP and APP layers In the original state definition, delays and PRR at RLC, PDCP and APP layers are included. However, these metrics are gathered in an idealized way, because they cannot be computed autonomously by the

vehicle, without interaction with the gNB. Indeed, calculating the delay requires to know (at least) the timestamp of transmission (available at UE) and reception (not available at UE). Similarly, the PRR requires to know the number of packets transmitted by the vehicle (available at UE) and packets received at the gNB (not available at UE).

Consequently, being realistic with the distributed approach means that each vehicle can only exploit metrics that are locally available to it, or that a mechanism to exchange E2E metrics must be modelled, together with relative overhead.

To solve the problem of calculating the delay at the application layer, which is a key metric also for the learning of the agent since it is part of the actual definition of the reward, a mechanisms of acknowledgments (ACKs) can be implemented in the simulation. The ACK is sent to the vehicle as soon as the entire transmitted point cloud is received by the gNB. This system is easy to implement at the application layer. As alternative, instead of the precise value of the application delay, a rough estimate can be calculated starting from the knowledge of transmitted bytes and physical layer metrics, i.e., SINR or MCS. In this case, the definition of the reward must be redesigned on the basis of this new estimate.

On the contrary, it is more difficult to build ACKs at intermediate layers (RLC and PDCP) of the 5G protocol stack to compute the delays. For this reason, the overall state of the transmission buffers can be monitored to understand the network status at these layers, instead of computing the precise delay. It is reasonable to think that in a network that is not extremely congested or in extreme poor conditions, the delay at the RLC layer is a function of the amount of data inside the buffers.

ACK AT APP LAYER

A mechanism to transmit ACKs is triggered as soon the gNB receives all the fragments of a point cloud from that vehicle, sending back a timestamp. Only when the ACK is received, the agent can acquire the network state, including the application delay, and take the PQoS countermeasure. More precisely, a timestamp can be problematic if the two systems have not an aligned clock, for this reason the E2E delay computed by the gNB (remote host) can be directly shared. Indeed, the gNB knows the slot when the UE starts transmitting, since it decides the scheduling of the resources, and the moment in which the point

cloud is completely received. In the simulated scenario, the traffic generated by a vehicle is related only to the point clouds transmission.

In the simulation, a `UdpClient` instance is installed in the remote host node and a `PacketSink` instance in each vehicle node: every time the remote host receives a point cloud, the `BurstRx` trace of the `BurstyApplication` is triggered and the UDP client sends a small packet with size of 40 bytes (8 bytes for the UDP header, plus 32 bytes of data) to the vehicle. This packet contains average, standard deviation, minimum and maximum of the perceived application delay; the last three statistics are not essential, but they are useful in case a point cloud is retransmitted. Only when the packet is received by the vehicle, i.e., the `RxWithAddresses` trace of the `PacketSink` is called, the status update is triggered and the metrics are shared to the RL agent.

Another improvement to this mechanism can be done on the target of the ACKs. Indeed, since the distributed scenario makes vehicles not directly aware of the overall networks status, the delay can be transmitted to all the vehicles inside the simulation, not only to the one that is interested, i.e., the generator of the point cloud. Broadcasting the ACKs can be useful in the multi-agent learning scenario. The impact should be not so heavy because of the small size of the ACK packets.

As far as concern the estimation of the application delay, the number of transmitted bytes and the MCS are key metrics that can be exploited in this process. The delay is directly proportional to the size of the transmitted data, instead it is inversely proportional to the MCS. Indeed, the MCS is able to express the channel quality (as the SINR), but at the same time provides a more direct link with the current data rate, passing through the modulation scheme. A strategy consists in computing experimentally an optimal threshold to estimate a parallelism between the ratio $\frac{tx\ bytes}{MCS}$ and a delay greater than the maximum tolerated one.

RLC BUFFERS

RLC buffers represent a very interesting source of information related to the network status: their state can be acquired by the RL agent in the place of the delay at this layer.

From the official model library of the ns-3 LTE module, the AM RLC entity installed on the UE exploits 3 buffers for the transmit operations.

- Transmission Buffer: it is the queue where the AM RLC entity receives and stores SDUs from the upper PDCP entity.

- Transmitted PDUs Buffer: it is the queue of transmitted RLC PDUs for which an ACK/NACK has not been received yet. When the AM RLC entity sends a PDU to the MAC entity, it push a duplicate of the transmitted PDU in the Transmitted PDUs Buffer.

- Retransmission Buffer: it is the queue of RLC PDUs to be retransmitted (previously NACKed). The AM RLC entity moves this PDU to the Retransmission Buffer, when it retransmits a PDU from the Transmitted Buffer.

In the simulation, a punctual mechanism to track the volume of data inside Transmission Buffer and Transmitted PDUs Buffer is implemented. The Retransmission Buffer is not considered because several statistics representing the channel quality are already present in the state (e.g., the SINR).

Every time that a PDU is transmitted by the RLC entity, with the traces properly configured in ns-3, the `MmWaveBearerStatsCalculator` entity keeps track of the size of data inside the two buffers by monitoring the `m_txPdu` callback. When the state is acquired, average, standard deviation, minimum and maximum of the load of Transmission Buffer and Transmitted PDUs Buffer are computed.

2.6.4 Performance Evaluation

The performance of the proposed realistic solutions are evaluated though a simulation campaign in ns-3, where the distributed RL multi-agent learns dynamically how to maximize the reward from the metrics perceived only locally at the vehicle. The reward takes into account both the QoS and the QoE. The analysis starts from the results achieved by the different state definitions in terms of (average) reward, E2E delay and action probability. As in the centralized case, the agent with the original state is considered as the baseline for the following evaluation of the results.

Simulation campaign structure The simulation campaign is made up of 800 episodes, where each episode is an independent simulation of nearly 31 s in ns-3. Inside every episode, 800 acquisitions of the local metrics are performed per vehicle (UE) by the correspondent agent, every time a point cloud is received

at the remote host. As a consequence, the agent has to decide 800 actions per vehicle about the compression mode to adopt.

The number of acquisitions of the state is larger than the centralized case because of the reduced perception of the network status by a single vehicle, so more inputs data are given to the agent during a single episode.

RL algorithm and parameters The same RL algorithms and the same parameters of the centralized case are considered in order to keep fairness in the learning. Differently from the centralized case, there is an independent RL agent for each vehicle, capturing only local state acquisitions. In this multi-agent RL scenario, each vehicle has an independent DDQL network to feed with local metrics.

Results Only a single scenario with 3 vehicles is considered for the distributed approach. The objective is the exploration of the learning performance in case of a multi-agent RL problem, focusing on the limits of the distributed case, related to the metrics availability and to the sharing of the networks status.

3 VEHICLES

First of all, the learning of the agent involving the RLC buffers state and the exchange of ACKs at the application layer is described. Then, the learning is modified replacing the real application delay with an estimate. The reward function changes accordingly.

Proposed State 1 In the Proposed State 1, the TBS is added to the state definition, among the physical layer parameters.

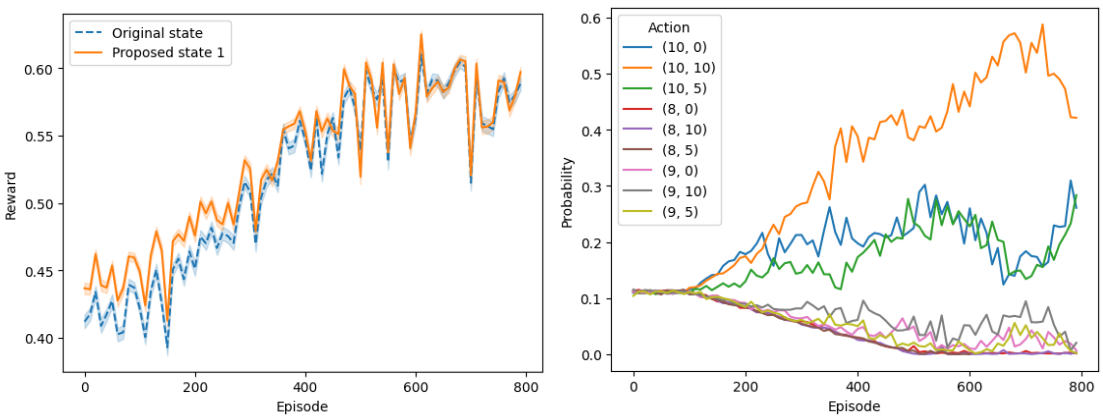
The metrics related to the RLC and PDCP layers are removed, specifically the delay statistics and the PRR. They are replaced with the available statistics related to the data volumes inside the Transmission Buffer and the Transmitted PDUs Buffer, to consider a more realistic state.

Furthermore, the application E2E delay remains in the state, but the mechanism to exchange ACKs is modelled inside the simulation. More precisely, the ACK to compute the delay is broadcasted to all the vehicles in the simulation, such that this information is shared by the whole network. In this context, the number of transmitted bursts and bytes from the APP layer are inserted into the state definition related to the APP layer, since they are freely available at the ve-

hicle. The APP PRR is removed due to the unavailability of metrics perceived at the remote host.

For completeness, the following metrics are extracted and then gathered by the `SendStatusUpdate()` method in the `MmWaveUeNetDevice` class to define the state.

- SINR
- MCS
- OFDM symbols
- TBS
- Average, standard deviation, minimum and maximum of the Transmission Buffer size
- Average, standard deviation, minimum and maximum of the Transmitted PDUs Buffer size
- APP transmitted bursts
- APP transmitted bytes
- Average, standard deviation, minimum and maximum of the APP delay



(a) Average Reward (b) Action probability

Figure 2.11: Results training of distributed RL multi-agent with Proposed State 1 vs. baseline (3 vehicles)

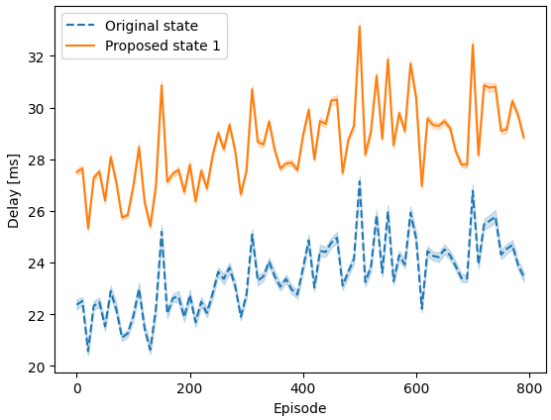


Figure 2.12: Average E2E Delay from the distributed training with Proposed State 1 vs. baseline (3 vehicles)

The results achieved by the proposed state configuration are very promising. The reward (see Fig. 2.11a) gained by the RL agent is perfectly aligned with the reward of the ideal baseline approach: at the end of the training, they both reach an approximate average reward of 0.6. The main difference is that the proposed solution does not exploit the punctual delays and reception rates of the RLC and PDCP layers, but it monitors the buffers status.

Notice that the proposed realistic network state outperforms the ideal baseline at the beginning of the training. This faster convergence can be due to the awareness of each vehicle of the overall network status, thanks to the sharing of application delays via the ACKs.

The removal of the idealization of the application delay is quite expensive in terms of overall network E2E delay, as shown in Fig. 2.12. The increment with respect to the baseline is of 4 ms, on average. Indeed, even if the ACK mechanism contributes to a realistic learning procedure and a knowledge sharing between the agents on the vehicles, from the other side it creates a higher load in the network. Calling n the number of vehicles, there is the transmission of n UDP packets every time a point cloud is received by the remote host, and there are on average n receptions of point clouds every 33.333 ms (generation interval). The time required for transmitting and processing these control packets generates congestion in the point clouds exchange.

The impact on the delay performance is for sure linked to the ACKs exchange, since the action probability plot (Fig. 2.11b) shows clearly that the less aggressive compression modes (highest QoE) continue to be preferred, as in the baseline solution where the delay is much smaller.

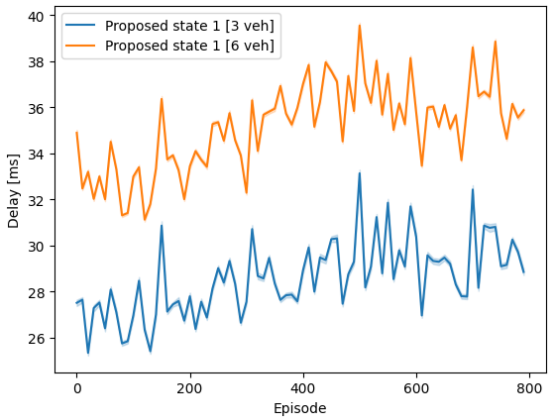


Figure 2.13: Average E2E Delay from the distributed training with Proposed State 1 with 3 vehicles vs. 6 vehicles

Increasing the number of vehicles, the overhead generated by this mechanism in terms of E2E delay becomes intolerable. In Fig. 2.13, the average delay with 3 and 6 vehicles is shown. 40 ms is the maximum value reached considering 6 vehicles, it is much larger than the case with 3 vehicles.

Proposed State 2 The Proposed State 2 is identical to the first proposal, but the statistics related to the application delay are removed from the state. Indeed, the aim is to explore a new learning procedure, which involves metrics that can be gathered only locally from the vehicle. The exchange of parameters with other entities is not possible: a fully distributed scenario is simulated. For completeness, the following proposal of state definition is evaluated.

- SINR
- MCS
- OFDM symbols
- TBS
- Average, standard deviation, minimum and maximum of the Transmission Buffer size
- Average, standard deviation, minimum and maximum of the Transmitted PDUs Buffer size

- APP transmitted bursts
- APP transmitted bytes

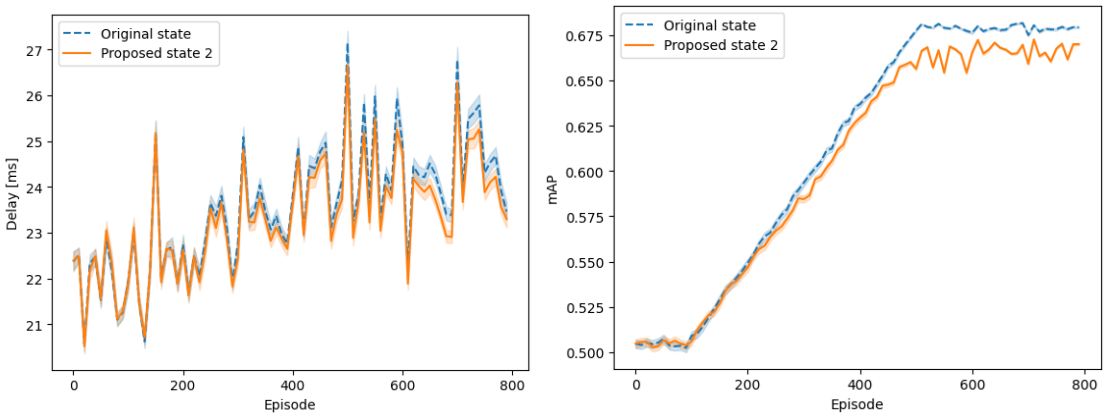
As mentioned in Sec. 2.6.3, to be consistent with state modification, also the reward function must be redesigned. Since the application is not available, the ratio between transmitted bytes and MCS is used to decide if the KPIs are satisfied or not.

