A MEC-based Extended Virtual Sensing for Automotive Services

Giuseppe Avino*, Marina Giordanino†, Pantelis A. Frangoudis‡, Christian Vitale*, Claudio Casetti*, Carla Fabiana Chiasserini*§, Kalkidan Gebru*, Adlen Ksentini ‡, Aleksandra Stojanovic †

* Electronics and Telecommunications Department, Politecnico di Torino, Italy
† CRF-FCA, Torino, Italy
‡ EURECOM, Sophia Antipolis, France

***Abstract*—Multi-access Edge Computing (MEC) promises to enable low-latency applications and to reduce the impact of edge service traffic on the core network. Leveraging on the extension of the popular OpenAir Interface (OAI) architecture to include MEC functionalities, in this paper we show the impact of edge computing resources on a crucial vertical domain, i.e., the automotive domain. As a key example, we focus on a relevant class of automotive services, namely, the Extended Virtual Sensing (EVS) services. With EVS, the network infrastructure collects and makes available measurements gathered by sensors aboard vehicles, as well as by smart city sensors, to improve road safety and passengers/driver comfort. Specifically, we select the EVS application that extends the vehicle sensing capability for supporting vehicle collision avoidance at intersections, and we describe its implementation within the OAI MEC platform. We evaluate the performance of the designed solution emulating the Cooperative Awareness Messages (CAMs) of several vehicles, using a Software Defined Radio (SDR) equipment. We then show experimentally that the MEC infrastructure is pivotal to meeting low-latency requirements and allows detecting all collisions between vehicles, thus proving to be of great benefit to the support of critical automotive services.**

***Index Terms*—5G networks, Road safety services, Extended virtual sensing, Multi-access edge computing, V2I communications.**

I. Introduction

The death toll on our roads has reached staggering numbers: every year, road accidents globally claim the lives of 1.25 million people, leaving up to 50 million with serious, permanent injuries. Calls for a paradigm shift in people's mobility abound, and none is more promising than the use of accident prevention technology on roadsides and onboard new vehicles. The latter is a specific application of a class of automotive services known as Extended Virtual Sensing (EVS), which leverages vehicular communication to collect the output of onboard vehicles sensors, merge them with smart city sensors, and distribute up-to-date information to increase the awareness of the surrounding environment.

While consolidated standard families for vehicular communication by IEEE and ETSI have been around for more than a decade, the rise of 5G networks is delivering the vision of a one-stop solution for integrated vehicle-to-vehicle (V2V), vehicle-to-infrastructure (V2I), and vehicle-to-pedestrian (V2P) operations. 5G networks are nearing the commercial readiness and the automotive industry is eyeing its first deployment by 2021. It is thus to the capabilities of 5G networks that one must look to assess the potentiality of the C-V2X (Cellular Vehicle-to-Anything) technology for road safety. Among all the key performance indicators of 5G that operators, manufacturers, and researchers alike hope to exploit, it is the low latency that matters most in view of devising effective EVS services. The key piece of the architecture that enables the reduction of the latency is the localized computational infrastructure represented by Multi-access Edge Computing (MEC).

In this paper, we take as reference EVS service the one for collision avoidance between vehicles at an intersection. We present a testbed implementation of such EVS service, deployed on an OpenAir Interface (OAI) architecture including MEC functionalities. We include a performance evaluation with a hardware-in-the loop simulation approach showcasing the effectiveness of both the collision avoidance algorithm and its MEC implementation.

II. Multi-access Edge Computing Architecture

Edge computing comes with the promise of enabling low-latency applications, exploiting distributed heterogeneous computing and network resources close to the user end, and reducing core network load by offloading traffic to edge service instances. Recent standardization efforts have brought about detailed architectures for MEC. The ETSI MEC Industry Specification Group (ISG) has provided a reference MEC architecture [2], specifying its components and their interfaces, as shown in Fig. 1. The main entities it includes are as follows:

- **MEC host**: It provides the execution environment to run (virtualized) Mobile Edge (ME) applications, and includes a MEC Platform (MEP) and a virtualization infrastructure, where ME applications are deployed.
- **MEP**: This component acts as the interface between the mobile network and ME applications. The MEP interacts with the mobile network over the (non-standardized) *Mp2* interface to access the user data plane, while it exposes MEC services via the *Mp1* reference point. The MEP Manager (MEPM) component is responsible for MEP configuration and ME application lifecycle management, under the control of the Mobile Edge Orchestrator (MEO).

