Simulation of Tele-Operated Driving over 5G Using CARLA and OMNeT++

Valerio Cislaghi[1], Christian Quadri[1], Vincenzo Mancuso[2], and Marco Ajmone Marsan[2]
[1]*Computer Science Department*, *Università degli Studi di Milano*, Milan, Italy
[2]*IMDEA Networks Institute*, Madrid, Spain

***Abstract*—Tele-Operated Driving (ToD) allows a remote operator to drive a vehicle through the services provided by a mobile radio network. ToD can replace on-board driving in many different occasions, such as dangerous environments, but can also provide assistance to autonomous driving systems in difficult and unexpected situations. ToD is a bandwidth-demanding and latency-sensitive service, which requires transmitting a large amount of sensor data from vehicle to operator, and driving instructions from operator to vehicle. The data exchange must comply with strict real-time requirements. The low latency and high bandwidth offered by 5G Radio Access Networks (RANs) open new opportunities for an effective deployment of ToD services in different contexts. However, the rapidly changing channel quality and network conditions can raise many challenges in meeting bandwidth and latency requirements.**

In this paper, we report on the development of an elaborate simulation framework combining the realism of vehicle dynamics simulated by CARLA and the detailed network models provided by OMNeT++. We demonstrate the capabilities of the simulation framework by describing results about the feasibility of ToD services in a simple scenario under different network and application configurations. We simulate the implementation of the ToD service in a slice of a 5G RAN, with varying application and network parameters, also considering a variable amount of background traffic. Our simulation results show that the ToD service performance is heavily impacted by the amount and shape (i.e., the selected 5G NR numerology) of radio resources allocated to the 5G slice.

***Index Terms*—Tele-Operated Driving; 5G Radio Access Network; Simulation; CARLA; OMNeT++**

I. Introduction

Tele-operated Driving (ToD) is a technology that assists and complements semi- and fully-automated driving in various scenarios, allowing a remote operator to drive a vehicle through the services of a Radio Access Network (RAN). ToD is classified as SAE Level 4, i.e., a system entirely in charge of all driving and navigational tasks, under specific circumstances, without requiring passengers to be ready to take control of the vehicle [1]. The 5G Automotive Association (5GAA) proposes an architecture and defines the system requirements for ToD [2] to operate properly in heterogeneous contexts with the support of the 5G mobile network. The ToD technology can be employed to support autonomous driving (AD) systems which, despite the enormous improvements in the last decade, are still unable to operate properly in many borderline situations, where human intervention is required. This is for example the case of the Baidu taxis developed in Chinese cities[1]. ToD can also be used to assist a driver who is experiencing difficulties, like a sudden health problem, or to remotely perform specific driving tasks of non-automated vehicles without the physical presence of a driver in the vehicle. Moreover, ToD can be employed to maneuver industrial vehicles in particularly dangerous situations guaranteeing the safety of workers.

[1]See e.g., https://techwireasia.com/2021/05/baidu-rolls-out-chinas-first-paid-driverless-taxi-service/

In all these scenarios, the mobile network plays a central role and must guarantee a suitable level of quality of service (QoS). In particular, the RAN must be able to handle the uplink traffic containing vehicle data captured by the on-board sensors (e.g., cameras and LiDAR), as well as the downlink traffic consisting of driving instructions sent by the remote driver. The 5GAA defines a minimum uplink bitrate of 32 Mb/s with a maximum delay of 100 ms, while in the downlink direction, latency is more stringent than bandwidth, i.e., 20 ms and 1 Mb/s, respectively [2]. Guaranteeing such QoS levels in heterogeneous environments is challenging, mainly due to the rapid variability of the radio channel characteristics caused by vehicle mobility, shadowing effects, and object reflections, resulting in unpredictable channel quality. Therefore, it is fundamental to understand under which network conditions and application settings the ToD service is able to operate properly, guaranteeing secure and comfortable remote driving.

In this paper, we report on the development of an elaborate simulation framework integrating the CARLA [3] and OMNeT++ [4] simulators, and including on top of them a vehicle driver agent that mimics real human drivers better than the agents available in CARLA and other simulators. The resulting framework offers a remarkable level of physical and networking details, together with a large set of configurable parameters. Using this framework, we are able to create different network and application configurations to test the effectiveness of the ToD service. The code of the simulation framework will be released upon publication of this work.

Using the developed framework, we perform a first investigation of the feasibility of ToD services under different network and application configurations. We measure the performance of ToD by comparing the vehicle's trajectory w.r.t. a baseline trajectory driven by a simulated human driver within the car. We conduct an extensive simulation campaign on a realistic scenario by varying a number of application parameters, such as video frame rate and video quality, as well as network settings, like 5G-NR numerology and background traffic. The results show that the ToD service is feasible (according to the 5GAA minimum requirements) under specific combinations of application and network settings. In general, performance is highly sensitive to radio channel conditions, availability of radio resources at the base station, and presence of background traffic, but also to the different difficulties of driving trajectory segments.

II. Related work

ToD was considered in a number of research works. Zhang [5] presents a vision of intelligent ToD and discusses its benefits and challenges. Gnatzig *et al.* [6] tested a ToD prototype at low speeds with a low-quality multi-camera system of 640×480 pixels, generating up to 2 Mb/s bitrate, and incurring around 600 ms total delay. Liu *et al.* [7] developed a prototype of ToD system by emulating a LTE mobile network to investigate how human remote drivers perform, and assessed the system performance under realistic network conditions. The main limitation of these contributions is that they are tested in a controlled and simple environment, using application settings significantly lower than the one specified by 5GAA [2]. Moreover, the results are potentially biased by the human test (e.g., driving style, reaction time, and training level), blurring the effect of the network component.

Some recent studies investigated the effect of network configuration on driving performance. Pérez *et al.* [8] define a fully parametrizable model which relates network performance (throughput, latency, loss rate) with the perceived video quality. They analyze the performance of different network scenarios, using existing databases and field measurements, considering LTE and pre-commercial 5G pilots. Neumeier *et al.* [9] investigate the effects of latency on performance and perceived workload for different driving scenarios. They run an experiment with 28 participants, measuring quantitative metrics accounting for their driving behavior and asking them to complete a questionnaire. The results show that latency negatively affects driving behavior and becomes a perceived problem above 300 ms.

