Vehicular-OBUs-As-On-Demand-Fogs: Resource and Context Aware Deployment of Containerized Micro-Services

Hani Sami, Azzam Mourad, *Senior Member, IEEE*, and Wassim El-Hajj, *Senior Member, IEEE*

***Abstract*— Observing the headway in vehicular industry, new applications are developed demanding more resources. For instance, real-time vehicular applications require fast processing of the vast amount of generated data by vehicles in order to maintain service availability and reachability while driving. Fog devices are capable of bringing cloud intelligence near the edge, making them a suitable candidate to process vehicular requests. However, their location, processing power, and technology used to host and update services affect their availability and performance while considering the mobility patterns of vehicles. In this paper, we overcome the aforementioned limitations by taking advantage of the evolvement of On-Board Units, Kubeadm Clustering, Docker Containerization, and micro-services technologies. In this context, we propose an efficient resource and context aware approach for deploying containerized micro-services on on-demand fogs called Vehicular-OBUs-As-On-Demand-Fogs. Our proposed scheme embeds (1) a Kubeadm based approach for clustering OBUs and enabling on-demand micro-services deployment with the least costs and time using Docker containerization technology, (2) a hybrid multi-layered networking architecture to maintain reachability between the requesting user and available vehicular fog cluster, and (3) a vehicular multi-objective container placement model for producing efficient vehicles selection and services distribution. An Evolutionary Memetic Algorithm is elaborated to solve our vehicular container placement problem. Experiments and simulations demonstrate the relevance and efficiency of our approach compared to other recent techniques in the literature.**

***Index Terms*— Vehicular on-boarding units (OBUs), on-demand fog placement, vehicular fog computing, vehicular edge computing, vehicular clustering, orchestration, container, micro-services, Kubeadm, Docker, memetic algorithm.**

I. Introduction

VEHICLES are mobile sensors generating high volume of data that should be processed in real-time. Hence, the demands of time critical applications require fog computing to bring cloud intelligence near the edge by hosting services next to users. Fog devices minimize the load on the cloud by doing the major work, reduce networking delay, enhance user's devices battery life, and improve QoS for various types of applications [1]. One of the essential time-sensitive vehicular applications that demand fog computing is the self-driving feature. Autonomous driving requires low latency and high processing power to offer an accurate analysis of sensed data in order to make decisions, identify objects, and plan trajectories. These data are generated by integrated sensors, beams of radars, data shared by neighboring vehicles, etc.

Many literature works explore the advantage of adding a fog layer at the edge to support users. For instance, [2], [3] [4] use static locations of fogs with specific services running on them to target specific group of IoT devices. Similarly, [5]–[7] use pre-defined locations of fogs, such as road side units (RSU) to support vehicular users. These approaches use Virtual Machines (VM) to migrate and host fog services based on the user's needs (on-demand). Furthermore, Authors in [8] proposes a model for offloading vehicular computations to RSUs. Despite the advantages RSUs bring to support vehicles, major limitations still need to be addressed. For instance, geographically fixed fogs (RSUs) have limited network coverage and, therefore, limited availability to serve cars. Besides, using a VM to host services on fogs leads to high deployment delay and major resource consumption. This results in low Quality of Service (QoS) and limited service flexibility when updates are needed. Furthermore, considering the case when RSUs are overloaded with requests, its performance can degrade to affect the response time.

In parallel, personal devices (smartphones, laptops...) and more importantly vehicles' on-board units (OBU) have evolved from simple devices that can track the vehicle location and speed, to networking devices capable of communicating with neighboring vehicles and opening a stable connection with servers on the cloud. OBUs are also capable of performing various computation tasks, depending on their resource capabilities. Buses in modern cities are now equipped with multiple OBUs that can support applications to help drivers and provide luxury to passengers. In this context, the improvement in wireless technology, the new development of services, and the evolution in OBUs are disclosing a potential in enhancing the vehicular applications. This potential lies in employing the available technologies and resources into vehicular fog computing clusters that can enrich the user experience and allow a new development paradigm targeting the vehicular market. Yet, the main challenges are:

- OBUs and personal electronic devices still lack the resources capacity to efficiently download and start heavy VMs by their own [9].

Manuscript received October 21, 2019; revised December 28, 2019 and January 23, 2020; accepted January 26, 2020; approved by IEEE/ACM Transactions on Networking Editor H. Shen. Date of publication March 11, 2020; date of current version April 16, 2020. This work was supported in part by the Lebanese American University (SRRC-P-2019-65) and in part by a grant from the University Research Board of the American University of Beirut (URB-AUB-25514). *(Corresponding author: Azzam Mourad.)*

Hani Sami is with the Department of Computer Science, American University of Beirut, Beirut 1107 2020, Lebanon, and also with the Department of Computer Science and Mathematics, Lebanese American University, Beirut 1102 2801, Lebanon.

Azzam Mourad is with the Department of Computer Science and Mathematics, Lebanese American University, Beirut 1102 2801, Lebanon (e-mail: azzam.mourad@lau.edu.lb).

Wassim El-Hajj is with the Department of Computer Science, American University of Beirut, Beirut 1107 2020, Lebanon.

This article has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors.

Digital Object Identifier 10.1109/TNET.2020.2973800

1063-6692 © 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Bibliothèque ÉTS. Downloaded on June 25,2021 at 06:28:33 UTC from IEEE Xplore. Restrictions apply.

- Car mobility and erratic behavior of drivers make it a challenge to maintain a stable network connection between cars and service providers.
- Connecting users with vehicular fogs can exhaust the Base Stations (BS) due to their limited resources [10].

Therefore, forming vehicular fog requires computing and storage resources, dynamic service updates, and reliable connection between vehicles.

In this paper, we overcome the aforementioned limitations by proposing Vehicular-OBUs-As-On-Demand-Fogs. We make use of one of the well-known containerization technologies, Docker, to deploy our services on OBUs on the fly while transforming cars to fog devices. Vehicles in our architecture form a cluster that can host micro-services related to one service or more. The advantage of micro-services can be summarized as a lighter load on small machines, enabling distributed processing, and easier build and maintenance for applications [11]. These clusters are managed by orchestrators/masters using Kubernetes utility Kubeadm [12]. We also adopt a technique proposed in [10] to provide a long-time stable connectivity between users and serving clusters. To the best of our knowledge, we are the first to propose a scheme that uses OBUs and personal devices to host fogs embedding containers of micro-services on the fly, and at the same time, provide long term support for moving users regardless of their connection status. Through our approach, we can open the door for a new development area serving the new generation of vehicular applications. The main contributions of this paper can be summarized as:

1) Proposing a novel Vehicular-OBUs-As-On-Demand-Fogs framework that efficiently initializes clusters of vehicles to benefit from OBUs and on-board resources to push and manage services with the least possible costs.
2) Building an efficient and adaptable networking architecture combining cellular technologies and the vehicular ad-hoc wireless network (802.11p) to maximize the connection time between vehicles.
3) Formulating and solving a resource selection and micro-services placement problem on clusters of vehicles, named "Vehicular Container Placement (VCP)", and adapt its evaluation to different scenarios such as the number of pushed services, micro-services connection time, cluster lifetime, and the number of active vehicles.

The rest of this paper is organized as follows. In section II, we go over the related work and discuss their limitations. In section III, we discuss our proposed architecture and methodology. We then extend our approach by formulating and presenting a solution for the VCP in section IV. We then discuss a solution to recover from cluster failures in section V. Sections VI and VII are depicted for testing our architecture and the VCP solution. Finally, we conclude with future directions in section VIII.

II. Related Work

In this section, we overview the contributions of several related literary works and discuss their limitations. The areas selected are related to supporting mobile and vehicular applications through (1) the use of cloud and RSUs as processing power and VMs for hosting services, (2) the use of vehicular nodes as computing infrastructure to serve other vehicles, (3) the use of containerization in a fog-related contexts to serve IoT devices, (4) the use of hybrid network architecture to keep vehicles connected, and (5) the use of Memetic algorithm (MA) to solve service placement problems on resource infrastructure.

A. Approaches Supporting Mobile and Vehicular Applications

Several approaches proposed to support mobile and vehicular applications by migrating tasks to the cloud in the context of mobile cloud [13], [14]. However, these techniques are subject to network delays with the cloud, which affects time sensitive applications. To solve these problems, several approaches proposed the use of fog/edge computing to support the future of VANET/MANET applications, such as [7]. The authors in [7] proposed the idea of RSU clouds that are managed by an SDN controller and uses virtualization for services migration to support the internet of vehicles. Three drawbacks affect this architecture. First, SDNs are subject to network and computational delays when overloaded with traffic. Second, VMs are time-consuming to download. Third, is the assumption of RSUs presence anytime, which is not always true in real life. In [15], the authors proposed a follow-me cloud (FMC) approach that migrates services on anchor routers near the edge to support mobile users. Micro data-centers are used to cache VMs copies for later migration on the edge. The authors assumed that routers can cover users in every BS cell and used VMs to host and migrate services while a user is moving.

In this paper, we prove by experiments how our proposed approach can overcome the limitations of [7] and [15], which also applies to different approaches using VMs and RSUs to support mobile and vehicular users.