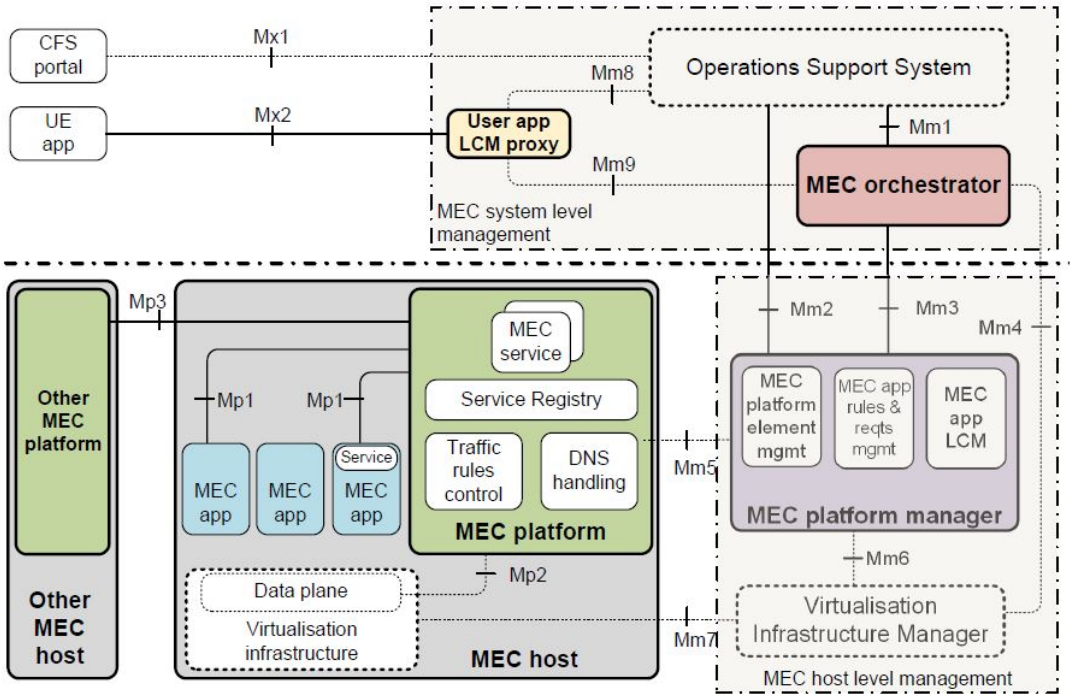

Fig. 1. ETSI MEC architecture [1].

- **MEC services**: These services are discovered and consumed by ME applications over the Mp1 reference point. The ETSI MEC standards specify a set of MEC services that are provided natively by the MEC platform, as is the case for the Radio Network Information Service (RNIS) [3], local DNS or traffic rules control. At the same time, via Mp1, third-party ME applications can register and provide their own services. Note that not all ME applications necessarily provide or consume MEC services.
- **Mobile Edge Orchestrator (MEO)**: This component maintains a global view of the whole mobile edge network and is in charge of managing ME applications. The MEO is the interface between the BSS/OSS and the MEC platform and host. By interacting with the MEP and the virtual infrastructure, it supports the lifecycle management (e.g., instantiation and termination) of ME applications.

Further extensions of the MEC architecture towards network slicing and for deploying a MEC framework in a Network Function Virtualization (NFV) environment are currently under study.

III. Extended Virtual Sensing Application

We now present the EVS application we chose for exemplifying the use of the MEC in the automotive domain. Details on the implementation are provided in Section. IV-B.

The EVS application, running in a VM on a host machine in the MEC platform, is activated in the proximity of an intersection and aims at avoiding the risk of collisions involving approaching vehicles. The collision detector is designed to alert drivers about the presence of unseen vehicles or other unexpected obstacles and, possibly, it is the enabler for vehicles to autonomously activate the emergency braking system.

