Alam, F., Toosi, A. N., Cheema, M. A., Cicconetti, C., Serrano, P., Iosup, A., Tari, Z., & Sarvi, A. (2023). Serverless vehicular edge computing for the internet of vehicles. *IEEE Internet Computing*, 27(4), pp. 40-51.

DOI: https://doi.org/10.1109/mic.2023.3271641

Serverless Vehicular Edge Computing for the Internet of Vehicles

F. Alam
Monash University, Australia

A. N. Toosi
Monash University, Australia

M. A. Cheema
Monash University, Australia

C. Cicconetti
University of Pisa, Italy

P. Serrano
University Carlos III de Madrid, Spain

A. Iosup
Vrije Universiteit Amsterdam, Netherlands

Zahir Tari
RMIT University, Australia

A. Sarvi
University of Melbourne, Australia

***Abstract*—Rapid growth in the popularity of smart vehicles and increasing demand for vehicle autonomy brings new opportunities for vehicular edge computing (VEC). VEC aims at offloading the time-sensitive computational load of connected vehicles to edge devices, e.g., roadside units. However, VEC offloading raises complex resource management challenges and thus remains largely inaccessible to automotive companies. Recently, serverless computing emerged as a convenient approach to the execution of functions without the hassle of infrastructure management. In this work, we propose the idea of serverless VEC as the execution paradigm for the Internet of vehicles applications. Further, we analyze its benefits and drawbacks, and identify technology gaps. We also propose emulation as a design, evaluation, and experimentation methodology for serverless VEC solutions. Using our emulation toolkit, we validate the feasibility of serverless VEC for real-world traffic scenarios.**

Introduction

Vehicles such as cars, trucks, and other modes of transportation play a critical role in modern society. With nearly 1.5 billion vehicles already in existence on Earth in 2022 and increasing demand for digital processing and onboard applications, the future of transportation and mobility presents numerous new opportunities across the hardware, software, networking, and services industries. We envision a paradigm shift in this ecosystem.

Although sensory inputs attached to the vehicles provide many of the current advancements in Intelligent Transport Systems (ITS) and Connected and Autonomous Vehicles (CAVs), the next generation of automobiles will require fast and reliable external computation services to offset the growing complexity and costs of onboard computers [1]. Modern CAVs can generate data at a velocity and volume that may exceed by far the network capacity towards the Internet, and the high Round Trip Times (RTTs) may be incompatible with many emerging applications requiring real-time capabilities. To cater to extra computing needs and stringent latency requirements, we need an approach that can leverage more resources than available in-vehicle, yet make sure the resources can compute and send back the results in-time, and can do so automatically and efficiently for a variety of time-critical vehicular applications, including the perilous tasks of every autonomous driving system.

We propose in this work an approach that addresses these needs by combining two emerging paradigms: vehicular and serverless computing. *Vehicular Edge Computing (VEC)* combines vehicular and edge computing by offloading delay-sensitive computational tasks from vehicles to the nearby edge computing nodes. VEC utilizes available infrastructure that includes primarily Road Side Units (RSUs) and decentralized edge data centers in the proximity of RSUs. A large body of research also proposes to utilize the leftover capacity of the neighbouring and parked vehicles for the computation of time-sensitive tasks [2]. There are significant challenges to achieving the potential of VEC. Firstly, in contrast to cloud-native applications, the development and deployment of VEC-compatible applications in dynamic and diverse VEC environments are extremely challenging for developers and operators. In fact, any VEC solution should allow developers to focus on the application's business logic instead of handling the management and orchestration aspects. Moreover, addressing the scalability requirements of the time-critical applications to respond to the load variability at the edge is not a trivial task. Finally, due to the mobile nature of the vehicles at the network edge, using traditional stateful application architectures, which are tightly coupled with storage to hold states of the application (stored inputs and outputs), significantly compromises the agility, elasticity and efficiencies of ITS applications.

In recent years, a new paradigm of Function-as-a-Service (FaaS) has emerged in the cloud computing domain to address similar challenges, with recent explorations in the context of edge [3] and mobile computing [4], [5]. FaaS allows users to deploy an independent, standalone piece of code (a "function") on the infrastructure where the computational backend requirements for the functions are assessed, provisioned and maintained by the platform provider. Advantages include: i) high agility in application development without operational expertise; ii) effortless scalability to cater to the surge in functions calls, and iii) efficient use of resources through seamless multi-tenancy. Since FaaS relieves the developer from server management-related issues, the concept is also known as *serverless computing*.

In accordance with these contemplations, this paper coins a new term called "Serverless Vehicular Edge Computing" or simply "Serverless VEC". *Severless VEC* refers to the deployment of serverless execution model on edge devices, RSUs, and vehicles for the purpose of processing data from connected vehicles and supporting the development of applications. In this architecture, functions, and tasks are executed on edge devices located near the source of data, rather than in a central server. The "serverless" aspect means that the edge devices can dynamically allocate computing resources as needed, without the need to manage and maintain a dedicated server infrastructure. This provides low latency, real-time processing, and increased efficiency, flexibility, and scalability for the management and analysis of the large amounts of data generated by connected vehicles and the increasing number of emerging

real-time applications.

Multiple efforts have been made in the recent past for efficient task offloading on the vehicular edge [1]. These works are centered around finding the best methods for distributing the load to connected computational units, e.g., neighboring vehicles, RSUs or traditional cloud for optimizing time, energy, or computational capacities. However, there has been no or little effort in providing a comprehensive solution that caters to VEC application deployment issues such as ephemeral connectivity of moving vehicles, failure handling, provisioning, monitoring, and scalability.