Thus, $R_t(s, a)$ can be redefined as:

$$R_t(s,a) = \begin{cases} m_a & \text{if } \frac{tx\ bytes}{MCS} \leq \gamma \\ -\frac{\frac{tx\ bytes}{MCS}}{10000} & \text{otherwise} \end{cases} \tag{2.8}$$

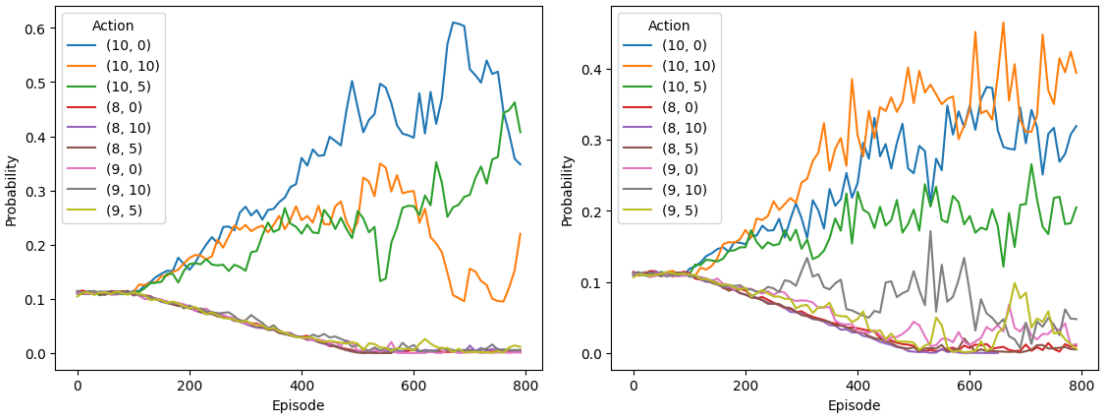
Finding the optimal value for threshold γ is done by simulation: several simulations are executed in an idealized distributed scenario where the application delay is known. A statistical analysis is performed on the values of the measure $\frac{tx\ bytes}{MCS}$, only when the delay perceived is larger than the maximum tolerated delay, fixed to 50 ms from the TD specifications.

The tuning of threshold γ is a critical aspect: a too small value could penalize heavily the transmission of point clouds with conservative compression modes, conversely, with a too large value, the congestion of the network is not considered, generating a weak penalization when QoS constraints are not satisfied.



(a) Average E2E Delay (b) Average mAP

Figure 2.14: Results training of distributed RL multi-agent with Proposed State 2 vs. baseline (3 vehicles)



(a) Original state

(b) Proposed state 2

Figure 2.15: Action probability of RL agent with Proposed State 2 vs. baseline (3 vehicles)

By simulating different scenarios and performing a weighted mean of $\frac{tx\ bytes}{MCS}$ values (filtering out the outliers) gathered from ns-3 simulator, the optimal threshold is γ = 10000 with 3 vehicles. Indeed, the performances in terms of delay, action probability and transmitted bytes are very similar to the original case (see Figures 2.14, 2.15), with the important difference that in this case the delay is not considered by the reward function. More precisely, in the long run, the E2E delay and the amount of transmitted APP bytes is slightly smaller than the values obtained with the original state. This reduction can be mainly attributed to the action selection mechanism that the RL agent has learned with the new state representation and reward function, which demonstrates a larger probability for selecting more aggressive compression modes.

This situation reflects the more conservative behavior of the RL agent. The agent prioritizes improvements in the QoS, by reducing the transmitted bytes, consequently, the latency and the network load, ensuring that the system can handle higher demands with more users. However, this conservative approach has a trade-off: it generates a small degradation in the overall QoE with respect to the original configuration. This is due to the lower mAP achieved (Fig. 2.14b), which indicates that the system's final accuracy in the object detection decreases on average. Instead, in the idealized case, where the delay is considered, the RL agent learns better how to optimize the transmission of less compressed point clouds, without generating critical congestion. The delay is slightly larger, but also the QoE is improved because less compressive modes are still preferred.

3

Federated PQoS

In this chapter, the federated learning approach for PQoS is explored in detail. This approach belongs to the decentralized scenario, as the distributed one, involving only metrics that are gathered locally by the RL agent in the vehicle or that are shared with it. The decisions of the agent are taken directly onboard.

In order to speed up the convergence to an optimal policy, in the federated approach, all the vehicles periodically share their own model parameters with a central server (e.g, in the gNB or at remote host). The aim is training collaboratively a global model aggregating the local models (at the vehicles) in an iterative way. Moreover, the availability of a global model combining the knowledge from different heterogeneous sources contributes to generate a more robust generalization of the multi-agent policy.

This approach can be considered as an hybrid solution between centralized and distributed cases, since local metrics are adopted by the agent and there are updates to share the local knowledge. As a consequence, there is a limited impact on the latency from the model exchange and agent's convergence is faster. Furthermore, as in the distributed case, since actions are computed locally, privacy of the users is ensured.

From the point of view of the simulation, the federated approach is almost identical to the distributed learning approach. In both scenarios, the UE-AI entity is a key element to enable the learning procedure.

In general, the federated learning approach offers a robust solution for decentralized PQoS, collaborative learning, preserving data privacy and accelerating the model training with parameter sharing between the vehicles.

3.1 Problem Formulation

The general federated learning problem [29] involves multiple decentralized entities (e.g., mobile devices, vehicles, sensors), which build a single and global model in a collaborative way. These entities are called clients, and there is a central server that coordinates them.

However, local data of the clients is sensitive, so it cannot be shared with others or with the central server. As a consequence, each client trains and computes locally an update to the current global model in the server, and only this update is publicly shared with the server. The server manages the training process, aggregating model updates from clients to build a global model.

To formalize the problem, the goal is the minimization of the following objective function [29], where w is the vector with parameters of the local model, n is the number of clients, F_k is the local objective function (e.g., loss function) of the client k and p_k is the importance of the client k:

$$\min_{w} F(w), \quad \text{with} \quad F(w) := \sum_{k=1}^{n} p_k F_k(w). \tag{3.1}$$

The state-of-the-art method for the federated learning is Federated Averaging (FedAvg) [30]. The FedAvg requires that each client computes single (or multiple) iterations of stochastic gradient descent (SGD) on the current model using its own local dataset.

$$\forall k, \quad w_{t+1}^{k} \leftarrow w_t - \eta g_k \tag{3.2}$$

Then, the server aggregates the actual parameters of each client doing a weight average.

$$w_{t+1} \leftarrow \sum_{k=1}^{K} p_k w_{t+1}^{k} \tag{3.3}$$

In this way, the global model is updated and shared again with all the clients. The procedure is iteratively repeated.

Another important algorithm is the Federated SGD (FedSGD). It involves the direct exchange of the gradient computed by each client on the local data. In the following phase, the server aggregates all these gradients and applies the update to the global model. This method is more expensive than FedAvg in

terms of network load, because, in general, gradients of each local model must be transmitted at every iteration [31]. In FedAvg, the frequency of exchange can be every few iterations.

When clients are homogeneous, i.e. Independent and Identically Distributed (IID) assumptions are valid for local data, the FedAvg reduces to the FedSGD [30].

The objective of this chapter is the improvement of the federated learning, following different strategies in order to optimize the performance of the RL agent.

- Adoption of adaptive federated optimization methods to aggregate the parameters: FedAdagrad, FedYogi, FedAdam
- Use of compression techniques on the model parameters to transmit: pruning, quantization and clustering
- Assignment of different utilities/priorities to the clients during the aggregation step, according to several definitions

3.2 Simulation Framework

In the simulation campaigns the network scenario remains unchanged with respect to Chapter 2.

A decentralized learning framework is enabled by the UE-AI entity installed on each vehicle. The number of vehicles involved in the simulations is set to 3, such that it is possible to evaluate concretely the impact of different strategies in the model aggregation, without creating an extreme congestion where the QoS could potentially alter the learning performance.

In this context, the assumption of knowing the application delay is considered, because a two-way communication with the central server is already present for the federated update. It is reasonable that the computed delay is sent to the vehicles together with the updated global model.

Federated Update The main difference with the distributed case is the implementation of the federated update to exchange and aggregate model parameters. The federated update requires that each vehicle has performed an action (i.e., a

learning step) before the sharing of parameters. If this is not the case, vehicles continue the learning until each vehicle has done at least one PQoS action. Only at this moment the federated update is triggered. The unbalance of learning steps per vehicle must be keep into account during the aggregation.

Inside the simulator, a mechanism to model the federated update is implemented. A `UdpClient` and `PacketSink` are installed both at the remote host and at each vehicle. For achieving a realistic behavior, each vehicle must share its model every time that a local learning step (i.e., action) is performed, because it cannot know whether or not other vehicles have already taken one action. It is not aware of the overall status. From the other side, when the server (sink) at the remote host has received at least one packet with model parameters from every vehicle, it is able to compute the global model and send back the new parameters to all the agents. The number of packets (i.e., actions) needed for a federated update is a tunable parameter in the simulation.

The exchange of federated updates starts only when RL agents have filled the ER buffer. Indeed, sufficient experience tuples (state, action, reward) must be collected by interacting with the environment to ensure that the RL agents have a representative and complete set of data for starting the learning. In addition, the agents have time to understand better the dynamic of the environment and share a meaningful local model, generating a more effective and stable federated learning.

The baseline approach used to evaluate the proposed solutions involves the transmission of all model parameters and the importance of each client represented by the fraction of learning steps (actions) done by the client over the total number of steps (actions) of all the vehicles since the last federated update. At the beginning, traditional FedAvg is applied as federated algorithm.

Inside the DDQL algorithm, a FCNN with a simpler architecture is considered with respect to Chapter 2. The new network contains 2409 parameters between weights and biases. The size of this network model, assuming a floating point of 32 bits data type, corresponds to 9636 bytes.

3.3 Adaptive Federated Optimization Methods

The main idea behind adaptive federated optimization methods [32] is the introduction of an adaptive gradient-based optimizer at the server side, in order to optimize the global model. This approach differs from the SGD optimizer

applied in FedAvg (in an IID context). Roughly speaking, the clients have to minimize the federated objective function exploiting (Eq. 3.1) their local data, instead the server optimizes it from a global perspective.

In [32], the following adaptive optimization methods are implemented:

- Adagrad in FedAdagrad;
- Adam in FedAdam;
- Yogi in FedYogi.

A brief description of each adaptive optimizer is provided.

Adagrad Adagrad [33] is an optimized version of the SGD, where the learning rate is automatically decayed to control the variance of the gradient updates and ensure convergence. Unlike SGD, which uses a unique global learning rate for all parameters, Adagrad employs a different learning rate for each dimension of the parameter space, based on the cumulative sum of squared gradients. This property makes Adagrad very effective in sparse settings, but with poor performance in non-convex and dense settings [34]. The accumulation of squared gradients can result in a too aggressive decay of the learning rate, becoming extremely small and interrupting the learning procedure.

Adam Adam [35] combines the advantages of two optimizers: Adagrad and RMSProp. Adam is an exponential moving average based adaptive method, meaning that gradients are scaled per dimension by square roots of exponential moving averages of squared past gradients. In this way, the learning rate tuning depends only on the recent gradients, addressing the problem of rapid decay of learning rate in Adagrad.

More precisely, exponential moving averages of the gradient and of the squared gradient are computed and updated according to two hyperparameters, β_1 and $\beta_2 \in [0, 1)$, which control the correspondent exponential decay rates. The moving averages are estimates of the mean (first moment estimation) and of the uncentered variance (second moment estimation) of the gradient.

Yogi Yogi [34], as Adam, uses an adaptive gradient, keeping a controlled learning rate decay. Adam exploits the exponential moving average technique, that is by definition a multiplicative approach. While this method can effectively adapt

Algorithm 1 Server's federated update with adaptive optimizers

Initialization: $w_0, \beta_1, \beta_2 \in [0, 1), \eta$ (learning rate), τ (adaptivity)
for step $t = 0, \dots, T-1$ **do**
  for each vehicle $v \in \mathcal{V}$ in parallel **do**
    Update w_v using Adam optimizer
    $\Delta_v^t = w_v^t - w_t$
  end for
  $\Delta_t = \sum_{v \in \mathcal{V}} p_v^t \Delta_i^t$
  $m_t = \beta_1 m_{t-1} + (1 - \beta_1)\Delta_t$
  $v_t = \begin{cases} v_{t-1} + \Delta_t^2 & \text{FedAdagrad} \\ v_{t-1} - (1-\beta_2)\Delta_t^2 \text{sign}(v_{t-1} - \Delta_t^2) & \text{FedYogi} \\ \beta_2 v_{t-1} + (1-\beta_2)\Delta_t^2 & \text{FedAdam} \end{cases}$
  $w_{t+1} = w_t + \eta \frac{m_t}{\sqrt{v_t} + \tau}$
end for

learning rates based on recent gradients, it can also lead to a rapid vanishing of past gradients, especially in sparse settings. The alternative approach exploited by Yogi involves simple additive updates.

Differently from Adam's second moment estimation, Yogi updates the second moment considering the direction of the gradient. Specifically, the use of the sign function ensures that the variance is updated in a controlled fashion, preventing it from growing or decaying too rapidly.

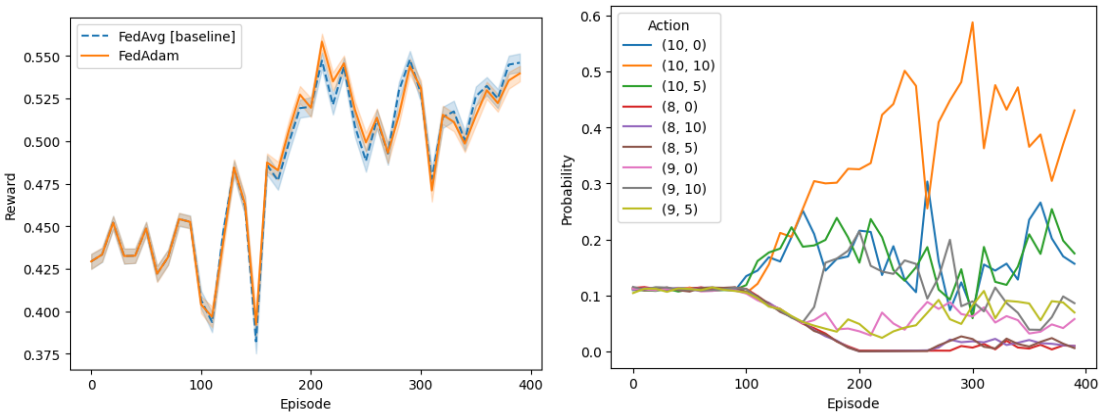
3.3.1 Implementation and Results

At the server side, the adaptive optimizers are implemented in the aggregation of local models from the RL agents at the vehicles. In this V2X scenario, vehicles represent the clients and the gNB/remote host represents the server.