B. Volunteering Vehicular Fogs as Infrastructure

Authors in [16] proposed the use of vehicular fogs as infrastructure named vehicular fog computing (VFC), where a central main fog covering a shopping mall can benefit from volunteering cars present in the parking area. This paper was limited to the use of known resources such as parked cars' computation power only. Furthermore, the technology used to push and run services on newly joining fog volunteers was not studied. Authors in [17] elaborated on the ability to use vehicles as infrastructure to build a fog environment. In their study, moving and parked vehicles can be used as infrastructure to support moving users. Consequently, we use in this paper OBUs and personal devices to build our fog solution for continuous vehicle support. In [18], authors create an on-demand vehicular cloud counting on RSUs. Finding a Star on the road is the key idea. A Star offers computation and storage power while moving. This information is shared with and published by the nearest RSU so that any user can reserve the star's resources to host its requested services. The coverage range of RSUs, network disconnection with the Star, and service updates/installments affect the feasibility of this solution. In [19], authors use neighboring mobile devices' resources to offload mobile tasks by forming a mobile edge cloud, which confirms the potential of using mobile devices to perform computational tasks.

C. Fog Containerization

To the best of our knowledge, none of the available work in the literature considers the use of containerization technology to host services on vehicles. However, many others used containers in the context of fog devices to serve IoT users

Authorized licensed use limited to: Bibliothèque ÉTS. Downloaded on June 25,2021 at 06:28:33 UTC from IEEE Xplore. Restrictions apply.

located in fixed locations. In this section, we discuss some of these contributions.

Authors in [20] prove the potential of using Docker technology to run containers on fog devices with the ability to adjust services hosted whenever needed. A model was proposed in [21] where the dynamic deployment of services on helper nodes of the main server using Docker is possible. They called these helper nodes "fogs". So it is feasible to remove, add, stop, and run any service on a physically known fog anytime. In [22], authors focused on the ability to use lightweight Docker containerization technology to support service provisioning over IoT devices.

In our recent work [23], [24], we were able to implement a framework capable of forming fog devices on the fly on any volunteering devices. This was possible with the help of Kubeadm utility and Docker containerization technology to form the cluster and push the needed services. In this paper, we adopt the use of the same technologies but in the context of Vehicluar-OBUs-As-On-Demand-Fogs, where it is more challenging to maintain the service availability and provide an optimal service placement on randomly moving vehicles.

D. Vehicular Hybrid Network Architecture

Authors in [10] proposed a hybrid network architecture to keep a VANET connected with the help of e-NodeBs. The master node uses two network adapters: (1) Long Term Evolution LTE to connect cluster's masters and (2) Ad-hoc vehicular wireless connection 802.11p to connect vehicles in the same cluster. In our work, we adopt this architecture for the purpose of keeping vehicles connected by adding another layer of RSUs that can further minimize the load on BSs.

E. Vehicular Container Placement Problem

In [25], researchers proposed an interactive MA to solve the proposed multi-objective formulation of the virtual machine placement problem on the cloud. The goal is to find an efficient distribution of VMs on corresponding available hosts concerning the conflicting objective functions. This problem can be mapped to our VCP problem by considering the VMs as containers, and the available hosts to run the VMs as the vehicular fog devices.

III. Vehicular-OBUs-As-On-Demand-Fogs Architecture

A single OBU is not enough to satisfy all users' requests; therefore, hosting services on neighboring cars having similar driving patterns can be a solution. The connection time between vehicles should be maximized in order to maintain a high rate of request/response packets delivery. Furthermore, OBUs must be clustered and monitored in a Master-Slave approach in order to increase resource capacity. Services should be divided into micro-services for lighter deployment in the form of containers. Another problem that arises while trying to initialize the vehicular Kubeadm cluster is the master node election for a group of available OBUs. Also, whenever a failure or disconnection happens in the cluster, it is costly to re-initialize a new cluster while the user is waiting for services. Therefore, a recovery method must be applied to maintain service availability and cluster connectivity.

In this section, we present our proposed architecture to overcome the aforementioned challenges. We then provide a description of the role of each component in the architecture per layer. Finally, we discuss a possible interaction between the components using a test case.

A. Architecture Overview

The architecture shown in Fig. 1 is composed of five linked layers: BS, Road Side Unit Controllers (RSUC) - RSUs, Kubeadm masters, fog devices (Kubeadm worker nodes), and requesting users. This architecture is layered by power and importance from top to bottom. In other words, a layer above has supervision of what is happening on the layers below. Moreover, if a layer below fails to maintain a connection between a fog and a user, the request is escalated to the layer above. A connection between BS-RSUC-RSU is called infrastructure to infrastructure (I2I). The top layer is composed of the BS or cellular towers that are connected using Ethernet. In our architecture, the task of a BS is to connect a requesting user to a cluster master node in another BS range and broadcast a user request for service hosting to the underlying RSUs through RSUCs using Ethernet. The RSUC routes requests between RSUs and shares their information, including the position of the master nodes they have in range, the services hosted by every Kubeadm cluster, and the list of Docker images they have. The third layer is composed of RSUs storing images of the users' requested services, and it tracks the position of the serving Kubeadm master node to connect users with services. This type of connection is called vehicle to infrastructure (V2I). The purpose of adding the RSUC-RSU layer is to minimize the traffic load and management on BSs. The next layer comprises the Kubeadm master nodes, which cluster the fog devices. Masters or orchestrators are dual-interface devices able to connect using 802.11p or through the cellular network. Master cars are elected locally, and they keep a vital connection with the BS, RSU, and fogs in the range all the time. The main job of the master node is to decide on the best distribution of services on the set of selected vehicular fogs and to monitor the containers' status. The master node sends resources offers to the user and waits for approval before pushing services to its cluster. A failure on the master node can be recovered whenever enough resources on the cluster are available to enable the high availability feature provided by Kubeadm. A device in the fog layer can communicate to a nearby car or the RSU using 802.11p only. It is responsible for hosting the assigned micro-services by the master node. The fog also updates the master node with its profile and availability. The communication between vehicles in a Kubeadm cluster is called vehicle to vehicle (V2V). This interconnection is kept through exchanging "Hello" packets using 802.11p. In Kubeadm, any failure in a worker node is reported to the master. The user initiates a request to host services nearby, and then receives several offers coming from available nearby clusters. A special decision algorithm running on the user side decides to accept offers that maximize its support time. The user can send another request to host other or similar services if not satisfied with the QoS level received.

B. Node Architecture

In this section, we discuss the functionalities of each component per layer in our proposed approach. The BS, RSUC, and RSU communicate together to keep the user connected with the master node by running the Master Manager. Moreover, the RSUC runs the Container Registry Manager responsible for providing Docker images to vehicular fogs. The Kubeadm Master and Fog nodes coordinate to form a stable Kubeadm cluster with high service availability and maximum duration of user support while driving. The master manages fogs and containers running on them through the Fog/Micro-Service

Authorized licensed use limited to: Bibliothèque ÉTS. Downloaded on June 25,2021 at 06:28:33 UTC from IEEE Xplore. Restrictions apply.

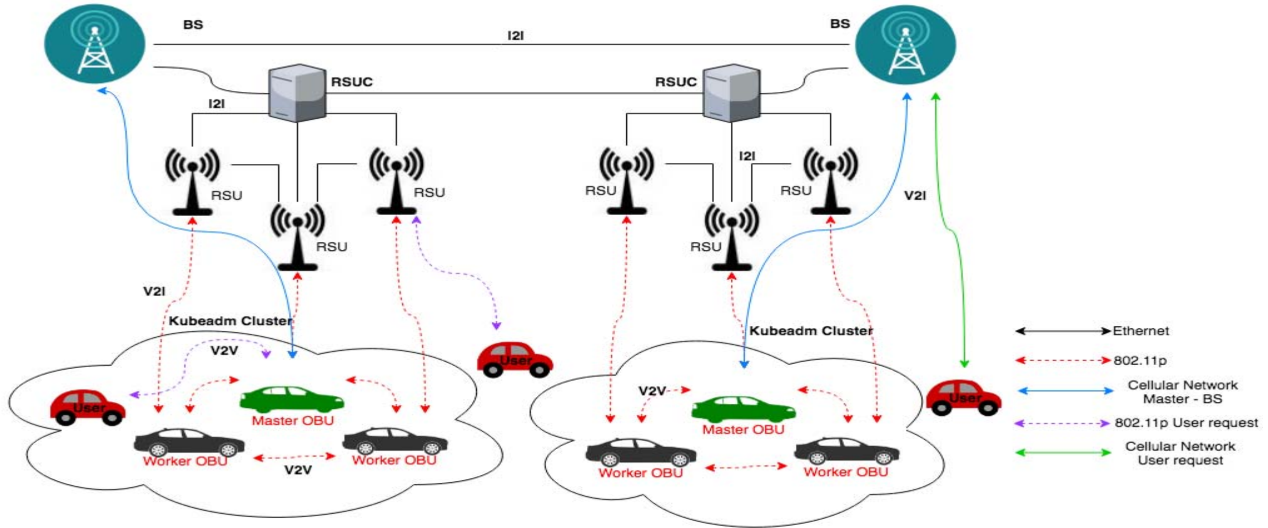

Fig. 1. Proposed Architecture.

TABLE I
NODE ARCHITECTURE PER LAYER

Layer	Component
Cellular Tower	- Master Manager
RSUC	- Master Manager - Container Registry Manager
RSU	- Master Manager - Container Registry Manager
Kubeadm Master	- Containerization Required Modules - Fog/Micro-Service Manager - Cluster Resource Manager - VCP - Profiler
Fog	- Containerization Required Modules - Kubeadm Cluster Initializer - Profiler
User	- Offer Acceptance Decision Module - QoS Manager

Manager, and decides on the best distribution of services on the set of available vehicles in its cluster. Containerization required modules should be installed on the cluster nodes. The user monitors the performance of the serving Kubeadm clusters through the QoS manager and also decides to send a service hosting request when needed. A user accepts an offer through the Decision Module for Offer Acceptance. The architecture components per layer are shown in Table I.

In what follows, we provide a detailed explanation of the functionalities within each component.