In addition, without empirical evidence, theoretical concept such as Serverless VEC may be deemed infeasible and impractical for time-sensitive real-world scenarios. To bolster its feasibility and usability in reality, we provide an emulation architecture and toolkit for Serverless VEC using open-source frameworks. Through experiments, we demonstrate the viability of Serverless VEC for real-world applications and show that it can provide improved response times for resource-intensive applications such as object detection. Our future work will aim at using this architectural framework to provide more extensive experiments with various policies providing a platform for others to conduct similar experiments.

This work aims to provide a comprehensive and practically feasible solution that distributes the load and creates, manages, and scales the VEC applications for optimal latency while minimizing development and deployment costs. We extend the idea of serverless computing that is successful in the traditional cloud to provide a feasible solution to manage the VEC infrastructure [3]. Our key contributions are: i) a platform agnostic infrastructure management for serverless VEC with built-in autoscaling, and load balancing on the edge; ii) an analysis of the advantages of serverless VEC; iii) a review of the challenges expected in early adoption; and iv) a detailed architecture for the emulation of serverless VEC, along with an exemplary scenario to showcase the feasibility of serverless VEC.

Operational Model and Background

We consider an operational model that combines vehicular, edge, and serverless computing.

Vehicular AdHoc Network (VANET)

The next generation of vehicles, called CAVs, will be equipped with communication technologies to communicate with each other—Vehicle-to-Vehicle (V2V)—with roadside infrastructure—Vehicle-to-Infrastructure (V2I)—and in some cases, even with pedestrians—Vehicle-to-Pedestrian (V2P). The ambit term for these communication technologies is Vehicle-to-Everything (V2X) communication. This Point-to-Point communication infrastructure creates an adhoc network on the roads called Vehicular Ad-Hoc Network (VANET). CAVs utilise VANET for applications such as pre-crash sensing, blind intersection, and forward collision. VANET enables message and information passing using multihop strategies on the P2P connections established by V2X [6].

Task Offloading in VANET

The number of sensors in vehicles is spiraling with the cost of electronics in vehicles, which was around 35% in 2010, is expected to reach 50% by 2050. Vehicles with higher autonomy generate around 25GB of data per hour, according to an estimate by McKinsey. With the advent of VANET and V2X, ITS applications would pave the way for newer and more advanced applications. However, the present computational infrastructure would not be sufficient to process this deluge of data for advanced applications. To cater to the growing demands of vehicular computation, we need to offload their computational tasks using the VANET infrastructure. Thus, a large body of research has been attributed to efficient task offloading [1].

Vehicular Cloud/Edge Computing

The onboard computational power in current vehicles is not keeping up with the advanced application demands. Due to an ongoing bottleneck in semiconductor production, producing and scaling enough chips to meet this growing demands is unlikely to happen for the next coming years. Mobile Cloud Computing (MCC), a paradigm that allows both storage and computation of data outside the mobile device, is being employed to keep up with this growing demand.

As an extension to MCC, Mobile Edge Computing/Multi-Access Edge Computing

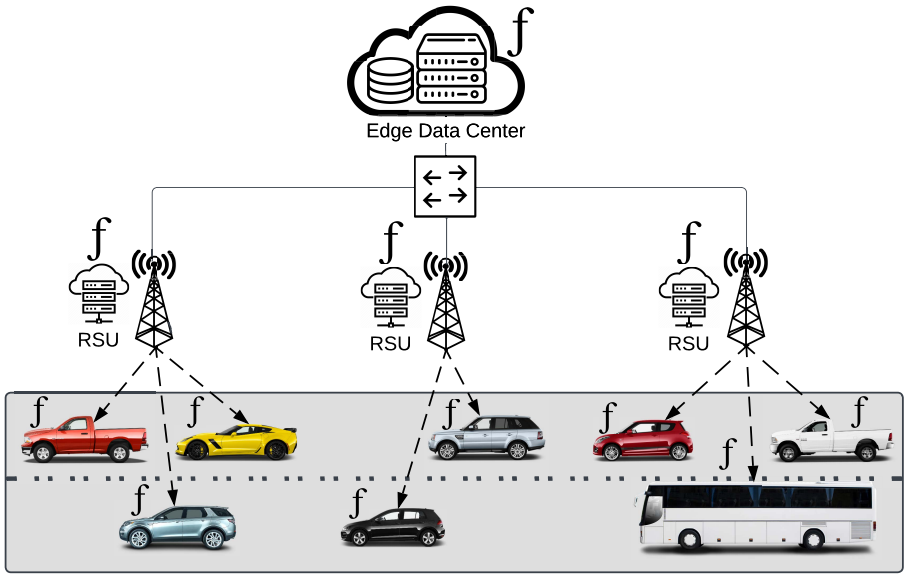

Figure 1. A serverless vehicular edge computing Scenario. f represents serverless functions that can be deployed on various nodes.

(MEC) has been explored to minimize latency of transmission to cloud and further improve the quality of service (QoS). MEC is a paradigm where the resources for infrastructure-as-a-service (IaaS), platform-as-a-service (PaaS), and software-as-a-service (SaaS) can be accessed at the edge of the network. When the MEC concept is extended to ITS, it is known as Vehicular Edge Computing (VEC). In a VEC, the data would be computed at the RSUs, or at decentralized computing centers nearby RSUs, which is called edge data centers. For sudden peaks in API triggers, even neighboring vehicles and central cloud facilities can be employed to augment the edge infrastructure. Various studies have been conducted to study the modalities and techniques for VEC [1].

Software-Defined Networking (SDN)

Software-defined networking (SDN) is an approach to networking that provides separation between the network control plane and the forwarding plane with centralized administration identified as the key enabler for 5G. It also offers operation flexibility and network management at scale for VANETs. A programmable and intelligent 5G network provided by SDN is highly agile and boosts the usability of the high bandwidth and low latency offered by 5G. It also paves the way for more innovation, and advanced service and product offerings, thereby improving overall efficiency.

