Multi-Technology Cooperative Driving: An Analysis Based on PLEXE

Michele Segata, *Member, IEEE*, Renato Lo Cigno, *Senior Member, IEEE*, Tobias Hardes, *Student Member, IEEE*, Julian Heinovski, *Student Member, IEEE*, Max Schettler, *Student Member, IEEE*, Bastian Bloessl, *Member, IEEE*, Christoph Sommer, *Member, IEEE*, and Falko Dressler, *Fellow, IEEE*

We report in this Additional Material some background and results that may be useful to fully appreciate and exploit the contribution we provide in the main paper. Section 1 reports a short primer on platooning and longitudinal vehicles' control. The material is all available in the original publications, but we think that having it all available at hand with a uniform notation makes the paper easier to read and appreciate. Section 2 presents the results and plots that are not included in the main paper because they are somewhat repetitive, but can be useful to build further contributions beyond our work. Finally, Section 3 adds some details on PLEXE and show some use cases that highlight the potentialities of the tool.

1 PLATOONING CONTROL

The goal of this short Section is simply to provide the reader with all the information needed to easily follow the paper without the need to browse in the literature and with a uniform notation. All the equations we provide here are reported from the cited original works, and correspond to the implementation in PLEXE. We assume the reader has some familiarity with control theory and distributed systems as this is not meant to be a primer or a tutorial on cooperative driving and platooning. Symbols are described in the text as soon as they are introduced, but we summarize them in Table 1 for reader's convenience. In the following x_i, v_i and a_i indicate the position (relative to the vehicle in front), the speed, and the acceleration of the vehicle in position i in the platoon. A dot on a symbol, as in $\dot{a}$, indicate a time derivative.

- *M. Segata is with the Faculty of Computer Science, Free University of Bolzano, Italy, E-mail: michele.segata@unibz.it.*
- *R. Lo Cigno is with DII, University of Brescia, Italy, E-mail: renato.locigno@unibs.it.*
- *T. Hardes is with TU Dresden, Faculty of Computer Science, Germany and the Software Innovation Campus Paderborn (SICP), Paderborn University, Germany, E-mail: tobias.hardes@upb.de.*
- *J. Heinovski, M. Schettler, and F. Dressler are with the School for Electrical Engineering and Computer Science, TU Berlin, Germany, E-Mail: {heinovksi, schettler, dressler}@ccs-labs.org.*
- *C. Sommer is with TU Dresden, Faculty of Computer Science, Germany, E-mail: sommer@cms-labs.org.*
- *B. Bloessl is with the Secure Mobile Networking Lab, TU Darmstadt, Germany, E-mail: mail@bastibl.net.*

Table 1
List of symbols.

Symbol	Meaning
$\dot{\square}$	first derivative of $\square$
$\square_i$	value of a variable $\square$ for the i-th vehicle in a platoon
u	control input (desired acceleration)
a	acceleration
v	speed
x	position
l	length of the vehicle
τ	first-order lag time constant
Δ_t	simulation time step
H	time headway for inter-vehicle gap (ACC and Ploeg)
λ	weight factor between spacing and speed errors (ACC)
k_p	distance error gain (proportional gain, Ploeg)
k_d	speed error gain (derivative gain, Ploeg)
d_d	fixed, desired gap (PATH)
C_1	leader and preceding vehicle acceleration weight (PATH)
ξ	controller damping ratio (PATH)
ω_n	controller bandwidth (PATH)

The first requirement to implement an automatic control on a vehicle is knowing (and modeling) it's dynamic behavior, or, in other words, understand how the vehicle responds to an input or command normally given in terms of desired acceleration.

The simplest possible model available in PLEXE is a first order lag characterized by the following differential equation for the acceleration a and the control input u (which is the command computed by the control law or, equivalently, a desired acceleration):

$$\dot{a} = -\frac{1}{\tau}a + \frac{1}{\tau}u. \tag{1}$$

In Equation (1) τ indicates the time constant of the lag: the higher the value, the slower the response of the engine and the braking system. A typical value found in the literature is 500 ms [1]. Within the simulator, Equation (1) is implemented using the following discrete update rule

$$a[k+1] = \alpha \cdot u[k] + (1-\alpha) \cdot a[k], \quad \alpha = \frac{\Delta_t}{\tau + \Delta_t}, \tag{2}$$

where k is the simulation step and Δ_t the sampling time of the simulator. PLEXE includes more sophisticated models of vehicles' dynamics that include the role or air-drag, the engine power, the gears and so forth. As these models are

brand-and-model specific, they are not suited to derive general results, but they can be used to verify general hypotheses onto specific vehicles, and can also be used as templates to include other models and possibly to state minimum requirements that vehicles have to meet to be part of a cooperative driving system. The interested reader can find a complete description of the model in [2, Section 2.3].