As regards simulation tools for ToD, the research community has proposed frameworks for performance evaluation and emulation. TELECARLA [10] is an extension of the CARLA simulator that integrates an emulated network using the Linux built-in tool *tc-netem* and provides an interface for the adaptation of the temporal resolution and target bitrate of the compressed video streams. Schimpe *et al.* [11] propose a ROS-based software stack to support research in the field of ToD. The system is modular, allowing easy integration with existing AD software. OpenROUTS3D [12] is a Unity-based simulator specifically designed for ToD, which offers a highly detailed model of vehicle physics and integration with external tools for map generation. These simulation frameworks mainly focus on reproducing a realistic remote driving experience, but unfortunately, they do not include realistic models of the mobile network. The recent work of Mason *et al.* [13] focuses on proposing a reinforcement learning solution for guaranteeing QoS for ToD applications. They evaluate their solution using a simulation framework based on the ns-3 network simulator and SUMO [14]. Opposite to the aforementioned simulators, this framework offers a realistic network model, but limited vehicle dynamics.

Different from previous works, in this paper we report a new ToD simulation framework providing detailed descriptions of both vehicle dynamics and data exchange. Exploiting this simulation tool we perform a first analysis of the feasibility of a ToD service implemented within a slice of a 5G RAN.

III. Network Assisted Tele-Operated Driving

The activity cycle of the ToD service is divided into three distinct phases. The first phase corresponds to the ToD start, either because the Host Vehicle (HV) is started, or because the AD that drives the HV recognizes that a remote intervention is necessary to manage a critical situation, which is unknown or unexpected. This first phase is named *ToD Activation*. The remote driver establishes an authenticated and secure communication channel with the HV. Once the connection is in place, HV provides information about its vehicle type, position, destination, and other details that will assist the ToD operator to create a model of the vehicle and of the surrounding area. This information includes vehicle motion, camera view, and data from other on-board sensors, possibly complemented by information coming from nearby vehicles.

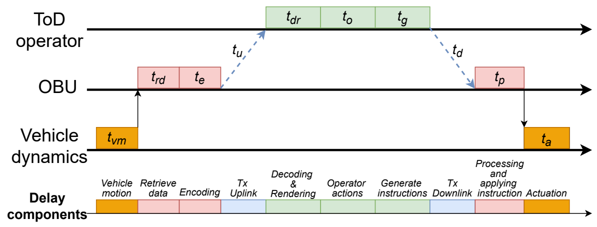

Fig. 1: Delay components of ToD service as identified in [15].

The second phase, namely the *Driving* phase, begins when the ToD activation has been completed. At this time, the Tele-Operator evaluates the vehicle context, selects the most appropriate strategy to reach the HV destination or to resolve a critical situation, and starts driving. For the whole duration of the driving phase, the HV continuously collects and transmits sensor data to the Tele-Operator through the network at predetermined intervals. This data includes HV status information such as high-quality video streaming to see the road, as well as motion data, nearby vehicles position and movement, and other context information. Based on the received information, the remote driver obtains a model of the vehicle's surroundings and produces the driving instructions which are sent to the HV through the network. As soon as the HV receives the driving information, it applies the maneuver instructions subject to the onboard security checks.

Once the HV reaches its destination or resolves the critical situation, the remote driving process ends. In this last phase, namely *ToD De-activation*, the control of the HV returns to the on-board controller (AD system or human).

All actors involved in the ToD service must satisfy several requirements. The HV must be able to periodically send state information to the ToD Service Provider, receive driving instructions from the ToD operator, and implement them using on-board actuators. In order to securely operate the remote vehicle, the ToD operator must be able to see on a display the driven vehicle's environment together with additional sensor data in real-time.

In Fig. 1 we show the main delay components of a ToD service as identified in [15]. Vehicle motion between consecutive status samplings (spaced by time t_{vm}) and actuation time due to the vehicle's inertia (t_a) are unavoidable sources of error that have to be taken into account in the definition of system requirements. Other delay components reside in the vehicle's On-Board Unit (OBU). In particular, we must consider the time for retrieving status data according to on-board sensors' duty cycle and CAN bus transmissions (t_{rd}); the processing time for encoding video streaming from cameras (t_e) and the time for processing and sending instructions to actuators (t_p). These delay components strongly depend on the vehicle's specifications and OBU computation capabilities. Moreover, also ToD operations involve latency which can be modeled in three components. First, a non-negligible time is required to decode and render the video data on the operator's monitors (t_{dr}). Second, tele-operator's reaction time (t_o) must be considered. Last, some time is necessary for the tele-operator's console to generate the instructions (t_g) based on the interactions between the operator and the console's actuators[2]. Note that the time required for computing the instruction varies between human and AD operators. In the latter case, it is reasonable to assume that a new instruction is generated based on the last received status and recent history each time the vehicle status is updated in the operator system. While a human gives new maneuvers with less time precision depending on several factors, including reaction time.

The last delay components are network data transfer delays in both the uplink (t_u) and downlink (t_d) directions. These represent the most critical and variable delay components in the whole ToD system. In particular, the uplink delay strongly impacts the coherence of the context perceived by the remote operator: a high uplink delay causes a significant discrepancy between the actual context of the vehicle and the one displayed on the operator's monitors. Similarly, the downlink latency must be low, in order not to compromise the effectiveness of instructions. The high variability of network delays, due to several factors such as available bandwidth, channel quality, background traffic, and amount of data to transmit, poses many challenges to a ToD service that must be able to operate properly under complex and heterogeneous network conditions. In this work, we investigate the feasibility of ToD services focusing on network requirements and evaluating different network and application configurations.