Serverless and Function-as-a-Service (FaaS)

With the advent of virtualization and cloud computing, the notion of serviceability enhanced by the advancement of technologies led to the creation of Function-as-a-Service (FaaS). FaaS allows the user to specify logical, independent pieces of code to be deployed as microservices. Functions are deployed in software containers which are self-contained units holding the function and related libraries bundled together in an isolated space running as a process. The service provider provides the required resources to execute functions based on their footprint and elastically scales them according to the demand. This is beneficial as the user does not need to maintain or scale the backend resources.

Figure 1 illustrates how FaaS operates in VEC. FaaS acts as a serverless way to deploy functions on the infrastructure provided and maintained by the service provider. It is mainly

used in an event-driven context where functions are triggered by typical events such as HTTP requests, message queues, database or storage operations, etc. The event driven nature of serverless aligns well with the nature of many VEC applications that require event triggering based on sensing/actuation.

Benefits of Serverless VEC

In the cloud, serverless computing and the FaaS programming model are becoming increasingly popular due to the many advantages they offer [7]. In this section we frame these advantages in the context of serverless VEC.

Serverless scales well with non-uniform dynamic vehicular offloading. R equests f or VEC applications follow vehicle traffic p atterns, which are bound to be highly non-uniform in both space and time. With serverless, provisioning does not need to be based on peak traffic conditions, which can waste resources. In other words, the stateless nature of FaaS invocations makes it straightforward to scale up and down the number of instances of a given platform based on the request rate and commensurate resource requirements. This would also help optimize the infrastructure cost of the service provider as the system specifications a re s et b ased o n favorable traffic c onditions, k eeping t he s ystem e lastic to meet growing demands. Scaling also helps reduce overall operational costs and energy consumption. Ideally, the exact amount of resources needed for the current traffic l oad s hould b e a ctive a t any time and platform [8].

Flexible billing model helps on-demand vehicular applications. With the rate of advancements and investment in ITS, the applications and services attributed to the vehicles will increase as well. This creates an opportunity to implement opportunistic mechanisms to share the compute resources available on the RSUs and the vehicles. Some preliminary studies on incentive mechanisms for edge computing have been proposed, e.g., [9], but the research is still in its infancy. We can speculate that the sandboxed environment provided by serverless computing, coupled with fine-granularity b illing (typically i n v olume of function calls), will provide a fertile environment for a market of computation power for VEC. For example, low-end vehicles with little or no computational resources would have the option to get the services offered by high-end vehicles or nearby infrastructure by paying for the services only as needed.

Statelessness property helps opportunistic deployment of VEC functions. The FaaS paradigm is inherently stateless. The stateless nature of serverless functions makes them an attractive platform for real-time VEC applications that need to deploy services opportunistically. Avoiding state enables serverless to be ephemeral. Thus, it makes it easier for the serverless framework to allocate resources to functions and host functions across RSUs, Vehicles, etc. Moreover, in VEC, the state of some applications may only make sense for a given physical location (e.g., at a junction) and for a limited amount of time (e.g., until a traffic jam is resolved) which is matching the ephemeral states in Serverless.

Stateless property addresses the challenge of service migration for moving vehicles. Due to the dynamic nature of moving vehicles, a vehicle may trigger an API request at the edge, but leave the network coverage area of the RSU before receiving the response. In order to meet the low-latency requirement of real-time VEC applications, this may require service migration over the network, particularly for stateful applications, which can be a costly and time-consuming process. In a stateless serverless scenario, the same task can be requested again at the next connected RSU without impacting the overall application flow. On the other hand, in traditional edge computing with stateful containers or VMs, service continuity requires the transfer of the current state to the new edge, thus increasing the management plane complexity and affecting both the system resources and the user's Quality of Experience (QoE).

Serverless VEC handles massive parallelization. FaaS invocations are independent of one another; thus, the execution of complex applications through many smaller tasks in parallel may be done in a straightforward manner [10]. Some vehicular applications may take advantage of this, especially when running on relatively modest hardware, such as that made available by RSUs. Such a granular function decomposition opens the

door to sophisticated optimization strategies and high-performing architecture. The request scheduler can place sub-tasks independently on different threads, processes, and machines or schedule them for parallelism on hardware accelerators.

Serverless VEC favors interoperability. With vehicles of different car makers and models on the road, vehicle computation infrastructure would be heterogeneous in nature. Handling various devices, hardware, operating system, and library requirements is a challenge that needs to be addressed within industry alliances (e.g., AUTomotive Open System ARchitecture – AUTOSAR[1]), which slows down the innovation and time-to-market. With serverless, the container is a self-contained abstraction with function code and relevant libraries pre-bundled in a single unit that can be deployed over different hardware and architectures, thus enabling faster cycles of software deployment, adoption, and updates.

Serverless helps faster development of VEC applications. A serverless architecture relieves the developers from the burden of platform management, maintenance, and scaling. This allows them to focus solely on the functionality and business logic of the application they want to provide to clients, paving the way for faster development cycles and lower operational costs.

Event-driven nature of serverless matches vehicular use cases. The event-driven architecture of serverless makes it a perfect match for many VEC applications working in real-time based on data-rich sensing and actuation events and state changes. In other words, the distributed and asynchronous architecture pattern of VEC applications can be simply handled by the event-driven function services. Serverless can rely on events to trigger and communicate between decoupled services of VEC applications built on microservices.

Challenges and open issues of using Serverless in VEC

Despite the advantages to using serverless architecture in a VEC scenario, there are also some challenges and open issues that need to be addressed to fully exploit its potential.