1) Master Manager: The RSU shares the master node information in range with the RSUC and BS. When the master node moves into the range of a new RSU, the old RSU notifies its neighbors of the joining master node. The new RSU, in turn, starts sending "Hello" packets to the master to ensure a vital connection. The Hello reply messages contain the master profile. The main functionalities of the master manager are to:

Connect User and Master Node:

- In case the module is running on BS: The BS receives a connection request from a user to a master node in two cases: either they are in the range of the BS but cannot communicate through an RSUC/RSU, or they are in the range of two different BSs. This connection uses the wireless cellular network.
- In case the module is running on RSUC/RSU: The RSUC/RSU is used to connect a user with a master node even if under different RSUs ranges. The RSU is aware of the master location within an RSU range. This helps to minimize the network load by limiting the broadcast messages into one RSU range. This connection uses 802.11p.

Send Requests and Collect Offers: The BS, RSUC, or RSU receive requests from users to allocate resources on available Kubeadm OBUs clusters. This request is broadcasted to all master nodes either through the BS or the RSU. Master nodes reply with the available resources. The BS or RSU replies to the user with the available clusters to serve (offers sent from the masters). Furthermore, a cluster can serve one or many users depending on their current load and suitability for the user based on the mobility metric. Accordingly, if clusters with the requested service(s) were already initialized, the RSU may also include them as other possibilities in the offer, along with their current load. The RSU then waits for an acceptance or rejection decision from the user.

Master Status Monitoring and Failure Recovery: In the case of a master node leaving the cluster, a recovery algorithm is triggered by the master manager to maintain the cluster connectivity and service availability. This algorithm is described in Algorithm 3 of Section V.

2) Container Registry Manager: Usually, the container registry is placed on the cloud for users to push and pull images from. In our architecture, we bring these registries closer to the user to minimize the time of pulling the required micro-services. The RSU has a container image registry holding micro-services frequently requested by users. All RSUs share information about services they are hosting to their RSUC by replying to the container registry manager's request. In case a fog node asks the RSU for an image it does not have, it asks the RSUC for the location of this service. Once found, the RSU uses Ethernet to get the image, and 802.11p at a maximum rate of 54Mb/s to send it to the fog.

3) Containerization Required Modules: Kubeadm is a suitable orchestration technology in our architecture because it supports using various types of devices forming the cluster. Kubeadm orchestration uses the de-facto

Authorized licensed use limited to: Bibliothèque ÉTS. Downloaded on June 25,2021 at 06:28:33 UTC from IEEE Xplore. Restrictions apply.

containerization technology, Docker. Kubelet should be installed on the device to ensure communication with the master node and allows different pods to communicate together. Kubelet also helps the master node in checking services' health. Kubectl is used to run Kubeadm commands on the master and worker nodes. Docker and Kubeadm, with the mentioned dependencies, should be installed on vehicles to form the clusters.

4) Profiler: This component runs on the master and worker nodes of a Kubeadm cluster. The module uses GPS to get the current position of the vehicle and calculate the speed. The profiler also gathers information about the available CPU, memory, and disk space on the vehicle. The route followed to reach the destination is provided through the car system. The vehicle might also offer its services for a specific period (can be used for billing purposes), so the time availability is also provided by the profiler.

5) Fog/Micro-Service Manager: This component runs on the master node and is responsible of managing the worker nodes in the Kubeadm cluster as well as the running containers. The purpose of this component is to keep the fog nodes available and to overcome any physical or service failures. The main functionalities can be described as follows:

Keep Connection Alive: Because of the random mobility of vehicles, a connection has to be checked periodically between fogs and master nodes. Likewise, the master exchange Hello packets with the RSU containing the cluster profile. Connection checks functionality is implemented in Kubeadm.

Assign Services to Fogs: After getting an offer approval from the user, the master assigns services to fogs based on the VCP algorithm. The master sends a pull command containing the list of micro-services to be installed on each node.

Fog Status Monitor and Failure Recovery: This functionality is triggered in two cases. Either a container on the fog stopped running and failed to restart, or the fog suddenly went out of the cluster. This algorithm is discussed in Algorithm 2 of Section V.

Load Balancing on Fogs: The master node is also responsible for monitoring the load on fog devices. When needed, the master node either creates more copies of the overloaded containers in its cluster. This feature is provided by Kubeadm.

6) Cluster Resources Manager: After receiving a service installment request from a user, the master uses the Cluster Resources Manager to get a matrix of the available resources. This cluster information is used to get the maximum time availability, speed, and maximum destination reached by all nodes in the cluster. This information are used to construct an offer message for the user to accept or reject.

7) VCP: The VCP is a decision module for volunteers selection and micro-services distribution. This module is presented in Section IV. Its purpose is to select the best set of vehicular resources to host a set of requested services.

8) Kubeadm Cluster Initializer and Master Node Election: Kubeadm cluster initialization includes the time to initialize a master node and for worker nodes to join and install the required modules. We avoid this cost by making the cluster ready beforehand. The first step in the cluster initialization is the election of a master node, which happens locally.

Master Node Election: Once a pair or group of cars approach each other, a master node election based on QoS metric takes place to initialize the Kubeadm master node on the proper device. The QoS metric includes the bandwidth, speed, distance from neighboring cars, serving time, and resources available to host the required orchestration modules. The high bandwidth is important to ensure reliability with the BS, and the speed, distance, and serving time are necessary to maintain cluster stability. Implementation of the master election algorithm is provided in [26] as part of the QoS-OLSR protocol. The election between vehicles is happening locally, where each car calculates its QoS score. Every vehicle receives all QoS scores from 2-hops neighbors based on multiple vote strategy. Based on the highest score, a master node is nominated. Therefore, multiple neighboring clusters can be created and connected through Multi-Points Relay nodes. The QoS is calculated based on the proportional bandwidth and proportional speed as follows:

$$QoS_j = \frac{BW_j}{N_j} \times \frac{RatioD_j}{RatioAvgS_j} \tag{1}$$

where:

- j is a vehicle in the set of available ones.
- BW_j is the bandwidth that j can offer.
- N_j is the set of neighbors of j.
- $RatioD_j$ is the ratio distance bypassed by j before reaching its destination.
- $RatioAvgS_j$ is the average ratio speed of j.

Cluster Formation: After electing the master node, Kubeadm init command is executed on the master. This command outputs a unique token to be shared with nodes willing to join using Kubeadm Join command. The required modules in our architecture are installed on the master node. The architecture modules on the fog layer are also installed on the worker nodes to make them ready to pull and run services on the fly.

Resource Sharing Between Clusters: Once a cluster is formed, the master waits for users' requests to host services and start utilizing the cluster's resources. However, because these resources may stay in some cases idle for a long time, a cluster can share resources with neighboring masters upon request. Consequently, an idle vehicle checks for the possibility of joining the requesting clusters based on its mobility and resources metrics.

9) Offer Acceptance Decision Module: In order to avoid the limitations of availability and computation overload on RSUs, this model should be placed on the user side. A user receives multiple offers from different master nodes. The user accepts the offer that helps in reaching the minimum required QoS for the longest period while moving towards its destination. A machine learning model is useful in this situation to study the history of accepted offers by the user. However, this is out of the scope of this work. For now, we assume that the user accepts the best offer by default.

10) QoS Manager: The QoS level served to the user is checked by the QoS Manager by measuring the processing and networking delays. If the minimum QoS required by the service is not met, the user sends a request for service hosting to the RSU.

C. Components Interactions When Hosting a New Service

Below, we show the flow of interactions between different components when a user issue a request to host services on a vehicular fog cluster. The flow is as follows:

1) The QoS manager initiates a service hosting request to the nearest RSU.
2) The master manager of this RSU receives the user request and broadcasts it to the available masters.

Authorized licensed use limited to: Bibliothèque ÉTS. Downloaded on June 25,2021 at 06:28:33 UTC from IEEE Xplore. Restrictions apply.

TABLE II
TABLE OF NOTATIONS

Variable	Description
m	Set of available vehicles
n	Set of requested services
$orch_j$	Kubeadm master node of V_j
n_{id}	Number of micro-services of service id
S	Set of services
V	Set of vehicles
L	Set of serving vehicles
S^i_{id}	Micro-service i of service id
C^i_{id}	CPU usage of $S_{id,i}$
M^i_{id}	Memory usage of $S_{id,i}$
D^i_{id}	Disk usage of $S_{id,i}$
T_{id}	Minimum time to host S_{id}
V_{cpu_j}	CPU available on V_j
V_{m_j}	Memory available on V_j
V_{d_j}	Disk space available on V_j
$V^t_{l_j}$	Current position of j as long and lat at time t
V_{t_j}	Departure time of V_js
T_j	Time for V_j to reach its destination
$AvgS_j$	The average speed of worker V_j
U_l	Maximum distance between user and cluster
$H_{cluster}$	End distance between user and cluster
$T_{cluster}$	Cluster time availability

3) A master node receives the service request and calls the cluster resources manager to check its availability.
4) The cluster resources manager collects all the workers' profile information from the fog/micro-services manager and asks the VCP for a placement decision. This information is sent back to the user as an offer.
5) When the user accepts the offer, the RSU starts preparing the micro-services to be hosted on the new fogs and informs the orchestrator to reserve the resources.
6) In case the RSU does not have the requested user services, it calls the RSUC's container registry manager.
7) The orchestrator pushes these services on its fogs.
8) The fog/micro-services manager uses the VCP output to place the micro-services on vehicular fogs.