Vehicles can count on data coming from multiple information sources: data generated by the ego vehicle on its own dynamics, onboard ADAS sensors (ultrasonic, lidar, radar, camera), and messages exchanged through V2I communication, which can be interpreted as virtual ADAS sensors. Nevertheless, having a centralized view of the monitored area, the EVS application can enhance the data provided by the aforementioned sources and can be exploited to provide key information to the control logic making decisions at the vehicle. Specifically, the EVS application exploits real-time data collected by a third-party entity, the Cooperative Information Manager (CIM), about position, heading, speed, and acceleration of all vehicles at the monitored crossing. Based on such information, the EVS application evaluates each pair of vehicles and estimates possible collisions with a trajectory-based algorithm. In this paper, we base our results on a state-of-the-art trajectory algorithm [4], which we report in the following section.

A. The Collision Detection Algorithm

The core of the EVS application is the detection algorithm, presented in Algorithm 1. Note that, although we focus on collisions between vehicles, being the algorithm based on generic trajectories, it can be applied to any kind of colliding entity (e.g., for collisions between vehicles and pedestrians). Also, for simplicity, we do not express in Algorithm 1 the

dependency on acceleration, even though it has been accounted for in the implementation of the EVS application.

The algorithm is run at each new Cooperative Awareness Messages (CAM) message generated by a vehicle travelling across the monitored intersection. The collision detection algorithm requires as input (Line 0):

- position and speed of the vehicle transmitting the last CAM in the area of interest (denoted below as ego vehicle), respectively identified by the two vectors $\vec{p_0}$ and $\vec{s_0}$, where the speed vector also includes information on the heading;
- the latest CAM sent by all other vehicles traveling in the area, which are stored in $\mathcal{V}$.

In Line 1, the set $\mathcal{Z}$ of entities with whom the ego vehicle could collide is initialized and, in Line 2, the future position of the ego vehicle is evaluated for each future time instant. Then the algorithm computes the position of each vehicle $v \in \mathcal{V}$ that recently sent a CAM (Line 4) and the distance $\vec{d}(t)$ between such a vehicle and the ego one (Line 5).

Thanks to the computation of $\vec{d}(t)$, we are now aware of the distance between the ego vehicle and the generic vehicle v, at any time t. To reduce false positives, our algorithm aims at producing an alert only for imminent collisions, i.e., for which the time to collision is below a given threshold $t2c$. Furthermore, we take into account the fact that the vehicles position available at the EVS application, i.e., $\vec{p_X}$, refers to the front bumper of the vehicle. To consider the actual space taken by of real vehicles, our algorithm raises an alarm if two cars distance goes below a threshold $s2c > 0$.

Since we are interested in the minimum value of $D(t)$, in Line 7 we compute $t^\star$, defined as the time instant at which the distance between the two considered vehicles is minimum. If $t^\star < 0$, the two vehicles are getting farther apart, whereas, if $t^\star$ is greater than the threshold $t2c$, a collision between them is not considered as imminent. In both cases, no action is required (Line 8). If $t^\star$ is between 0 and $t2c$, Line 11 computes the minimum distance $d^\star$ at which the two vehicles will be at time $t^\star$. The algorithm compares $d^\star$ against the threshold $s2c$: if $d^\star$ is lower, then vehicle c is added to set $\mathcal{Z}$, otherwise the algorithm skips to the next iteration of the cycle.

Once all CAMs in set $\mathcal{V}$ have been processed, the algorithm returns the set $\mathcal{Z}$ of vehicles with which the ego vehicle is on a collision course. If the set $\mathcal{Z}$ is empty, no action is taken, else an alert message is sent to the ego vehicle as well as to all those that are in set $\mathcal{Z}$. As discussed in [4], the optimal values of $t2c$ and $s2c$ for maximising the number of correctly detected collisions in the considered scenario are $t2c = 3.5\,\mathrm{s}$ and $s2c = 3.7\,\mathrm{m}$.