Management and Orchestration (MANO) of Serverless VEC has high overheads. Cluster management and orchestration is a complex operation involving keeping a list of running containers, driving autoscaling, managing placement, performing load balancing, and continuously monitoring the resources. Existing Serverless frameworks are designed to execute in purposely built cloud settings with computing nodes in static clusters operated on tightly coupled and often homogeneous servers connected through high-speed and reliable wired networks. However, in VEC, computing nodes are heterogeneous and dispersed over a mixture of wired and unstable and intermittent wireless networks, which makes management and orchestration more challenging and especially vexing for low-resource computing nodes such as vehicles and RSUs.

Cluster formation is not designed for dynamic vehicular topologies. Existing FaaS frameworks and orchestration tools are designed for back-end server-based solutions where cluster nodes are readily available for scheduling and are expected to be stable over time. Instead, in a VEC secnario, where vehicles can act as computing nodes and join and leave clusters dynamically. Today's procedure for cluster formation may be inadequate for such a volatile scenario due to the considerable time and resources required by the related procedures, which can negatively impact the overall performance.

Serverless suffers from cold-start effects. Experiments executed in the cloud have revealed that only a tiny fraction of the response time of a FaaS call invocation can be attributed to computation, while the rest is overhead due to network transfers, container activation, run-time environment set-up, virtualization costs [11] etc. The overhead is especially significant the first time a function is invoked after the orchestration system has scaled down to zero instances, in which case a cold-start phenomenon occurs. The container image has to be pulled from the repository and loaded, which can take orders of magnitude more time than the typical response time.In a VEC scenario, we speculate that cold-start effects may be much more widespread than in a cloud system because there is not a single serverless platform logically centralized in a data center, but many distributed over a territory. High jitter, and in particular tail latencies created by

[1] https://www.autosar.org/

cold-start effects, can be problematic for time-critical tasks and, if not addressed by research and technology, may become a barrier to the adoption of serverless VEC.

Resource scheduling optimization is more difficult. In the cloud, the main role of the autoscaler is deciding how many replica instances are needed for a given function. In an edge scenario, the problem becomes more complicated because the run-time environment is also in charge of deciding *where*, that is, on which edge node to activate or terminate an instance. Serverless VEC presents a new level of complexity because these environments are highly dynamic and consist of a heterogeneous, loosely coupled set of nodes connected with erratic and unreliable network connections. Given the time-variable dynamics of such a system, it is difficult to predict which node is best suitable for the given request, considering the locations of the nodes, their velocities, directions of movement, hardware characteristics, and so on.

New security concerns. Serverless in the cloud is secured by a firewall in a trusted environment [12], while in a VEC, nodes would be exposed to different heterogeneous and insecure vehicles. Moreover, with multi-tenancy, multiple clients would be serviced in the same cluster and overlapping nodes. This adds to the complexity of addressing security measures as the data would be distributed across vulnerable nodes in heterogeneous environments. Although sandboxing does help isolate the space, the data is still shared on a different platform. New studies on how to enable security and privacy in such a heterogeneous and ephemeral system are needed.

Ingress points are distributed Serverless systems in the cloud are logically centralized, with all client requests forwarded to a central gateway. Such a model does not capture the distributed nature of a typical VEC scenario, and investigation is required to establish a scalable, distributed, agile and/or hierarchical model for faster response times for clients and minimum resource utilization for servers. Furthermore, the choice of where to place the multiple gateways based on the continuously changing VEC environment is also an open research problem.

Modeling, simulation, and emulation of serverless VEC is not trivial. Models, simulators, and emulators are required to evaluate and test the performance of serverless VEC applications and mimic their behavior, hardware, software, etc. However, modeling, simulation, and emulation of a serverless VEC scenario where vehicles move in and out of range at varying high speeds in a short span of time with many distributed software and hardware components involved is challenging. In the remaining part of this paper, we focus on this challenge and propose our emulation for serverless VEC.

Serverless VEC Sample Use Cases

There could be many applications that would benefit from a serverless VEC. Some of the use cases are listed below:

Autonomous Driving. Serverless VEC can support high-speed processing of data from the cameras and other sensors onboard vehicles with full or partial offloading to assist extra computation and application development for autonomous driving. This improves the performance, reliability, and cost-effectiveness of autonomous driving systems.

Traffic Management. Serverless VEC can help with real-time data processing from vehicles to enable coordinated responses to traffic information such as road incidents and other safty hazard for efficient traffic management.

Infotainment Services. The presented framework can provide high-speed, low-latency processing for vehicle add-on services like entertainment systems such as gaming, streaming video, and more. Other add-on services could include emerging AI/ML use cases for gesture recognition and voice recognition, leading to improved QoE.

Emulation Architecture and Prototype

A typical VEC scenario involves a bunch of RSUs and vehicles generating requests for tasks such as image processing or object detection for various applications like autonomous driving, accident prediction algorithms on blind turns, and augmented and virtual reality applications for add-on comfort. Testing, performance analysis, and verifying serverless applications in a real-world VEC environment is costly and difficult, if not impossible, in many cases. In addition, it is essential to study the system's intricacies, like predicting the load on RSUs, average response

times, effects of various scheduling and orchestration strategies, the impact of traffic movement patterns on a load of RSUs and edge data centers, and other similar metrics. In these cases, simulation or emulation tools are instrumental in providing developers with accurate or near-real precision data. In this work, we put together an emulator toolkit to delve into the peculiarities of deploying a FaaS application in a serverless VEC environment.

The proposed emulator toolkit architecture is depicted in Figure 2. We use a mix of multiple off-the-shelf and open source simulator/emulator tools along with our software codes to create a software suite representing serverless VEC scenarios. In the following, we discuss main component of our proposed emulator.

