Mark8s - A Management Approach for Automotive Real-Time Kubernetes Containers in the Mobile Edge Cloud

Bernhard Blieninger
Automotive System Software and Architecture
fortiss GmbH
Munich, Germany
blieninger@fortiss.org

Aaron Dietz
Department of Informatics
Technical University Munich
Munich, Germany
aaron.dietz@tum.de

Prof. Dr. Uwe Baumgarten
Department of Informatics
Technical University Munich
City, Germany
baumgaru@tum.de

***Abstract*—This paper presents a management approach for real-time Kubernetes clusters in the automotive mobile edge cloud. As part of a vehicle-centric approach to future autonomous mobility the toolchain presented is positioned to extend, offload and enhance the computational capabilities for real-time tasks of a vehicle to the mobile edge cloud. The sensing of the environment by sensor-equipped MECs allows an extended preparation of driving tasks from the MEC to the vehicle. A similarity we seek to exploit further. With the help of a management prototype, we show the feasibility of our approach in principle and how such a system can be realised. A further timing analysis is derived in order to investigate the overhead the proposed management toolchain is introducing.**

***Index Terms*—mobile edge cloud, real-time systems, Kubernetes, automotive systems**

I. Introduction

The current trend towards fully autonomous driving up to SAE Level 5 requires future vehicles to provide a dynamic driving toolchain that is capable of holistically sensing the current traffic scenario, classifying and validating gathered sensory data, and then applying these data points as input to dynamic driving task algorithms [1]. Such toolchains often rely on machine learning (ML) or stochastic algorithms for image classification, object detection, or motion prediction at almost every step [2]. Since these approaches are mostly based on probability distributions, they often pose an inherent problem for the safety of autonomous driving and are therefore deliberately designed to be robust against perturbations. The highly dynamic nature of various traffic situations also places great demands on the available computing capacity and simultaneously inherits this variability of the situation into its computation [2].

As autonomous systems, vehicles suffer from limited computational capabilities, as well as energy, space, and weight constraints, so they are often supported by connecting additional computational resources, such as a mobile edge cloud (MEC) at the roadside [3], [4]. These MECs are often also equipped with sensors on gantries and can thus assess current driving situations at the respective location and prepare maneuver calculations, leveraging the computation demands off the vehicles [3], [5].

This research was funded by the Federal Ministry of Transport and Digital Infrastructure of Germany in the research project Providentia++.

In this paper we want to address this situation by exploiting and extending the similarities of vehicle and MEC from a point of view of task reusability and scheduling. This enables an increase in the safety of the system by increasing the predictability of its task behaviour while decreasing the effort on implementation and code / run time analysis.

We will first give an overview of the problem statement and motivation, then go into comparison with related work and show our approach. A prototype illustrating general feasibility of our approach is described and derived timing analysis is provided. Finally we conclude by depicting the achievements and future work to do.

II. Motivation and Problem Statement

Although the MEC is capable of providing real-time driving task functionality to vehicles in its surroundings, like offloading, extending (with gantry sensor data) or adding parts of the dynamic driving toolchain, the introduction of such decoupled functionalities always imply offloading or even transfer costs [6]. In addition to the time delay imposed by offloading, task runtime parameters have to be known in advance. These are required for the vehicle to safely start the offloading procedure without the risk of missing a deadline on one of its offloaded tasks and thus fail the current driving task requirements with potentially critical impacts [6]. Furthermore, sudden connection failures between MEC and vehicle can occur, rendering previously carried out offloading of tasks useless and potentially harmful. We therefore propose a vehicle edge cloud design, where the MEC at roadside is designed as a stationary twin of the vehicle. MEC and vehicle therefore share their applications, operating system and hardware design, which is upscaled (on the MEC side) with the use of Kubernetes containers. The management toolchain proposed in this paper allows for enabling this particular use case and the full self-sufficiency of both vehicle and MEC. Based on this design, the vehicle is provided with full