IV. Simulation Framework

In this section, we present our highly detailed and realistic simulation framework. The framework is based on the two state-of-the-art simulators CARLA [3] and OMNeT++ [4]. CARLA is an open-source simulator for autonomous driving research built on top of Unreal Engine[3] and provides developers with a full-fledged set of APIs for modeling, sensing, and controlling detailed physical environments. CARLA offers a remarkable level of physical and graphical realism. In particular, it provides a complete suite of onboard sensors (cameras, Radar, LiDAR) and a sophisticated vehicle dynamics model. OMNeT++ is a highly extensible network simulator that offers several options purposely designed to model specific parts of the communication network. In this work, we use the INET framework, which implements the whole suite of the standard TCP/IP protocol stack, and Simu5G [16], which offers a detailed model of a 5G RAN, focusing on 5G-NR, data-plane, and MEC implementation.

We designed the simulation framework following the separation of concerns principle, reusing and extending the existing frameworks, and exploiting modern and high-level programming languages such as Python.

The high-level integration of CARLA and OMNetT++ is depicted in Fig. 2. All networking components (in blue) are implemented in OMNeT++ and include all the protocol stacks of UE, gNodeB, UPF, and remote host. The OMNeT++ part is responsible only for realizing the communication between HV and ToD operator, without implementing the application logic. In particular, the *network application* layer is in charge of: *(i)* receiving/sending packets from/to the transport layer; *(ii)* generating the timed events of the application; *(iii)* interacting with CARLA application components for passing messages between HV and ToD through the communication network (the red arrows in the figure).

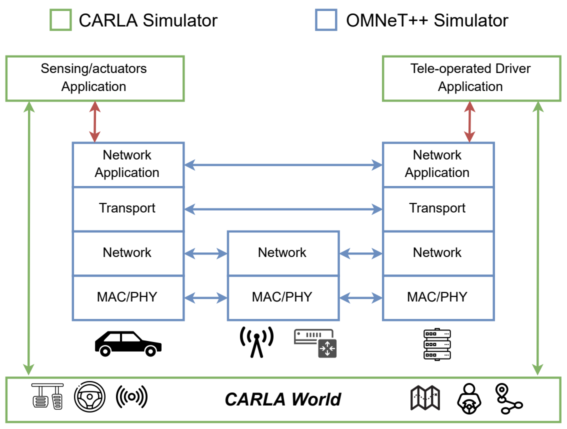

Fig. 2: Overview of CARLA and OMNeT++ integration. CARLA components in green, OMNeT++ components in blue, and interactions between CARLA and OMNeT++ in red.

The CARLA components (in green) are responsible for: *(i)* implementing the physical world, e.g., vehicle dynamics, sensors/actuators, road maps, traffic lights, and other moving objects such as vehicles, bicycles, pedestrians, and more; *(ii)* implementing a specific application deployed onboard of HV for sensing data from sensors and actuating driving instructions; *(iii)* implementing the remote ToD agent that uses the received data of the vehicle to generate the instructions to control the HV. The implementation of the ToD agent makes extensive reuse of the driving agents' library provided by CARLA and written in Python, which allows easier integration with the most common machine learning suite designed for autonomous driving prototyping. In this paper, we extend a base agent that emulates human driving behavior adding a realistic mechanism to implement realistic and continuous micro-driving decisions based on the visible road segment and the available sensing data (more details in Section V-D).

A. Co-simulation architecture

The architecture of the ToD simulation framework is shown in Fig. 3. It is a distributed architecture made of three main parts: *CARLA Server*, *CARLA Client*, and *OMNeT++* simulator. In the figure, the components we implemented are represented in colors, while the unchanged ones are in gray.

CARLA simulator is implemented as a client-server architecture. All the physics and graphic components reside in the server part, while the control agent is implemented on the client side. The interaction between the client and the server is realized through a TCP connection and a set of APIs (CARLA APIs in the figure) written in C++ to get data from the CARLA physical world and to send commands to actors[4]. Moreover, CARLA provides a wrapper for C++ API written in Python to ease the integration with the driving agents' library.

In our framework implementation, the interaction between CARLA client and OMNeT++ is realized through a

[2]This component can be neglected in the case of an AD operator.
[3]https://www.unrealengine.com/

[4]In CARLA, an actor is an object whose action commands are given by an external agent.

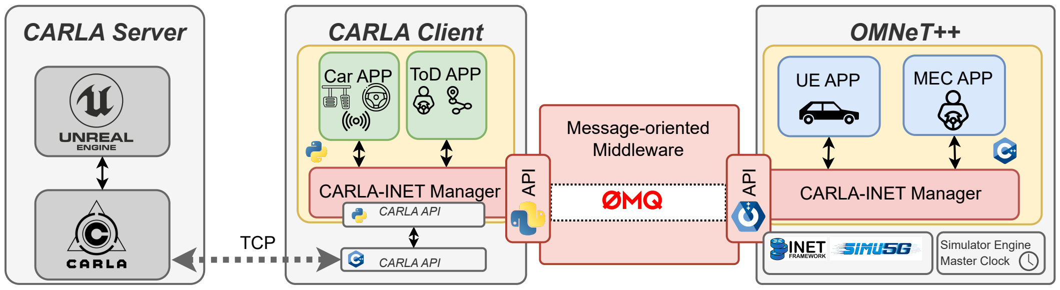

Fig. 3: Technical schema of the ToD simulator.

message-oriented middleware (red components in the figure) in charge of dispatching JSON-formatted messages between OMNeT++ and CARLA. This solution is inspired by the approach adopted by the Veins [17] framework, which implements the binary Traffic Control Interface (TraCI) of SUMO [14] and the communication between OMNeT++ and SUMO is realized through TCP connections. Differently from Veins, our solution relies on high-level messaging libraries such as ZeroMQ[5], rather than a bare TCP connection, which offers a simplified message passing interface allowing multiple communication patterns like publish-subscribe and push-pull-in, in addition to the request-reply schema. Another difference w.r.t. Veins is the utilization of JSON API. The reason for the adoption of JSON-formatted messages instead of a binary interface like TraCI is the following: TraCI is an interface designed for allowing external programs to interact with the physical world of SUMO providing well-defined features for getting data and controlling vehicles. In our case, in addition to preserving the consistency of the vehicles' position in both simulators, the APIs are designed for mapping the network application layer, implemented in OMNeT++, to the applications developed on top of CARLA API (the red arrows in Fig. 2). Therefore, the communication protocol between the two simulators consists of two types of messages: *(i)* predefined messages, which carry basic information such as initialization parameters, clock ticks, and position updates, and *(ii)* scenario-dependent messages, which are more complex and customizable according to the specific requirement of the scenario. For this reason, JSON represents a suitable format to easily extend and customize the set of APIs according to the specific scenarios and is also human-readable which simplifies the development and debugging.