IV. VEHICULAR CONTAINER PLACEMENT (VCP)

Vehicles in VANET tend to change their speed randomly, making it a challenge to maintain cluster stability. In our architecture, vehicles are the primary source of computational power, including any devices on board to host micro-services. Furthermore, fog nodes are asked to host different services based on the user's request, which makes it another challenge of mapping each micro-service to the proper vehicle. Finding the optimal migration of micro-services to available cars is an NP-hard problem. The objectives of getting the optimal solution for this problem are to maximize the cluster stability/connectivity or serving time, maximize the connection time between micro-services, maximize the number of pushed services, and minimize the number of active vehicles while meeting several constraints. In this section, we define the VCP Problem and prove it is NP-Hard. We then mathematically formulate our problem by identifying the input, output, constraints, and objective functions as a multi-objective optimization problem. Finally, we solve this problem through a MA. Notations are provided in Table II.

A. VCP Problem Definition

In this problem, we have a set of services having different requirements and a set of available vehicles that would potentially host a micro-service or more. The aim is to find the best distribution of these services on the set of available vehicles taking into account the resources available on them, requirements of services, network stability of the moving cluster, maintenance of attached micro-services, and proximity from the user during the serving time (required for low-latency applications). By reducing our problem to the Bin-Packing problem [27], we prove that VCP is NP-Hard. The traditional bin packing problem is described as follows. Suppose we have a set of objects with different volumes that need to be packed inside a finite number of bins of different capacities and volumes. The aim is to try maximizing the total objects packed in each bin and to minimize the number of used bins. This problem can be mapped to our problem as follows. Each bin is a vehicle having resources capacity, and the objects are services to assign for vehicles. Our objective is to maximize the number of pushed services while minimizing the number of active fog/vehicles, in addition to other objective functions. Thus our problem is NP-hard.

B. VCP Problem Formulation

We aim to optimize the number of pushed services, the number of serving vehicles, stability of the cluster, and attachment of related micro-services under similar vehicular conditions.

1) Input Data: As input, we have a set of services S having different requirements, a set of vehicles V with different offerings, and the mobility parameters to locate the user U.

- **Service:** The set of services S, each with a service id, are represented as a 2D matrix $S \in \mathbb{R}^{(n*n_{id})*4}$ having four attributes. A row in the matrix represents a microservice requirements as $S^i_{id} = [S^i_{cid}, S^i_{mid}, S^i_{did}, S_{tid}]$.
- **Vehicle:** The set of vehicles V is represented as a matrix $V \in \mathbb{R}^{n*7}$ that illustrates seven attributes of a vehicle. Each vehicle is described as follows: $V_j = [V_{p_j}, V_{t_j}, V_{s_j}, V_{c_j}, V_{m_j}, V_{d_j}, V_{oj}]$.
- **User:** A user U is represented as an array of four items as follows: $U = [U_s, U_t, U_p, U_d]$.

2) Output Data: The optimization solution aims to map each requested service to a vehicle node in its cluster. The output is a binary matrix K_{ij} of size $(n \times n_{id}) \times m$ where $K_{ij} \in\ 0, 1$. $K_{ij} = 1$ means that service S_i is hosted on vehicle V_j. Moreover, an offer message is constructed by K_{ij} and represents the serving time C_t of the cluster for each service, as well as the predicted end distance C_d from the user. In order to simplify the computation of C_t and C_d, we assume that v_s is a good representation of the vehicle movement, and that no recovery technique is used in case j leaves its cluster. C_t and C_d are calculated as follows:

- **Cluster Serving Time**: We write it as C_t, and is calculated as follows:

$$C_t = min(V_{a_j}) \quad \forall j | L_j = 1 \ and \ j \in \{1, \ldots, m\} \quad (2)$$

  In (2), V_{a_j} is calculated as the minimum between the time for j to reach its destination V_{t_j}, or the time to go out of range of its orchestrator VO_j.
- **Cluster Distance From User**: We write it as C_d. In order to compute C_d, we use C_t and VO_{s_j}. We also use the geographical coordinates of the vehicles (VO_p and U_p) as well as the degree indicating the direction of travel. The end position of the orchestrator and user specifies the distance separating them C_d. The initial distance between orchestrator and user is also calculated using their inital positions.

Authorized licensed use limited to: Bibliothèque ÉTS. Downloaded on June 25,2021 at 06:28:33 UTC from IEEE Xplore. Restrictions apply.

C_t and The initial and predicted end C_d are sent in the offer message. During the offer acceptance decision, the user takes into consideration the cluster time availability and proximity to the cluster in case the application is time sensitive.

3) Constraints: In this subsection, we present the different constraints that make a solution feasible. These constraints apply on services that are placed in the cluster.

Resources Limit: The total CPU, Memory, and Disk resources required by the hosted service on a vehicle should be less than its available resources. This constraint can be formulated as follows:

$$\sum_{id=1}^{n}\sum_{i=1}^{n_{id}} S_{c\,id}^{\,i} \times K_{ij} \leq V_{c_j} \quad (3)$$

$$\sum_{id=1}^{n}\sum_{i=1}^{n_{id}} S_{m\,id}^{\,i} \times K_{ij} \leq V_{m_j} \quad (4)$$

$$\sum_{id=1}^{n}\sum_{i=1}^{n_{id}} S_{d\,id}^{\,i} \times K_{ij} \leq V_{d_j} \quad (5)$$

$\forall j \in m,$ i.e., for all available hosts V_j

Minimum Cluster Serving Time: To guarantee that the cluster can serve the user for a reasonable time, we set a time threshold to be considered before sending an offer message to the user as follows:

$$C_t \geq S_{tid} \;\; \forall id \in n \quad (6)$$

All Coupled Micro-Services To Be Hosted: Micro-services should be coupled together to keep them connected in the same cluster and to avoid service downtime and delay issues. If all the micro-services composing service requirements cannot be met by a vehicular cluster, any micro-service of this service must not be pushed to the cluster. This constraint is formulated as follows:

$$\sum_{j=1}^{m}\sum_{i=id}^{n_{id}} K_{i_j} = n_{id} \quad \forall id \in n \quad (7)$$

Equation (7) implies that the number of micro-services of S_{id} placed in the cluster should be equal to the total number of micro-services of S_{id}.

Distance Threshold To User: In case a low-latency application requires the fog to be hosted near the user to avoid networking delay, VCP ensures that the distance between the cluster and the user does not exceed a certain value U_d already set based on the application need. This cluster-user distance constraint can be formulated as:

$$C_d \leq U_d \quad (8)$$

Weights Summation: Each objective function is multiplied by a weight associated with it. All the weights should add up to one. The purpose of these weights is to have a tradeoff in terms of the importance of each objective function. For example, to push as many services as possible no matter what the conditions are, we should set W_{f3} to a value greater than all other weights. In this case, the evaluation of f3 is affecting the sum of the optimization functions more than all other objective functions. It is expressed as follows:

$$W_{f1} + W_{f2} + W_{f3} + W_{f4} = 1 \quad (9)$$

4) Objective Functions: In this subsection, we present four contradicting objective for taking the VCP decision.

- **Maximize Cluster Lifetime:** Maximizing the cluster lifetime leads to maximizing the user serving time. This objective function aims to maximize the time availability of the kubeadm cluster by selecting fog vehicles that can keep connected to the master node for a longer period. We formulate this function by maximizing the minimum time it takes for one fog to go out of the cluster as follows:

$$F_1 = max(C_t \times W_{f1}) \quad (10)$$

- **Maximize Micro-Services Connection Time**: An alive connection between all micro-services of a service is important to ensure service availability. One micro-service does not function without the other. Therefore, in this function, our objective is to host micro-services in similar mobile conditions on different vehicles if possible. We make sure that a vehicle hosting micro-services of a service are approximately at the same distance from each other at different time steps as follows:

$$F_2 = min(\sum_{id=1}^{n}\sum_{i=id}^{n_{id}}\sum_{j=1}^{m}\sum_{i'=id}^{n_{id}}\sum_{j'=1}^{m}(((||V_{p_j}^{t} - V_{p_{j'}}^{t}| - |V_{p_j}^{t'} - V_{p_{j'}}^{t'}||) \times K_{ij} \times K_{i'j'}) \times W_{f2}) \mid i \neq i' and\; j \neq j' \quad (11)$$

where $V_{l_j}^{t} - V_{l_{j'}}^{t}$ indicates the distance separating V_j and $V_{j'}$ hosting service S_{id} at time t. In this function, S_{id} is selected in the first summation. The second and third summations retrieve the microservices of a service and their hosts. The host position of a micro-service is compared with all other hosts running the other parts of that service using the last two summations. The cycle repeats for the rest of services and hosts. This function minimizes the difference of distances between related hosts at different time steps $(t, t', t'' \ldots)$ to ensure that vehicles, hosting connected micro-services, stay close while moving.

- **Maximize Number of Pushed Services:** The aim is to maximize the number of pushed services the vehicular fog cluster as follows:

$$F_3 = max((\sum_{j=1}^{m}\sum_{id=1}^{n}\sum_{i=id}^{n_{id}} K_{ij}) \times W_{f3}) \quad (12)$$

By maximizing the number of pushed services, we guarantee that all users' requests for services are satisfied and services are deployed on vehicular fogs. F3 in (12) sums the number of placed micro-services of services on all vehicles.

- **Minimize Number of Active Vehicles**: The aim of this objective function is to minimize the number of active vehicles in order to minimize the monitoring load on the orchestrator, and make it easier to recover from workers leaving the cluster. It is expressed as:

$$F_4 = min((\sum_{j=1}^{m} L_j) \times W_{f4}) \quad (13)$$

Therefore, our multi-objective optimization problem is to optimize F, which includes optimizing the cluster lifetime,

Authorized licensed use limited to: Bibliothèque ÉTS. Downloaded on June 25,2021 at 06:28:33 UTC from IEEE Xplore. Restrictions apply.

the micro-services connection time, the number of pushed services, and the number of active vehicles:

$$F = [F_1, F_2, F_3, F_4] \tag{14}$$

These objectives are subject to the constraints of resources limits, minimum cluster serving time, and ensuring that either all micro-services of a service are pushed or none.