In our contribution we assume that when communications are not available the vehicle falls back to a standard, radar-based ACC. ACC implementations may differ slightly one another, but they can all be modeled by the control law in Equation (3) reported in [3, Chapter 6], which assumes a constant headway-time between vehicles, as mandated with human drivers, and thus a distance between platoon vehicles that increases with speed.

$$u_i = -\frac{1}{H}\left(\dot{\varepsilon}_i + \lambda\delta_i\right) \tag{3}$$

$$\delta_i = x_i - x_{i-1} + l_{i-1} + Hv_i \tag{4}$$

$$\dot{\varepsilon}_i = v_i - v_{i-1} \tag{5}$$

In Equation (3), H is the time headway and l_i is the length of vehicle i.

We note incidentally that this control law assumes that the radar can efficiently estimate the speed difference of the own vehicle from the vehicle in front, which is not necessarily true, thus this model can be somewhat optimistic in the evaluation of ACC performance. The parameter λ controls how aggressive the ACC is with respect to the distance from the vehicle in front. We used a constant value often reported in the literature in the experiments, which, as we have shown, can easily lead to accidents in case of abrupt transition from PATH Cooperative Adaptive Cruise Control (CACC) (see Equation (7)) to ACC. A larger value of λ, or better, a dynamic λ can be used to reduce the probability of rear-end collisions, but this is clearly beyond the scope of this paper. Also exploring what happens when the control of the vehicle is given back to the human driver instead of an ACC is interesting, but to do this analysis the Intelligent Driver Model (IDM) [4] and Krauss [5] car-following models already available in PLEXE are not enough, because they do not account for the "surprise" of a driver that all of a sudden is requested to be in control of a vehicle they were not driving till the moment before.

Integral parts of our study are the Ploeg's and PATH CACC [1], [6] that we use when all three communication interfaces are available (PATH) or when only one or two are available (Ploeg).

Ploeg's CACC has been designed to mimic an advanced ACC system, thus it follows the constant headway-time model:

$$\begin{aligned}\dot{u}_i = \frac{1}{H}\left(-u_i + k_p\left(x_{i-1} - x_i - l_{i-1} - Hv_i\right)\right. \\ \left. + k_d\left(v_{i-1} - v_i - Ha_i\right) + u_{i-1}\right)\end{aligned} \tag{6}$$

Ploeg's CACC allows a smaller headway-time compared to standard ACC because it can exploit the knowledge of the front vehicle input u_{i-1}, thus discounting the actuation lag of the vehicle in front. Given its design goals Ploeg's CACC is the ideal transition controller toward an ACC, as it can smoothly increase the vehicles' headway-time to the one which is safe for the ACC system.

PATH's CACC goal is instead the maximization of performance in terms of road usage and fuel consumption: both goals require the minimum possible vehicle inter-distance independently from the speed. To achieve this goal it uses information also from the leader of the platoon and not only from the vehicle in front as described by the following equations:

$$\begin{aligned}u_i = \alpha_1 u_{i-1} + \alpha_2 u_0 + \alpha_3(v_i - v_{i-1}) \\ + \alpha_4(v_i - v_0) + \alpha_5(x_i - x_{i-1} + l_{i-1} + d_d)\end{aligned} \tag{7}$$

where

$$\alpha_1 = 1 - C_1; \quad \alpha_2 = C_1; \quad \alpha_5 = -\omega_n^2 \tag{8}$$

$$\alpha_3 = -\left(2\xi - C_1\left(\xi + \sqrt{\xi^2 - 1}\right)\right)\omega_n \tag{9}$$

$$\alpha_4 = -C_1\left(\xi + \sqrt{\xi^2 - 1}\right)\omega_n. \tag{10}$$

Differently from Equations (3) and (6), we have no headway time H but a fixed desired distance d_d regardless of the cruising speed. C_1, ξ, and ω_n are control parameters regulating the weight between leading and preceding vehicle accelerations, the damping ratio, and controller bandwidth, respectively.

PLEXE includes also other CACC models [7]–[9] that we have not used in this study, and adding others is simple given the modularity of the simulation framework.

Table 2
Network parameters.

	Parameter	Value
	Packet payload size	200 Byte
	Beacon frequency	10 Hz
802.11p	Transmission power	20 dBm
	Bit rate	6 Mbit/s
	Noise floor	–95 dBm
	Path-loss model	Free-space, $\alpha = 2$
	Frequency	5.89 GHz
VLC	Transmission power	10 dBm
	Bit rate	1 Mbit/s
	Headlight max tx range	100 m (LOS)
	Taillight max tx range	30 m (LOS)
	Headlight max tx angle	45°
	Taillight max tx angle	60°
C-V2X	Transmission power (UE)	26 dBm
	Transmission power (eNB)	40 dBm
	Frequency	2.1 GHz
	Mode	C-V2X Mode 3 (D2D, eNB assisted)
	Channel configuration	Urban macrocell (SimuLTE provided)

2 Additional Results

This section shows additional results that are not included in the main manuscript. In addition, in Table 2, we list all the main network parameters used in the simulations. Figures 1 and 2 show the acceleration and distance behavior for the naïve fallback mechanisms for scenarios 2 and 3 respectively. It is clear in both cases that the crashes we observe are not a remote chance, but they are intrinsic to an abrupt change from a cooperative driving situation, where vehicles exploit the knowledge of each other dynamics and intentions, to