The JSON APIs are managed through *CARLA-INET Manager* component which is responsible for performing some basic procedures such as initialization and mobility synchronization, and exposing the APIs to the upper layer, i.e., the applications.

B. Co-simulation message sequence

In our co-simulation framework, the OMNeT++ component manages the simulation clock and schedules all discrete events including those for advancing the simulation on the CARLA side. This design choice allows the two simulators to run coherently maintaining the same simulation time on both sides. To implement this mechanism, we configure the

[5]https://zeromq.org/

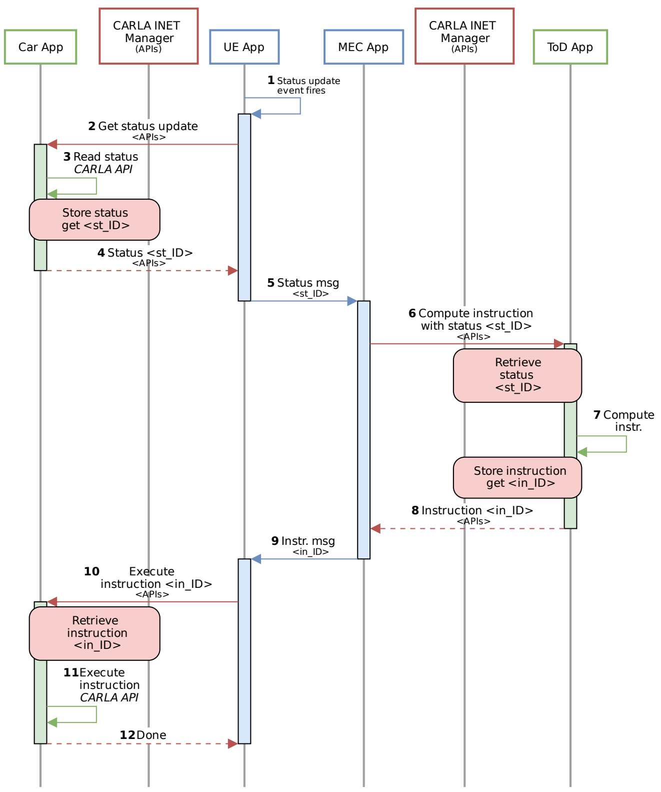
Car App
CARLA INET Manager (APIs)
UE App
MEC App
CARLA INET Manager (APIs)
ToD App
1 Status update event fires
2 Get status update <APIs>
3 Read status CARLA API
Store status get <st_ID>
4 Status <st_ID> <APIs>
5 Status msg <st_ID>
6 Compute instruction with status <st_ID> <APIs>
Retrieve status <st_ID>
7 Compute instr.
Store instruction get <in_ID>
8 Instruction <in_ID> <APIs>
9 Instr. msg <in_ID>
10 Execute instruction <in_ID> <APIs>
Retrieve instruction <in_ID>
11 Execute instruction CARLA API
12 Done

Fig. 4: Sequence diagram of the simulated ToD service. CARLA components are in green, OMNeT++ modules are in blue, and interactions with CARLA-INET Manager are displayed in red.

CARLA server to run in *synchronous* mode[6] with fixed time-step length, which makes the CARLA simulation engine wait for an external clock tick to execute the next simulation step. Moreover, we configure the message-oriented middleware to operate in the *request-reply* mode. In particular, the requests are performed by modules on the OMNeT++ side, while replies are generated by CARLA components. In this setting, when the execution of an OMNeT++ discrete event requires interaction with CARLA, it issues a request using APIs and waits for the response before proceeding to the execution of the next events, guaranteeing time coherence between the two simulators.

There are three main types of request-reply messages: *initialization*, *mobility*, and *ToD service*. Initialization messages are exchanged at the beginning of the simulation to configure the CARLA side from OMNeT++ exploiting the

[6]CARLA can also run in *asynchronous* mode, where the simulation clock is directly managed by CARLA using variable time-step length. This mode is suitable for real-time simulation with a human tele-operator as in the TELECARLA [10] simulator but cannot suit a co-simulated framework like ours.

TABLE I: Simulation parameters

Parameter	Value
	ToD service
Frame rate (f/s)	{25, 60}
Status packet size	{33, 66} kB
Instruction packet size	1000 B
Transport protocol	UDP
Data retrieval time (t_{rd})	$\mathcal{N}(15\ ms, 0.1\ ms)$
Encoding image time (t_e)	$\mathcal{N}(10\ ms, 0.5\ ms)$
Processing status time (t_p)	$\mathcal{N}(15\ ms, 0.75\ ms)$
	Mobile network
Path loss model	ITU Rural/Urban macro cell
Path loss exponent (α)	2.5
Car UE transmit power	26 dBm
Car UE antenna gain	+0 dB
gNodeB antenna gain	+18 dB
gNodeB height	25 m
gNodeB transmit power	43 dBm
gNodeB duplex mode	FDD
gNodeB scheduling discipline	DRR
Handover latency	50 ms
Target BLER	0.01
Numerology index (μ)	{0, 1}
gNodeB carrier	Single carrier (2 GHz)
gNodeB bandwidth	20 MHz $\mu = 0$, 40 MHz $\mu = 1$
Resource blocks	100 per TTI
	Miscellaneous
ToD background car per BS (N_{bg})	{0, 1, 2, 3}
Background ToD app	pkt = 33 kB, fps = 25
CARLA time-step	10 ms

powerful parameter studies tool. Mobility messages are sent by *CARLA-INET Manager* at fixed time intervals to advance CARLA simulation and to synchronize the motion status (i.e., position, velocity, and acceleration) in OMNeT++. The exchange of ToD service messages implements the retrieval of sensor data, the computation of the agent's instruction, and the instruction actuation.