autonomy, integrating the MEC as potentially unreliable and less performant ECU clone for real-time task executions into its scheduling algorithms and policy. Furthermore, applications can be developed, tested and verified only once if we assume that they have a modular design and that used algorithms can cope with certain differences of input values (e.g. traffic observation angle). Thus, the MEC can be used as a HIL pre-testing setup for updates or new rollouts of driving functions to the vehicle, in order to ensure correct application functionality. In addition, the use of identical hardware/OS (e.g. ARM, RISC-V / RTOS, RT-enabled Linux) in the MEC and vehicle means that similar system behaviour can be exploited and thus runtime data can be obtained, which in turn can provide clues to the runtime behaviour in the vehicle or, in the best case, even be highly similar. Previous research has shown, that such a scenario might be possible through the use of real-time kernel patches or co-kernel approaches, but also show drawbacks or unsolved challenges [7]. We assume that this approach can be combined with the ML-based deployment and migrations strategy researched in [8]. As of now, we solely focus on the combination and direct connection of MEC and vehicle, as further extensions to central cloud systems imply new limitations in terms of transfer overheads or hardware architectures.

III. Related Work

Previous research on this topic has already shown that it is possible to offload real-time applications [9] or whole workflows [10] in a cloud environment.

In [9], a framework to offload low critical tasks to a cloud environment is presented, where scheduling and offloading decision are supported by machine learning but require an always connected cloud to offload low critical tasks. The work misses some key points of our proposed MEC environment, like the dynamic driving situation and changing vehicle position.

In [11], a platform is designed for seamless deployment and management of container clusters, similar to our approach. They show very quick response times for requests ranging from 3ms to 370ms. Nevertheless, this approach focuses on a centralized cloud infrastructure, where a main cloud is delegating apps to clusters on the edge. A similar approach is shown in [12], where applications for the MEC are provided that help with predictive cloud bursting. However, they again use a centralized architecture and are addressing different usage scenario.

Whereas [13] investigates on the 5G connection and the custom Kubernetes scheduler, which improve latency and load up times, whenever multiple containers are allocated onto the edge server.

A traffic-aware dynamic container migration using LXC containers with real-time kernels on lightweight application containers is considered in [6].

Furthermore, the general idea of real-time containers and their deployment with respect to time guarantees [14], and with a focus on real-time and best effort container co-location inside Kubernetes [15] is just recently getting research interest. The described papers clearly show that offloading of real-time tasks to the MEC is possible and that a Kubernetes driven MEC is capable of hosting real-time applications, even with the means of machine learning-based scheduling. Nevertheless, to the best of our knowledge, there has not yet been an approach that unites a vehicle-centric MEC and a light-weight Kubernetes approach as we describe it.

IV. Approach

We separate the approach into three subsections framing the general idea, the setup based on this idea and possible use case scenarios.

A. General Idea

The overall idea of the proposed approach is that the MEC is a nearly identical hardware and software twin of the vehicle and that it is capable of providing extended RT-services to a vehicle if present in the current area of driving. The MEC is designed to be decentralized and divided into smaller units containing one or more server clusters and gantries equipped with sensors as well as 5G base stations for communication [16] . On top of these clusters, Kubernetes is introduced allowing for automated deployment and scaling, as well as a basic management of containerized real-time applications. In order to fully utilize and adapt the container management to the automotive real-time MEC environment, while also enabling task offloading, we introduce Mark8s, a **M**anagement toolchain for **a**utomotive **r**eal-time Kubernetes containers (**k8s**). It implements a communication gateway per MEC unit enabling fast vehicle-based requests for computational resources within a certain road sector the vehicle is passing. In order to be fully decentralized MEC units are equipped with adjacent neighbour discovery, as well as inter-gateway communication supporting the exchange of health information, such as available resources and the unit's uptime status between two or more adjacent MEC units. Combining the computing capability of the MEC unit's Kubernetes cluster with gantry sensor systems will also allow for offering real-time capable services beyond task offloading from the vehicle or providing pre-computed cloud information. This combination, in addition to a modularization of the automotive driving task chain (sensing, calculating, acting), enables the development and extensive testing of such tasks on the vehicle-like MEC before applying them to the more critical vehicle environment. A permanent sensing container, for example, will fuse and classify found objects for optimal driving pathway calculation before the information is sent to the vehicle and eventually cross-checked with internal sensor and calculation data. Although such driving scenarios and their associated automotive toolchains can vary widely, a common scheme consisting of three kinds of containers can be used as a basis:

- **One-shot containers** (individual/sensitive services) are only used by a single client/vehicle. If the container is no longer needed, gathered data will be deleted and it will be shut down (e.g. offloaded driving tasks)

• **Multi-session containers** (multi-user services) are started once at least one client has requested them. Additional users will simply get redirected to the already running instance. Containers will get shut down, if no client is actively using them after a certain grace period (e.g. driving convoy applications).
• **Permanent containers** (permanent services) start running without being requested and are a special kind of multi-session container, which do not implement a grace period (e.g. automatic emergency detection & alarming services).