Algorithm 1 Collision detection pseudocode

Require: $\vec{p_0}, \vec{s_0}, \mathcal{V}$

1: $\mathcal{Z} \leftarrow \emptyset$
2: $\vec{p_0}(t) \leftarrow \vec{p_0} + \vec{s_0}t$
3: **for all** $v \in \mathcal{V}$ **do**
4: $\quad \vec{p_v}(t) \leftarrow \vec{p_v} + \vec{s_v} \cdot t$
5: $\quad \vec{d}(t) \leftarrow \vec{p_0}(t) - \vec{p_v}(t)$
6: $\quad D(t) := |\vec{d}(t)|^2 \leftarrow (\vec{s_0} - \vec{s_v}) \cdot (\vec{s_0} - \vec{s_v})t^2 + 2(\vec{p_0} - \vec{p_v}) \cdot \cdot (\vec{s_0} - \vec{s_v})t + (\vec{p_0} - \vec{p_v}) \cdot (\vec{p_0} - \vec{p_v})$
7: $\quad t^\star := t\colon \frac{\mathrm{d}}{\mathrm{d}t} D(t) = 0 \leftarrow \frac{-(\vec{p_0} - \vec{p_v}) \cdot (\vec{s_0} - \vec{s_v})}{|\vec{s_0} - \vec{s_v}|^2}$
8: $\quad$ **if** $t^\star < 0$ **or** $t^\star > t2c_t$ **then**
9: $\quad\quad$ **continue**
10: $\quad d^\star \leftarrow \sqrt{D(t^\star)}$
11: $\quad$ **if** $d^\star \leq s2c_t$ **then**
12: $\quad\quad \mathcal{Z} \leftarrow \mathcal{Z} \cup \{v\}$
13: **return** $\mathcal{Z}$

IV. Implementation

We now illustrate the implementation details of our EVS application. Our testbed can be divided into two main blocks, namely, (i) the MEC-enabled EPC Network (Section IV-A); (ii) the EVS and the CIM services running as ME applications (Section IV-B). Furthermore, in the testbed, two realistic implementations of cellular User Equipments (UEs), based on OAI, act as vehicles. Such UEs send periodically the information related to position, speed, and direction of emulated vehicles towards the CIM.

Fig. 2 provides an overview on the interactions between such building blocks. Vehicle movements are taken from simulated traces obtained by the well-known Simulation of Urban MObility (SUMO) [5] tool. In turns, the MEC-enabled EPC redirects the traffic received towards the CIM. The EVS application retrieves periodically the latest vehicle information from the CIM. When needed, the EVS application sends alerts towards the vehicles.

A. The MEC Platform

Our system builds on OAI [6], an open source implementation of a full LTE network, spanning the RAN and the Evolved Packet Core (EPC), with current developments focusing on 5G technology. On top of this, we have implemented our MEC platform, which exposes REST-based API endpoints to the ME applications, so that the latter can discover, register, and consume MEC services (e.g., the EVS application).

We provide extensions to the OAI RAN and the core network elements to implement the Mp2 reference points. Core network extensions are necessary for traffic offloading to ME application instances, while specific support is needed at the RAN level for retrieving radio network information from eNBs, such as per-UE Channel Quality Indications (CQI), and exposing them to subscribing ME applications. In our MEC testbed, ME applications run on the MEC host as Virtual Machines (VMs) directly on top of the kvm [7] hypervisor. However, our MEC platform is also compatible with Virtualized Infrastructure Managers (VIMs) such as OpenStack [8], while it has been tested with containerized ME applications managed by lxd [9].

Fig. 2 presents our MEC testbed setup and the interactions and interfaces between its components. The OAI EPC is virtualized, with the HSS, MME, and SPGW running as separate kvm VMs on a single physical machine, which also hosts the MEP. Note that the latter can also be executed as a virtual