In Fig. 4 we show in detail the sequence of a ToD service messages exchange. The whole procedure is triggered by a *status update event* which is periodically scheduled according to the configured frame rate. The message sequence (2–4) represents the vehicle status retrieval, (6–8) implements the remote instruction computation, and (10–12) performs the instruction actuation. Messages 5 and 9 represent the upload of the status and the download of the instruction, respectively. As a design choice, we decide to not pass the raw data of the status/instruction, but rather to use unique identifiers to refer to a specific vehicle status/instruction read/computed at a particular time, avoiding complex marshaling/unmarshaling of raw data. This solution allows us to configure arbitrary packet sizes on the OMNeT++ side emulating different video coding algorithms and sophisticated video compression techniques [18], [19].

V. Simulation Setup

In this section, we present the simulated scenario, discussing the different configurations of network and application parameters. Table I reports the main parameters we use throughout the simulations. Note that $\mathcal{N}(m, d)$ indicates the normal distribution with mean m and standard deviation d.

A. Map and trajectory

In Fig. 5 we illustrate the entire simulation map, which is an adaptation of `World 04`, one of the predefined maps

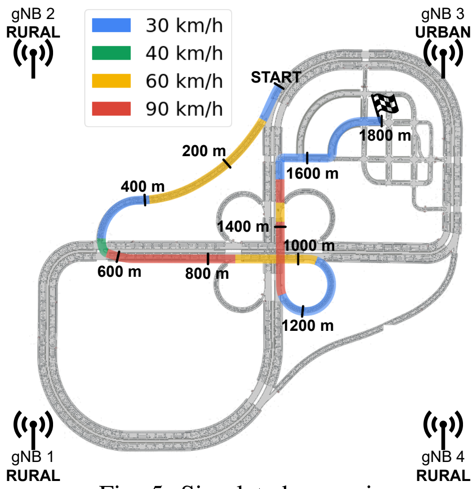

Fig. 5: Simulated scenario

of CARLA. It covers an area of approximately one square kilometer and includes both urban streets and highways. The remote operator is in charge of driving HV from the highway exit on top, marked START, to the town center marked by the checkered flag. The route is about 1.8 km long and the service lasts approximately 160 seconds. Distances from start are reported along the driving path. The resulting trajectory allows us to simulate a realistic scenario combining different degrees of difficulty and speed limits. In particular, in suburban and highway segments, speed limits are 60 and 90 km/h, respectively, whereas they lower to 40/30 km/h along the entrance/exit ramp and within the town center. We can see that the HV must exit from the highway close to the top of the figure, follow a ramp to reenter at the left, exit end reenter again at the center, finally exit from the highway close to the start, but in opposite direction, and follow urban roads to reach its destination. The most challenging segments for ToD are the first entrance ramp (400-650 m) and the town center part (1500-1800 m). The former is particularly complex because it combines turning, lane alignment, and progressively accelerating to reach the highway speed limit of 90 km/h. We assume that the vehicle is remotely driven for the entire journey, without simulating the context-switching phases described in Section III.

B. Network

Four base stations (BSs) are placed in correspondence to the four corners of the map; the BS closest to town (*gNB 3* in Fig. 5) is modeled with the ITU urban path loss model, while the other three BSs with the rural path loss model. The vehicle associates with all four BSs during its journey, varying the quality of the cellular signal and causing handovers. Each BS provides the ToD service through a slice that is configured with a single carrier having a total bandwidth of 20 MHz or 40 MHz, considering numerology index μ equal to 0 and 1, respectively. Both these configurations lead to 100 resource blocks per TTI, i.e., the configuration with $\mu = 1$ offers double RB/s w.r.t. $\mu = 0$. The ToD agent is deployed on a remote host located at 10 km distance from the driving path, and the connection with the mobile network is realized with a 10 Gb/s wired link. This setting avoids bottlenecks in the transport network and introduces marginal further network latency.

To simulate a realistic scenario, we configure different levels of background traffic, considering the presence of other HVs using the ToD service in the same area, but not interfering with the movement of the considered vehicle. In particular, we simulate up to 3 ToD service instances per BS, leading to a maximum of 12 additional HVs. This represents a slightly more challenging environment than what described in the 5GAA specifications, which limit to 10 the number of ToD services active in an area of 1 square kilometer. In order to reduce the variability of the BS resource usage by background ToD service, we assume that background HVs have always a good channel quality. This setting allows us to better understand the behavior of the main ToD service without introducing arbitrary side effects of background traffic.

C. Application

At the application level, we can tune two parameters: the frame rate and the video quality. The frame rate determines the status update interval, while the video quality characterizes the size of the status packet. We consider two values for the frame rate, 25 and 60 frames/s, and two different packet sizes, 33 and 66 kB. The configuration with 60 frames/s and 66 kB packet size leads to a total uplink bitrate of 32 Mb/s, a value that agrees with the 5GAA requirements.

As for the background traffic, we configure the application using the lowest settings, i.e., 25 frames/s and 33 kB of packet size, which leads to an average bitrate of 6.5 Mb/s each.

D. ToD Agent

The ToD agent is an extension of the base proportional–integral–derivative (PID) controller available in CARLA. It is a combination of two PID controllers (lateral and longitudinal) to perform the low-level control of the HV from the client side. The agent aims at traveling at the maximum allowed speed while remaining at the center of the lane. Given a start and destination point, it computes a trajectory as a set of waypoints at a prefixed distance, connected through linear segments. To obtain the route toward the destination we use an inter-waypoint distance of 0.5 m, which represents a suitable value for approximating the sharpest turns, e.g., those in the town center in Fig. 5. The agent aims at reaching the next waypoint by computing the suitable corrective maneuvers to reach that waypoint following a linear segment from the current vehicle position. However, the CARLA approach exhibits good performance only if the vehicle is not too far from the ideal trajectory and requires a careful setting of the inter-waypoint distance parameter. To overcome this problem, we adopt the trajectory optimization algorithm presented in [20], which implements a threshold-guided sampling. The key intuition behind this approach is to discard samples that do not reveal a relative change in the course of a trajectory. This produces a trajectory with a dynamic inter-waypoint distance as a function of the road curvature.