The implementation of the server's federated update to compute the global model is reported in Alg. 1. The algorithm considers jointly 3 adaptive federated optimization methods: FedAdagrad, FedAdam and FedYogi. Their names derive from the considered optimizer.

An offline training is executed to evaluate the optimizer with best performance. FedAdam has reached the highest performance in terms of reward and delay, therefore, only its results are presented. A learning rate η of 10^{-4} is considered, together with $\beta_1 = 0.9$, $\beta_2 = 0.99$ and $\tau = 10^{-9}$.

FedAdagrad and FedYogi performs similarly, also with respect to FedAdam.



(a) Average Reward (b) Action probability

Figure 3.1: Results training of federated RL agent with FedAdam vs. FedAvg (baseline)

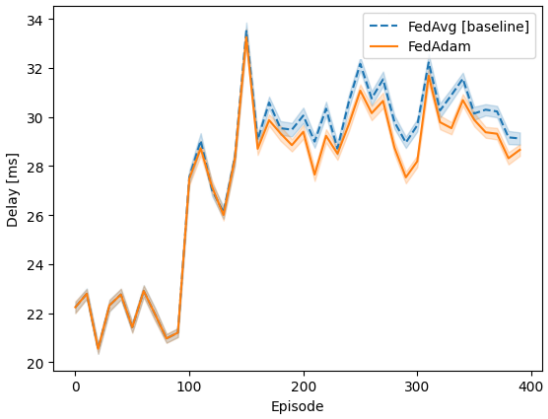


Figure 3.2: Average E2E Delay from the federated training with FedAdam vs. FedAdvg (baseline)

The main problem is that QoS becomes more degraded: the E2E delay increases by a few milliseconds. That may be due to the fact that a different direction is followed during the parameters optimization, thus, a new learning evolution skewed toward QoE is globally generated. The choices of RL agent reflect a huge preference for more conservative compression modes, so more aggressive approaches are rarely applied. The reward should be very high, but considering the congestion created by RL choices, together with the impact of model parameters exchange, a reduction of the reward and an increment of the latency happen. However, the objective is to minimize the QoS degradation due to the additional traffic in the network generated by federated updates, while keeping a good value of QoE.

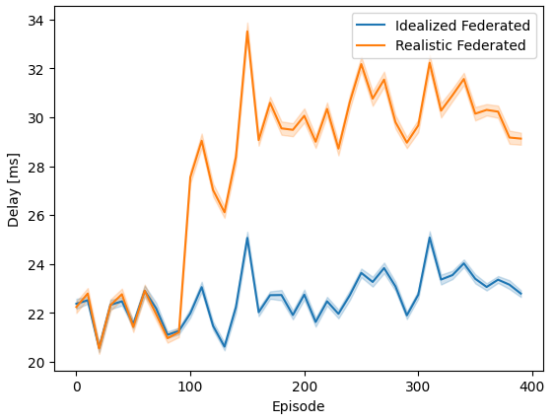


Figure 3.3: Average E2E Delay of the federated training with idealized vs. realistic exchange of federated updates

The reward achieved by FedAdam (Fig. 3.1a) is quite aligned to the reward of FedAvg. The main benefit of using Adam optimizer in the federated update is that the computed global model of RL agent optimizes the overall delay of the network. The new model takes into account the overhead due to parameters exchange and the overall QoS in a more significant way. Indeed, in certain steps, more aggressive compression is selected to reduce the network congestion over time. In addition, among the more conservative compression modes maximizing the QoE, the one with the highest compression level ($c = 10$) is chosen. Plots of delay and action probability are reported in Figures 3.2 and 3.1b.

3.4 Model compression

The federated update generates congestion in the network because model parameters are frequently exchanged between vehicles and remote server. Indeed, at each learning step, every RL agent must share the weights of its model with all the other agents. This periodic communication can be very disruptive, especially in environments with high vehicular density.

The impact of sharing federated updates on the E2E delay can be notice in Fig. 3.3. Obviously, the idealized transmission does not generate any effect on the delay, reaching a value near to 24 ms, as in the distributed case. Conversely, the curve of the simulation involving a realistic transmission mechanism shows an increment of nearly 6 ms, due to the additional traffic in the network.

In this context, model compression techniques represent a valuable strategy

in order to reduce, at the source, the amount of data injected in the network during federated updates. However, as in every compression context, a trade-off between accuracy and compression rate must be considered. High compression rates create significant loss of information, which can degrade the performance and accuracy of the RL models. On the other side, lower compression rates keep more information but they don't sufficiently reduce network congestion.

There are several methods to achieve compression of a NN model [36]. In this section, three compression techniques are applied to the weights of a RL agent model.

- Pruning
- Post-Training Quantization
- Clustering

3.4.1 Pruning

Pruning is a powerful compression technique to reduce the size and the complexity of NN models. Especially, when dense networks are employed, such as FCNNs.

More precisely, the unstructured pruning consists in removing weights that are not essential for the model. This means that some connections between neurons are selectively dropped, keeping ones that most affect the performance. In this way, the original dense network is transformed into a sparser network with fewer parameters to share.

Several criteria can be considered to perform the pruning. For example:

- magnitude-based pruning [37] [38], where weights with the smallest absolute values are removed, since their small contribution to the output;
- sensitivity-based pruning [39], where the removing procedure evaluates the contribution of each weight to the overall model accuracy.

In this work, a post-training magnitude-based approach is applied. Indeed, more complex techniques to estimate the importance are quite expensive in terms of time and computation, introducing additional overhead that can limit the benefits of pruning.

Pruning is applied two times in the federated step: first of all, before the sharing of every model from vehicles to the server (UL transmission), then, after

the aggregation at the server, before the global model is resend back to the RL agents (DL tranmission).

Magnitude-based pruning The magnitude-based pruning requires ranking the weights based on their absolute values and removing those with the smallest magnitudes. The reason behind is that weights with smaller absolute values have a lower contribution on the overall performance of the model and can be safely pruned without significantly affecting accuracy.

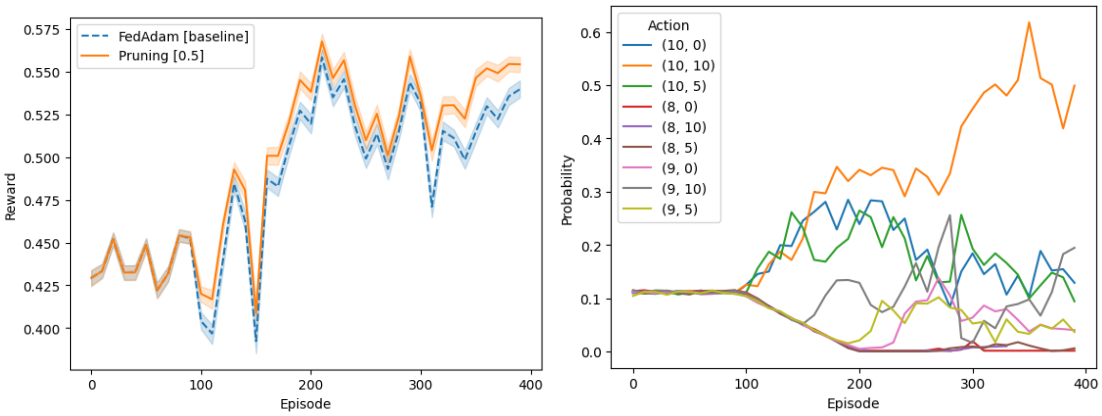
In this context, the fraction of pruned weights depends on the specified sparsity ratio. This ratio can be either static or dynamic. In the first case, the amount of pruning is kept fixed during the learning, instead, a dynamic sparsity ratio changes over training episodes. This dynamism can be useful for very large and dense networks, where a gently reduction of connections helps to not degrade brutally the model performance at the beginning.

For completeness, the following steps are involved in the implementation:

1. a static/dynamic sparsity ratio is received in input;
2. computation of number of weights to preserve;
3. ranking of weights based on the absolute values;
4. computation of the threshold under which weights are pruned;
5. setting to 0 all pruned weights;
6. sharing of the pruned model in the federated update.

Packet Structure As far as the exchange of the pruned model is concern, it is not possible to create trivially a packet containing only the fraction of kept parameters in the payload. Indeed, the receiver does not know the received values to which model parameters correspond to. That missing correspondence between parameters must be transmitted. Otherwise, the receiving entity is not able to reconstruct locally the shared model.

To solve this problem, a new header is created and encapsulated into the packet. This header, generated by the transmitting entity, contains a binary bitmap indicating for each parameters whether (bit 1) or not (bit 0) it is present in the packet. In other words, if that parameter is shared or not, due to the pruning.



(a) Average Reward (b) Action probability

Figure 3.4: Results training of federated RL multi-agent with FedAdam and Pruning (0.5) vs. FedAdam (baseline)

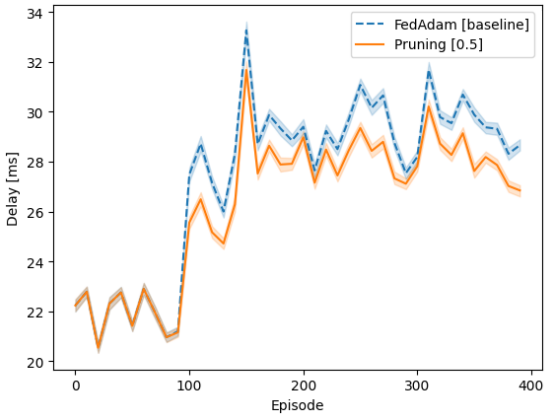


Figure 3.5: Average E2E Delay from the federated training with FedAdam and Pruning (0.5) vs. FedAdam (baseline)

Results

An offline training is executed to evaluate the sparsity ratio with best performance. It is an hyperparameter to tune in order to balance the trade-off between accuracy and compression rate. The optimal sparsity ratio is set to 0.5.

The results (see Figures 3.4 and 3.5) achieved with FedAdam algorithm combined with pruning technique are very good, considering that only half of the weights are preserved in the RL agent. The new reward is larger than the baseline approach. The main reason is that the halved size of the packets containing federated updates contributes to reducing the overall traffic in the network, consequently the congestion decreases. Negative rewards associated to violations of

TD performance indicators (i.e., delay) become less frequent since the reduction of network congestion.

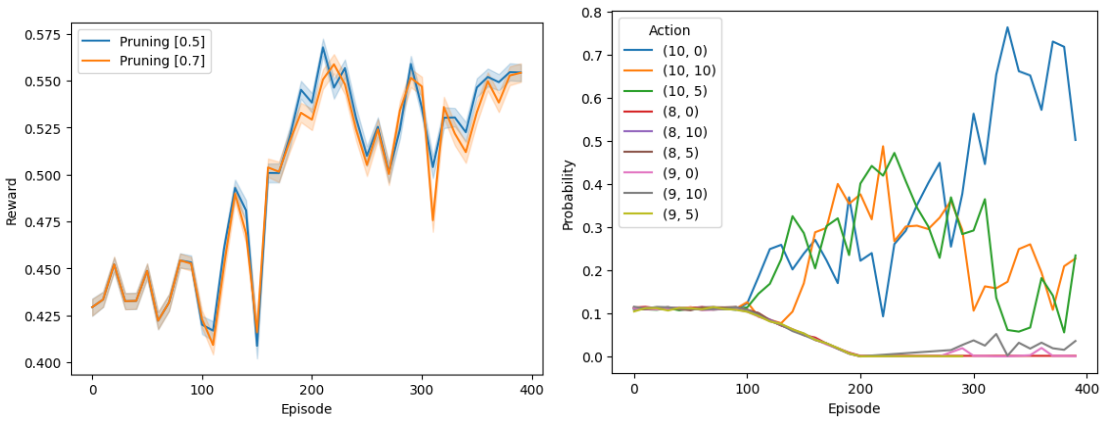
Furthermore, this fact is shown also by the action probability plot in Fig. 3.4b, where most compressive modes are practically never considered by the RL agent. Instead, their probability is higher with full-size packets (Fig. 3.1b). Indeed, there is a network status which requires in some conditions aggressive actions to achieve the expected QoS because of the major impact of federated step packets on the data traffic.

In this context, among the less compressive modes, the one with the highest compression level ($c = 10$) is preferred anyway. This because there is still an important overhead, even if it is halved, for the federated update.

In parallel, a good learning happens even if half of the model parameters are removed. So two observations can be done:

- the contribution of smaller parameters is negligible with respect to the model output
- the frequent sharing of model parameters, in general at every local step, contributes to limit the divergence of the agent during learning, even if much less parameters are present in the model.

As far as the E2E delay is concerned, an average value of 28 ms is reached. A decrement of 2 ms with respect to the baseline learning approach with full-size federated updates can be noticed.



(a) Average Reward (b) Action probability

Figure 3.6: Results training of federated RL multi-agent with FedAdam and Pruning with sparsity ratio 0.5 vs. 0.7

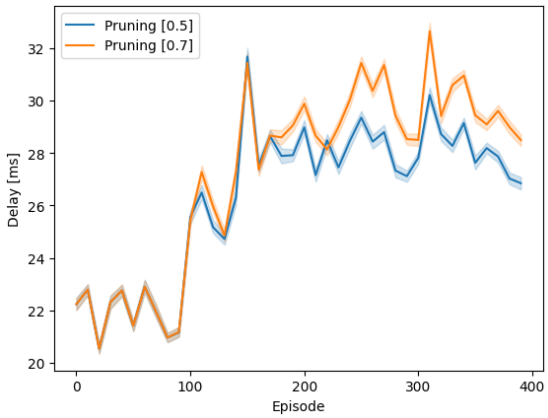


Figure 3.7: Average E2E Delay from the federated training with FedAdam and Pruning with sparsity ratio 0.5 vs. 0.7

Finally, a brief comparison with a sparsity ratio of 0.7 is presented. It seems reasonable that if more parameters to share are removed from the model, less overhead is generated by the federated step and the final delay decreases. However, the overall performance is not improved with respect to the sparsity ratio of 0.5, as shown by the reward in Fig. 3.6a. In this case, the level of network resources exploited to exchange the model is lower, while the latency is increased by 7% (Fig. 3.7) because the least compressive mode ($q = 10$, $c = 0$) is preferred during the learning for its highest reward (Fig. 3.6b). The new pruned model promotes the QoE, increasing the average value of mAP from 0.65 to 0.68, but, on the other side, several violations of TD constraint on the E2E delay (QoS) happen and are the main reason of inferior results.