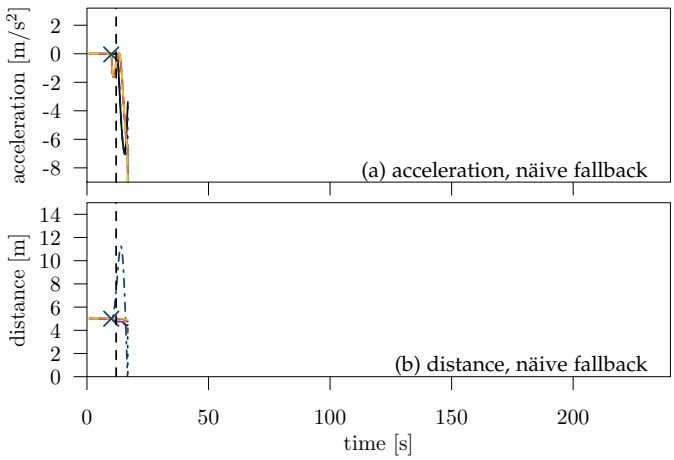


Figure 1. Acceleration and inter-vehicle until the crash for the näive fallback in scenario 2.

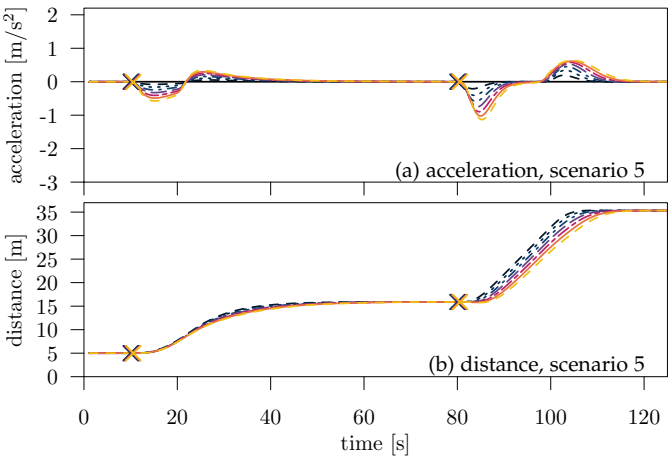


Figure 3. Acceleration and inter-vehicle distances for scenarios 5 (multiple failures).

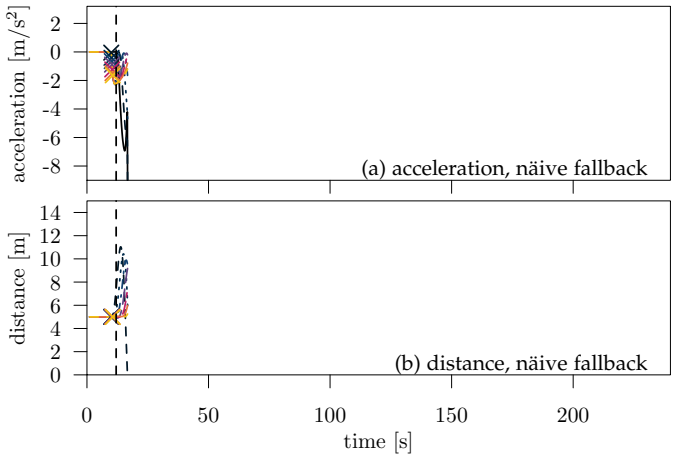


Figure 2. Acceleration and inter-vehicle until the crash for the näive fallback in scenario 3.

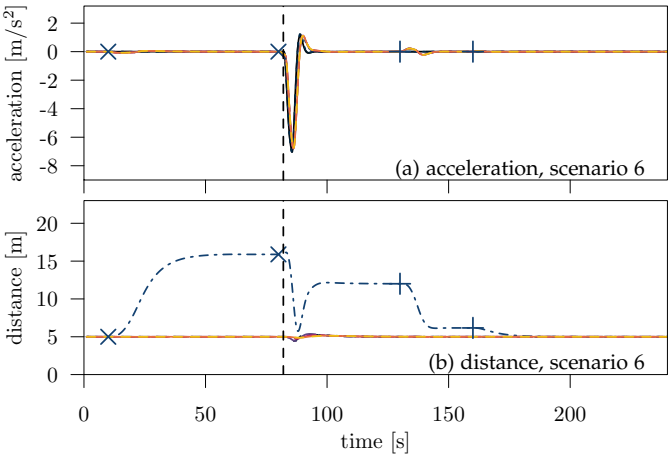


Figure 4. Comparison of acceleration and inter-vehicle distances for scenarios 6 (multiple failures).

one of autonomous driving, where vehicles can rely only on the local sensors. The crashes occur immediately after the communication failure with the emergency braking; in practice none of the vehicles attempt a reaction to V_0 abrupt braking, and collisions occur. The experiments stop as soon as one vehicle collides.