We configure different simulation scenarios by varying some of the system parameters: the frame rate (*fps*), the status packet size (*pkt*), the numerology index (μ), and the number of background ToD services (N_{bg}). Using the values of the parameters shown in Table I, we simulate 32 different configurations, performing 10 runs for each scenario.

VI. Performance evaluation

In this section, we present the results of the simulations to evaluate the consistency of our simulation framework and to provide a preliminary analysis about the feasibility of ToD serving under realistic 5G network conditions.

A. Metrics

To guarantee safety, the remote operator must be able to keep HV as close as possible to the center of the lane. The optimal trajectory is the one that maintains the vehicle perfectly aligned to the lane center. However, strictly following the optimal trajectory is a challenging and unreasonably too precise requirement. In particular, the experiments have shown that also the agent deployed on board HV is unable to perfectly follow the optimal trajectory, mainly due to the actuation time (t_a in Fig. 1). For this reason, we evaluate the performance of the remote operator by measuring the trajectory error w.r.t. a benchmark obtained by placing the agent on board the vehicle, i.e., removing all delay components except the actuation time. To compare two trajectories, we implement a simple similarity algorithm based on the Dynamic Time Warping (DTW) metric [21]. In particular, we exploit the best-match computation of DTW to measure the trajectory error for every single point.

As for network metrics, we measure round trip time (RTT), considering the delay components from t_u to t_d in Fig. 1, the mobile channel quality indicator (CQI) in uplink direction, and the percentage of the resource blocks allocated by the base station that is serving HV.

For all metrics we report the average value[7] over all the simulation runs.

B. Base scenario

In the first set of experiments, we vary the packet size and the numerology index, while we assume no background traffic and fixed frame rate at 25 frames/s. These configurations lead to an uplink bitrate of about 6.6 and 13.2 Mbps with 33 and 66 kB packet size, respectively. These settings allow us to investigate the feasibility of the ToD service under mid-low application requirements. In Fig. 6 we report all the metrics described above as a function of the HV position along the trajectory. We also indicate the points when handovers are triggered (dashed vertical lines). By observing the trajectory error (top figure), we can see that the error is always below 0.3 m, which is less than 10% of displacement w.r.t. the reference trajectory considering an average road lane 3 m wide[8]. The relatively largest error is in the proximity of the first entrance ramp around 600 m, this being the most difficult part of the trajectory (see Section V-A).

The very small trajectory error indicates that the considered RAN configurations are capable of handling the entire uplink traffic in all configurations. As a matter of fact, RTT is at most 65 ms in case of 66 kB packet size and numerology 0, which is far below the 5GAA requirements. Moreover, the slight increase in RTT is coherent with the decrease in channel quality (see the CQI plot), which leads to more retransmissions at the radio link level occupying some more base station resources, up to 50% in the worst case, as shown in the bottom figure.

[7]Since the four metrics have different sampling periods, we perform a common re-sampling every 250 ms averaging over the sample period.

[8]In Europe, the minimum widths of lanes are generally between 2.5 to 3.25 meters [22]

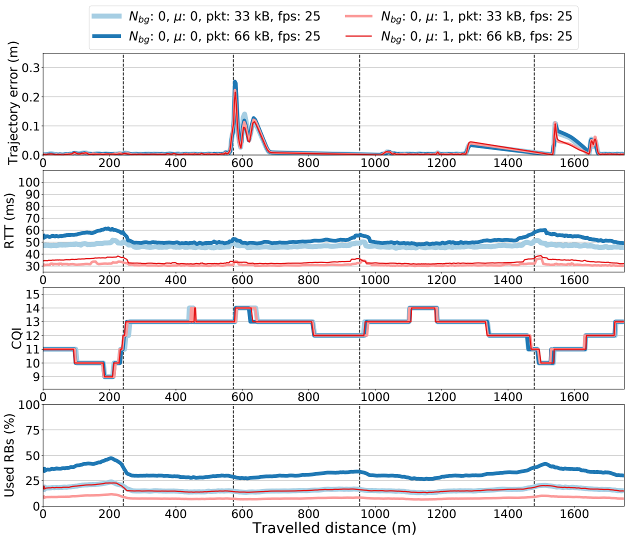

Fig. 6: *Base scenario*. 25 frame/s, packet size 33 and 66 kB, numerology index (μ) 0 and 1, and no background traffic, $N_{bg} = 0$.

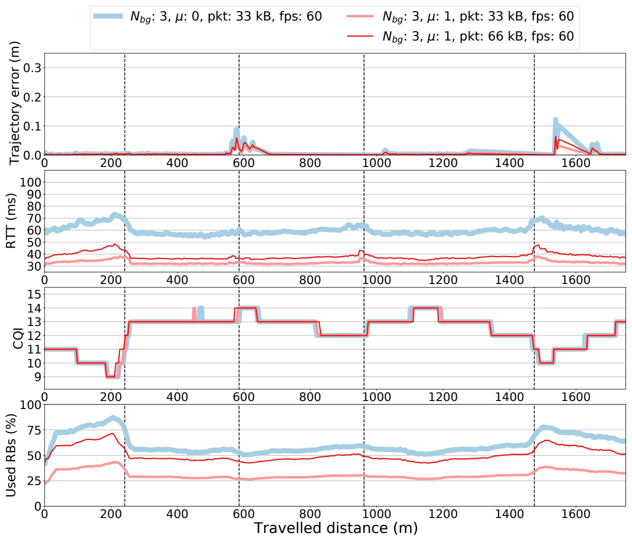

Fig. 8: *Worst case scenario*. 60 frame/s, packet size 33 and 66 kB, numerology index (μ) 0 and 1, and maximum background traffic, $N_{bg} = 3$. Configuration with packet size 66 kB and μ 0 is not shown because it ends with an accident.