3.4.2 Post-Training Quantization

Quantization technique is another powerful strategy to achieve the reduction of federated step impact. A very aggressive quantization can reduce heavily the amount of data generated by the sharing of model parameters.

Two types of quantization exist: post-training quantization and quantization-aware training [40]. The first approach applies the quantization after that the model has been trained; the second approach trains the model in full precision simulating the effect of quantization on the weights and activation functions in the forward pass, allowing it to learn how to mitigate the quantization errors, and only after the training the model is quantized. In this way, the accuracy is increased but more time and computation are required.

Focusing on post-training quantization, a first naive approach is to select a less-representative data type for storing the trained parameters. For example, instead of a floating point of 32 bits (double precision), a floating point of 16 bits (reduced precision) or an integer with 8 bits (or even smaller) can be employed. However, despite the large gain in terms of size required by the model, a too smaller precision of the weights can significantly compromise the learning procedure.

Instead of exploiting a conservative quantization of the model parameters, an approach with an aggressive quantization of the updates of model parameters can introduce several benefits.

At the beginning, the server shares a default global model, e.g., randomly generated. Each vehicle, after a local training step is performed, computes the difference of its local model parameters with respect to the current global model parameters, i.e., the update of the global model generated by that vehicle.

Since the server knows the previous global model, it is able to compute the current clients parameters only receiving the updates from the vehicles. Sharing all the parameters is not necessary. This can be very useful because a more aggressive quantization can be applied to the updates. Indeed, in general, the maximum values of updates are smaller compared to the parameters, so the quantization error is kept limited.

Furthermore, the sharing of updates is typically done at each learning step, so the any quantization error introduced has limited time to affect the overall training procedure. Subsequent updates can correct or mitigate the errors from previous steps. Also, the federated updates are often averaged in the server aggregation, this can for sure help to smooth individual quantization errors, reducing the impact on the global model.

The same procedure can be exploited in DL, for the sharing of current global model towards the agents in the vehicles. Only the difference with respect to the previous global model can be transmitted. Every agent is aware of the previous global parameters. The overall algorithm of the federated update involving quantization is shown in Alg. 2.

Two quantization approaches are exploited: binary quantization and ternary quantization. That are very aggressive schemes.

Binary quantization The binary quantization adopted is proposed in [41].

Algorithm 2 Federated update with quantization

Initialization: global model parameters $w_0 = \bar{w}_0$ (randomly generated), $\Delta\bar{w}_t^0$ (all zeros)
for step $t = 1, \ldots, T$ **do**
  for each vehicle $v \in \mathcal{V}$ in parallel **do**
    $w_t = w_{t-1} + \Delta\bar{w}_q^{t-1}$
    Update w_v using Adam optimizer
    $\Delta w_v^t = w_v^t - w_t$
    $\Delta w_{v,q}^t \leftarrow$ Quantize Δw_v^t
    Transmit to the server $\Delta w_{v,q}^t$
  end for
  $w_v^t = \bar{w}_{t-1} + \Delta w_{v,q}^t$ for each $v \in \mathcal{V}$
  Update $\bar{w}_t$ using FedAdam
  $\Delta\bar{w}_t = \bar{w}_t - \bar{w}_{t-1}$
  $\Delta\bar{w}_q^t \leftarrow$ Quantize $\Delta\bar{w}_t$
  Transmit to each vehicle $\Delta\bar{w}_q^t$
end for

A first phase of sparsification, according to a specified sparsity ratio, is necessary to maintain only the largest and the smallest values of the update, the other values are set to zero. Similarly to the pruning method described previously.

Then, the averages of largest and smallest values are computed.

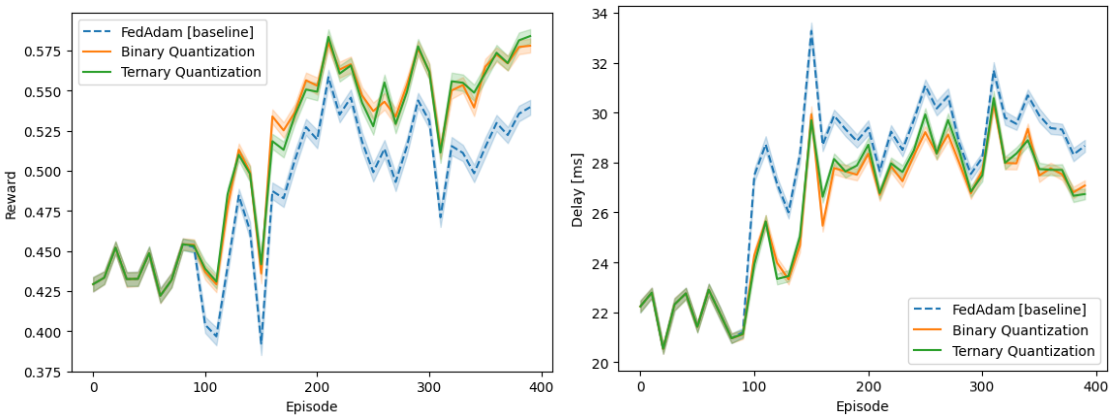
- If the positive mean is greater than the negative mean, positive values are quantized to the positive mean, while negative values are set to 0.
- If the negative mean is greater than the positive mean, negative values are quantized to the negative mean, while positive values are set to 0.

Ternary quantization The ternary quantization adopted is proposed in [42].

A first phase of sparsification, according to a specified sparsity ratio, is necessary to maintain only the values with the largest absolute magnitude. The remaining values are discarded, so they are set to zero.

Then an overall average is computed based on the absolute values of maintained elements.

- The positive elements are quantized to the positive.
- The negative elements are quantized to the negative of the average.



(a) Average Reward

(b) Average E2E Delay

Figure 3.8: Results training of federated RL multi-agent with FedAdam and quantization vs. FedAdam (baseline)

Packet Structure A simple encoding mechanism is created in order to optimize the size of federated updates and have an efficient transmission.

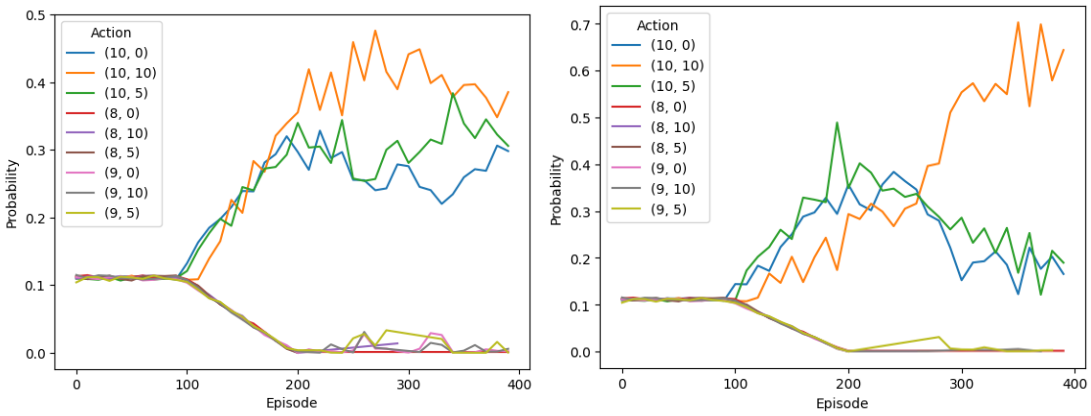
In the case of binary quantization, a packet containing only a single bit for each parameter (in the same order of appearance in the model) is sufficient to describe the result of the quantization, since only two values are produced. Of course, an initial exchange of the non-zero value must take place, which is negligible with respect to all the transmission.

In the case of ternary quantization, a binary bitmap is exploited to indicate which updates are filtered out by the sparsification and which are not, i,e., which values are transmitted by the client. Therefore, a packet containing only a single bit for each non-zero quantized parameter update (in the same order of appearance in the model) is sufficient to describe the two non-zero values produced by the quantization. Of course, also in this case, an initial exchange of non-zero values must take place, which is negligible with respect to all the transmission. In this context, the more aggressive the sparsification, the smaller the number of bits required to transmit the non-zero quantized parameters.

Results

In the quantization algorithms, a sparsity ratio of 0.5 is considered in order to preserve a discrete range of values before the compression.

The results achieved with quantization are very impressive. The reward (Fig. 3.8a) is much higher than the reward achieved with non-quantized federated



(a) Binary quantization

(b) Ternary quantization

Figure 3.9: Action probability of federated RL multi-agent with FedAdam and quantization vs. FedAdam (baseline)

updates, both with binary and ternary quantization. This is mainly due to the very aggressive compression applied to the size of federated update packets. The network status becomes much less congested thanks to lighter overhead for the exchange of model parameters. The delay (Fig. 3.8b) reflects this condition, with a decrement of almost 2 ms with the baseline, reaching an average of 28 ms. The action probability, plotted in Fig. 3.9, shows that only the most conservative compressions are taken by the agents, without never employing the more aggressive modes (unless for RL exploration).

The quantization of updates does not deteriorate the learning performance of RL agents. The reasons are linked to the fact that the updates are shared very frequently and their magnitudes are relatively small due to the low learning rate of the RL algorithm. In addition, the aggregation done at the server contributes to mitigate the quantization error.

Briefly comparing the binary with ternary quantization, the binary approach performs slightly better.

- Binary quantization is able to transmit all the updates requiring basically only a single bit for each model parameter, optimizing both reward and delay metrics. The less aggressive compression modes are chosen (Fig. 3.9a) because they have the largest reward and the network status is able to manage them.

- Ternary quantization requires larger packets for updates in comparison to the binary quantization, this creates a very minimal impact on the re-

ward of RL agents and network E2E delay with respect to the binary case. The main difference can be notice in the action probability plot (Fig. 3.9b): the less aggressive compression modes are preferred anyway, but the one with the highest compression level is taken more frequently in the long run. This behaviour can be explained by the larger size of the federated updates compared to the binary quantization case. More precisely, the size is increased by a factor of 1.5 (302 vs. 451 bytes), considering a sparsity ratio of 0.5. In general, depending on the sparsity ratio, the increment can be up to a factor of 2.

Finally, in comparison to the pruning, the performance is improved because of more aggressive compressions. The QoS contribution (i.e., the delay) on the reward is basically unchanged, but the QoE part increases due to the larger probability of selecting compression modes with an higher mAP.

3.4.3 Clustering

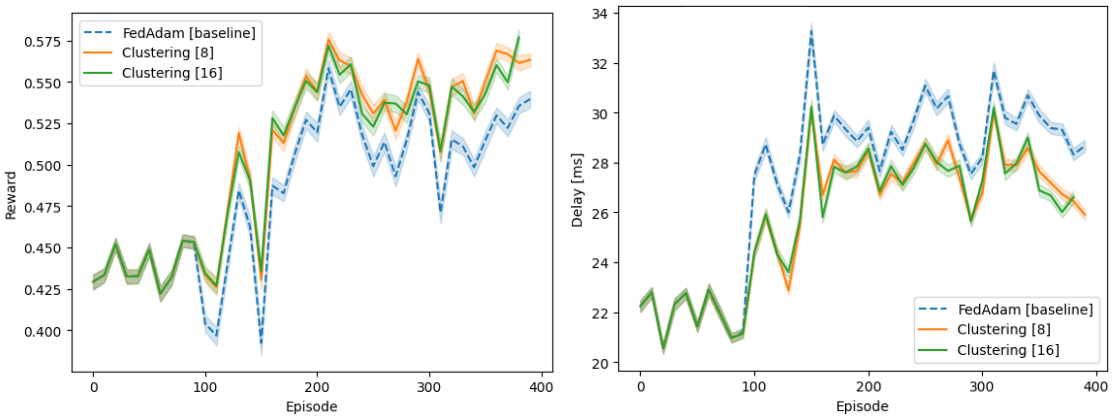
Clustering technique can be very effective in limiting the dimension of federated updates and thus improving the overall network performance.

Clustering is a machine learning technique whose task is grouping a set of objects such that similar objects belong to the same group (i.e., cluster) and dissimilar objects are separated in different groups [43].

Clustering can be applied to compress federated updates. The local update from each agent can be in a certain sense quantized by grouping the contained values into clusters, and then a representative element from each cluster is computed. Each component of the update takes the value of the representative value of cluster it belongs to. In this context, it is convenient to share only the index of the cluster for each value. This can help to reduce the communication overhead and improve the performances of RL agents.

K-means is a suitable clustering algorithm for these compression purposes. Indeed, it automatically computes for every cluster the relative cluster center, called centroid. That centroid is a perfect candidate to represent all the values in the cluster, in an average perspective.

K-means K-means [44], also known as Llyod algorithm, is the simplest distance-based clustering algorithm. It takes in input a fixed number of clusters k to identify and the distance function (typically the Euclidean distance). It



(a) Average Reward

(b) Average E2E Delay

Figure 3.10: Results training of federated RL multi-agent with FedAdam and clustering vs. FedAdam (baseline)

returns cluster centers and relative value-cluster allocation. Clusters are computed in order to minimize the error from the approximation of the values with cluster centroids. An iterative algorithm is performed until convergence, with the following steps:

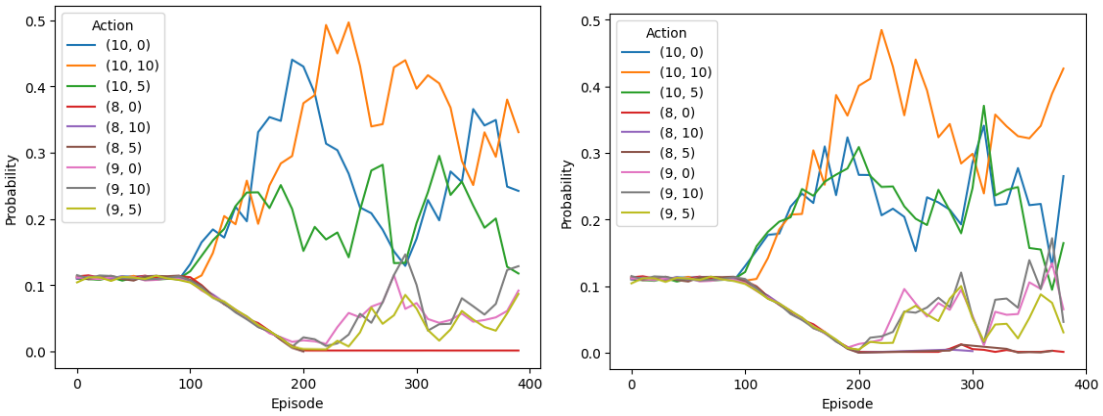
1. cluster centers (centroids) are fixed, each value is assigned to the closest cluster;

2. value-cluster allocation is fixed, new centroids are computed.