Figure 3 presents the performance of the proposed Safe Autonomous Switchover Algorithm (SafeSwitch) for scenario 5, i.e., when a technology fails for all the vehicles in the platoon at the same time and with failure of two technologies. This can happen in rare cases, for instance if there is a jamming attack on IEEE 802.11p, or because an LTE base station fails, thus stopping to serve the entire platoon. Clearly the occurrence of both cases is extreme. SafeSwitch works as intended, safely distancing vehicles without abrupt accelerations. PLEXE highlights the same amplification of the deceleration observed in scenario 1 (main paper). The amplification is however more severe and this is due to the fact that a different controller (i.e., Ploeg) is used, so the behavior is slightly different.

The situation of scenario 6, multiple failures and recoveries at the same vehicle with an emergency brake after the second failure, leads to the results presented in Figure 4. The deceleration due to the emergency braking is abrupt, but this is unavoidable, and during the braking maneuver the distance of V_3, the one with failing communications, reduces drastically, but remains safe, and then quickly returns to the distance dictated by Ploeg's controller in state F_1, after the communication recovery the platoon returns to normal cruising with PATH's controller.

Finally, Figures 5 and 6 present two additional runs relative to the scenario with realistic communication failures. It is evident that the stochastic pattern of losses affects the quantitative behavior of the platoon, as the dynamic patterns observed in the lower plots of these figures are different and are also different from those reported in the main paper. However, from a qualitative point of view the behavior is consistent, with IEEE 802.11p and LTE losses driving the performance, with Visible Light Communication (VLC) ones often induced by the increased distance between vehicles, a situation that can be recovered only if the other two technologies work well for a long enough period of time, as can be observed in Figure 6. We also observe that handovers sometimes result in burst of losses and sometimes not, hinting that handover procedures can still be improved. This is very important, in light of 5G advent, where with a high density of gNodeB and very frequent handovers, it is important that these latter are loss-free for cooperative

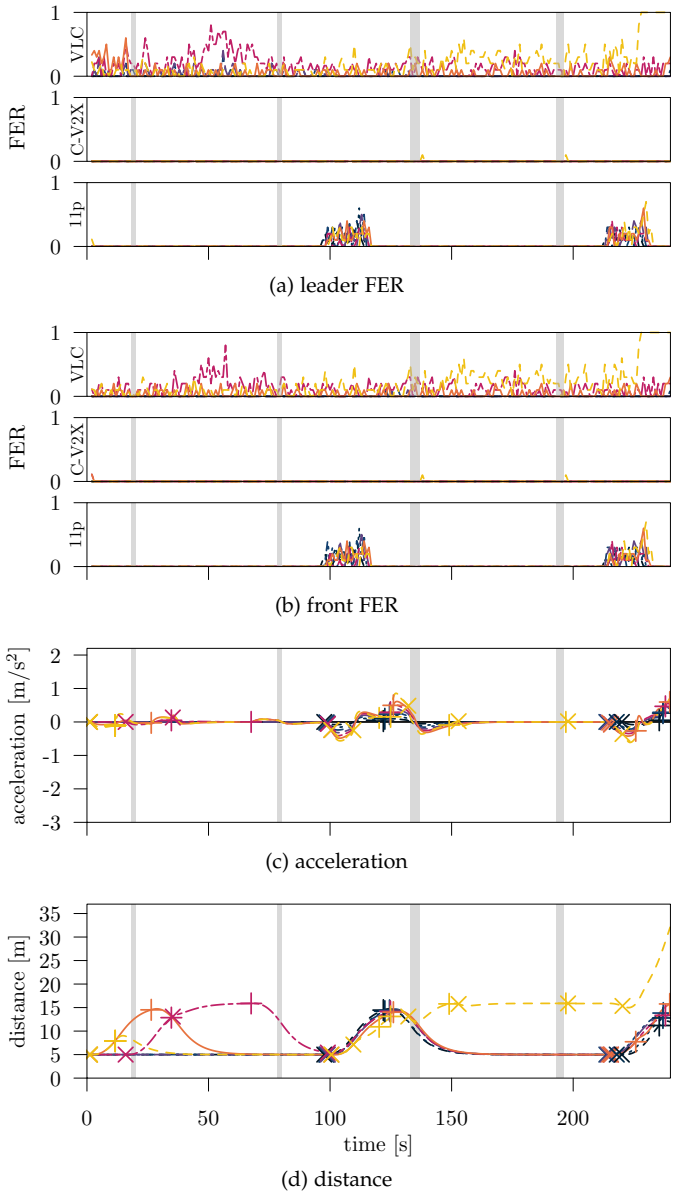


Figure 5. Vehicle dynamics induced by the fallback FSM and frame error rates for the scenario with realistic failures, second replica.