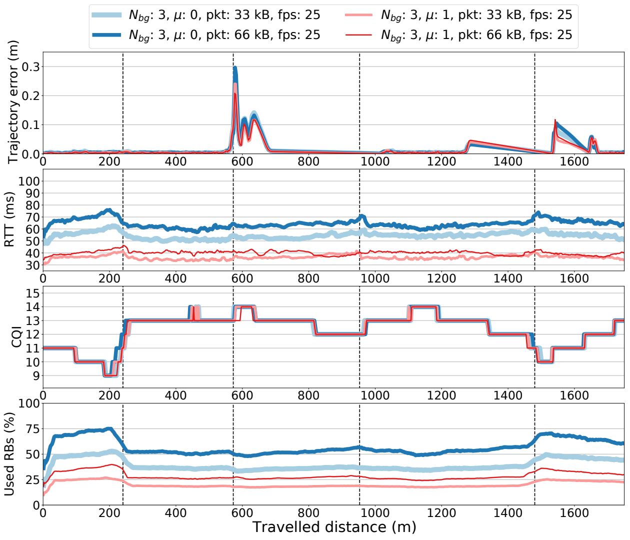

Fig. 7: *Impact of background traffic*. 25 frame/s, packet size 33 and 66 kB, numerology index (μ) 0 and 1, and maximum background traffic, $N_{bg} = 3$.

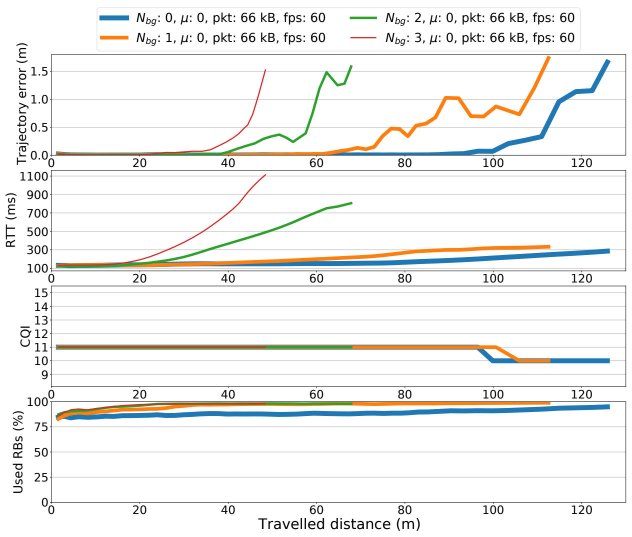

Fig. 9: *Accident scenarios*. 60 frame/s, packet size 66 kB, numerology index (μ) 0, and all conditions of background traffic, N_{bg} from 0 to 3.

C. Impact of background traffic

The second set of simulations investigates the impact of background network traffic on the ToD service. We consider the same network and application configuration as before, and we add up to 3 background ToD services at each base station. Here we report the results only for $N_{bg} = 3$, which represents the most challenging configuration.

The results of this second set of simulations are reported in Fig. 7. Comparing the trajectory error with the one in Fig. 6, we observe a negligible worsening with a maximum error of 0.3 m with the configuration of 66 kB packet size and $\mu = 0$. As for the previous experiments, the network is still able to process all uplink traffic (around 33 Mbps at most), at the cost of increasing RTT and resource blocks usage. In particular, RTT is up to 10 ms more than in the case with no background traffic, while base station resource utilization increases up to 75%. Also in this set of experiments, the network delays do not cross the limit of 100 ms and the ToD service is able to run properly throughout the trajectory.

D. Worst case and accident scenarios

In the last set of experiments, we consider more challenging application settings, configuring all simulations with 60 frame/s, thus meeting the 5GAA specifications.

In Fig. 8, we report results for the most demanding configuration, i.e., 60 frame/s and maximum amount of background traffic. As we can observe, the configuration with 66 kB packet size and numerology 0 is missing because all runs terminate with an accident, as we will discuss later. On the contrary, in the other three configurations, the remote operator is able to complete the journey. Compared to the results shown in the previous sections, we note that the trajectory error is significantly lower. This result is due to a higher frequency of the control loop, which reduces the interval between two consecutive status samplings (t_{vm} in Fig. 1).

Looking at the RTT metric, we observe a larger difference between numerology configurations w.r.t. the previous simulations. In particular, we can see that numerology index 0 leads to RTT above 70 ms, whereas with numerology

1 RTT remains below 50 ms all over the trajectory. As for base station resources, the more demanding application configuration, around 51.5 Mbps with 66 kB packet size, leads to considerably high utilization of RBs, up to 80% in the case of $\mu = 0$ in presence of non-optimal channel quality.

We now briefly discuss scenarios leading to car accidents. In Fig. 9 we report the results obtained for the configurations having 66 kB packet size and numerology index 0, considering all levels of background traffic. Since the behavior of the remote agent is highly unpredictable with very large network delays, we show results until the displacement w.r.t. the benchmark trajectory is below 1.5-1.7 m. Even though 1 m leads to a non-negligible drift into another lane, we tolerate extra displacement to allow the remote agent to recover. From the figure we clearly see that in this case the network is unable to handle the uplink traffic, which leads to a rapid increase in the trajectory error. In the bottom figure, we can see the cause of the failure. The resources at the base station rapidly saturate, leading to an increase in network delays, which in this scenario is always above the limit of 100 ms. The results in Fig. 9 show that the base station does not have enough resources to support the highest application settings even in presence of good channel quality (CQI is equal to 11 at the beginning of the journey).

To summarize, the performance analysis reported in this section has shown the correctness of our simulation framework and provided preliminary insights into the network requirements to properly deploy ToD service. In particular, we have observed that the performance is highly affected by the availability of resources at the base station; thus, provided enough free RBs within the network latency constraints, the ToD service is feasible. On the contrary, a saturation of BS's resources leads to rapid degradation of performance with dangerous consequences.