The number of clusters is a critical parameter for the performance of the algorithm. Fewer clusters with many values could not be able to capture the variance within the data, resulting in an excessive generalization (underfitting). On the contrary, too many clusters with a smaller amount of data for each cluster could capture a lot of noise rather then the hidden model, resulting in a poor generalization (overfitting). The number of cluster has a direct impact on the trade-off between the data compression and the accuracy of the model updates.

Packet Structure A simple encoding mechanism is created in order to optimize the size of federated updates and have an efficient transmission.

In case of clusters, a packet containing for each parameter the number of bits to represent the index of clusters (i.e. $\lceil \log_2 k \rceil$) is enough to describe the result of clustering, since only cluster centroids are shared in place of values in the update. Of course, an initial exchange of the centroids must happen, that is negligible with respect to all the transmission.



(a) 8 clusters (b) 16 clusters

Figure 3.11: Action probability of federated RL multi-agent with FedAdam and clustering vs. FedAdam (baseline)

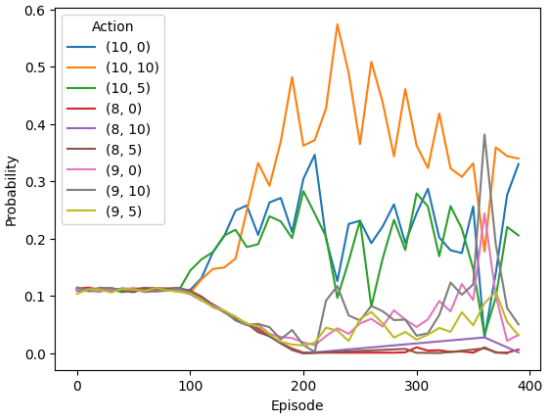


Figure 3.12: Action probability of federated RL multi-agent with FedAdam and clustering (4 clusters)

Results

The results achieved with a compression by clustering of federated updates (Fig. 3.10) are very good. The reward is increased and the delay is reduced with respect to uncompressed updates, thanks to the smaller amount of data injected by the federated step in the network. The reasons are similar to the ones presented in the quantization approach. Indeed, clustering can be considered as a variant of traditional quantization.

Three different numbers of clusters are tested: 4, 8 and 16. Thus, 2, 3, and 4 bits for each parameter are respectively required in transmission phase.

The number of clusters k can be considered as a controller of both compres-

sion and accuracy of the RL learning. With a greater k, values in the updates are represented with more centroids, so clusters are able to identify finer details. On the contrary, with a too aggressive k, larger clusters impact on the granularity of updates, as a consequence the convergence of the aggregated global model.

The effect of k can be notice in the action probability plots. A greater number of clusters (8 and 16 in Fig. 3.11) shows a RL agent preferring less aggressive actions, that is reasonable with a reduced dimension of the updates (i.e., less congestion) in comparison to the uncompressed model. More precisely, a difference can be notice between 8 and 16 clusters. With more clusters, a greater number of bits must be transmitted and aggressive compression modes ($q = 8$ and $q = 9$) are slightly more probable. In addition, among the more conservative actions, the ones with smaller compression level c become less frequent.

Instead, with a smaller number of clusters (4 in Fig. 3.12), so with a smaller size of updates, the agent starts to select mainly more compressive modes at a certain step. It is not the expected behavior in this context. Probably, the compression is too aggressive and the centroids are not well representative of the original punctual values, so the federated update creates divergence in learning with respect to the optimal choice of actions. For that reason, the other results are omitted.

3.5 Utility Functions

The federated aggregation performed at the server averages parameters from each client (i.e., agent in the vehicle) in a weighted manner. In this aggregation, each agent $k \in \{1, ..., n\}$ contributes to the global model according to a certain factor p_k. Consistency is maintained with the formula reported in Eq. 3.1.

The factor is related to agent's importance: it is an estimate of the utility of an agent's local update for the global model. In more practical terms, this utility factor controls the impact of each agent in the federated learning procedure.

Furthermore, the utility can be used to build a priority scheduling mechanism for selecting only specific updates in a more constrained setup.

Several approaches can be exploited for defining agent's utility [45]. In this section, 3 different measurements of utility are explored:

- Sample-based utility;

- Model-based utility;
- System-based utility.

Sample-based utility This measurement considers the amount and the quality of data samples available to agent k for quantifying its utility U_k. Agents with more data or higher-quality samples receive a larger utility.

A simple approach to deal with the quantity of data is considering the number of learning steps N_k performed by agent k since the last federated update.

$$U_k^t = N_k \tag{3.4}$$

A possible approach to deal with the quality of learning samples involves the loss function (averaged over the batch B of input states s_k^t) at training step t of agent k. In this way, larger importance is assigned to the agent that more diverges from the model in that iteration (of parameters θ_k^t).

$$U_k^t = \frac{1}{|B|} \sum_{i \in B} L(\theta_k^t, s_{k,i}^t) \tag{3.5}$$

Model-based utility This measurement considers the relationship between the global model and the local model of agent k for quantifying its utility U_k. Agents whose weights θ_k are more divergent from the global parameters θ_g receive a larger utility.

A possible approach to deal with the divergence between global and local model is performing the computation of the Euclidean distance (L2 norm) between the two sets of parameters. If the local model at time t has a small divergence in comparison to the global model at time $t-1$, i.e., the global model without considering the local update at time t, its impact on the next global model is less relevant.

$$U_k^t = \| \theta_g^{t-1} - \theta_k^t \|_2 = \sqrt{\sum_{i=1}^{|\theta|} (\theta_g^{t-1} - \theta_{k,i}^t)^2} \tag{3.6}$$

System-based utility This measurement considers system-level aspects during the training of agent k for quantifying its utility U_k. Agents whose network conditions are better may be associated with higher utility factors.

A possible approach to consider system utility is taking into account the channel condition of agent k measured by the SINR at time t. SINR has the advantage to be independent from the data transmission, so from the congestion of the network, expressing directly the channel quality. In this context, the network congestion has been already taken into account by the model update through the reward mechanism.

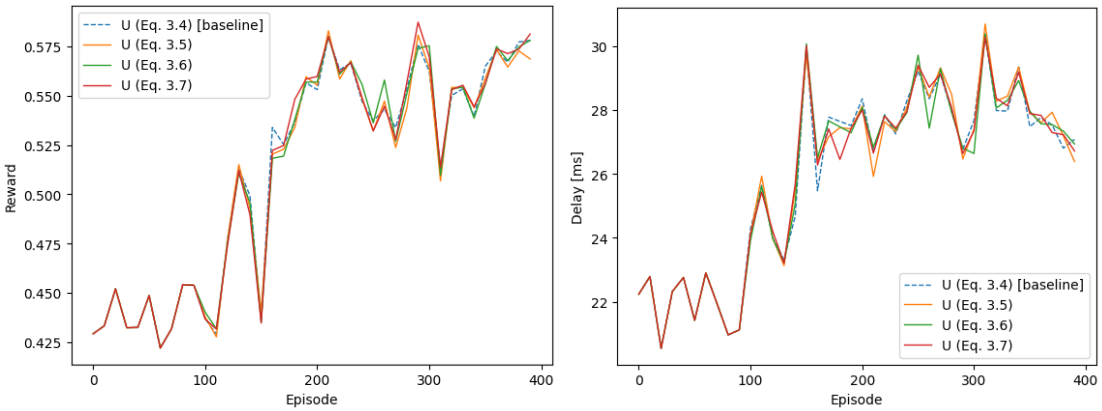
Direct proportionality is exploited in order to reduce latency and ensure higher reliability. Indeed, a higher SINR means a better communication link, with faster data transmission and reduced latency. This results in a prioritized aggregation allowing accelerated convergence of the global model. Furthermore, a higher SINR implies more robust communication with fewer transmission errors. In this way, the server gives greater weight to more accurate updates it receives, since they are less likely to be corrupted, generating a more stable learning.

$$U_k^t = \text{SINR}_k^t \tag{3.7}$$

Inverse proportionality can be also employed to promote fairness among the agents, especially for ones with poorer channel conditions. However, there is a risk of failing to reach convergence in the learning process, as experienced by offline tests.

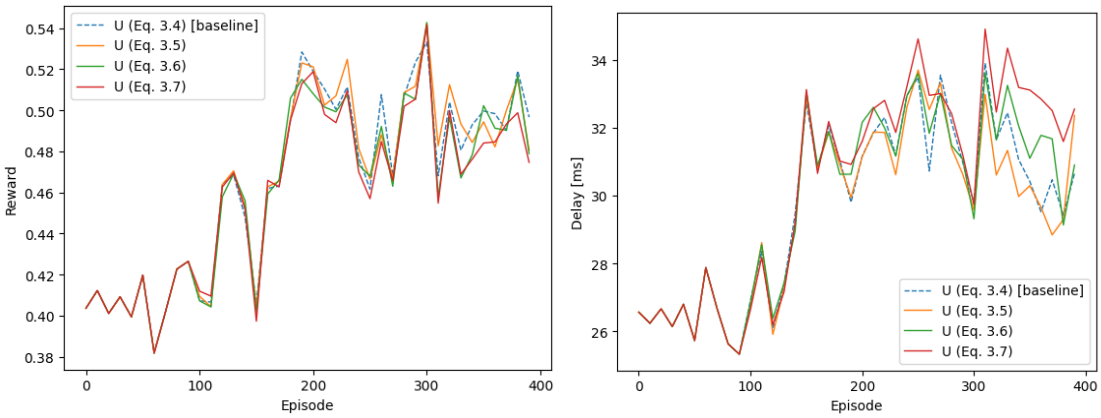
Results

The federated framework used in the evaluation phase involves:



(a) Average Reward (b) Average E2E Delay

Figure 3.13: Results training of federated RL multi-agent with utility functions (3 vehicles)



(a) Average Reward (b) Average E2E Delay

Figure 3.14: Results training of federated RL multi-agent with utility functions (6 vehicles)

- FedAdam optimizer to aggregate updates from the clients;

- binary quantization (sparsity ratio of 0.5) to reduce the overhead of exchanging packets with model parameters.

3 vehicles The results achieved from the learning with several utility definitions are shown in Fig. 3.13. The RL multi-agent performance is quite aligned, without revolutionary changes among the different proposals. In general, reward and delay follow the curves of the baseline approach, which consists in the simplest approach of considering the number of learning steps performed by agent as weight factor.

This fact can be motivated by the quite small number of vehicles in the system: the range of variability of network conditions during the learning is limited. As a consequence, similar federated updates are generated and the impact of different utilities in the aggregation becomes minimal.

Some differences can be noticed in certain episodes where slightly better results are obtained. In these moments, temporary changes in network conditions or in local data distribution (also due to the exploration component) could be happened. These small variations highlight the potential for proposed utility definitions to generate slight advantages in specific situations, especially if more vehicles are considered.

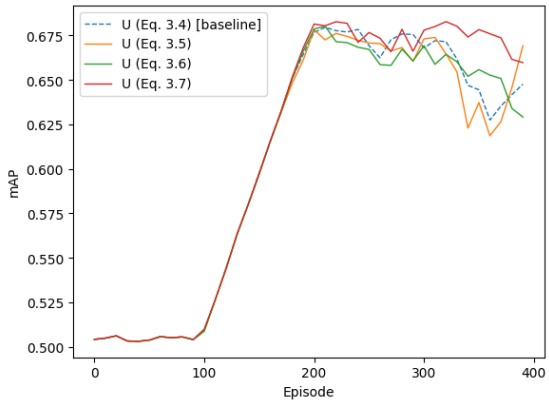


Figure 3.15: Average mAP of the federated RL multi-agent with utility functions (6 vehicles)

6 vehicles The results achieved from the learning with several utility definitions are shown in Fig. 3.14. The RL multi-agent performance is quite heterogeneous among different utility definitions. In general, the increased number of vehicles created appreciable trends during the learning.

Overall, the loss-based (Eq. 3.5) utility presents the best results. In comparison to the baseline, the reward is quite aligned on average, while the delay is reduced in a significant manner (nearly 1 ms, see Fig. 3.14b) towards the last part of the training. This represents a very good result. Indeed, only tuning the weights in the aggregation of federated updates, i.e. scaling the federated steps, agent's behaviour is modified such that some additional compression is performed, without impacting the reward (on average). The reward, on average, remains unchanged because the QoE factor decreases by a small percentage (3%) with respect to the baseline, as it can be noticed in the mean mAP plot in Fig. 3.15. This is the usual trade-off between QoS and QoE: in this TD scenario, especially with a large number of vehicles, reducing the quality of experience is acceptable in order to satisfy strict constraints in terms of QoS.

The other two utility definitions show worst performances. In particular, the SINR-based utility (Eq. 3.7) has a detrimental impact on E2E delay, increasing it up to a maximum of 3 ms with respect to the baseline. This can be explained with the fact that SINR is not able to capture the network congestion, so a prioritization with mismatches between optimal QoS countermeasure based on network status and channel quality is applied.

4

Meta-learning agent for PQoS

Being able to choose the proper learning approach is not a trivial task. The decision between centralized vs. decentralized setups must consider several aspects:

- the quality of the channel;
- the congestion of the network;
- the TD requirements.

It is more convenient to adopt a centralized approach if the channel is good and the network is not congested so that PQoS countermeasures taken by the RL agent take into account the full network state, increasing the accuracy of the decisions.

Conversely, it is more convenient to adopt a decentralized approach if the channel is bad and the network is congested so that the communication overhead for the state aggregation is lower. The federated update can accelerate the overall convergence.

In this chapter, after a comparison between centralized vs. distributed vs. federated approaches, a new meta-learning agent that can dynamically choose when to perform centralized vs. decentralized models is implemented. Starting from the global condition of the vehicles described in terms of network metrics, the optimal learning approach is chosen to maximize at the same time QoS and QoE.

The meta-learning agent controls the learning approach for PQoS at each time step. Two levels of control can be identified in the PQoS framework:

1. PQoS Learning Approach, to optimize the choice between centralization and decentralization;
2. PQoS Countermeasures, to optimize the choice of the compression mode, so as to jointly satisfy QoS and QoE requirements in the TD scenario.