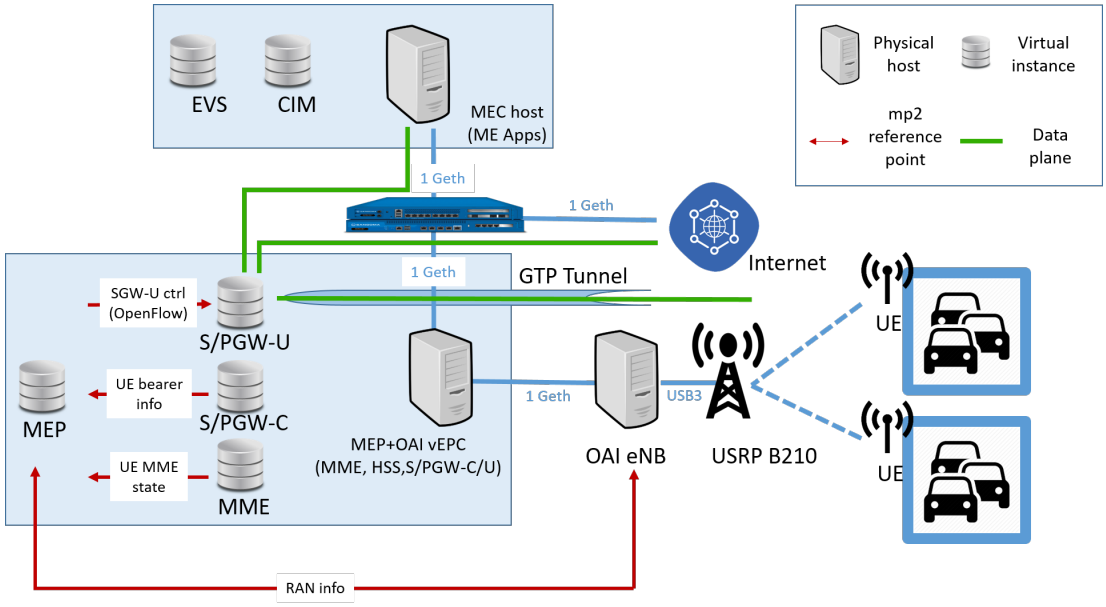

Fig. 2. Interaction between the testbed building blocks.

instance on the MEC host. Due to its real-time constraints, the OAI eNB software is running on a dedicated host, to which a USRP B210 RF board is attached. Each UE RF front-end is also connected to a dedicated host. Over-the-air transmissions are carried out in LTE band 7; the transmission range of the eNB is in the order of a few meters.

B. The Automotive MEC Service

As previously mentioned, we run two automotive MEC services: (i) the CIM, which acts as a collector of CAMs transmitted by the vehicles in the monitored area, and (ii) the EVS application onboarding a trajectory-based algorithm, which aims at avoiding collisions between vehicles.

At run time, the CIM decodes received CAMs messages and then performs the following two actions:

(i) it stores a record of the received CAM in a PostgreSQL database for post-processing purposes,

(ii) it passes a copy of the received CAMs to the so called CAM managers. A CAM manager is an agent that is initiated by the CIM to keep in memory only CAMs related to a specific circle of the monitored area. When CAMs are passed to the CAM manager, the CAM manager checks if the CAMs fall in the assigned circle and, if so, it stores them in a dedicated area of the RAM memory, ready for the queries of the EVS application.

The EVS application onboards the algorithm we described in Sec. III-A. Furthermore, the EVS application queries the CAM managers at the CIM covering the area of interest. The EVS application queries the latest CAMs to the CAM managers every 5 ms over a dedicated TCP connection. When the CAM managers provide new CAMs, the detection algorithm checks if its sender is at risk of collision. When the EVS application detects a pair of vehicles on a course of collision, they are warned by unicast alert messages (Decentralized Environmental Notification Messages (DENMs)).

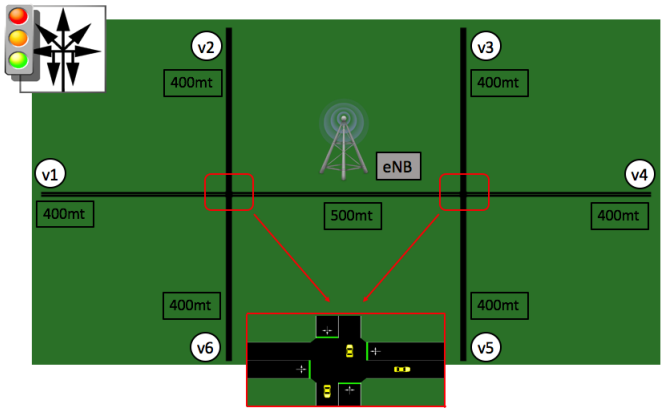

Fig. 3. Screenshot of the scenario simulated in SUMO.

V. Performance Evaluation

We now present the scenario we used for our performance evaluation Sec. V-A, and then we evaluate our MEC implementation in terms of: (i) end-to-end delay, and (ii) percentage of correctly detected collisions.