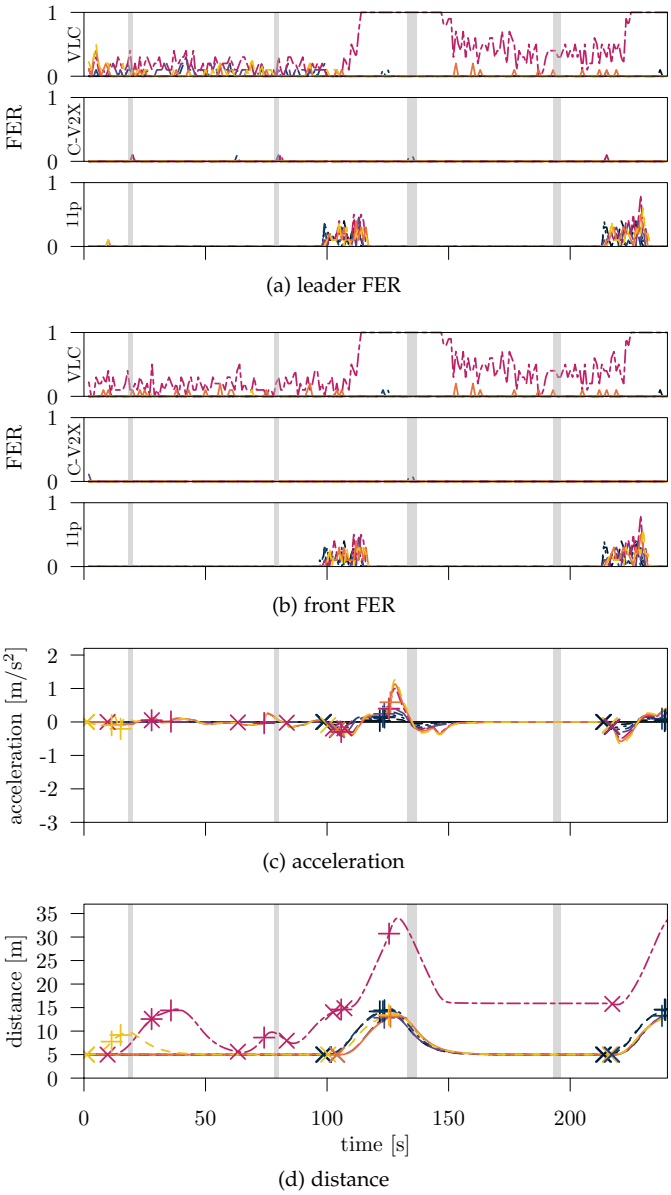


Figure 6. Vehicle dynamics induced by the fallback FSM and frame error rates for the scenario with realistic failures third replica.

driving applications.

3 Sample use cases and subprojects

This section describes the structure of PLEXE's code base for future users, together with the default scenarios it provides. The following description assumes the user to be familiar with the OMNeT++ ecosystem. PLEXE is divided in three main sub-folders: main source code, examples, and subprojects. The implementation of all the functionalities is found under the `src/plexe` folder, starting from the root, and this is where the core of PLEXE resides. The `examples` folder includes all the sample simulations users can play with to get acquainted with the frameworks. The `subprojects` folder, instead, includes the source code and examples that involve external libraries such as VLC or Cellular V2X (C-V2X) modules. By default, PLEXE depends only on SUMO and *Veins*, and the other components are optional.

3.1 Flexibility through dynamic linking

Optional components are managed through a `configure` script, which is present in each subproject. The role of such script is to configure and check the dependencies of the subproject (that is, other frameworks such as *Veins VLC* or *SimuLTE*), generate the `Makefile` needed for building, and the files required to run simulations. To better explain this, we describe the `veins_vlc` subproject, which includes example simulations like the ones provided by basic PLEXE, but its vehicles are using VLC to communicate. We are going to consider the file system structure in listing 1, which shows a partial view of the folder structure and the most relevant files. Listing 2 shows a portion of the `configure` script. Lines 2 to 18 define the dependencies which, in this specific case, are PLEXE, *Veins*, and *Veins VLC*. For each dependency the user needs to provide:

- `name`: a name of the dependency;

- `library`: the name of the shared library which the subproject will be linked against;
- `default_path`: the default root path where the library is located. The user can override this via command line arguments when running the script;
- `versions`: a list of accepted versions of the library;
- `source_folder`: the location (starting from the root path of the library) where the source files are located;
- `lib_folder`: the location (starting from the root path of the library) where the compiled shared library is located;
- `images_folder`: the located (starting from the root path of the library) where additional images provided by the library are located. It is possible to make use of such images (e.g., a car or a pedestrian) when running the OMNeT++ simulation in graphical mode;
- `version_script`: name of a script that prints (or a file that contains) the version of the library.

Lines 20 to 24, instead, define the properties of the `plexe_vlc` subproject itself, i.e., the name of the shared library (being built), and the parameters required to run simulations (the shared library itself and where OMNeT++ will find the `.ned` files).

The script automatically checks for the presence of all the components and that the versions correspond to the required ones, generating an error for the user if such task fails. Upon a successful execution, the script generates a `Makefile` for building the subproject and the script required to run the simulation, which automatically indicates to OMNeT++ where to find all the dependencies. The user can now develop custom OMNeT++ modules (in `src/plexe_vlc`) and simulations (in `examples/platooning_vlc` or any other folder). For the former step, it is important to remember that if the user adds new C++ source files, the `configure` scripts needs to be run again to update the `Makefile`. For example, `PlatoonVlcCar.ned` defines an OMNeT++ communication node representing a vehicle which, differently to standard PLEXE, uses a VLC interface. As the configure script automatically resolves the dependencies, the user can now import the modules provided by *Veins VLC* and sobstitute the 802.11p interface with VLC. The user has clearly the possibility to add a new interface to study heterogeneous communication systems.