SUMO: For road network traffic simulation, we use Simulation of Urban MObility (SUMO),[2] an open source traffic simulator. SUMO can handle large scale and continuous traffic simulation on vast road networks. One can also create road networks of choice for their setup. Maps downloaded from OpenStreetMaps (OSM)[3] are used as an input of the road network to SUMO for real life traffic generation. As an output, SUMO generates traffic trace file with position of every vehicle at every time step on the map using a given traffic model.

Mininet-WiFi: The proposed architecture uses WiFi based DSRC network for V2X communication. We use open source emulator Mininet-WiFi.[4] Mininet-WiFi is an extension of open source network emulator Mininet and allows for creation of WiFi access points and WiFi stations. Mininet-WiFi allows specifying the positions of the stations and access points. The WiFi connection is established by virtualizing the wireless medium that connects all WiFi devices. This virtual medium is called *wmediumd*. It also handles handover by stations getting connected to the access point based on two different strategies: strongest signal first (ssf) and least loaded first (llf).

Containernet: Containernet[5] is another project that extends Mininet-WiFi and allows creation of *Docker* containers as hosts and stations in the emulated network topology. We use containernet to create vehicles (stations in Mininet-WiFi) and RSU or edge devices as Docker Containers. Containers let us get a segregated computing environment for experimentation closely resembling the independent environment over any vehicle or edge device.

ONOS: Mininet-WiFi also integrates with Software-Defined-Networking (SDN), where networking of nodes can be managed by an SDN controller. We use Open Network Operating System (ONOS)[6] as the SDN controller. All the nodes connect to ONOS for updating their routing entries. SDN helps provide more networking control and related optimization in a dynamic experimental setup.

Container Orchestrator: Any serverless framework requires a running cluster on which function can be deployed and executed. We use *Docker Swarm*[7] to create and orchestrate the VEC cluster due to its lightweight architecture suitable for edge computing. In our setup, the Docker Swarm instance runs on the RSU container. The vehicles running on other docker containers can join the cluster as nodes using the connection maintained by Mininet-WiFi. Vehicles joining the cluster are available for the function replica placements by the swarm manager at RSU. Other edge devices and RSUs can also join the cluster similarly. The cluster manager is responsible for maintaining a list of nodes (vehicles or other RSUs and edge devices) associated with this cluster, providing a platform for the placement of functions and helping in the scheduling and orchestration of nested containerized functions. We used Docker Swarm instead of Kubernetes as the time required for nodes to join a cluster in Kubernetes is much high compared to Docker Swarm. If a vehicle moves fast and keeps crossing RSUs, much of the time is spent on disconnecting from the previous cluster and connecting to the closer ones, providing little time for the actual computation of functions.

OpenFaaS: OpenFaaS[8] is an open-source serverless framework for the deployment of

[2]SUMO. https://sumo.dlr.de/docs/index.html
[3]OpenStreetMap. https://www.openstreetmap.org/
[4]Mininet-WiFi., https://mininet-wifi.github.io/
[5]Containernet. https://mininet-wifi.github.io/containernet/
[6]ONOS. https://opennetworking.org/onos/
[7]Docker Swarm. https://docs.docker.com/engine/swarm/
[8]OpenFaaS.https://www.openfaas.com/

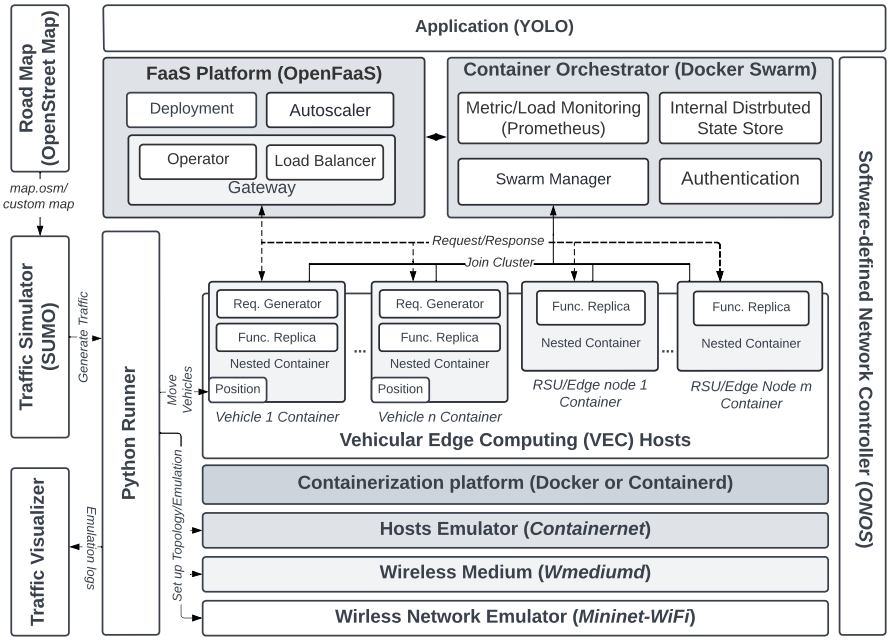

Figure 2. Software architecture of the emulation toolkit.

Function-as-a-Service. This is used to deploy functions as a service on the cluster setup by Docker Swarm. The user needs to specify the functions to be used, and OpenFaaS deploys these functions on the Docker Swarm cluster. OpenFaaS helps set up API gateway for the functions and performs autoscaling as the load changes.