A. Reference Scenario

Two UEs emulate flows of vehicles traveling on the roads of the map shown in Fig.3. Vehicles only traverse the scenario from north to south (or viceversa), and from east to west (or viceversa). Collisions happens only between vehicles belonging to the opposite groups, since we assume that all cars travel with constant speed (namely, 50 km/h). The mobility traces describing the pattern of the vehicles are obtained with SUMO. We sample the mobility traces of each vehicle every 0.1 s (10 Hz) and we derive key information of their movement, among which, speed, acceleration, and direction. For each obtained sample, we create a CAM, which we transmit through one of

the UEs towards the eNB of the OAI cellular network. Finally, to evaluate how performance changes with the number of cars in system, we used three different values of vehicle density: (i) high, i.e., 20 vehicles, (ii) medium, i.e., 15 vehicles, and (iii) low, i.e., 10 vehicles.

B. End-to-end Delay

The end-to-end delay is computed considering only CAMs that trigger an alarm, and it is defined as the time that elapses between the transmission of a CAM message by a vehicle and the reception, by the same vehicle, of the alarm that such CAM triggered.

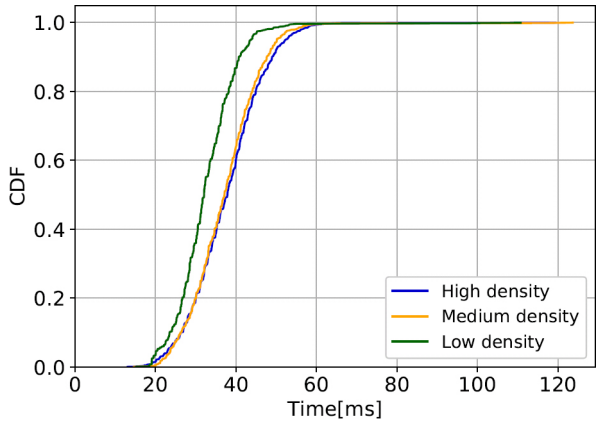

Fig. 4. CDF of the end-to-end delay for varying vehicle density.

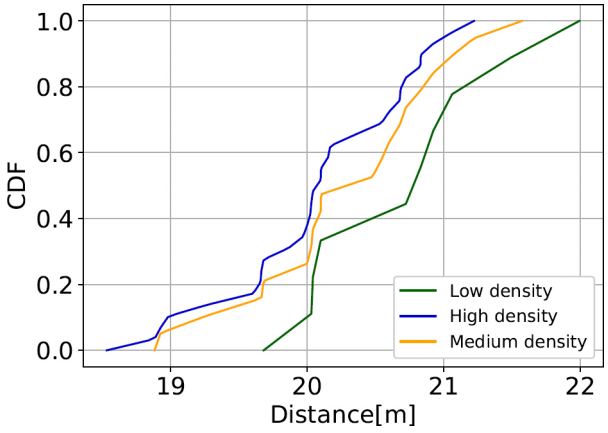

Fig. 5. CDF of the minimum distance reached by vehicles on collision cource.

Fig.4 reports the delay experimental cumulative distribution function (CDF): each curve has been obtained averaging the values of end-to-end latency of 5 different runs. In our tests, the average number of collisions for the "high density" case is 6, 4 for the medium density and 2 for the low density. Given the algorithm used and the parameters set (i.e., the threshold *t2c* and *s2c*), the maximum number of DENMs that can be generated for a given collision is 70. Indeed, in the best case, two cars in course of a collision start receiving DENMs $t2c = 3.5$ s in advance, once for every CAM they transmitted, i.e., two CAMs every 100 ms.

In all scenarios, the 80% of the end-to-end latency values we obtained is below 45 ms. In particular, the average end-to-end latency is 32.5 ms for the low-density case, 37.2 ms for the medium density, and 37.7 ms for the high density. The lower end-to-end latency for the low-density case is due to the smaller number of CAMs processed by the EVS/CIM applications. Indeed, while the delay introduced by the network remains constant (on average 22.5 ms), the processing time at the EVS/CIM applications is proportional to the number of cars in the system. Finally, we can observe that 99.999% of the latency values we recorded are lower than 60 ms.