4.1 Centralized vs. Distributed vs. Federated Results

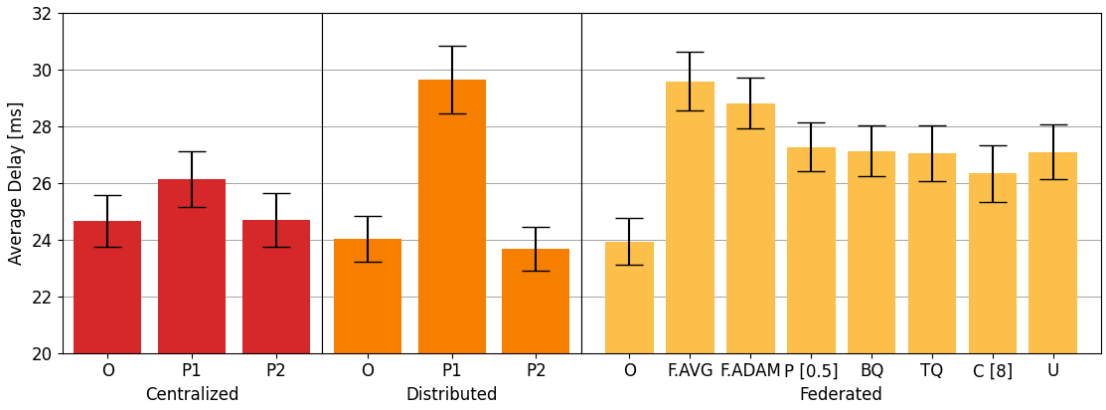


Figure 4.1: Average E2E Delay (QoS) achieved by the proposed configurations in centralized, distributed and federated approaches (3 vehicles) in the last 30 episodes

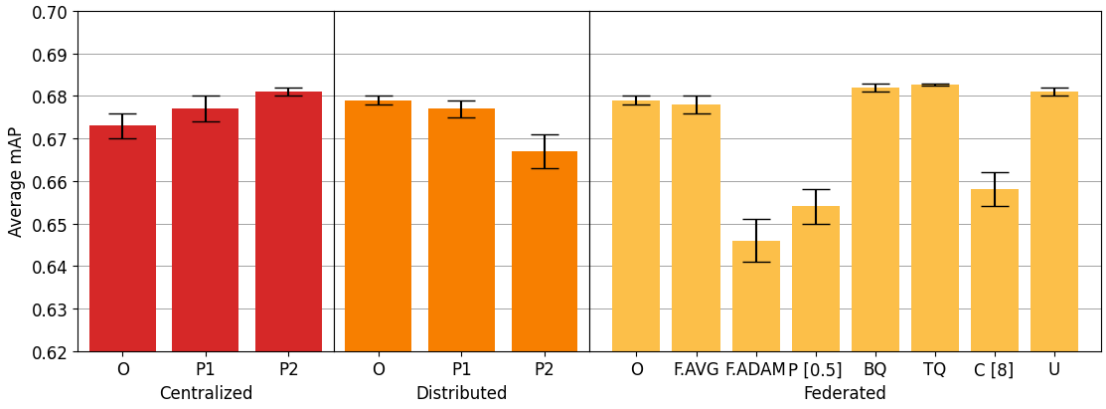


Figure 4.2: Average mAP (QoE) achieved by the proposed configurations in centralized, distributed and federated approaches (3 vehicles) in the last 30 episodes

A comparison between the overall results achieved with centralized, distributed and federated approaches is presented. The several configurations for the PQoS learning framework with 3 vehicles are analyzed in terms of average E2E delay for evaluating the level of QoS, and average mAP for evaluating the QoE. The results are shown in Figures 4.1 and 4.2.

First of all, centralized and fully decentralized (i.e., distributed) scenarios are compared. Considering the same original idealized state (O), where all the metrics exploited by the RL agent are supposed to be available at the agent(s) with no additional overhead, the delay of the distributed approach is 3% smaller than its centralized counterpart because the overhead to communicate with the coordinator entity, i.e., the gNB, is ideally zero. Furthermore, this fact impacts in the average QoE, where the less aggressive compression modes (with higher mAP) are adopted more frequently. However, the reward function defined in Ch. 2 achieves higher values in the centralized case (0.63) vs. the distributed case (0.58), at the end of the learning. This means that the distributed approach receives stronger penalization through the reward mechanism because TD constraints are exceeded in some punctual steps, even if the delay is lower on average, making slower the convergence. The choice of less aggressive compression modes leads to an increment in the QoE but also to an increased network load.

Two state proposals are designed to improve the realism of the simulation.

In the centralized case, the first proposed state (P1) requires the realistic transmission of the SINR perceived at the UE, which results in an increase of the average E2E delay by 6% compared to the baseline ideal approach (O). The new state definition (involving TBS, BLER and BSR) generates a new learning evolution where the average QoE is slightly improved. The second proposal (P2) employs a state identical to P1 unless that CQI reporting mechanism is exploited in place of SINR: the delay is comparable with the original state (O), the average mAP reaches a value of 0.68, slightly larger than the value obtained by P1 because of less congested network status.

On the other side, in the distributed case, the unavailability of E2E delays is solved with the monitoring of RLC buffers and the transmission of ACKs at the application layer to compute the E2E latency. In this way, the first proposed state (P1) generates a very large degradation of the QoS with respect to the O state, while the QoE value remains unchanged. This approach does not scale with

the number of vehicles, for this reason a new state is proposed (P2). P2 does not take into account the delay metric in the learning, but it tries to estimate it, looking at the transmitted bytes from the application layer and the MCS. Very good results are achieved in terms of delay, pretty much similar to the original idealized configuration with a delay of 24 ms. As far as concern the mAP, the value is a bit smaller because the selected compression modes are on average more aggressive: the new reward function is slightly more conservative in terms of amount of transmitted bytes.

The federated approach represents a compromise between the two previous approaches, it is identical to the distributed case during the inference phase but involves a higher cost in terms of communication resources during the training for exchanging the model parameters. In the original idealized state (O), where the federated update is transmitted in a idealized way, the same results of the original distributed state for QoS and QoE are obtained. More precisely, the delay is slightly better (0.1 ms) for the faster convergence to the optimum action (i.e., compression mode) selection.

Then, the exchange of packets for the federated step is implemented and the delay increases by 6 ms, considering the traditional FedAvg (F.AVG) algorithm. Subsequently, adaptive optimizers are exploited to aggregate the global model instead of FedAvg. Better performance is achieved by the FedAdam (F.ADAM), which is able to optimize the overall reward, improving the QoS factor at the expense of the QoE. The delay is decreased by 1 ms, while the average mAP gets worst by 5%.

In this context, the exchange of model parameters creates an additional amount of traffic impacting the network performance, for this reason a method to compress the overhead generated by the federated steps is explored. Pruning over model parameters, quantization and clustering over model updates are the techniques employed in this work and they all reduce the E2E delay in the network between 1.5 and 2 ms. Clustering with 8 clusters (C) reaches the minimum delay of almost 26 ms, but also the mAP is reduced to 0.66. This is a small value if compared to the other approaches, this technique potentially results too much aggressive during the learning. At the same way, pruning (P) achieves a poor mAP because more aggressive compression is needed to counteract the additional network load due to the packets with parameters. Even if the halved size of federated model updates reduces the delay, the federated step is still creating congestion in the network. Therefore, quantization is the best approach to opti-

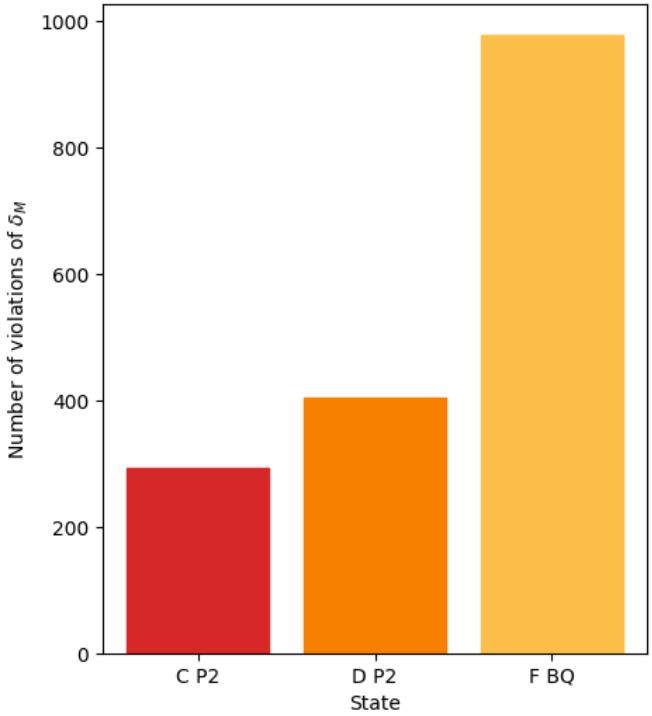


Figure 4.3: Number of violations of the maximum tolerated E2E delay δ_M during the last 10000 learning steps in centralized, distributed and federated approaches (3 vehicles)

mize both the QoS and the QoE. The binary (BQ) and ternary (TQ) quantization reach basically the same delay (27 ms) and mAP (more than 0.68), being able to compress efficiently the updates, without deteriorating the learning procedure. BQ slightly outperforms TQ in terms of QoE because smaller packets (302 vs. 451 bytes) are shared, in this way, the more conservative compression with the highest compression level (smaller mAP with respect to other compression modes with same q) is adopted less frequently in the long run due to the smaller congestion.

In conclusion, a prioritization approach measuring the utility of each vehicle in the federated update is adopted, considering FedAdam and binary quantization. Several definitions are explored: sample-based, model-based and system-based utility. With few vehicles, the performance are quite similar (Fig. 3.13) with respect to the baseline approach with the number of learning steps as weight factor. For completeness, the system-based utility with the SINR (U) is plotted. With more vehicles there are meaningful difference, as discussed in Ch. 3.5. In particular, the sample-based approach involving the loss value is

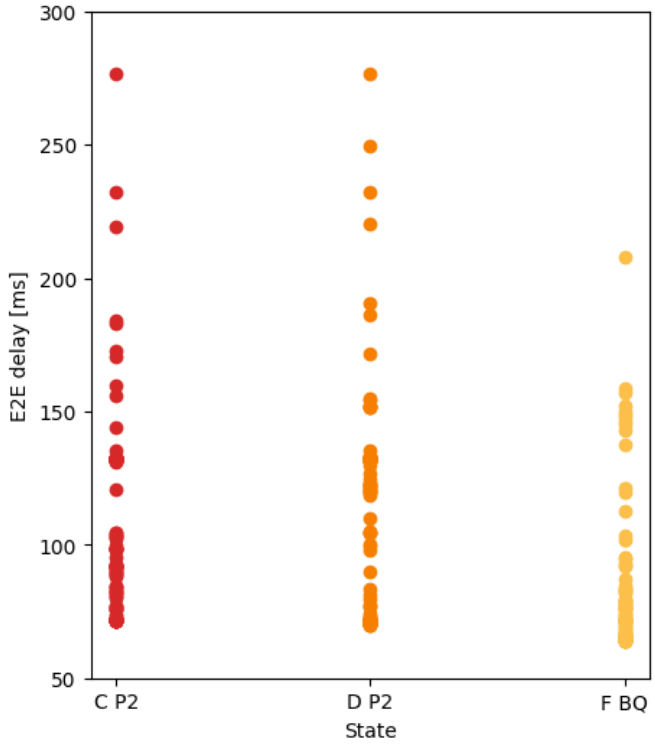


Figure 4.4: Top 1% values of the E2E delay during the last 10000 learning steps in centralized, distributed and federated approaches (3 vehicles)

able to improve the QoS of the communication.

Comparing the best configurations for each approach, i.e., P2 for centralized and distributed solutions, BQ for federated solution, an interesting aspect can be noticed from the delay distribution. Notably, considering the number of violations during the last 10000 learning steps of the maximum tolerated E2E delay δ_M in Fig. 4.3, the distributed approach exceeds more frequently the TD constraint of 50 ms than the centralized approach (400 vs. 300 times), even if the delay is smaller on average (24 vs. 25 ms). This means that the centralization results in an improved global optimization of the QoS, because the full network state is perceived. In contrast, the distributed setup suffers the limited perception of the overall network: the RL multi-agent can exploit only local measurements, being unable to fully converge to an optimal policy for all vehicles. Therefore, there is more penalization by QoS violations from that vehicles with degraded network conditions. Looking at Fig. 4.4, where the top 1% of E2E delay values is illustrated, the magnitude of the violations is comparable between centralized and

distributed approaches. Thus, the number of QoS violations in the distributed scenario is larger, but the spikes reached by the delay are quite aligned to the centralized scenario.

Considering the federated approach, there are many more violations of the maximum tolerable delay because of the congestion generated by the federated updates. However, the magnitude of the largest delay values is smaller than the centralized and distributed solutions, as shown in Fig. 4.4. Thus, the number of QoS violations is quite large, but the values of E2E delay are less critical than the other approaches. This may be due to the accelerated convergence towards an optimal policy selection.

To summarize, in general, the federated approach increases the QoE factor with respect to centralized and distributed cases exploiting a more efficient generalization from the aggregation of local models. On the other hand, this happens at the cost of QoS, because the sharing of model parameters increases the burden on the channel, generating an increment in the E2E delay. A interesting direction to explore is the combination of the federated approach with the centralized one, in order to discover the sweet point between centralization and collaborative decentralization.

4.2 Meta-learning Agent Model

RL techniques are exploited to build the meta-learning agent. A suitable framework involving state, actions and reward function definitions is created. The model is designed to be compatible with the framework of RL *sub-agents*, i.e., the centralized and decentralized agents in charge of deciding for the optimal compression modes (PQoS countermeasures).

4.2.1 State

The discrete state space for n vehicles contains vectors $\mathbf{s} \in R^{11n}$. For each vehicle, the state contains a set of 11 measurements related to the UL direction. For completeness, they are reported in the underlying list.

Considering a step:

- Average E2E latency
- Average reward of the agent selecting PQoS countermeasures

- Frequency of selected actions (i.e., compression modes)

4.2.2 Action

The discrete action space involves 2 possible choices for the PQoS learning approach to adopt: centralized and decentralized framework.

In the centralized case, sub-agents choosing PQoS countermeasures take into account the metrics related to the centralized Proposed State 2.

Instead, in the decentralized case, the federated learning scenario with FedAdam and binary quantization is considered. The accelerated convergence to the optimal policy generates superior performance in comparison with the distributed approach, because of the sharing of the models. In addition, the federated approach can be converted to the distributed approach by turning off the federated updates, in case the channel load increases or the quality deteriorates significantly.

4.2.3 Reward

The reward function at step t, $R_t(s)$, depends on the average rewards $\overline{R}_k^t$ of the RL agent (centralized) or RL multi-agent (decentralized) selecting PQoS countermeasures for vehicle k during step t. In this way, the reward of meta-learning agent is a direct mapping of the reward of RL agent selecting compression modes, which is based on QoS and QoE parameters.

With n vehicles, the formula of $R_t(s)$ is defined as:

$$R_t(s) = \frac{\sum_{k=1}^{n} \overline{R}_k^t}{n} \tag{4.1}$$

4.3 Double Deep Q-Learning Algorithm

A model-free RL algorithm learning from the experience, which follows a value-based strategy and integrates also the deep NNs, is implemented: the DDQL [46] algorithm. The DDQL is a variation of the Deep Q-Learning (DQL) [47], which derives from the traditional effective Q-Learning [17] method, whose objective is finding the optimal Q (state-action) value function. The huge difference is that DQL is not a tabular approach, so it does not require to save in memory a table for each possible state and action: it would be not suitable and

quite problematic for an high-dimensional discrete spaces, both in memory and complexity terms, as in the vehicular scenario. Instead, in DQL the table is replaced by a NN that approximates, according to some parameters (weights) to train, the Q function: it produces in output a value for each action, given in input a certain state.