Underlining the similarity of MEC and vehicle even further, all examples given could be executed on the vehicle itself, just requiring different sensor data and action receiver modules. Besides other advantages, the containerized applications enable the car manufacturers (OEMs) to keep the whole chain of software components within their development workflows and to guarantee an OEM secured car control when deploying on potentially foreign MEC infrastructure. It also facilitates further follow-up questions on (ethical) responsibility and accounting [17].

B. Setup

The setup derived from the general idea is depicted in figure 1. As mentioned before, the MEC units consist of one ore more gantries and server clusters running Mark8s with Kubernetes. MEC/Mark8s units are connected to the vehicle via low latency 5G radio cells allowing for direct wireless network connection with the prototype gateway at the MEC, where requests are forwarded to the k8s master, which in turn manages different k8s nodes and the containers executed on them.
As the proposed overall architecture of the MEC is decentralized, its MEC units or prototype clusters act autonomously. If requests cannot be handled, the Mark8s cluster gateway will redirect the client to an adjacent prototype gateway/cluster on the predicted future path of the vehicle. Information about such adjacent clusters is asynchronously gathered from a central status aggregation, where newly started prototype clusters report their location and availability after installation. Further health and connectivity status of adjacent clusters is checked bilaterally between neighbouring prototype clusters. Thus, newly installed clusters will report themselves to the global discovery, get neighbouring cluster information and start checking their status and availability. Afterwards, the central status aggregation would only be needed for newly added clusters, or if clusters change their connection parameters and are thus no longer bilaterally detectable. Such unreachable or crashed clusters are no longer used for load balancing or computation considerations but do not affect the availability of adjacent clusters, as long as the deployment/networking infrastructure of each cluster is set up in a self-sufficient manner.

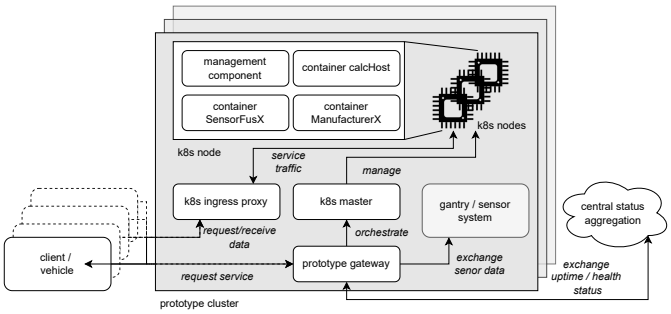

Fig. 1: Sketch of the setup scenario

C. Use Case Scenario

Illustrating the value and functional principle of the setup, we constructed a use case scenario enabled by the prototypical implementation. A vehicle approaching the feeder road of a highway is setting up a connection via 5G network to a MEC/prototype cluster unit capable of task offloading, enhancing or extending. Once the connection is established, an automated request for needed services is sent to the prototype gateway. The gateway will then, based on the workload of the downstream Kubernetes cluster, either schedule the requested service and return the service address after it was successfully launched (SCHEDULED_HERE), redirect it to another gateway/cluster (OTHER_GATEWAY) or deny (CAN_NOT_SCHEDULE) the request.
In the event that the requested service could be scheduled (SCHEDULED_HERE), the responsible prototype cluster is starting up and providing the containerized application, leveraging the computational needs of the vehicle, e.g. preprocessing of a video stream for entertainment or augmented transparency of surrounding vehicles. Other driving-related use cases are also conceivable, such as a predictive lane assistant outsourced to the MEC with increased potential for smoother driving maneuvers and an extended prediction period due to a dramatically expanded field of view at gantry bridges. Figure 2 shows a sequence diagram of a successful client request, which is being scheduled on the local prototype cluster, finishing with the Kubernetes objects being deployed. Once requested, such a service container has to be periodically flagged as still active by the client. This is done because, as the vehicle progresses on its way, it might lose connection or autonomously connect to a following prototype cluster unit and radio cell. On top of these automatically initiated driving tasks, the vehicle can offer additional manually triggered services for passengers, which are supported by the prototype cluster.