C. Collision Detection Performance

To evaluate the ability of our EVS implementation to detect imminent dangers in the area of interest, we first build a ground truth for the collisions. Thanks to the SUMO error-log file, for a specific mobility trace, we obtain the actual collisions between vehicles. Given the fact that the same mobility trace is also used in our testbed, we can easily compute the percentage of detected collisions analyzing the DENM messages delivered in the testbed. Furthermore, if the alert for a collision was correctly transmitted, we also look at when it was received and processed by the involved entities. In this way, we can determine if the vehicle had sufficient time to brake before the impact. To do so, we compute the distance at which the colliding vehicles stop after having received the DENM (Fig.5).

To compute the position in which the two vehicles stop, we retrieve their position at the moment of the first DENM reception. The driver cannot start braking as soon as the alert message is received for two different reasons: first, the vehicle human-to-machine interface (HMI) needs some time to elaborate the warning message; second, a human driver does not react immediately to a danger. We set the HMI time to 400 ms, whereas the human reaction time to 1 s. Considering these two factors, we can compute the position $\vec{p^f}$ of the front bumper when a car stops. Finally, we reconstruct the shape of the vehicles (approximating to a rectangle) and find the minimum distance between these two polygons.

The result of our evaluation is that all collisions occurred in SUMO are avoided by the EVS application and, on average, the distance at which the vehicles on a collision course stop is between 20 and 21 m. Furthermore, the results found here are consistent with what was observed in the previous section. Indeed, in the high-density case, the processing time of the EVS/CIM applications is higher, the end-to-end latency grows, DENMs are delayed, and drivers travel larger distances before starting to brake (see Fig.5).

VI. Related Work

Several works have dealt with applications in the automotive domain (e.g., [10]). Many of them, e.g., [11], [12], propose collision avoidance and collision detection applications that do not leverage any mobile network infrastructure. In particular,

[11] focuses on collisions between vehicles and pedestrians in industrial plants, with no specification on the type of wireless communication used. [12] proposes instead a way to automatically detect a collision after it has occurred, using smart-phone accelerometers to reduce the time gap between the actual collision and the first aid dispatch.

The use of the cellular network for the automotive domain as supporting infrastructure has been considered in the last few years. A considerable body of works, e.g., [13]–[15], perform comparisons between IEEE 802.11p and the standard LTE network (non-V2V) for vehicular applications. In particular, [13] highlights the higher capacity, coverage, and penetration of LTE with respect to 802.11p, which is also affected by scarce scalability and unreliable transmissions. [15] confirms these observations stating that LTE offers superior network capacity with respect to 802.11p and is suitable for all case studies. On the contrary, [14] considers LTE unsuitable for collision avoidance applications, due to issues caused by the Doppler effect and LTE handoff procedures. The choice of the best communication technology is still subject of an intense debate in the scientific community.

To assess the ability of the cellular network to support automotive services, in this paper we present a first *real* implementation of a road safety application. Furthermore, contrarily to all previous approaches, our work shows how latency and reliability may be improved when the automotive domain is assisted by the cellular network and the availability of computational capabilities at the edge of the network.

An extensive body of works focuses on MEC [16], but concrete MEC system implementations are few. On this front, [17] presents the design and implementation of a MEC platform with the design goal of requiring no modifications at the RAN and EPC. CDS-MEC [18] aims to eliminate the interactions between the EPC and the mobile edge system. However, no implementation allows a full, yet transparent to the network, MEC implementation. Thus, we opted for tailoring our solution to OAI, particularly regarding the RAN part. In fact, our work is closer to the LL-MEC [19], a MEC design also focused on OAI. LL-MEC uses SDN techniques for control-user plane separation, as well as the same southbound protocol [20] as ours for retrieving RAN-level information from OAI eNBs. Nevertheless, apart from implementation differences (e.g., different interfaces towards the EPC), our MEC system further includes a standards-compliant implementation of the Mp1 reference point of the MEO towards the OSS/BSS, an RNIS interface that fully complies with ETSI MEC 012 [3], and platform components for MEC service discovery and registration.

VII. Conclusion

The provision on a global scale of low-latency communication services to vehicular applications will soon be realized by the deployment of 5G networks and the introduction of edge infrastructure such as MEC. Such a technological enabler will be a key in the development of sophisticated road safety applications, of which the intersection collision avoidance is one of the most egregious examples. In this paper we have presented a MEC-based architecture and its testbed implementation, supporting a collision avoidance algorithm that leverages inter-vehicular communication to alert drivers of impending, potential crashes. A hardware-in-the-loop simulation campaign has shown the effectiveness of our approach in avoiding collisions and stopping vehicles well clear of their potential crash.