With respect to the DQL, the adjective Double means that DDQL involves two different NNs:

- Q Network to predict the Q values for the current state (of experience tuple) in input
- Target Network to predict the Q values for the next state (of experience tuple) in input

The two NNs have an identical structure, but the Target network is not updated at each step as the Q Network, but only after some steps they are synchronized, copying the weights of the Q Network in the Target Network. In this way the algorithm ensures that the Target Q values, which are used in the computation of the loss function to optimize, are kept stable for a certain period, so the target does not change continuously decreasing the stability of the learning. After a precise number of steps, the Target network parameters are updated in order to improve the output estimation.

The ER technique is exploited. A circular buffer (queue) is created to collect the experience tuples *(current state, action, reward, next state, end)* gathered by the agent interaction with the environment at each step. This happens because the buffer improves the stability of the learning. In this context, the agent store its experience and learn by sampling at random from collected data, this contributes to breaking the high correlation between sequence of tuples, so the Adam optimizer can work on IID experience. Moreover, the collected samples can be reused during the training, being more data efficient [47].

Another trick present in the implementation is considering an ϵ-greedy strategy to deal with the usual exploration vs exploitation dilemma: the rate of exploration ϵ is updated in a exponentially decreasing manner, so at the beginning the agent will explore a lot in order to discover better action combinations, then the best action is gradually preferred. This trade-off to explore is very important especially in the vehicles scenario, which is characterized by a huge

number of combinations state-action.

4.3.1 Implementation and Parameters

The implementation of the DDQL algorithm is made up of the following steps. B is the batch size, K is an arbitrary positive number.

1. Initialize Q Network (weights $\boldsymbol{\theta}$) and Target (T) Network (weights $\boldsymbol{\theta}'$), the replay buffer, the agent and the environment
2. For every epoch and for every step t:
   (a) Choose action a_t from state s_t according to ϵ-greedy policy
   (b) Agent takes a_t, observe reward r_t and the next state $s_t^{'}$
   (c) Store in the buffer the tuple $(s_t, a_t, r_t, s_t^{'}, end_t)$
   (d) If size of buffer $> K * B$
      i. Sample a minibatch from the buffer
      ii. For any tuple i in the minibatch
         A. If end_i: $y_i = r_i$, else $y_i = r_i + \gamma max_{a'} T(s_i^{'}, a^{'}, \boldsymbol{\theta}')$
         B. Compute the MSE loss $L = \frac{1}{B}\sum_{i=1}^{B}(y_i - Q(s_i, a_i, \boldsymbol{\theta}))^2$
         C. Update $\boldsymbol{\theta}$ using Adam optimizer on L
         D. If $t = UPDATE\ STEP$: $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}$

More precisely, in the code, the Q Network and the Target Network are implemented using a FCNN. They take in input the state vector relative to the network status of the vehicles, then the elaboration goes through several hidden layers, the final output layer contains 2 neurons, as the number of possible actions (centralized and decentralized). In output we must have a Q value for each action taken from the input state. with the following structure of the hidden part:

1. fully connected layer with 64 neurons and ReLu activation function, followed by a batch normalization layer;
2. fully connected layer with 16 neurons and ReLu activation function, followed by a batch normalization layer.

The weights of the NN are updated using the Adam algorithm, with a learning rate $\alpha = 10^{-4}$.

The discount factor λ is set to 0.95 so that the agent takes into account the previous history. DDQL stores the learning transitions (state, action, reward, next state, end) in an ER buffer with a capacity B = 5000. Inside the DDQL algorithm, the weights of the Target Network are replaced every 1000 steps with the weights of the Q Network. During the training of the agent, a batch-approach is exploited, with a batch size set to 128.

An ϵ-greedy strategy with an exponential decay factor along the learning steps is adopted to deal with the exploitation-exploration dilemma. Specifically, the epsilon factor decays exponentially from 1.0 to 0.1 in 100 episodes.

4.4 Performance Evaluation

The performance of the proposed meta-learning RL agent is evaluated though two simulation campaigns in ns-3, where the agent learns dynamically to adopt a centralized or decentralized learning approach in order to maximize the reward. The first campaign considers a vehicle scenario with good channel conditions, i.e., high received power at the UEs, namely a high perceived SINR. Then, the second campaign considers a scenario characterized by a more degraded channel (on average).

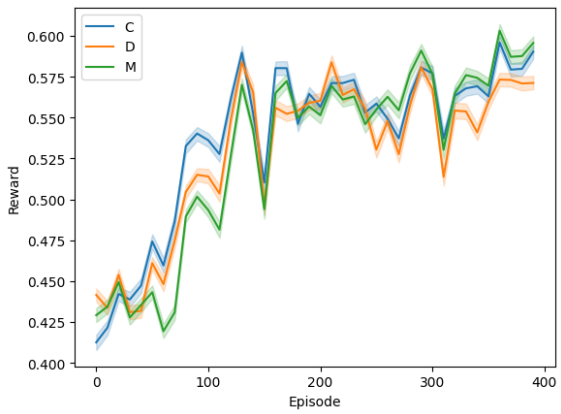
Simulation campaign structure The simulation campaign of the meta-agent is made up of 400 episodes. Each episode corresponds to an independent ns-3 simulation of nearly 90 s, where the multi-agent performs 90 steps. As a consequence, it has to decide for 36000 times the PQoS learning approach to adopt.

Each step of the meta-agent is computed after that 30 steps are performed by centralized/decentralized RL sub-agent (i.e., the one choosing PQoS countermeasures). This occurs 90 times within an independent episode (simulation), thus a MDP framework is built.

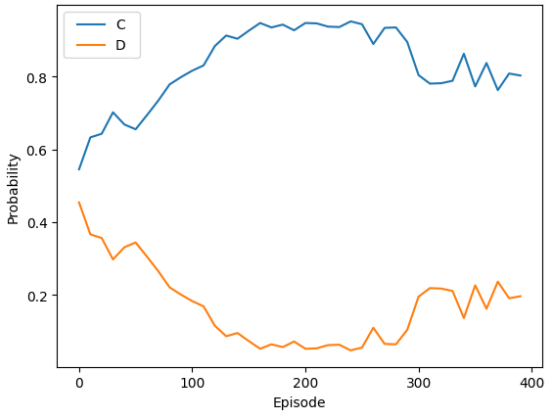
Results

A PQoS scenario involving 3 vehicles is considered in the ns-3 simulation. Furthermore, the policies of selecting statically either centralized or decentralized approach are used as baseline for evaluating the meta-learning agent.

4.4.1 Good Channel

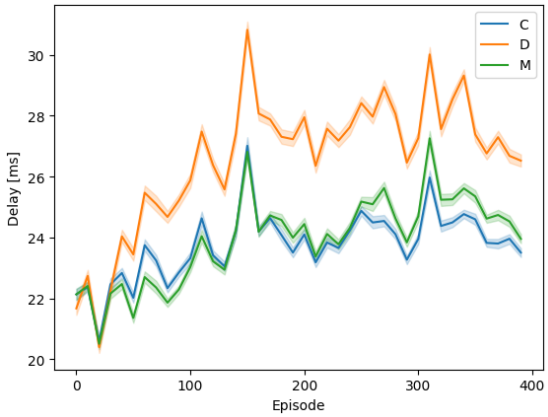


(a) Average Reward

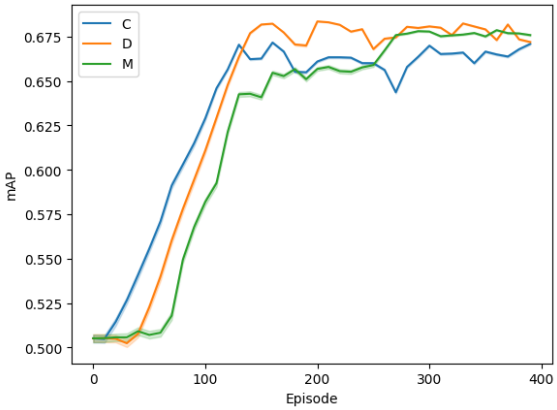


(b) Action probability

Figure 4.5: Results training of meta-learning agent vs. static approaches (baseline) in good channel conditions



(a) Average E2E Delay



(b) Average mAP

Figure 4.6: QoS and QoE indicators achieved by meta-learning agent vs. static approaches (baseline) in good channel conditions

As expected, in good channel conditions, the centralized approach produces a larger reward since the RL agent gathers more data from all the vehicles, being able to learn and optimize the overall network status. The decentralized federated approach generates a similar but smaller reward than the centralized one (Fig. 4.5a). Indeed, the convergence is still quite fast thanks to the federated updates, but a significant drawback is present in terms of impact on E2E latency (Fig. 4.6a). However, the value of the latency still satisfies the TD constraints, so

there are no strong penalizing effects on the reward. In addition, the decentralized learning achieves a higher QoE, in terms of mAP of selected compression modes, with respect to the centralized case, as illustrated in Fig. 4.6b. This fact could be linked to the different state definition and the multi-agent setup, where the agent may learn different trajectories and strategies to achieve its goal (i.e., maximization of long-term reward).

The meta-learning agent (denoted as M in the plots) achieves very interesting results in terms of QoS (delay) and QoE (mAP) of the simulated TD scenario. With respect to the static stand-alone approaches, the dynamic selection of centralization vs. decentralization generates important improvements in the learning performance.

The first key aspect to notice is the curve of the average reward in Fig. 4.5a. In the first phase of the learning, after an exploration moment, the meta-learning agent performs very similarly to the centralized baseline, until the 250^{th} episode. After this moment, an important increment of the metric occurs and the intelligent agent collects a reward of almost 0.6, slightly better than the best baseline, i.e., the centralized approach. Analyzing deeply the action probability plot (Fig. 4.5b), this behaviour is associated with an increase in the probability of selecting the decentralized federated approach.

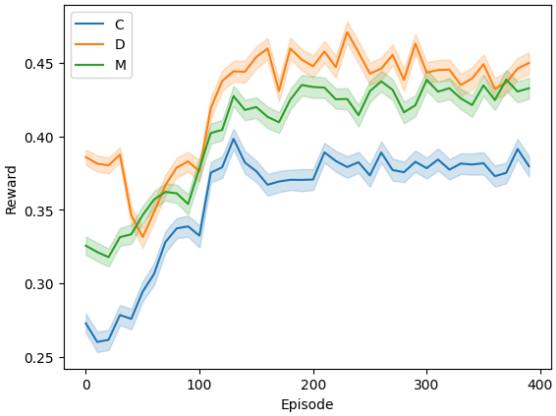
The centralized approach produces the best results from the point of view of the reward, because of low delay and rich availability of network metrics to the agent. Indeed, the meta-agent prefers clearly the centralization, but at a certain point the usage of federated approach becomes more frequent. At the end of the learning, on average, the centralized approach is chosen with 80% of probability vs. 20% of probability of the decentralized federated approach. In this context, the average reward goes beyond the reward of a fully centralized agent.

This happens because the decentralized federated (sub-)agent starts selecting compression modes with an higher mAP (on average), resulting in an larger contribution of the QoE on the reward. The average E2E delay increases because of the federated steps but it does not explode. Indeed, the final latency is smaller than the federated baseline because the centralized approach remains the preferred option during the simulation. In this way, the QoS factor is not penalizing the reward.

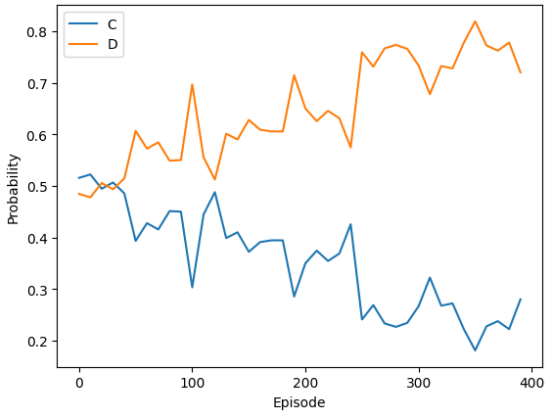
In conclusion, in some cases, involving dynamically the decentralization in the centralized approach contributes to improve the performance of PQoS

framework in the TD scenario. Furthermore, user privacy and data security are ensured when the decentralized approach is employed.

4.4.2 Bad Channel

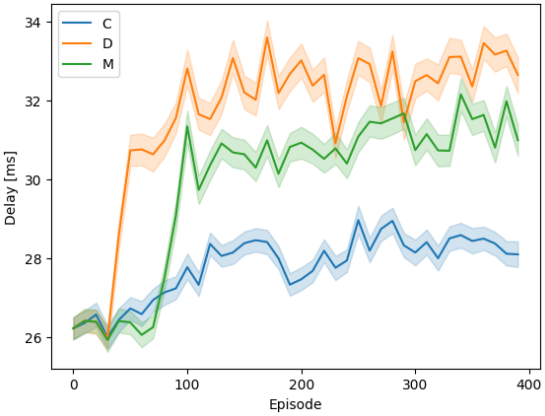


(a) Average Reward

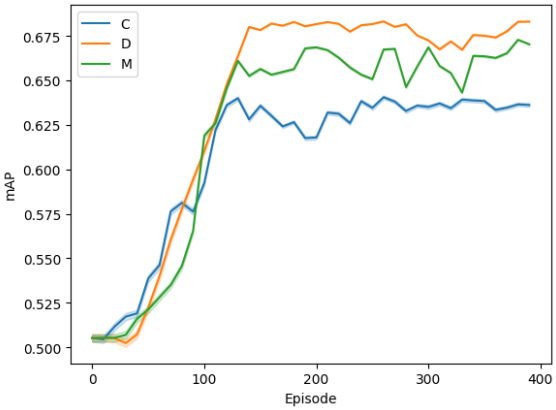


(b) Action probability

Figure 4.7: Results training of meta-learning agent vs. static approaches (baseline) in bad channel conditions



(a) Average E2E Delay



(b) Average mAP

Figure 4.8: QoS and QoE indicators achieved by meta-learning agent vs. static approaches (baseline) in bad channel conditions

In worst channel conditions, the centralized approach is more penalized by outage events happening during the learning. In these moments, the communication with the gNB is not feasible, as a consequence, the centralized agent at the RAN is not able to gather all the local metrics and measurements

related to the vehicles to build the state. The penalization impacts heavily the average reward (Fig. 4.7a). As shown in Fig. 4.8, the E2E delay is increased and the mAP is decreased in comparison to the good channel simulation because more retransmissions are necessary to share correctly the point clouds, generating more congestion. Indeed, the QoE is more degraded due to the higher probability of selecting more aggressive compression modes. On the contrary, the decentralized federated scenario achieves better performance (Fig. 4.7a) because local metrics are sufficient to build the state and take PQoS countermeasures, so outages have a weak impact during the learning. Anyway, with respect to the centralized approach, the delay (Fig. 4.8a) is higher due to the federated updates. While the QoE (Fig. 4.8b) is improved because the decentralization exploits local information, without relying on a continuous communication with the gNB, and this autonomy reduces the negative effects of poor channel conditions on the learning of the optimal compression policy.