V. Demonstrator and Timing Analysis

In our setup two multi-core Cavium ThunderX servers represent the MEC in figure 1 and are running vanilla Ubuntu 20.04. Within this prototype setup all participants are connected via Ethernet. A direct 5G connection is omitted and assumed working as in [18], [16]. To achieve real-time scheduling capabilities, the PREEMPT_RT patched source

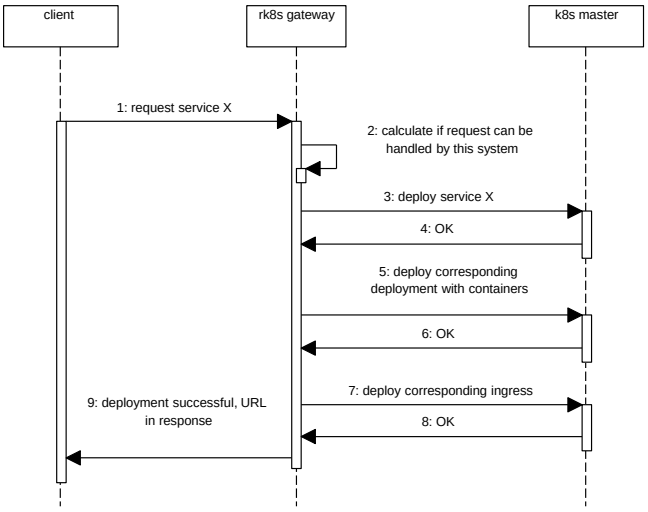

Fig. 2: Sequence diagram of a successful client request

code of the Linux kernel was used and compiled enabling the option Fully Preemptible Kernel. Kubernetes is deployed and runs on top of this real-time environment. The main component of the MarK8s prototype cluster is the gateway, which was designed to be light-weight and is therefore written in node.js for this demonstration. It uses a simple SQLite database to store operating information in it, like deployment service parameters or health status. To exchange data between vehicle and gateway - as well as for other communication means - a REST-API is used.

As previous research has already shown the feasability of a real-time capable Kubernetes as well as the potential of offloading real-time applications (section III), we want to focus on the measurement and analysis of the service request (container upstart and prior decision making) as a main bottle neck of our approach. Runtime and benchmarking tests concerning the real-time capability showed that the PREEMPT_RT was successfully applied to the benchmarking task containers. Measurements upstarting an NGINX (alpine) example native systemd app (4829 ms) and Kubernetes preloaded container (5010 ms) only revealed an overhead of 181 ms in average. Cold starts with containers from a remote source took 8731 ms and multi-session container starts - forwarding the request to a running container - took 85 ms. However, the most important measurement, showing the applicability of the prototype, is the request-response period between a client's request and the answer from the Mark8s prototype gateway. The gateway could either schedule a requested service locally (SCHEDULED_HERE), redirect the client to another gateway (OTHER_GATEWAY) or reject the request as not schedulable at the moment (CAN_NOT_SCHEDULE). No matter what the final result of the request is, the gateway needs to send the responses as fast as possible to enable the client to make an informed decision. Every scheduling decision case was tested 50 times. Additionally we carried out tests for CAN_NOT_SCHEDULE with the global discovery reachable and not reachable to cover the worst case where the MEC unit is isolated. Figure 3 shows the accumulated times measured on average and the division into the different logical code sections. It depicts that SCHEDULED_HERE has the longest response time (272.1 ms), the redirection OTHER_GATEWAY is in the middle (52.25 ms) and the CAN_NOT_SCHEDULE response is the quickest (55,08 ms).

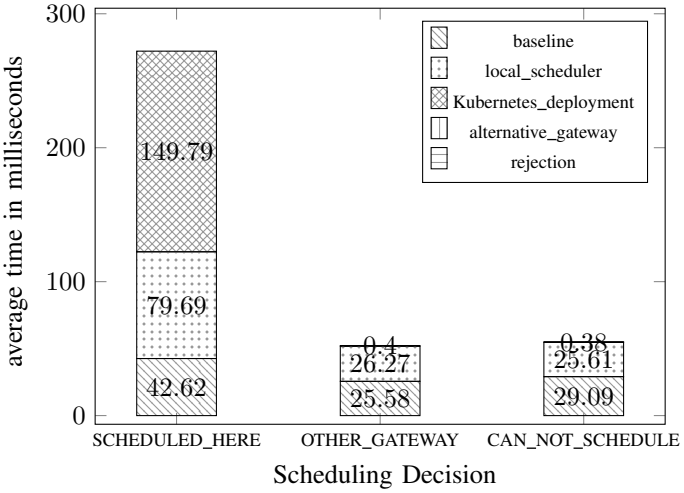

Fig. 3: Request response time diagram for the 3 different response variants following a client's request