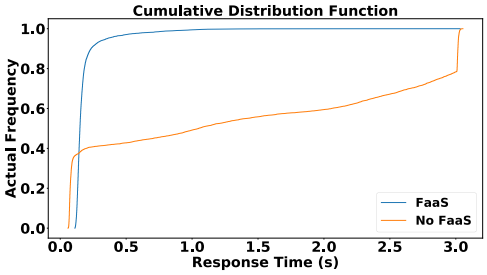

Figure 3. CDF of response times

Experimental Setup

An experiment is conducted to prove the feasibility of the presented architecture. A road map with a single crossing and a 1km road length on each side is employed. Traffic is randomly generated for a 5-minute interval using *randomTrips.py* utility of SUMO. Traffic traces are generated in XML format, maintaining vehicle positions at each time step (seconds) throughout the vehicle journey.

A python script parses these traces and invokes a routine for Mininet-WiFi. It starts an access point, instantiates *wmediumd* wireless medium with IEEE802.11n protocol, *logDistance* as the propagation loss model and *Strongest Signal First* for association and handover. It creates vehicles as Mininet-WiFi stations and emulates them on the docker container provided by the containernet. RSU is instantiated as a docker host and connected with the access point on an ethernet link. Each vehicle container is allocated a 1GB memory and RSU a 4GB memory.

The YOLO object detection model [9] is deployed on OpenFaaS as the use case application for experimentation. A docker container image is created for YOLO with an exposed port for HTTP requests. Before starting the emulation, Docker Swarm and OpenFaaS are installed on the RSU with RSU as the manager node.

After that, the python script starts the movement of vehicles as per traffic traces generated by SUMO. The vehicle movement is performed by changing the position parameter associated with each vehicle container (as provided by Mininet-

[9] YOLO, https://pjreddie.com/darknet/yolo/

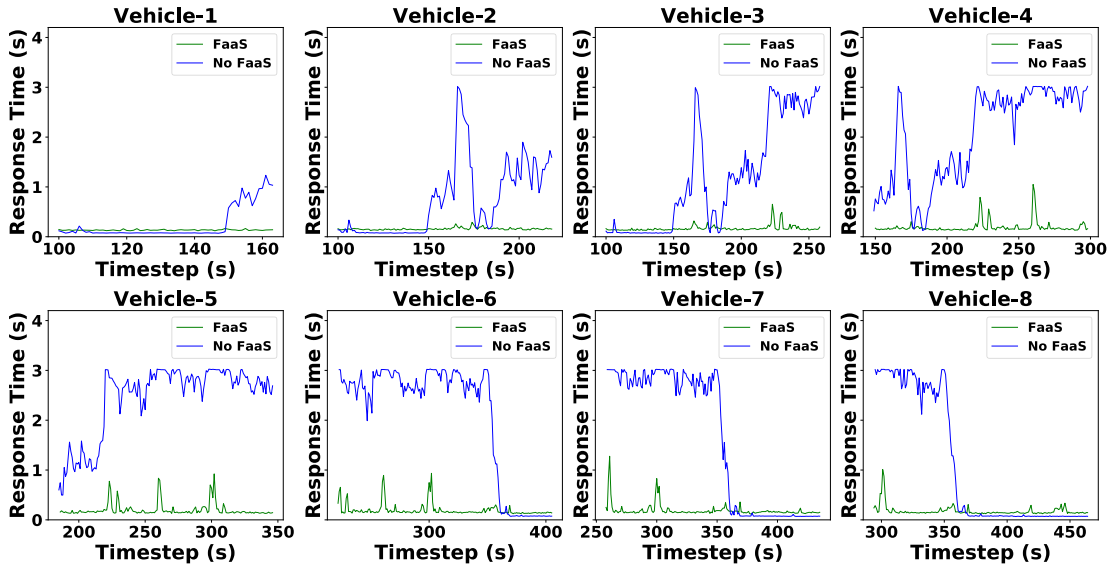

Figure 4. Response time with FaaS and without FaaS per HTTP request

WiFi). Mininet-WiFi automatically updates the received signal strength based on the updated vehicle position and establishes a connection with the RSU accordingly.

A Graphical User Interface (GUI) is also created to visualize the movement of vehicles on the road. Figure 5 depicts a sample screenshot of the traffic visualizer where the green triangle represents RSU and blue circles represent vehicles labeled with the number of running containers (pod) hosted by that RSU/vehicle at different times. When a vehicle connects to the RSU, a python script in the vehicle sends a command to the swarm manager to connect the vehicle as a worker node. To emulate requests, a traffic generator script runs at each vehicle which sends HTTP requests with an image as payload to the deployed API gateway for the YOLO functions every 0.25 seconds asynchronously. The response time for each request and the number of replicas at each time step is recorded for evaluation.

For comparison, we perform another experiment without OpneFaaS (No FaaS) by installing YOLO function of size 1.35GB directly at the RSU with no autoscaling option. With the same SUMO traffic traces and the same workload generator at each vehicle as above, we check the response times and success rates for the requests sent. The emulation is run on the road for 465 seconds, in which eight vehicles were generated

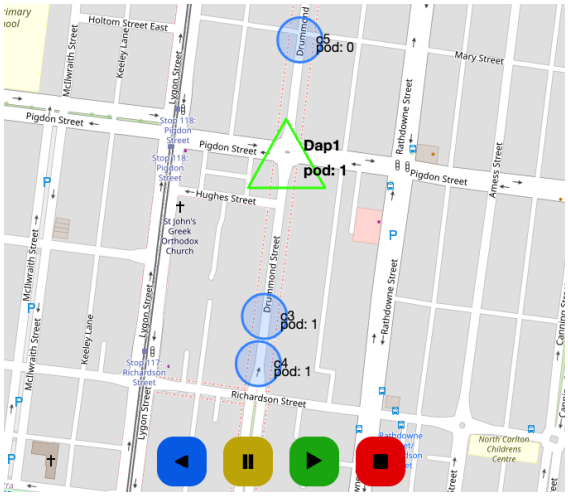

Figure 5. Screenshot of the traffic visualizer.

in the scene. We exclude the first 100 seconds of results for a warm-up and system stability.