The meta-learning agent achieves very interesting results in terms of QoS (delay) and QoE (mAP). It learns that the decentralized approach has better performances in degraded channel conditions, prioritizing this approach over the centralized solution. Indeed, the collected reward (Fig. 4.7a) and the QoS, along with the QoE metrics (Fig. 4.8) of the meta-agent are significantly close to the results achieved by the static decentralized baseline.

In this context, the action probability plot (Fig. 4.7b) shows a very high probability (75%) of selecting decentralization and a smaller probability (25%) of selecting centralization. There are two possible reasons why the latter is not null. First, the exploration component ($\epsilon = 0.1$) allows the meta-agent to take actions not maximizing the immediate reward, with the aim of discovering better strategies over the time (exploration vs. exploitation dilemma). The second reason is that the centralized approach performs better in absence of outages. Consequently, the meta agent prefers to select it in presence of good channel conditions. Indeed, in general, the simulated scenario presents, on average, degraded channel conditions, but not always. Otherwise, the network would be unable to ensure the communication required by the TD application.

5

Conclusions and Future Works

The TD scenario is characterized by strict requirements in terms of QoS, especially E2E latency and reliability. In this context, being able to control and predict future degradation in the network state is fundamental, as well as taking countermeasures in case QoS is not satisfied. For this reason, the PQoS technique is explored. The aim of this thesis has been the design, implementation and evaluation of learning algorithms for improving PQoS, by exploiting RL strategies.

In the first part, realism is introduced in the collection of metrics to describe the network state to RL agents. Considering the centralized approach, new metrics related to the 5G protocol stack are collected (BLER and TBS) and a new mechanism to exchange the SINR perceived at the UE to the gNB, according to a realistic policy, is implemented in ns-3. In addition, the CQI and BSR reporting mechanism are exploited. The best results obtained considering the new state definition with the CQI are very close to the idealized case: the E2E delay is slightly higher, but the QoE is improved. Considering the distributed approach, metrics that cannot be computed autonomously by the vehicle are discarded. Instead, PHY layer metrics and the size of transmission buffers at the RLC layer are collected. Two solutions are proposed to collect the E2E delay: ACKs at the APP layer and estimation based on APP transmitted bytes and MCS. The first solution does not scale with the number of users, generating a large delay; while the second approach presents better results in terms of QoS with respect to the idealized setup, slightly degrading the QoE. Overall, the distributed approach exceeds the TD limit of 50 ms more frequently than the centralized approach,

even if the delay is smaller on average (24 vs. 25 ms). This means that the centralization results in an improved global optimization of the QoS, because the full network state is perceived.

Subsequently, in the federated approach, a realistic sharing of the model parameters is implemented, which heavily degrades the QoS (+6 ms in the E2E delay) due to the larger burden on the channel. For this reason, an adaptive optimizer (Adam) and compression techniques are exploited to limit the overhead generated by the federated steps, ensuring a high QoE of the optimal compression policy. In this context, the quantization technique outperforms the other approaches. Moreover, different utility definitions are considered to measure the importance of each vehicle in the federated update, showing interesting results with many vehicles.

In the last part, a meta-learning agent is implemented to dynamically choose between centralized vs. decentralized learning models, jointly optimizing QoS and QoE. This agent shows very promising results, being able to switch autonomously in time to the best approach based on the network state. More precisely, the centralized approach is preferred in good channel conditions, since the RL agent perceives the full network state. On the contrary, the decentralized approach is preferred in degraded network conditions, because outages have no impact on the state collection during the training.

As a future work, the main direction is to improve the meta-learning agent to allow a dynamic choice between centralization vs. decentralization not only in time, but also per vehicle. Consequently, the heterogeneity in local channel conditions can be exploited. Furthermore, in this thesis, the PQoS countermeasure (i.e., the choice of the RL agent) is restricted only to the tuning of the compression mode, at the APP layer. The PQoS action can be extended to parameters related to the resource allocation and scheduling, e.g., adjusting the sub-band and power level or selecting the optimal group of resource blocks.

References

[1] Marco Giordani et al. "Toward 6G Networks: Use Cases and Technologies". In: *IEEE Communications Magazine* 58.3 (2020), pp. 55–61. DOI: 10.1109/MCOM.001.1900411.

[2] Junil Choi et al. "Millimeter-Wave Vehicular Communication to Support Massive Automotive Sensing". In: *IEEE Communications Magazine* 54.12 (2016), pp. 160–167. DOI: 10.1109/MCOM.2016.1600071CM.

[3] Fengxiao Tang et al. "Survey on Machine Learning for Intelligent End-to-End Communication Toward 6G: From Network Access, Routing to Traffic Control and Streaming Adaption". In: *IEEE Communications Surveys Tutorials* 23.3 (2021), pp. 1578–1598. DOI: 10.1109/COMST.2021.3073009.

[4] Tao Zhang. "Toward Automated Vehicle Teleoperation: Vision, Opportunities, and Challenges". In: *IEEE Internet of Things Journal* 7.12 (2020), pp. 11347–11354. DOI: 10.1109/JIOT.2020.3028766.

[5] 5GAA. "C-V2X Use Cases Volume II: Examples and Service Level Requirements". In: *White Paper* (Oct. 2020).

[6] Mate Boban, Marco Giordani, and Michele Zorzi. "Predictive Quality of Service: The Next Frontier for Fully Autonomous Systems". In: *IEEE Network* 35.6 (2021), pp. 104–110. DOI: 10.1109/MNET.001.2100237.

[7] M. Todescato et al. "Machine Learning meets Kalman Filtering (with proofs)". In: *55th IEEE Conference on Decision and Control (CDC16)*. IEEE. Las Vegas, Dec. 2016, pp. 4594–4599.

[8] Mowei Wang et al. "Machine Learning for Networking: Workflow, Advances and Opportunities". In: *IEEE Network* 32.2 (2018), pp. 92–99. DOI: 10.1109/MNET.2017.1700200.

[9] Federico Mason et al. "A Reinforcement Learning Framework for PQoS in a Teleoperated Driving Scenario". In: *2022 IEEE Wireless Communications and Networking Conference (WCNC)*. 2022, pp. 114–119. DOI: 10.1109/WCNC51071.2022.9771590.

[10] Filippo Bragato et al. "Towards Decentralized Predictive Quality of Service in Next-Generation Vehicular Networks". In: *IEEE Information Theory and Applications Workshop (ITA)*. 2023.

[11] Marco Mezzavilla et al. "End-to-End Simulation of 5G mmWave Networks". In: *IEEE Communications Surveys Tutorials* 20.3 (2018), pp. 2237–2263. DOI: 10.1109/COMST.2018.2828880.

[12] Matteo Drago et al. "Artificial Intelligence in Vehicular Wireless Networks: A Case Study Using ns-3". In: *Proceedings of the 2022 Workshop on Ns-3*. WNS3 '22. Virtual Event, USA: Association for Computing Machinery, 2022, pp. 112–119. ISBN: 9781450396516. DOI: 10.1145/3532577.3532605. URL: https://doi.org/10.1145/3532577.3532605.

[13] 3GPP. "Service requirements for enhanced V2X scenarios (Release 16)". In: *TS 22.186* (Nov. 2020).

[14] E. Dahlman, S. Parkvall, and J. Skold. *5G NR: The Next Generation Wireless Access Technology*. Academic Press, 2021. ISBN: 978-0-12-822320-8. DOI: 10.1016/C2019-0-04564-5.

[15] 3GPP. "5G NR and NG-RAN Overall description (Release 16)". In: *TS 38.300* (July 2020).

[16] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction (2nd ed.)* 2018. URL: http://incompleteideas.net/book/RLbook2020.pdf (visited on 06/20/2024).

[17] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8 (1992), pp. 279–292. DOI: 10.1007/BF00992698.

[18] G. Rummery and Mahesan Niranjan. "On-Line Q-Learning Using Connectionist Systems". In: *Technical Report CUED/F-INFENG/TR 166* (Nov. 1994).

[19] Daniel Krajzewicz et al. "SUMO (Simulation of Urban MObility); An open-source traffic simulation". In: Jan. 2002, pp. 183–187. ISBN: 90-77039-09-0.

[20] Mate Boban, João Barros, and Ozan K. Tonguz. "Geometry-Based Vehicle-to-Vehicle Channel Modeling for Large-Scale Simulation". In: *IEEE Transactions on Vehicular Technology* 63.9 (2014), pp. 4146–4164. DOI: 10.1109/TVT.2014.2317803.

[21] Google. *Draco 3D Data Compression*. 2017. URL: https://google.github.io/draco/ (visited on 06/21/2024).

[22] Paolo Testolina et al. "SELMA: SEmantic Large-Scale Multimodal Acquisitions in Variable Weather, Daytime and Viewpoints". In: *IEEE Transactions on Intelligent Transportation Systems* 24.7 (2023), pp. 7012–7024. DOI: 10.1109/TITS.2023.3257086.

[23] Hao Yin et al. "ns3-ai: Fostering Artificial Intelligence Algorithms for Networking Research". In: *Proceedings of the 2020 Workshop on Ns-3*. WNS3 '20. Gaithersburg, MD, USA: Association for Computing Machinery, 2020, pp. 57–64. ISBN: 9781450375375. DOI: 10.1145/3389400.3389404. URL: https://doi.org/10.1145/3389400.3389404.

[24] Alex H. Lang et al. "PointPillars: Fast Encoders for Object Detection From Point Clouds". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 12689–12697. DOI: 10.1109/CVPR.2019.01298.

[25] 3GPP. "5G NR Physical layer procedures for data (Release 16)". In: *TS 38.214* (July 2020).

[26] 3GPP. "5G NR Medium Access Control (MAC) protocol specification (Release 16)". In: *TS 38.321* (July 2020).

[27] 3GPP. "5G NR Radio Link Control (RLC) protocol specification (Release 16)". In: *TS 38.322* (July 2020).

[28] 3GPP. "5G NR Packet Data Convergence Protocol (PDCP) specification (Release 16)". In: *TS 38.323* (July 2020).

[29] Tian Li et al. "Federated Learning: Challenges, Methods, and Future Directions". In: *IEEE Signal Processing Magazine* 37.3 (2020), pp. 50–60. DOI: 10.1109/MSP.2020.2975749.

[30] Brendan McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data". In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. Ed. by Aarti Singh and Jerry Zhu. Vol. 54. Proceedings of Machine Learning Research. PMLR, 20–22 Apr 2017, pp. 1273–1282. URL: https://proceedings.mlr.press/v54/mcmahan17a.html.

[31] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. pmlr. 2015, pp. 448–456.

[32] Sashank Reddi et al., eds. *Adaptive Federated Optimization*. 2021. URL: https://openreview.net/forum?id=LkFG3lB13U5.

[33] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: 12.null (July 2011), pp. 2121–2159. ISSN: 1532-4435.

[34] Manzil Zaheer et al. "Adaptive methods for nonconvex optimization". In: NIPS'18. Montréal, Canada: Curran Associates Inc., 2018, pp. 9815–9825.

[35] Diederik Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations (ICLR)*. San Diega, CA, USA, 2015.

[36] Sunil Vadera and Salem Ameen. "Methods for Pruning Deep Neural Networks". In: *IEEE Access* 10 (2022), pp. 63280–63300. DOI: 10.1109/ACCESS.2022.3182659.

[37] Song Han et al. "Learning both weights and connections for efficient neural networks". In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'15. Montreal, Canada: MIT Press, 2015, pp. 1135–1143.

[38] Michael Zhu and Suyog Gupta. "To Prune, or Not to Prune: Exploring the Efficacy of Pruning for Model Compression". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net, 2018. URL: https://openreview.net/forum?id=Sy1iIDkPM.

[39] R. Reed. "Pruning algorithms-a survey". In: *IEEE Transactions on Neural Networks* 4.5 (1993), pp. 740–747. DOI: 10.1109/72.248452.

[40] Xiaotian Zhao, Ruge Xu, and Xinfei Guo. "Post-training Quantization or Quantization-aware Training? That is the Question". In: *2023 China Semiconductor Technology International Conference (CSTIC)*. 2023, pp. 1–3. DOI: 10.1109/CSTIC58779.2023.10219214.

[41] Felix Sattler et al. "Sparse Binary Compression: Towards Distributed Deep Learning with minimal Communication". In: *2019 International Joint Conference on Neural Networks (IJCNN)*. 2019, pp. 1–8. DOI: 10.1109/IJCNN.2019.8852172.

[42] Felix Sattler et al. "Robust and Communication-Efficient Federated Learning From Non-i.i.d. Data". In: *IEEE Transactions on Neural Networks and Learning Systems* 31.9 (2020), pp. 3400–3413. DOI: 10.1109/TNNLS.2019.2944481.

[43] Jelili Oyelade et al. "Data Clustering: Algorithms and Its Applications". In: *2019 19th International Conference on Computational Science and Its Applications (ICCSA)*. 2019, pp. 71–81. DOI: 10.1109/ICCSA.2019.000-1.

[44] S. Lloyd. "Least squares quantization in PCM". In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137. DOI: 10.1109/TIT.1982.1056489.

[45] Lei Fu et al. "Client Selection in Federated Learning: Principles, Challenges, and Opportunities". In: *IEEE Internet of Things Journal* 10.24 (2023), pp. 21811–21819. DOI: 10.1109/JIOT.2023.3299573.

[46] Hado van Hasselt, Arthur Guez, and David Silver. "Deep reinforcement learning with double Q-Learning". In: AAAI'16. Phoenix, Arizona: AAAI Press, 2016, pp. 2094–2100.

[47] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533. DOI: 10.1038/nature14236.

Acknowledgments

Voglio ringraziare profondamente i miei genitori, Nicoletta e Alessandro, e mio fratello Luca per avermi dato la possibilità di arrivare a questo traguardo con serenità, motivazione, forza ed energia. Tanto merito è anche vostro.

Voglio ringraziare il Prof. Zorzi, il Prof. Giordani e Filippo Bragato per avermi continuamente stimolato con le loro idee, aiutandomi a uscire da momenti bui e a cogliere aspetti nuovi. Un grande ringraziamento va anche a tutti i membri del laboratorio SIGNET che mi hanno accompagnato e supportato durante il periodo di tesi.

Voglio ringraziare tutti i miei amici più stretti e i miei amici della pallavolo per i momenti di allegria, gioia e divertimento.

Voglio ringraziare tutti i miei amici dell'università per aver contribuito ad alleggerire le giornate piene di lezioni e studio a Padova, durante gli ultimi 5 anni.