VII. Conclusions

In this paper, we presented a sophisticated simulation framework for tele-operated driving that integrates CARLA and OMNeT++, providing a detailed modeling of the driven vehicle dynamics as well as of the data transfer necessary to remotely drive the vehicle. The resulting framework provides high flexibility in configuring many parameters of both network and application components of the ToD service. Moreover, the simulator is suitable for exploring the parameter space relying on the high repeatability of the scenario settings. We have demonstrated some of the main features of the simulation framework in a preliminary investigation of the effectiveness of tele-operated driving, verifying good performance for those cases in which the specifications of the 5GAA are met. On the contrary, longer round trip times in the data transfer lead in a number of cases to the inability to remotely drive the vehicle, possibly causing accidents.

While here we have presented a novel co-simulation framework and its potential applications, a more detailed and exhaustive simulation campaign is necessary to completely characterize in a number of different driving scenarios, which we leave as future work. In there we will, e.g., also evaluate the impact of interfering vehicular traffic, pedestrians and obstacles, so as to be able to assess safety and efficiency of ToD in cellular networks.

References

[1] SAE International Automotive, "Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles." https://www.sae.org/standards/content/j3016_202104/, 2021.

[2] 5GAA, "Tele-Operated Driving (ToD): Use Cases and Technical Requirements." https://5gaa.org/wp-content/uploads/2021/08/ToD_D1.1-Use-Cases-and-Technical-Requirements.pdf, 2021.

[3] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, pp. 1–16, 2017.

[4] A. Varga, "Omnet++," in *Modeling and Tools for Network Simulation* (K. Wehrle, M. Güneş, and J. Gross, eds.), pp. 35–59, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[5] T. Zhang, "Toward automated vehicle teleoperation: Vision, opportunities, and challenges," *IEEE Internet of Things Journal*, vol. 7, no. 12, pp. 11347–11354, 2020.

[6] S. Gnatzig, F. Chucholowski, T. Tang, and M. Lienkamp, "A system design for teleoperated road vehicles," in *International Conference on Informatics in Control, Automation and Robotics*, 2013.

[7] R. Liu, D. Kwak, S. Devarakonda, K. Bekris, and L. Iftode, "Investigating remote driving over the lte network," in *Proceedings of the 9th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, AutomotiveUI '17, (New York, NY, USA), p. 264–269, Association for Computing Machinery, 2017.

[8] P. Pérez, J. Ruiz, I. Benito, and R. López, "A parametric quality model to evaluate the performance of tele-operated driving services over 5g networks," *Multimedia Tools and Applications*, vol. 81, pp. 12287–12303, Apr 2022.

[9] S. Neumeier, P. Wintersberger, A.-K. Frison, A. Becher, C. Facchi, and A. Riener, "Teleoperation: The holy grail to solve problems of automated driving? sure, but latency matters," in *Proceedings of the 11th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, AutomotiveUI '19, (New York, NY, USA), p. 186–197, Association for Computing Machinery, 2019.

[10] M. Hofbauer, C. B. Kuhn, G. Petrovic, and E. Steinbach, "Telecarla: An open source extension of the carla simulator for teleoperated driving research using off-the-shelf components," in *2020 IEEE Intelligent Vehicles Symposium (IV)*, pp. 335–340, 2020.

[11] A. Schimpe, J. Feiler, S. Hoffmann, D. Majstorović, and F. Diermeyer, "Open source software for teleoperated driving," in *2022 International Conference on Connected Vehicle and Expo (ICCVE)*, pp. 1–6, 2022.

[12] S. Neumeier, M. Höpp, and C. Facchi, "Yet another driving simulator openrouts3d: The driving simulator for teleoperated driving," in *2019 IEEE International Conference on Connected Vehicles and Expo (ICCVE)*, pp. 1–6, 2019.

[13] F. Mason, M. Drago, T. Zugno, M. Giordani, M. Boban, and M. Zorzi, "A reinforcement learning framework for pqos in a teleoperated driving scenario," in *2022 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 114–119, 2022.

[14] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic traffic simulation using SUMO," in *IEEE International Conference on Intelligent Transportation Systems*, 2018.

[15] J.-M. Georg, J. Feiler, S. Hoffmann, and F. Diermeyer, "Sensor and actuator latency during teleoperation of automated vehicles," in *2020 IEEE Intelligent Vehicles Symposium (IV)*, pp. 760–766, 2020.

[16] G. Nardini, D. Sabella, G. Stea, P. Thakkar, and A. Virdis, "Simu5g–an omnet++ library for end-to-end performance evaluation of 5g networks," *IEEE Access*, vol. 8, pp. 181176–181191, 2020.

[17] C. Sommer, R. German, and F. Dressler, "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis," *IEEE Transactions on Mobile Computing (TMC)*, vol. 10, pp. 3–15, January 2011.

[18] M. Hofbauer, C. B. Kuhn, M. Khlifi, G. Petrovic, and E. Steinbach, "Traffic-aware multi-view video stream adaptation for teleoperated driving," in *2022 IEEE 95th Vehicular Technology Conference: (VTC2022-Spring)*, pp. 1–7, 2022.

[19] S. Neumeier, V. Bajpai, M. Neumeier, C. Facchi, and J. Ott, "Data rate reduction for video streams in teleoperated driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 10, pp. 19145–19160, 2022.

[20] M. Potamias, K. Patroumpas, and T. Sellis, "Sampling trajectory streams with spatiotemporal criteria," in *18th International Conference on Scientific and Statistical Database Management (SSDBM'06)*, pp. 275–284, 2006.

[21] R. S. D. Sousa, A. Boukerche, and A. A. F. Loureiro, "Vehicle trajectory similarity: Models, methods, and applications," *ACM Comput. Surv.*, vol. 53, sep 2020.

[22] SAFESTAR FP4 EU Project, "Safety Standards for Road Design and Redesign." https://trimis.ec.europa.eu/project/safety-standards-road-design-and-redesign, 1998.