Results

Results show that 99.3% of total HTTP requests are successful with FaaS while only 90.4% of requests are successful for No FaaS when request timeout is set at 3 seconds. Figure 3 shows the Cumulative Distribution Function (CDF) of the response times. We can see that with FaaS, 90% of requests are completed within 250ms while only 40% of requests are completed in the same for No FaaS. Figure 7 shows the total

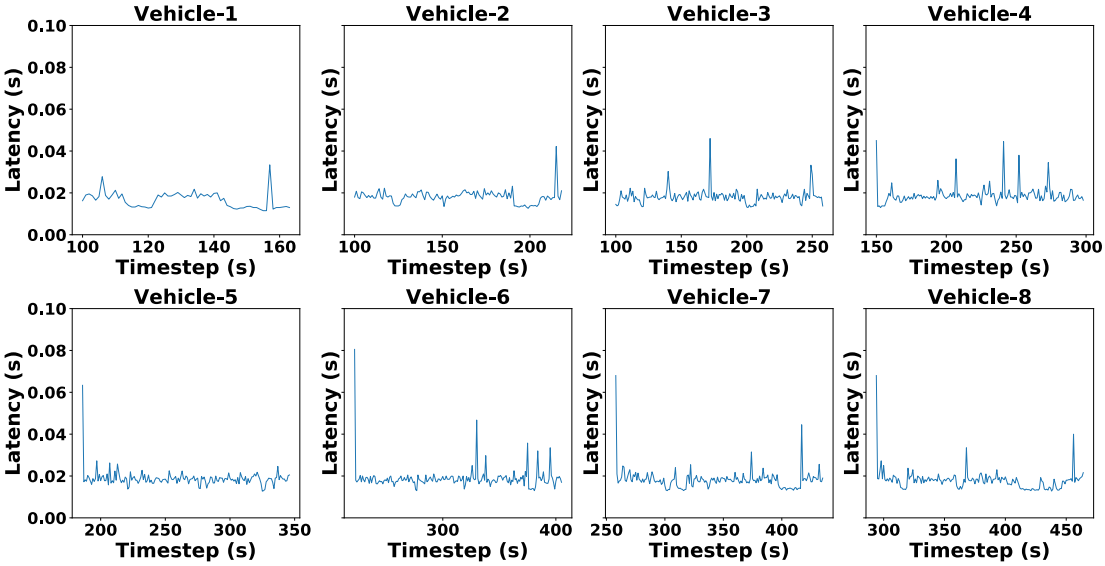

Figure 6. Network latency per HTTP request

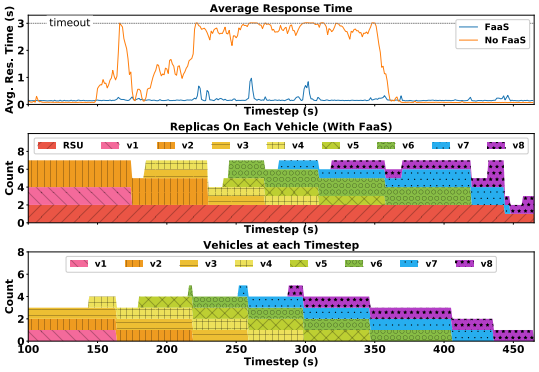

Figure 7. Number of vehicles and replicas

number of replicas and the number of vehicles present at each time step. With multiple vehicles in the scene and each sending requests for function execution, OpenFaaS detects a surge in HTTP requests and auto-scales function replicas accordingly. Docker Swarm uses *Spread* strategy for replica placement as default. Thus it tries to spread replicas evenly on all the vehicles in the vicinity and the RSU itself. As it can be seen in Figure 7, the number of replicas increases with an increase in the number of vehicles. It can also be observed that despite the increase in load, autoscaling results in a stable response time, whereas the response time increases considerably with load in No FaaS. Figure 4 illustrates the response times for each request made by vehicles in the vicinity. It can be observed that the response times for each vehicle using *FaaS* are mostly lower compared to *No Faas*. Figure 6 shows the average latency encountered in sending the HTTP request to the RSU. It can be seen that the latency encountered by the network is much lower when compared with the corresponding response times.

Similarly, Figure 4 displays the response time of each vehicle with FaaS and No FaaS.

Conclusions and Future Directions

In this work, we advocated the idea of serverless for vehicular edge computing and discussed its advantages, limitations, and challenges in early adoption. To study and experiment on serverless VEC, we presented an emulation built on open-source frameworks and developed a prototype. Through experimenting with our prototype, we identified that serverless VEC can provide promising for task offloading and provides reasonably low and stable response times even for compute and bandwidth-intensive functions like object detection in images.

The area of vehicular edge computing has recently attracted significant interest in the scientific community. This work is only a preliminary step towards a mature realization of the serverless VEC concept. As a straightforward continuation of our activities, we foresee building a custom swarm manager to cherry-pick appropriate vehi-

cles for replica placement, where the choice of vehicle for placement could depend on how long and well it could cater to the offloaded tasks. Furthermore, since the vehicles are moving at various speeds with varying bandwidths, optimized load-balancing schemes are required for better response times. Further, the emulation architecture can be improved in terms of scalability; one solution could be to load balance such that the heavy tasks of simulation and function execution are distributed across multiple servers. To further enrich the innovation of this work, we are considering conducting more extensive experiments in our future work, which will include comparison with other baselines and additional metrics such as analyses from network, latency, and fairness perspectives.