C. Memetic Algorithm to Solve VCP

It is important to get an efficient set of solutions for the container placement problem in a short time. The MA is a suitable solution for such problems [28]. It is built on top of the genetic algorithms. However, in addition to the optimization operators, it has a local optimization (local search) algorithm that can reach efficient solutions in early generations [29]. The MA proposed by the authors in [25] is amended to solve our optimization problem; Algorithm 1 illustrates the updated algorithm.

Algorithm 1 Multi-Objective Memetic Algorithm

Data: Set of containers
Result: Pareto set approximation p_{known}
1: Check if the problem has a solution
2: Initialize set of solutions P_0
3: P_0' = Repair infeasible solutions ofP_0
4: P_0'' = Apply local search to solutions of P_0'
5: Update set of non-dominated solutions p_{known} from P_0'
6: $t = 0$
7: $P_t = P_0''$
8: **While** (Stopping criterion is not met), do
9: Q_t = Selection of solutions from $P_t \cup p_{known}$
10: Q_t' = Crossover and mutation of solutions of Q_t
11: Q_t" = Repair infeasible solutions of Q_t'
12: Q_t"' = Apply local search to solutions of Q_t"
13: Increment t
14: Update set of non-dominated solutions p_{known} from Q_t"'
15: P_t = fitness selection from $P_t \cup Q_t$"'
16: **End while**
17: **Return** Pareto set approximation p_{known}

First, the algorithm checks that the problem has at least a feasible solution. If yes, a random set of solutions P_0 is initialized by randomly assigning images of services to available cars. In step 3, the set of available solutions in P_0 is repaired to avoid violations of the constraints. These violations are repaired in three ways: (1) Moving containers to other available vehicles in the cluster, (2) adding available vehicles to the list of running ones and moving Docker containers to them, and (3) removing containers from the list of services to be pushed. Step 4 of the MA is to apply a probabilistic local search method to optimize feasible solutions. If the probability is less than 0.5, the number of pushed services is maximized. On the other hand, if the probability > 0.5, the number of available volunteers is minimized. This way, the agent is trying to converge to an efficient solution at early the stages. Then the Pareto set approximation is generated at step 5. After the initialization of step 6, selection, crossover, and mutation operators are applied, infeasible solutions are repaired, optimization of solutions is done using probabilistic local search, iteration counter is incremented, and finally, the Pareto set is updated if any improvements happened. After that, a new population is selected. This process keeps on iterating until the algorithm meets the stopping criteria (e.g., the maximum number of iterations is reached). Finally, the fittest set of the solution p_{known} is returned. In this MA, we use the binary tournament for selecting individuals from the population to apply crossover and mutation on them. The crossover operator used is the single point cross-cut. The mutation used is the bit string, where each gene is mutated with probability $1/n$ where n is the number of services. This prevents stagnation in a local optimum.

The complexity of this algorithm is divided into four parts, as discussed in [30]: The generation of M chromosomes, crossover, mutation, and local search complexity time. Let M and N be the number of chromosomes and the number of nodes, respectively. The MA starts off using $O(M \times (n-1) \times log(n-1))$ time units to generate the random population. Also, let p_c and p_m be the probability of the mutation and crossover, respectively. The number of offsprings generated by the crossover uses $O(N \times p_c \times [M \times (N+1)])$, while the ones created by the mutation consumes $O(p_m \times [M \times (N+1)])$ of time units. The local search algorithm consumes $O(n)$. Therefore the combined time complexity of the MA is shown in equation 15 (given $p_m = 1/2$).

$$O((M \times (n-1) \times log(n-1)) + (N \times pc \times [M \times (N+1)]) + (1/2 \times [M \times (N+1)] + N)) \tag{15}$$

It is important to note that in [25], based on which we built our algorithm, the MA was proven to reach the exact optimal solution when the input size is small, and a near optimal solution when the input is large.

V. Cluster Recovery Algorithm

Vehicles in a Kubeadm cluster can host one service/micro-service or more. If a serving fog or master becomes undecided or isolated, the user loses connection with the service. Hence the service can either be requested from an RSU or from the cloud. To avoid such scenarios, we propose a Kubeadm cluster recovery algorithm. In this section, we described the proposed Kubeadm fog and master recovery algorithms.

A. Kubeadm Fog Recovery

If a fog vehicle is about to leave the cluster, or a connection is lost, the master node runs Algorithm 2 to recover from any potential failures. The master tries to check if another fog in its cluster can host the service of the leaving vehicle. In case there are no resources available on the workers, the master uses its resources to host the service, if possible. If not, the master asks the RSU or BS for temporary resources to host the missing service and maintain its availability. As a last resort, the RSU examines the available clusters it has and sends additional offer messages to the user for acceptance. Once the user accepts the offer of the new cluster, the missing service(s) is/are migrated to the new cluster.

B. Kubeadm Master Recovery

In Kubeadm, if the master node leaves its cluster, the cluster goes down [12]. The RSU monitors the behavior of all the underlying master nodes. If a potential leave for a master node is detected, the RSU calls Algorithm 3 to recover from cluster failures before occurring. The algorithm checks first if a secondary master node is running to replace the primary.

Authorized licensed use limited to: Bibliothèque ÉTS. Downloaded on June 25,2021 at 06:28:33 UTC from IEEE Xplore. Restrictions apply.

Algorithm 2 Kubeadm Fog Recovery Algorithm

1: **procedure:** Fog Recovery
2: The master Searches for another fog availability by
3: following these checks:
4: Run VCP to check if another fog node can host
5: the service
6: Check if the master node can host the service
7: Check if another cluster or a single vehicle can
8: temporally host the service by contacting the RSU or BS
9: if checks fail, **then:**
10: The RSU prepares a backup cluster: generate
11: new offer messages and ask for the user acceptance.
12: **end procedure**

Algorithm 3 Kubeadm Master Recovery Algorithm

Procedure: Master Recovery
1: The RSU searches for another master availability by
2: following these checks:
3: Check for a running secondary master
4: Run master election and Check for a fog ability
5: to transition for a master state
6: if checks fail, **then:**
The RSU prepares a backup cluster: generate new offer messages andask for the user acceptance.
7: **end procedure**

If no secondary is found, the RSU asks for the election algorithm to run locally and elect a new master node in case available resources are found. As a last resort, the RSU collects offers from other available Kubeadm clusters. Once an offer is accepted from the user side, the services are migrated to the new backup cluster, which can replace the original one in case of failures.

VI. Simulations and Experimental Results

In this section, we provide experiments of four scenarios illustrating the limitations of existing vehicular fog approaches and presenting how our proposed scheme outperforms these solutions. We also provoke a worst-case scenario where the vehicular cluster fails to maintain a connection. Therefore, the importance of adapting our recovery method is also explored.

A. Combined Testing Scenarios Showing Our Approach Advantages

1) Experiment Setup: In our experiments, we use the Mininet-Wifi simulator as a base environment. Mininet-Wifi offers the ability to run a full network on one machine using wireless technology for vehicular connections [31]. This is done using the notion of processes as hosts. Wireshark is used to track the processes interactions during the simulation. Because of the need to represent VANETs and simulate them in real life, we make use of the integration between Mininet-Wifi and SUMO simulator to build near real-life scenarios of moving vehicles. Sumo is a road traffic simulator programmed to control and display the movement of vehicles on any chosen map [32].

Cellular Networks and RSUs are represented in SUMO as well as the moving vehicles at predefined speeds and routes to follow. We simulate the vehicle's behavior using a container. Automated python scripts using Mininet-Wifi library are used to build experiments discussed in the following Section. A ping web service is built using the Flask framework [33]. In this service, the hosting device listens on a certain port waiting for the user's request. The fog then replies with a message. The aim of this service is to measure the service availability and networking delays between users and serving vehicles. The networking delay is a good measure to show the ability of our approach to hosting services on the fly on fogs. We build a VM that uses Ubuntu minimal image of size 520MB that runs our web service, in addition to a Docker image using Ubuntu minimal of size 30MB running the same service. The internet speed during the simulation is on average 50Mb/s.

The experiment conducted is a combination of four sub experiments aiming to show how our approach overcomes the existing fog limitations. The first experiment aims to show the improvement achieved by installing containerized micro-services vs. VMs. The second part presents the importance of hosting services on vehicles vs. RSUs by avoiding the handover delays caused by SDNs. The third fold proves that our approach overcomes the RSU availability issue. The last part of our experiment implies the need for a recovery algorithm to handle any fog technical failures or non-availability. Therefore, we set our testing environment as follows: Four vehicles, with 100m wireless coverage are installed on the road following the same path. Also, five RSUs are arranged in sequence on that road. All RSUs are aligned in a way that covers all parts of the road, except a gap of 300m between RSU3 and RSU4. All vehicles start moving at a speed of 10 m/s. V1, V2, V3, and V4 start moving at time 5s, 10s, 5s, and 0s, respectively, hence creating a distance of 50m between V2, V3, and V4. V1 is the user who starts requesting the web service after 5s of the simulation time. V3 is elected as the master node of the cluster V2, V3, and V4, where the Kubeadm cluster is initialized. At 5s, the RSU starts pulling the VM containing the flask service. At the same time, V4 starts pulling the same service from Docker Hub. When V1 joins the range of RSU2, a computation delay of 100ms is manually provoked on the SDN controller in order to simulate the RSU handover issue. After 195s, V4 speed is doubled to reach 20m/s. This behavior was provoked to simulate speed randomness. V4 leaves the cluster (range of the master) at 200s. In this part of the experiment, we do not use any recovery algorithm. The response time of each ping request sent to RSUs vs. V3 is recorded.