To run a simulation, after defining all the classic OMNeT++ configuration files such as `omnetpp.ini` in a folder of choice, the user can simply exploit the script generated by `configure`. For example, running `../../bin/plexe_vlc_run` from the `examples/platooning_vlc` folder will start the OMNeT++ GUI, permitting the user to choose which simulation to run. The user can also specify to run through command line arguments, as in standard PLEXE.

This brief description shows how flexible PLEXE is even considering its complexity, and that integrating a new communication technology or any other OMNeT++ framework simply requires to add a dependency to a project.

3.2 Base scenarios

Base PLEXE scenarios are located in the `examples` folder and the main one resides inside the platooning subfolder. This example simulates two scenarios using the available ACC and CACC algorithms. The scenarios include a sinusoidal speed profile, a classical test of the stability of control systems, and an emergency braking scenario, which is of obvious interest for safety reasons. Obtaining simple performance metrics of control systems starting from the examples is just a matter of changing the parameters. For example, by reconfiguring the emergency braking scenario and introducing an artificial frame error rate ranging from 0 % to 80 %, we obtain the results in Figure 7. In the scenario we have a platoon of 8 vehicles with the leader performing an emergency braking maneuver with a deceleration of $8\,\mathrm{m/s^2}$. We repeat each simulation point 10 times and we compute the minimum inter-vehicle distance between each pair of vehicles. We then plot the average over the 10 repetitions with the relative 95 % confidence intervals. In the graph, we only show the results for the PATH [6] and the Ploeg [1] CACCs.

```
1  /Users/user/src/
2  |-- plexe/
3  |   |-- print-plexe-version
4  |   |-- src/
5  |   |   |-- libplexe.dylib
6  |   |   |-- plexe/
7  |   |-- subprojects/
8  |       |-- plexe_vlc/
9  |           |-- configure
10 |           |-- bin/
11 |               |-- plexe_vlc_run
12 |           |-- examples/platooning_vlc/
13 |           |   |-- omnetpp.ini
14 |           |-- src/plexe_vlc/
15 |               | PlatoonVlcCar.ned
16 |               | VlcRepropagationProtocol.cc
17 |               | VlcRepropagationProtocol.h
18 |               | VlcRepropagationProtocol.ned
19 |-- veins/
20 |   |-- print-veins-version
21 |   |-- src/
22 |       |-- libveins.dylib
23 |       |-- veins/
24 |-- veins-vlc/
25     |-- print-veins_vlc-version
26     |-- src/
27         |-- libveins-vlc.dylib
28         |-- veins-vlc/
```

Listing 1. File system structure (only most relevant folders and files shown).

The main differences between the two is that the PATH CACC employs a leader- and predecessor-following control topology with a fixed inter-vehicle spacing policy (which in our simulation is set to 5 m), while Ploeg's CACC employs a predecessor-following control topology with a time-headway spacing policy of the form

$$d = d_0 + Hv, \tag{11}$$

where $H = 0.5\,\mathrm{s}$ is the time-headway and v is the speed in m/s. The $d_0 = 2\,\mathrm{m}$ term is defined as the stand-still distance, which avoids the vehicles colliding with each other when the speed goes to zero.

What the figure shows is, first of all, the robustness of CACCs to packet losses. The performance is unaffected for losses up to 20 %, and for vehicles to become dangerously close we need to have packet loss rates higher than 50 %.

This is a very basic scenario, but it is of extreme interest in the cooperative driving community, and obtaining such

```
[...]
plexe = Library(name="Plexe", library="plexe", default_path="../../",
                versions=["3.0"], source_folder="src/plexe",
                lib_folder="src", images_folder="images",
                version_script="print-plexe-version")
veins = Library(name="Veins", library="veins", default_path="../../../veins",
                versions=["5.1"], source_folder="src/veins", lib_folder="src",
                images_folder="images", version_script="print-veins-version")
veins_vlc = Library(name="Veins␣VLC", library="veins-vlc",
                    default_path="../../../veins_vlc", versions=["1.0"],
                    source_folder="src/veins-vlc", lib_folder="src",
                    images_folder="images",
                    version_script="print-veins_vlc-version")

libraries = LibraryChecker()
libraries.add_lib(plexe)
libraries.add_lib(veins)
libraries.add_lib(veins_vlc)

makemake_flags = ["-f", "--deep", "--no-deep-includes", "--make-so", "-I", ".", "-o", "plexe_vlc",
                  "-O", "out", "-p", "PLEXE_VLC"]
run_libs = [join("src", "plexe_vlc")]
run_neds = [join("src", "plexe_vlc")]
run_imgs = []

libraries.check_libraries(makemake_flags, run_libs, run_neds, run_imgs)
[...]
```

Listing 2. Partial content of the configure file of the veins_vlc subproject.