Acknowledgments

This work was supported by the European Commission through the H2020 5G-TRANSFORMER project (Project ID 761536).

References

[1] F. Giust, X. Costa-Perez, and A. Reznik, "Latency Critical IoT Applications in 5G: Perspective on the Design of Radio Interface and Network Architecture," *IEEE 5G Tech Focus*, vol. 1, no. 4, Dec. 2017.

[2] *Mobile Edge Computing (MEC); Framework and Reference Architecture*, ETSI Group Specification MEC 003, Mar. 2016.

[3] *Mobile Edge Computing (MEC); Radio Network Information API*, ETSI Group Specification MEC 012, Jul. 2017.

[4] M. Malinverno, G. Avino, C. Casetti, C.-F. Chiasserini, F. Malandrino, and S. Scarpina, "Performance analysis of c-v2i-based automotive collision avoidance," in *2018 WoWMoM*. IEEE, 2018, pp. 1–9.

[5] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "Sumo–simulation of urban mobility," in *The Third International Conference on Advances in System Simulation (SIMUL 2011), Barcelona, Spain*, vol. 42, 2011.

[6] "OpenAirInterface, 5G software alliance for democratising wireless innovation," http://www.openairinterface.org, accessed: 2018-14-12.

[7] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the Linux virtual machine monitor," in *Proc. Linux Symposium*, 2007.

[8] "Openstack webpage," https://www.openstack.org/, 2019, [Online; accessed 10-March-2019].

[9] "Linux container 1xd webpage," https://linuxcontainers.org/lxd/, 2019, [Online; accessed 10-March-2019].

[10] L. Gallo and J. Haerri, "Unsupervised long-term evolution device-to-device: A case study for safety-critical v2x communications," *IEEE Vehicular Technology Magazine*, vol. 12, no. 2, pp. 69–77, June 2017.

[11] Z. Riaz, D. Edwards, and A. Thorpe, "SightSafety: A hybrid information and communication technology system for reducing vehicle/pedestrian collisions," *Elsevier Automation in construction*, 2006.

[12] J. White, C. Thompson, H. Turner, B. Dougherty, and D. C. Schmidt, "Wreckwatch: Automatic traffic accident detection and notification with smartphones," *Springer Mobile Networks and Applications*.

[13] G. Araniti, C. Campolo, M. Condoluci, A. Iera, and A. Molinaro, "Lte for vehicular networking: a survey," *IEEE Communications Magazine*, vol. 51, no. 5, pp. 148–157, May 2013.

[14] Z. Xu, X. Li, X. Zhao, M. H. Zang, and Z. Wang, "Dsrc versus 4g-lte for connected vehicle applications: A study on field experiments of vehicular communication performance," *Journal of Advanced Transportation*, 2017.

[15] Z. Hameed Mir and F. Filali, "Lte and ieee 802.11p for vehicular networking: a performance evaluation," *EURASIP Journal on Wireless Communications and Networking*, vol. 2014, no. 1, p. 89, May 2014. [Online]. Available: https://doi.org/10.1186/1687-1499-2014-89

[16] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 3, pp. 1657–1681, 2017.

[17] T. Subramanya, G. Baggio, and R. Riggio, "lightmec: A vendor-agnostic platform for multi-access edge computing," in *Proc. 14th International Conference on Network and Service Management (CNSM '18)*, 2018.

[18] E. Schiller, N. Nikaein, E. Kalogeiton, M. Gasparyan, and T. Braun, "CDS-MEC: nfv/sdn-based application management for MEC in 5g systems," *Computer Networks*, vol. 135, pp. 96–107, 2018.

[19] N. Nikaein, X. Vasilakos, and A. Huang, "LL-MEC: enabling low latency edge applications," in *Proc. IEEE CloudNet*, 2018.

[20] X. Foukas, N. Nikaein, M. M. Kassem, M. K. Marina, and K. P. Kontovasilis, "Flexran: A flexible and programmable platform for software-defined radio access networks," in *Proc. ACM CoNEXT*, 2016.