2) Experimental Results: The results in Fig. 2 are separated into four parts, as mentioned in the Experiment Setup. The x-axis in the graph corresponds to the simulation time in seconds, vs. the response time at different stages or time of the simulation on the y-axis.

During the first part of our experiment, a VM instance is being downloaded on the vehicle. It takes around 45s for the VM to get downloaded. During this time, all vehicle requests are served by the cloud. In conclusion, such time for a simple service cannot be tolerated by vehicular applications, especially when services are hosted on RSUs, where routes should be updated to reach the service or a new VM has to be pushed again. In contrast, the container image took 5s to download and boot. Therefore, more than 90% improvement

Authorized licensed use limited to: Bibliothèque ÉTS. Downloaded on June 25,2021 at 06:28:33 UTC from IEEE Xplore. Restrictions apply.

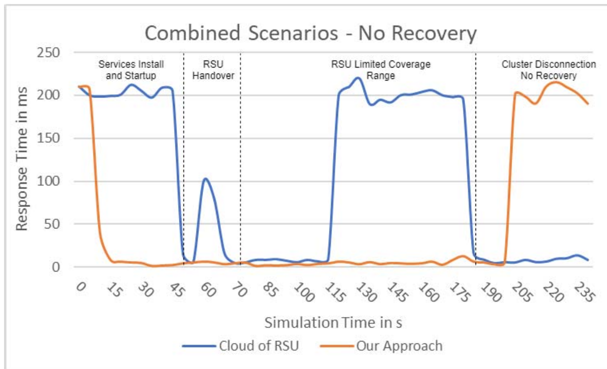

Fig. 2. Approach Advantage in Combined Scenarios Compared to [7], [15] - No Recovery.

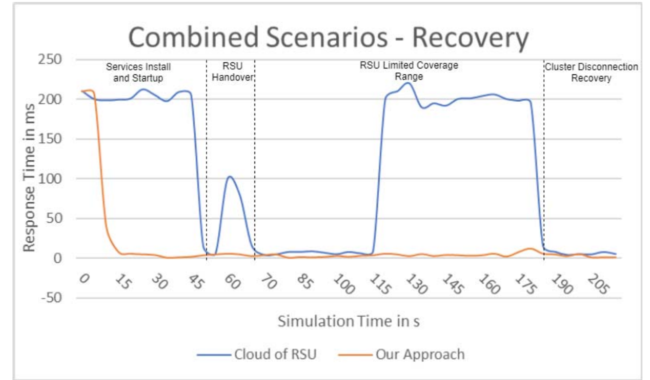

Fig. 3. Approach Advantage in Combined Scenarios - Recovery.

in image size and booting time is achieved when using containerization technology for pulling and running services. RSUs limitations such as handover and range of coverage, studied in parts two and three, are solved in our approach because of the ability to host services on vehicles after clustering them and electing the right orchestrator. Based on [7], the controller re-computes the routing table whenever the vehicle moves to a new RSU. Because of the networking and computational delays added to the SDN during our experiment, we can observe the jump of response time in the second part of Fig. 2, whereas our approach uses dynamic routes updates between vehicles hosting services through the orchestrator or the upper level in our architecture like the RSUs. Therefore, we are able to maintain low response time in case of change in the network topology. Thereafter, because services are being hosted on vehicles and orchestrator is reachable through the cellular technology or 802.11p, the service is made available throughout the cluster lifetime, as shown in the third part of Fig. 2. On the other hand, counting on static fogs (RSUs or anchor routers) to reach services [7], [15] is not feasible because of their limited range of coverage. This is illustrated in the third part of Fig. 2 representing the cloud of RSUs.

After 195s of running the simulation, V4 doubles its speed to 20m/s causing it to leave the cluster at around 200s (part 4). Because V4 is the only running fog in V3's cluster, the service requested by V1 becomes unavailable. In this case, thereafter requesting the service from the cloud. In [7]'s approach, the service is available on RSU1 and reachable through RSU5.

B. Recovery Algorithm

In the second part of this experiment, the recovery algorithm is installed to run on V3 to avoid any cluster connectivity or physical failures. We reproduce the above experiment after adding the fog recovery solution. The results are shown in Fig. 3 (Recovery part). V3 is now able to push the service to V2 before the current running fog (V4) leaves the cluster. Moreover, the master recovery algorithm can also be used when connection breaks occur with the orchestrator.

C. Comparative Study Over Existing Approaches

In the combined experiments, our approach is compared to the RSU cloud approach [7] and FMC [15]; however, this comparison applies to all approaches trying to serve vehicular applications using RSUs and VMs. In table III, we summarize all the advantages of our approach compared to existing ones such as the cloud of RSUs. Adapting the containerization technique allowed us to download and update services faster.

TABLE III

PERFORMANCE COMPARISON BETWEEN OUR APPROACH VS CLOUD OF RSUs [7] AND FMC [15]

Compared Based On	Our Approach	RSU Cloud and FMC
Time to download service	low	high
Time to update service	low	high
SDN Delays	none	Network/ Computation
Service Availability	always	within RSU/MDC range
Push service again	use same base OS	download VM again

Our approach does not consider the use of SDNs because of the timely low-cost reporting between master nodes and RSUs. In addition, the use of our proposed hybrid networking architecture makes it possible for users and vehicular fogs to keep connected. The recovery algorithm plays a vital role here by avoiding any potential clusters failures.

VII. VCP EXPERIMENTS

In the previous experiments, the best selection of volunteers and optimal distribution of services on vehicles are taken by default. In the following experiments, we build test cases with three scenarios to show the importance of our objective functions and their effect in taking the selection and placement decision of services. The first experiment illustrates the importance of keeping micro-services connected because loosing one micro-service before the other leads to the non-availability of the main service. A decision can be taken with the best distribution of connected micro-services; however, services can be redistributed in a way that maximizes the number of pushes while maintaining the same fitness value of the micro-services connectivity or tolerating a small fitness loss. In the second experiment, we show the ability of the VCP to maximize the number of pushed services. While maintaining these two objectives, the service will not be available for a long time if the cluster is about to reach its time availability limit. Therefore, we study the importance of maximizing cluster lifetime in the third experiment.

In all of the mentioned experiments, we ran the MA ten times while considering individuals of size 100. We performed several experiments in order to assign stopping criteria based on a predefined number of iterations, which resulted in adopting the 1000 iterations. We then take the best solution out of the 10 runs as the output of our Memetic solution. We discuss the MA output and compare the results to a search algorithm to place the services. This is done to show the ability of our VCP to maintain an equilibrium between all of the objective

Authorized licensed use limited to: Bibliothèque ÉTS. Downloaded on June 25,2021 at 06:28:33 UTC from IEEE Xplore. Restrictions apply.

TABLE IV
VEHICLES DATASET COMBINING MOBISIM AND GOOGLE CLUSTER TRACE 2011-2

	Starting Position	Dest Time	AvgS	CPU	Memory	Disk	Dep Time	Orch
V1	(40.740, -73.994)	20	7.5	0.5	0.55	0.6	0	V1
V2	(40.740, -73.994)	5	7.35	0.6	0.6	0.4	1	V1
V3	(40.740, -73.994)	30	25.0	1	1	1	2	V1
V4	(40.743, -73.996)	40	7	1	1	1	1	V4
V5	(40.743, -73.996)	35	7.1	0.7	0.7	0.7	1	V4
V6	(40.743, -73.996)	30	6	0.5	0.5	0.8	0	V6
V7	(40.165, -73.736)	4	3	0.8	0.7	0.4	3	V7
V8	(40.365, -73.756)	100	20	0.2	0.3	0.1	4	V8

TABLE V
SERVICES DATASET

Service	ID	CPU	MEM	DISK
S1	1	0.4	0.3	0.5
S2	1	0.1	0.1	0.05
S3	1	0.2	0.2	0.25
S4	1	0.5	0.4	0.4
S5	2	0.2	0.1	0.25
S6	2	0.3	0.4	0.25
S7	3	0.6	0.5	0.58

TABLE VI
SCENARIOS

Scenario	Available Vehicles(s)	Requested Service(s)
Scenario1	V1, V2, V3, V4, V7, V8	S1, S2, S3, S4
Scenario2	V4, V6, V7, V8	S5, S6, S7
Scenario3	V2, V4. V7, V8	S7

functions in one decision. Furthermore, we consider equal weights for each objective during the experiments.

To develop our scenarios, we use two well-known datasets, the Google trace 2011 dataset [34], and another one generated by Mobisim tool [35]. Google cluster trace dataset contains data about micro-services' resources requirements in terms of CPU, memory, and disk, as well as data about the available cluster's resources. In our case, each node in this cluster is a vehicle. In their data, all nodes have identical specs where the given values are normalized. The mobility conditions of our scenarios are selected from a generated Mobisim dataset. In Mobisim, we can generate a behavior of real vehicles with random directions and speeds. From this data, we selected the average speed and distance crossed by a group of vehicles. The services and vehicles datasets are shown in tables IV and V. Table VI shows the combination of available vehicles and services used to build our test cases. The testing environment is created using Mininet-Wifi and SUMO simulators. Because vehicles are changing their speed in Mobisim, small fluctuations in the response time are observed in experiments results.

The starting location of the user is (40.740, -73.994), who departs at time 4 after the simulation starts. The user mobility is described in each of the below experiments setup.

A. Experiment 1: Importance of Keeping Micro-Services Connected

All micro-services must reach each other to keep the service available. In this experiment, we show the ability of our VCP algorithm to keep the microservices connected for the maximum time possible. We compare the performance of the VCP algorithm to a search algorithm that looks into the available vehicles and finds the first vehicle close to the user and capable of hosting the service.