Furthermore table I show more detailed numbers and gives further values derived from the measurements. The overall value distribution can be explained by taking a deeper look into the functionality and code structure of the three response options. They all share a baseline part and a local_scheduler part, which checks the schedulability of the request locally. However, if the request is schedulable, more effort has to be done for the local deployment. If it is not schedulable locally, the list of adjacent gateways has to be checked and its address has to be forwarded to the requesting client. If neither is true and there simply is no available adjacent gateway the request can be denied quickly.

Given the above mentioned use case, where a vehicle is entering the MEC-enabled section of a smart road, these first timing analyses show promising results which can only be further evaluated with real workload containers and the full integration of Mark8s with other approaches like [15] and [14].

VI. Conclusion

We propose a **M**anagement approach for **a**utomotive **r**eal-time Kubernetes Containers in the Edge Cloud (Mark8s). The idea of a vehicle-centred automotive future is presented, in which single MEC units are designed as independent hardware and software alike stationary vehicles. The overall idea as well as the design of the toolchain aims to improve fault-tolerance, availability and reusability of applications (modules) from the MEC to the vehicle, while also allowing for rapid development and extensive testing of new driving applications for robustness. The presented prototype shows the feasibility

Scheduling Decision	Phase	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
SCHEDULED_HERE	baseline	26.544	29.117	32.135	42.628	36.942	149.990
	local_scheduler	24.157	28.923	32.562	79.694	39.572	1286.690
	Kubernetes_deployment	99.597	112.066	126.370	149.799	160.977	422.249
OTHER_GATEWAY	baseline	21.412	22.483	24.205	25.587	26.400	58.927
	local_scheduler	21.757	22.770	23.823	26.275	25.462	99.184
	alternative_gateway	0.299	0.340	0.378	0.408	0.410	1.125
CAN_NOT_SCHEDULE with global discovery	baseline	21.414	22.419	23.552	29.094	24.430	134.322
	local_scheduler	20.995	22.787	23.545	25.619	24.968	101.707
	rejection	0.313	0.346	0.367	0.385	0.386	0.879
CAN_NOT_SCHEDULE without global discovery	baseline	19.818	20.905	22.497	24.393	24.278	51.964
	local_scheduler	19.700	21.460	23.277	23.762	24.940	34.613
	rejection	0.270	0.301	0.331	0.359	0.359	1.103

TABLE I: Measurements of Mark8s' scheduling decision timings: SCHEDULED_HERE, OTHER_GATEWAY and CAN_NOT_SCHEDULE with and without working global discovery given in milliseconds

of offloading real-time tasks within such a scenario and gives a first timing analysis in such a system, which can be further enriched by enabling built-in safety and automation features of Kubernetes. Finally, the proposed approach is not very invasive and uses existing software components, is therefore a good basis for future extensions and enhancements, like RT-Kubernetes [14] or REACT [15].

VII. Future Work

While we can show that our approach is feasible for a real-time automotive mobile edge cloud environment, certain restrictions, like privilege escalation due to the used SYS_NICE capability in the prototype, remain. This restriction could be lifted using other available methods like Co-Kernels [7], depending on the use case. Furthermore, useful extensions to the orchestration and managing capabilities of Mark8s, like predictive container start on gateways at the pathway of a vehicle, are not implemented yet. As research on the excluded vehicle part is, as well, still ongoing, we currently try extending the toolchain with a vehicle-centric ML-supported schedulability analysis, as presented in [8].

References

[1] Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles, ISO/SAE PAS 22736 (2021-08), 2021.

[2] E. Yurtsever, J. Lambert, A. Carballo and K. Takeda, "A Survey of Autonomous Driving: Common Practices and Emerging Technologies," in IEEE Access, vol. 8, pp. 58443-58469, 2020, doi: 10.1109/ACCESS.2020.2983149.