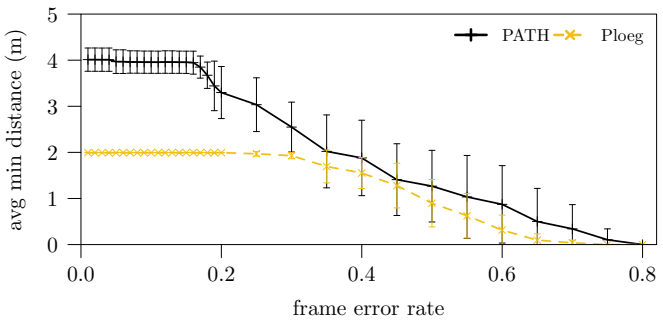


Figure 7. Average minimum inter-vehicle distance with 95 % confidence intervals in an emergency braking scenario as function of the frame error rate for the PATH and the Ploeg CACCs.

results is a matter of a few minutes.

Another sample scenario of interest is the aforementioned join at back maneuver. In this example, a lone vehicle joins an existing platoon of 4 cars by first requesting permission to join, then being instructed to approach the platoon, and finally joining the platoon.

Other examples include a scenario where a human vehicle is inserted into the simulation, to show how to simulate mixed scenarios where human-driven vehicles can interfere from a mobility and a communication perspective. In addition, we find a simple scenarios showing the behavior of the realistic engine model by simulating a drag race between three different types of vehicle.

3.3 Heterogeneous scenarios

PLEXE includes sample scenarios for the subprojects, which deals with VLC and C-V2X. In the VLC subproject, we have the same main scenario (sinusoidal plus emergency braking) but using VLC as the underlying communication technology as previously described. In addition, there is a modified communication protocol that performs re-propagation of the beacon messages as VLC works only in direct line-of-sight.

In the C-V2X subproject, instead, we have two platoons running on a ring-like freeway that are located far apart. The two platoons use the LTE uplink/downlink standard to communicate with a centralized Traffic Authority (TA) server. One platoon queries the TA for other platoons and, upon receiving the reply, requests the TA guidance to approach such platoon with the aim of merging. Once the approaching platoon is close enough to the other, the TA demands the former to contact the latter and performs the merge maneuver autonomously using direct Vehicle to Vehicle communication (V2V) communication via IEEE 802.11p. In addition, V2V beacons with control information are sent redundantly using C-V2X Mode 3. This example is particularly important as it shows how vehicles can be coordinated on different levels (locally with V2V communication and remotely through the infrastructure) and how easily PLEXE enables cooperative driving studies with heterogeneous networking technologies.

The final subproject (named `plexe_hetnet`) includes a ring road scenario with a platoon of vehicles using multiple communication technologies simultaneously to communicate in a V2V fashion. This is the scenario on top of which we developed the evaluation of SafeSwitch, and it can be extremely useful to users to understand how to integrate multiple communication technologies into their cooperative driving scenarios or even integrate additional simulation frameworks.

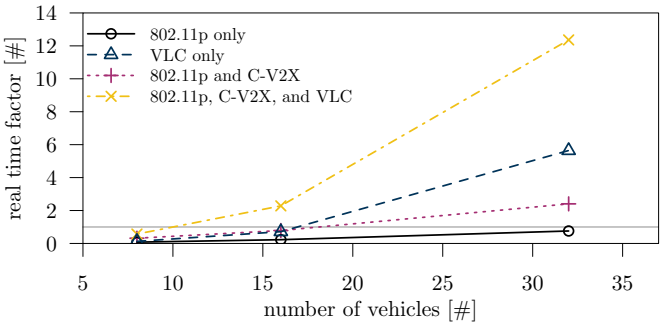
One fundamental aspect which always represents a parameter of choice when selecting a simulation framework is scalability. The scalability of PLEXE heavily depends on the communication models required for the analysis. Some of them can be very demanding in terms of computation. For example, *Veins VLC* employs geometrical models to compute the effect of vehicle shadowing, which clearly cannot be

neglected. In the past, when considering IEEE 802.11p only, we could easily simulate hundreds of vehicles [10], but such a large number of vehicles would be difficult to handle when using very detailed communication models. To give the reader an idea, we run a set of simulations with a single platoon composed by 8, 16, and 32 cars, using different communication technologies, and measuring the execution time. In the simulation, each vehicle sends 10 broadcast frames per second. Figure 8 plots the real time factor of the simulation, which is defined as follows. Let t_s and t_w the time elapsed in the simulation and in the real world, respectively. The real time factor is defined as $f = t_w/t_s$. For example, if $f = 0.1$, it means that simulating 1 s requires 100 ms in the real world. On the contrary, if $f = 10$ the user needs to wait 10 s for each second within the simulation. Simplistically speaking, the lower the value of f, the better. Clearly, the real time factor heavily depends on the hardware. The results in Figure 8 are obtained on a 2018 MacBook Pro with an Intel i9 processor. Running the simulations on a different hardware would quantitatively change the results, but qualitatively they should be, in general, the same.