1) Experiment Setup: Scenario 1 is used in this experiment, where push S1, S2, S3, and S4 are to be placed on vehicles V1, V2, V3, and V4. The vehicle has a network coverage range of 50m, the user is moving with V1 at almost the same average speed during the first 2s and then connects to it through the RSU. We implemented the micro-services in a way that each one pings the other whenever the user sends a request to V1 (selected orchestrator). If all services can reach each other, the user receives its response. The response time is recorded in the graph of Fig. 4. Whenever a micro-service is not reachable, we represent the response time to be -1 in the graph. This is where the user does not receive any response from V1. We pass scenario one as input to our VCP algorithm, and we compare its results to the ones generated by the simple search algorithm. 5ms of networking delay is added on the link between the user and each of V4 and V5. Concurrent requests are sent to the serving vehicles neglecting the time of initializing the cluster and downloading the service. The x-axis in Fig. 4 represents the request number, whereas the y-axis represents the response time of the request sent by the user to test the service availability.

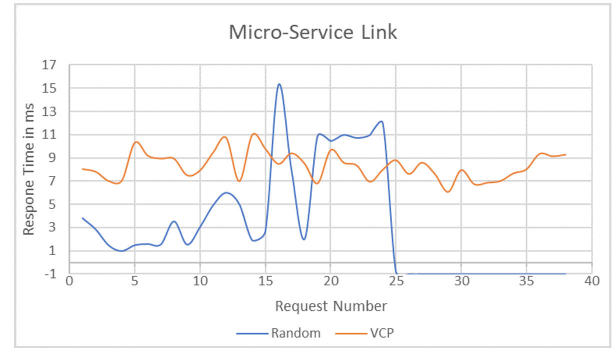

Fig. 4. Maintain Micro-Services Connection Using VCP.

2) Experimental Results: The VCP assigns S1, S2 to V5 and S3, S4 to V4. On the other hand, the search algorithm output for scenario 1 is to assign S1, S2 to V1 and S3, S4 to V3. In the simple search case, the user is being served until sending the 25th request where he stops receiving a response. At this time, the response drops to -1 and the service becomes unavailable. This is because V3 is moving at a faster average speed and leaves V1 range after around 3s from the simulation starting time. A jump in the response time for the search algorithm is shown during the 17th request. This is because the user connects to V1 through RSU after a certain time rather than a direct connection (the speed of the user is different than the cluster's speed). VCP tries to push micro-services to vehicles in a way that maximizes their time connectivity to their orchestrators. Following the VCP decision, the service is made available for the user all the time. We can notice that VCP is causing more delays when the service is available because V4 is far from the user and connected through an RSU from the beginning.

B. Experiment 2: Importance of Maximizing the Number of Pushed Services

We show in this experiment the ability of our VCP to push the maximum number of services on available cars to enhance the QoS of all requesting users.

1) Experiment Setup: In this experiment, we use scenario two as a testing environment and let the user move next to V1. To show the importance of this objective function, we compare the results of our VCP to the search algorithm. Using Mininet-Wifi integration with SUMO, we can simulate the

Authorized licensed use limited to: Bibliothèque ÉTS. Downloaded on June 25,2021 at 06:28:33 UTC from IEEE Xplore. Restrictions apply.

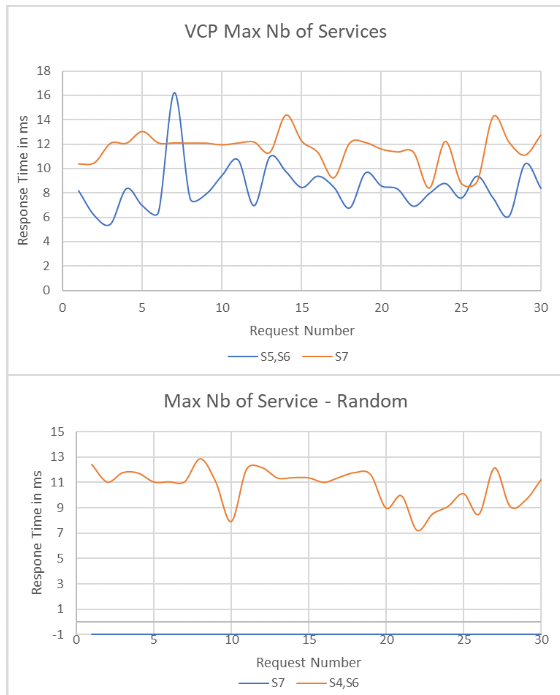

Fig. 5. Maximize Number of Pushed Services using VCP.

behavior of both criteria. A networking delay of 5ms is added to the link between the user and all vehicles. A comparison of the response time for each criteria is shown in the two graphs of Fig. 5, where each HTTP request sent to services hosted on selected vehicles is recorded.

2) Experimental Results: The VCP output shows that micro-services S5 and S6 together should be placed on V6 and S7 on V4; however, the search algorithm output for scenario 2 is to assign S5 and S6 to V4 and S7 to V6. Following the VCP decision, we can notice that both services (S5/S6 and S7) are available to the user all the time. The services response time is somewhat high because the user is connected to the fogs through RSUs. For the search algorithm output, S5 and S6 services are available, but S7 is not. This is because when S5 and S6 are pushed to V4, the remaining resources on V4 and V6 are not enough to meet the requirements of S7. Therefore, S7 cannot be hosted anymore on V6 because S7 resources requirements are more than V6 capacity. Hosting S5/S6 on V6 allows us to utilize almost its full available resources, and allows S7 to be hosted on V4. V7 and V8 are not considered in the search algorithm or VCP because they are further away from the user than V4 and V6. Based on the experiment's results, we explored that our Memetic solution is capable of maximizing the number of pushed services.

C. Experiment 3: Importance of Increasing the Cluster Lifetime

Considering the VCP decision without the objective of increasing the cluster lifetime, services can be placed on nearby vehicles having enough resources, which results in the least delay possible. However, if services are being pushed to clusters that do not have a stable connection, or where the orchestrator/worker nodes will soon reach their destinations and stop serving, this leads to shorter serving time. In this case, the decision to select serving vehicles is greatly affected by their serving time. Therefore, aim is to maximize the cluster

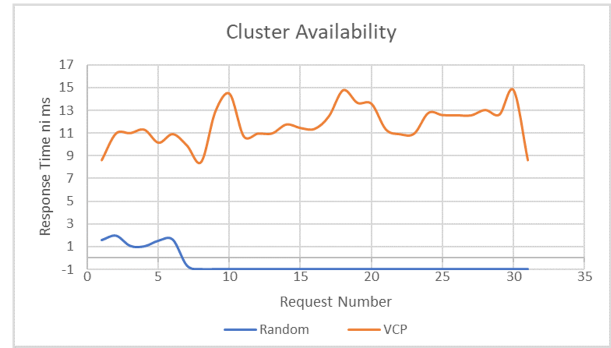

Fig. 6. Maximize Cluster Availability using VCP.

serving time. Through this experiment, we show the ability of VCP to achieve this task.

1) Experiment Setup: We use Scenario 3 for this experiment. Services are ready and running on the fogs before the vehicles start moving. We also let V2 leave the cluster after 54s from its starting time. The user drives next to V2 at the same speed, and following the same direction. A delay of 5ms is added between the user and V4. We assume that S7 is already running on both vehicles. The response time of user's requests sent to S7 either hosted by V2 or V4 is shown in Fig. 6.

2) Experimental Results: Based on the VCP output, the VCP decides to push S7 to V4. Differently, the search algorithm for scenario 3 assigns S7 to V2. As shown in the results of Fig. 6 and after the 7th request, S7 is no longer available on V2 because it reached its destination and stopped serving. We can see that S7 is available on V4 throughout the simulation time because V4 can serve for 40s with enough resources. Therefore, VCP's decision shows the ability to maximize the cluster availability. Depending on the available resources, user's requirements, and services importance, VCP tries to provide an effective selection and distribution of services based on multiple contradicting objectives.

VIII. Conclusion

In this paper, we explored that the use of RSUs to support real-time vehicular applications is not convenient in real-life scenarios. Moreover, On-Board Units are still constrained by the availability of their resources. Therefore, one or two OBUs are not enough to handle the processing of the vast amount of data generated by vehicles. To the best of our knowledge, there is no work in the literature capable of hosting services on the vehicle. In case vehicles are used as fog devices, they should keep an alive connection with the user to provide longtime support. Another problem arises in this context, which is finding the best fit of micro-services on available vehicles in a way that maximizes the vehicles serving time and maintains its reachability. In this context, we proposed the Vehicular-OBUs-As-On-Demand-Fogs solution capable of providing an on-demand fog and service placement on vehicles, considering a hybrid network architecture to maintain the connection between the requesting vehicle and vehicular fog cluster. Our solution embeds an adapted master election, recovery algorithm, and evolutionary MA for efficient service placement on vehicular fogs. Our experiments showed more than 90% improvement in the response time in cases such as SDN delay, RSU limited coverage, and RSU handover. In addition, the importance of our VCP solution is observed compared to a search algorithm for placing containers on vehicular fog clusters. As future work, the recovery algorithm should be

Authorized licensed use limited to: Bibliothèque ÉTS. Downloaded on June 25,2021 at 06:28:33 UTC from IEEE Xplore. Restrictions apply.

improved by introducing a reinforcement learning approach to identify the proper time to restore the cluster state. We can also benefit from our approach by introducing a security model for securing vehicular networks. This is a susceptible area because any attacks on the network can lead to data and decisions changes. On the other hand, upcoming cellular network technologies like 5G are very promising towards the support of real-time vehicular applications. Thus, our architecture and methodology can be adapted and enhanced based on the advancement of networking technologies.