Acknowledgments

We would like to thank Mr. Asama Qureshi for his contribution to the traffic v isualizer application. The authors would also like to acknowledge support through the Australian Research Council's funded projects DP230100081 and FT180100140.

REFERENCES

1. S. Talal, W. S. M. Yousef, and B. Al-Fuhaidi, "Computation offloading algorithms in vehicular edge computing environment: A survey," in *2021 International Conference on Intelligent Technology, System and Service for Internet of Everything (ITSS-IoE)*, 2021, pp. 1–6.
2. L. Liu, C. Chen, Q. Pei, S. Maharjan, and Y. Zhang, "Vehicular edge computing and networking: A survey," *Mobile networks and applications*, vol. 26, pp. 1145–1168, 2021.
3. M. S. Aslanpour, A. N. Toosi, C. Cicconetti, B. Javadi, P. Sbarski, D. Taibi, M. Assuncao, S. S. Gill, R. Gaire, and S. Dustdar, "Serverless edge computing: Vision and challenges," in *2021 Australasian Computer Science Week Multiconference*, ser. ACSW '21. New York, NY, USA: ACM, 2021.
4. M. Gramaglia, P. Serrano, A. Banchs, G. Garcia-Aviles, A. Garcia-Saavedra, and R. Perez, "The case for serverless mobile networking," in *IFIP Networking Conference (Networking)*, 2020, pp. 779–784.
5. N. Apostolakis, M. Gramaglia, and P. Serrano, "Design and validation of an open source cloud native mobile network," *IEEE Communications Magazine*, pp. 1–7, 2022.
6. S. Al-Sultan, M. M. Al-Doori, A. H. Al-Bayatti, and H. Zedan, "A comprehensive survey on vehicular ad hoc network," *Journal of Network and Computer Applications*, vol. 37, pp. 380–392, 2014.
7. S. Eismann, J. Scheuner, E. v. Eyk, M. Schwinger, J. Grohmann, N. Herbst, C. L. Abad, and A. Iosup, "The state of serverless applications: Collection, characterization, and community consensus," *IEEE Transactions on Software Engineering*, vol. 48, no. 10, pp. 4152–4166, 2022.
8. P. Patros, J. Spillner, A. V. Papadopoulos, B. Varghese, O. Rana, S. Dustdar, and S. Dustdar, "Toward sustainable serverless computing," *IEEE Internet Computing*, vol. 25, pp. 42–50, 11 2021.
9. R. Olaniyan, O. Fadahunsi, M. Maheswaran, and M. F. Zhani, "Opportunistic edge computing: Concepts, opportunities and research challenges," *Future Generation Computer Systems*, vol. 89, pp. 633–645, 2018.
10. V. Shankar, K. Krauth, K. Vodrahalli, Q. Pu, B. Recht, I. Stoica, J. Ragan-Kelley, E. Jonas, and S. Venkataraman, "Serverless linear algebra," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, ser. SoCC'20. New York, NY, USA: Association for Computing Machinery, 2020, p. 281–295.
11. I. E. Akkus, R. Chen, I. Rimac, M. Stein, K. Satzke, A. Beck, P. Aditya, and V. Hilt, "SAND: Towards High-Performance serverless computing," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. Boston, MA: USENIX Association, july 2018, pp. 923–935.
12. D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, "Keystone: An open framework for architecting trusted execution environments," ser. EuroSys'20. New York, NY, USA: Association for Computing Machinery, 2020.

Faisal Alam is a PhD student in DisNet Lab, at the Faculty of Information Technology, Monash University, Australia. He is working on resource management and scheduling algorithms for vehicular edge computing. Contact him at *faisal.alam@monash.edu*.

Adel N. Toosi is the director of DisNet Lab and a Senior Lecturer at the Faculty of Information Technology, Monash University, Australia. His research interests include Cloud and Edge Computing, Serverless Computing, and Sustainable IT. Contact him at *adel.n.toosi@monash.edu* and more information at http://adelnadjarantoosi.info/.

Muhammad Aamir Cheema is an ARC Future Fellow

and Associate Professor at Faculty of Information Technology, Monash University, Australia. He is the Co-Director of Urban Computing Lab and is interested in the development of sustainable cities. Contact him at *aamir.cheema@monash.edu*.

Claudio Cicconetti has a PhD in Information Engineering from the University of Pisa, Italy and he is a Researcher at IIT-CNR. He is interested in serverless edge computing and Quantum Internet architecture and protocols. Contact him at *c.cicconetti@iit.cnr.it*.

Pablo Serrano is an Associate Professor at the University Carlos III de Madrid. His research interests lie in the analysis of wireless networks. He currently serves as Editor for IEEE Open Journal of the Communication Society. Contact him at *pablo@it.uc3m.es*.

Alexandru Iosup is a tenured full Professor and University Research Chair with the Vrije Universiteit Amsterdam, the Netherlands. He is elected Chair of the SPEC-RG Cloud Group. For scientific a nd community merits, he has been knighted. Contact him at *a.iosup@vu.nl*.

Zahir Tari is a full professor in Distributed Systems at RMIT and the Research Director of the RMIT Centre of Cyber Security Research and Innovation (CCSRI). Zahir's expertise is in the areas of system's performance (e.g. P2P, Cloud, IoT) as well as system's security (e.g. SCADA, SmartGrid, Cloud, IoT). He is the co-author of over six books (John Wiley, Springer) and he has edited over 25 conference proceedings. Contact him at *zahir.tari@rmit.edu.au*.

Majid Sarvi is the chair of transport engineering and the director of Transport Technology program at the University of Melbourne, Australia. His expertise covers a range of topics, including Artificial Intelligence in Transport, connected and automated multimodal transport systems, and ITS. Contact him at *majid.sarvi@unimelb.edu.au*.