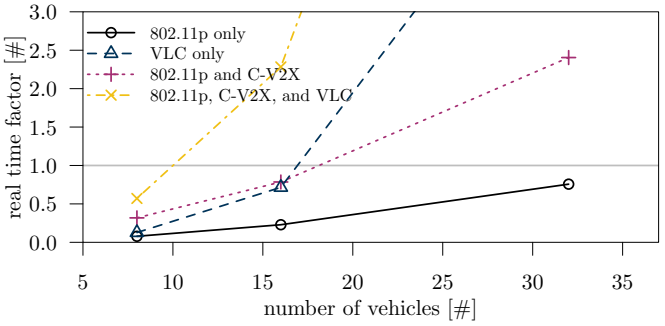
First of all, the plot highlights a well-known fact about network simulations, i.e., that the real time factor increases more than linearly in the number of network nodes. This is true when all nodes communicate with each other because, disregarding optimizations, the communication model needs to compute the probability of reception for each node in the simulation, so the complexity intuitively increases quadratically. Overall, in the worst case, i.e., with 32 cars and 3 simultaneous communication technologies, the real time factor is roughly 12, meaning that it requires 120 s in the real world to simulate a scenario of 10 s in PLEXE.

The graph finally highlights the impact of communication models: the more the technologies, the higher the simulation time. The IEEE 802.11p model provided by *Veins* is the most efficient one. When using only this technology, even a simulation with 32 vehicles runs faster than real time. What it is interesting to observe is the huge impact of the VLC model. For 8 and 16 vehicles, the VLC model is faster than the simulations using IEEE 802.11p and C-V2X, but then its real time factor drastically increases for the simulation with 32 cars. As mentioned before, the VLC channel layer requires geometrical computations to calculate the effects of shadowing. For a small number of vehicles, the computational effort is limited, but this quickly grows as we increase the nodes in the simulation, dominating over all the other algorithmic components.

These results show that PLEXE can potentially scale but there is clearly a limitation induced by the models that researchers need to consider. It is thus necessary to select the communication models and the scale of the simulation depending on the required granularity. If a study focuses on the physical layer and requires very low level channel details (e.g., ray tracing), it will certainly be unfeasible to run a simulation with hundreds of cars. On the other hand, if the user is interested in traffic-related metrics for which a large number of vehicles is required (e.g., throughput), considering multiple communication technologies might not be necessary. It is thus needed to find the right balance depending on the requirements but, as we have clearly shown, the flexibility of PLEXE enables its users to easily tune and configure the simulator for their purposes.



(a) full y axis



(b) zoomed y axis

Figure 8. Simulation real time factor as function of the number of vehicles, for different communication models. The gray line highlights a real time factor of 1.

REFERENCES

[1] J. Ploeg, B. Scheepers, E. van Nunen, N. van de Wouw, and H. Nijmeijer, "Design and Experimental Evaluation of Cooperative Adaptive Cruise Control," in *IEEE ITSC 2011*, Washington, D.C.: IEEE, Oct. 2011, pp. 260–265.

[2] M. Segata, "Safe and Efficient Communication Protocols for Platooning Control," PhD Thesis, University of Innsbruck, Innsbruck, Austria, Feb. 2016.

[3] R. Rajamani, *Vehicle Dynamics and Control*, 2nd ed. Springer, 2012.

[4] M. Treiber, A. Hennecke, and D. Helbing, "Congested Traffic States in Empirical Observations and Microscopic Simulations," *PRE*, vol. 62, no. 2, pp. 1805–1824, Aug. 2000.

[5] S. Krauß, P. Wagner, and C. Gawron, "Metastable states in a microscopic model of traffic flow," *APS Physical Review E*, vol. 55, no. 5, pp. 5597–5602, May 1997.

[6] R. Rajamani, H.-S. Tan, B. K. Law, and W.-B. Zhang, "Demonstration of Integrated Longitudinal and Lateral Control for the Operation of Automated Vehicles in Platoons," *TCST*, vol. 8, no. 4, pp. 695–708, Jul. 2000.

[7] A. Ali, G. Garcia, and P. Martinet, "The Flatbed Platoon Towing Model for Safe and Dense Platooning on Highways," *IEEE Intelligent Transportation Systems Magazine*, vol. 7, no. 1, pp. 58–68, Jan. 2015.

[8] S. Santini, A. Salvi, A. S. Valente, A. Pescapè, M. Segata, and R. Lo Cigno, "A Consensus-based Approach for Platooning with Inter-Vehicular Communications and its Validation in Realistic Scenarios," *TVT*, vol. 66, no. 3, pp. 1985–1999, Mar. 2017.

[9] G. Giordano, M. Segata, F. Blanchini, and R. Lo Cigno, "The joint network/control design of platooning algorithms can enforce guaranteed safety constraints," *Elsevier Ad Hoc Networks*, vol. 94, Nov. 2019.

[10] M. Segata, B. Bloessl, S. Joerer, et al., "Towards Communication Strategies for Platooning: Simulative and Experimental Evaluation," *TVT*, vol. 64, no. 12, pp. 5411–5423, Dec. 2015.