REFERENCES

[1] K. P. Saharan and A. Kumar, "Fog in comparison to cloud: A survey," *Int. J. Comput. Appl.*, vol. 122, no. 3, pp. 10–12, Jul. 2015.

[2] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for Internet of Things and analytics," in *Big Data and Internet of Things: A Roadmap for Smart Environments*. Cham, Switzerland: Springer, 2014, pp. 169–186.

[3] O. Salman, I. Elhajj, A. Kayssi, and A. Chehab, "Edge computing enabling the Internet of Things," in *Proc. IEEE 2nd World Forum Internet Things (WF-IoT)*, Dec. 2015, pp. 603–608.

[4] I. Stojmenovic, "Fog computing: A cloud to the ground support for smart things and machine-to-machine networks," in *Proc. Australas. Telecommun. Netw. Appl. Conf. (ATNAC)*, Nov. 2014, pp. 117–122.

[5] J. Li, J. Jin, D. Yuan, and H. Zhang, "Virtual fog: A virtualization enabled fog computing framework for Internet of Things," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 121–131, Feb. 2018.

[6] N. B. Truong, G. M. Lee, and Y. Ghamri-Doudane, "Software defined networking-based vehicular adhoc network with fog computing," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, May 2015, pp. 1202–1207.

[7] M. A. Salahuddin, A. Al-Fuqaha, and M. Guizani, "Software-defined networking for RSU clouds in support of the Internet of Vehicles," *IEEE Internet Things J.*, vol. 2, no. 2, pp. 133–144, Apr. 2015.

[8] I. Sorkhoh, D. Ebrahimi, R. Atallah, and C. Assi, "Workload scheduling in vehicular networks with edge cloud capabilities," *IEEE Trans. Veh. Technol.*, vol. 68, no. 9, pp. 8472–8486, Sep. 2019.

[9] T. Salah, M. J. Zemerly, C. Y. Yeun, M. Al-Qutayri, and Y. Al-Hammadi, "Performance comparison between container-based and VM-based services," in *Proc. 20th Conf. Innov. Clouds, Internet Netw. (ICIN)*, Mar. 2017, pp. 185–190.

[10] S. Ucar, S. C. Ergen, and O. Ozkasap, "Multihop-cluster-based IEEE 802.11p and LTE hybrid architecture for VANET safety message dissemination," *IEEE Trans. Veh. Technol.*, vol. 65, no. 4, pp. 2621–2636, Apr. 2016.

[11] D. Namiot and M. Sneps-Sneppe, "On micro-services architecture," *Int. J. Open Inf. Technol.*, vol. 2, no. 9, pp. 24–27, 2014.

[12] G. Sayfan, *Mastering Kubernetes*. Birmingham, U.K.: Packt, 2017.

[13] H. Tout, C. Talhi, N. Kara, and A. Mourad, "Selective mobile cloud offloading to augment multi-persona performance and viability," *IEEE Trans. Cloud Comput.*, vol. 7, no. 2, pp. 314–328, Apr. 2019.

[14] H. Tout, C. Talhi, N. Kara, and A. Mourad, "Smart mobile computation offloading: Centralized selective and multi-objective approach," *Expert Syst. Appl.*, vol. 80, pp. 1–13, Sep. 2017.

[15] T. Taleb, A. Ksentini, and P. A. Frangoudis, "Follow-me cloud: When cloud services follow mobile users," *IEEE Trans. Cloud Comput.*, vol. 7, no. 2, pp. 369–382, Apr. 2019.

[16] M. Sookhak *et al.*, "Fog vehicular computing: Augmentation of fog computing using vehicular cloud computing," *IEEE Veh. Technol. Mag.*, vol. 12, no. 3, pp. 55–64, Sep. 2017.

[17] X. Hou, Y. Li, M. Chen, D. Wu, D. Jin, and S. Chen, "Vehicular fog computing: A viewpoint of vehicles as the infrastructures," *IEEE Trans. Veh. Technol.*, vol. 65, no. 6, pp. 3860–3873, Jun. 2016.

[18] K. Mershad and H. Artail, "Finding a STAR in a vehicular cloud," *IEEE Intell. Transp. Syst. Mag.*, vol. 5, no. 2, pp. 55–68, Apr. 2013.

[19] T. Dbouk, A. Mourad, H. Otrok, H. Tout, and C. Talhi, "A novel ad-hoc mobile edge cloud offering security services through intelligent resource-aware offloading," *IEEE Trans. Netw. Service Manage.*, vol. 16, no. 4, pp. 1665–1680, Dec. 2019.

[20] P. Bellavista and A. Zanni, "Feasibility of fog computing deployment based on docker containerization over RaspberryPi," in *Proc. 18th Int. Conf. Distrib. Comput. Netw. (ICDCN)*, 2017, p. 16.

[21] H.-J. Hong, P.-H. Tsai, and C.-H. Hsu, "Dynamic module deployment in a fog computing platform," in *Proc. 18th Asia–Pacific Netw. Oper. Manage. Symp. (APNOMS)*, Oct. 2016, pp. 1–6.

[22] R. Morabito, I. Farris, A. Iera, and T. Taleb, "Evaluating performance of containerized IoT services for clustered devices at the network edge," *IEEE Internet Things J.*, vol. 4, no. 4, pp. 1019–1030, Aug. 2017.

[23] H. Sami and A. Mourad, "Towards dynamic on-demand fog computing formation based on containerization technology," in *Proc. Int. Conf. Comput. Sci. Comput. Intell. (CSCI)*, Dec. 2018, pp. 960–965.

[24] H. Sami and A. Mourad, "Dynamic on-demand fog formation offering on-the-fly IoT service deployment," *IEEE Trans. Netw. Service Manage.*, early access, Jan. 1, 2020, doi: 10.1109/TNSM.2019.2963643.

[25] F. López-Pires and B. Barán, "Many-objective virtual machine placement," *J. Grid Comput.*, vol. 15, no. 2, pp. 161–176, May 2017.

[26] O. A. Wahab, H. Otrok, and A. Mourad, "VANET QoS-OLSR: QoS-based clustering protocol for vehicular ad hoc networks," *Comput. Commun.*, vol. 36, no. 13, pp. 1422–1435, Jul. 2013.

[27] E. Falkenauer, "A hybrid grouping genetic algorithm for bin packing," *J. Heuristics*, vol. 2, no. 1, pp. 5–30, 1996.

[28] P. Moscato, C. Cotta, and A. Mendes, "Memetic algorithms," in *New Optimization Techniques in Engineering*. Berlin, Germany: Springer, 2004, pp. 53–85.

[29] Q. H. Nguyen, Y.-S. Ong, and M. H. Lim, "A probabilistic memetic framework," *IEEE Trans. Evol. Comput.*, vol. 13, no. 3, pp. 604–623, Jun. 2009.

[30] R. M. Ramadan, S. M. Gasser, M. S. El-Mahallawy, K. Hammad, and A. M. El Bakly, "A memetic optimization algorithm for multi-constrained multicast routing in ad hoc networks," *PLoS ONE*, vol. 13, no. 3, Mar. 2018, Art. no. e0193142.

[31] R. R. Fontes, S. Afzal, S. H. B. Brito, M. A. S. Santos, and C. E. Rothenberg, "Mininet-WiFi: Emulating software-defined wireless networks," in *Proc. 11th Int. Conf. Netw. Service Manage. (CNSM)*, Nov. 2015, pp. 384–389.

[32] D. Krajzewicz, G. Hertkorn, C. Rössel, and P. Wagner, "Sumo (simulation of urban mobility)-an open-source traffic simulation," in *Proc. 4th middle East Symp. Simul. Model. (MESM)*, 2002, pp. 183–187.

[33] M. Grinberg, *Flask Web Development: Developing Web Applications With Python*. Newton, MA, USA: O'Reilly, 2018.

[34] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: Format + schema," Google, Mountain View, CA, USA, Tech. Rep., version 2.1, Nov. 2011. [Online]. Available: https://github.com/google/cluster-data

[35] J. Härri, F. Filali, C. Bonnet, and M. Fiore, "VanetMobiSim: Generating realistic mobility patterns for VANETs," in *Proc. 3rd Int. workshop Veh. Ad Hoc Netw.*, 2006, pp. 96–97.

Hani Sami received the B.S. degree from Lebanese American University and the M.Sc. degree in computer science from the American University of Beirut. He worked as a Research Assistant at Lebanese American University. The topics of his research are fog computing, vehicular fog computing, smart vehicles, and reinforcement learning.

Azzam Mourad (Senior Member, IEEE) is currently an Associate Professor of computer science with Lebanese American University and also an Affiliate Associate Professor with the Software Engineering and IT Department, École de Technologie Supérieure (ÉTS), Montreal, Canada. He has served/serves as a TPC member, an Associate Editor for the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, the IEEE NETWORK, the IEEE OPEN JOURNAL OF THE COMMUNICATIONS SOCIETY, *IET Quantum Communication*, and the IEEE COMMUNICATIONS LETTERS, the General Chair of IWCMC2020, the General Co-Chair of WiMob2016, and the track chair and a reviewer of several prestigious journals and conferences.

Wassim El-Hajj (Senior Member, IEEE) is currently an Associate Professor of computer science with the American University of Beirut. He researches wireless communications, security, and machine learning. In networks, he works on designing scalable and energy efficient routing protocols. In security, he works on the development of effective techniques to stop attackers from gaining access to PCs via the network. In machine learning, he designs scalable algorithms and develops accurate sentiment analysis and emotion recognition models.

Authorized licensed use limited to: Bibliothèque ÉTS. Downloaded on June 25,2021 at 06:28:33 UTC from IEEE Xplore. Restrictions apply.