[3] S. Raza, S. Wang, M. Ahmed & M.R. Anwar (2019). A Survey on Vehicular Edge Computing: Architecture, Applications, Technical Issues, and Future Directions. Wireless Communications and Mobile Computing, 2019, 3159762. https://doi.org/10.1155/2019/3159762

[4] Intel Corp, "ECU Consolidation Reduces Vehicle Cost, Weight, and Testing", Intel Corp, Accessed: Oct. 27, 2021. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ecu-consolidation-white-paper.pdf

[5] T. Fleck et al. (2018). Towards Large Scale Urban Traffic Reference Data: Smart Infrastructure in the Test Area Autonomous Driving Baden-Württemberg.

[6] S. Maheshwari, S. Choudhury, I. Seskar, and D. Raychaudhuri. "Traffic-Aware Dynamic Container Migration for Real-Time Support in Mobile Edge Clouds." In: 2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS). 2018, pp. 1–6. doi: 10.1109/ANTS.2018.8710163.

[7] V. Struhár, M. Behnam, M. Ashjaei and A. V. Papadopoulos, "Real-Time Containers: A Survey" , 2nd Workshop on Fog Computing and the IoT (Fog-IoT 2020), pp. 7:1–7:9, 2020, doi: 10.4230/OASIcs.Fog-IoT.2020.7

[8] O. Delgadillo, B. Blieninger, J. Kuhn and U. Baumgarten, "A Generalistic Approach to Machine-Learning-Supported Task Migration on Real-Time Systems.", J. Low Power Electron. Appl., 2022, doi:10.3390/jlpea12020026

[9] M.A. Maruf and A. Azim, "Extending resources for avoiding overloads of mixed-criticality tasks in cyber-physical systems", IET Cyber-Physical Systems: Theory & Applications, pp. 60-70., 2020, doi: 10.1049/iet-cps.2018.5062

[10] J. Zhou, J. Sun, M. Zhang and Y. Ma, "Dependable Scheduling for Real-Time Workflows on Cyber–Physical Cloud Systems," in IEEE Transactions on Industrial Informatics, vol. 17, no. 11, pp. 7820-7829, Nov. 2021, doi: 10.1109/TII.2020.3011506.

[11] H. Mfula, A. Ylä-Jääski and J.K. Nurminen, "Seamless Kubernetes Cluster Management in Multi-Cloud and Edge 5G Applications.", In: International Conference on High Performance Computing & Simulation (HPCS 2020). 2021.

[12] F. Faticanti et al. , "Distributed Cloud Intelligence: Implementing an ETSI MANO-Compliant Predictive Cloud Bursting Solution Using Openstack and Kubernetes", In: K. Djemame et al. (eds) Economics of Grids, Clouds, Systems, and Services. GECON 2020. Lecture Notes in Computer Science, vol 12441. Springer, Cham. doi: 10.1007/978-3-030-63058-4_8

[13] M. C. Ogbuachi, A. Reale, P. Suskovics and B. Kovács, "Context-Aware Kubernetes Scheduler for Edge-native Applications on 5G," in Journal of Communications Software and Systems, vol. 16, no. 1, pp. 85-94, April 2020, doi: 10.24138/jcomss.v16i1.1027

[14] S. Fiori, L. Abeni and T. Cucinotta, "RT-kubernetes: containerized real-time cloud computing", In Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing (SAC '22), Association for Computing Machinery, New York, NY, USA, 2022, pp. 36–39, doi:10.1145/3477314.3507216

[15] V. Struhár, S. S. Craciunas, M. Ashjaei, M. Behnam and A. V. Papadopoulos, "REACT: Enabling Real-Time Container Orchestration," 2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2021, pp. 1-8, doi: 10.1109/ETFA45728.2021.9613685.

[16] V. Lakshminarasimhan and A. Knoll, "C-V2X Resource Deployment Architecture Based on Moving Network Convoys," 2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring), 2020, pp. 1-6, doi: 10.1109/VTC2020-Spring48590.2020.9128410.

[17] J. Gogoll and J.F. Müller, "Autonomous cars: in favor of a mandatory ethics setting" , Science and engineering ethics, 2017, vol. 23 , no. 3, pp. 681-700.

[18] M. Tao, K. Ota and M. Dong, "Foud: Integrating Fog and Cloud for 5G-Enabled V2G Networks," in IEEE Network, vol. 31, no. 2, pp. 8-13, March/April 2017, doi: 10.1109/MNET.2017.1600213NM.