Vehicular Communications 40 (2023) 100571

Contents lists available at ScienceDirect

Vehicular Communications

journal homepage: www.elsevier.com/locate/vehcom

Reinforcement learning based scheme for on-demand vehicular fog formation

Ahmad Nsouli [a], Wassim El-Hajj [a], Azzam Mourad [b,c,*]

[a] Department of Computer Science, American University of Beirut, Lebanon
[b] Cyber Security Systems and Applied AI Research Center, Department of CSM, Lebanese American University, Lebanon
[c] Division of Science, New York University, Abu Dhabi, United Arab Emirates

A R T I C L E I N F O

Article history:
Received 23 October 2022
Received in revised form 4 December 2022
Accepted 19 January 2023
Available online 26 January 2023

Keywords:
Reinforcement learning
Vehicular fog computing
Vehicular clustering
Kubeadm
Hungarian algorithm
Istio service mesh

A B S T R A C T

The high need of real-time vehicular applications for self-driving cars to maintain service availability and reachability, and to process huge amount of generated data within a small amount of time, rise the need to improve the vehicular network infrastructure. Fog computing has been introduced to reduce the amount of data sent to cloud by bringing processing power near the edge and reducing latency. In this paper, we overcome the aforementioned limitations by taking advantage of Istio service mesh and kubeadm clustering to handle inter-service communication and provide clusters to serve nearby vehicles, the evolvement of On-Board Units in vehicle supplies enough processing power to run reinforcement learning model in taking action of pushing microservices to cluster, and the efficiency of Hungarian Algorithm to assign microservices to vehicles. We propose in our scheme (1) a service mesh architecture that manages communication between multiple microservices across different clusters and tackle inter-service communication, (2) a Reinforcement Learning model deployed on road side units to predict on-demand placement of microservices on cluster or caching at RSU, and (3) Reformulating the vehicular container placement (VCP) problem as an assignment problem to map microservices to vehicles. Experiments and simulations show that the reinforcement learning model served 90% of requests when deploying microservices which is more efficient than current strategies in the literature. Given that only needed microservices are deployed in limited resource cluster, mesh network shows an improvement of 84% in the deployment time and inter-service communication across clusters, and the usage of Hungarian algorithm in solving the VCP improves Quality of Service (QoS).

1. Introduction

The growing requirement to generate data and perform computations across a network, as well as the rapid development of the Internet of Things, have given rise to the concept of Internet-of-Vehicles (IoV). IoV is a dispersed network of vehicles with sensors, software, and technologies that use the data provided by vehicles and their computing capacity to improve vehicle communication and interaction. For example vehicles can exchange information about road conditions in order to avoid traffic jams. Due to their expansion, several commercial applications (e.g., multi-media, infotainment) have also been integrated into such networks for marketing and business purposes [15]. It provides intelligent Vehicle-to-Everything (V2X) communication, allowing for heterogeneous network connectivity [4]. Moreover, the high demand of time critical applications require fog computing to bring cloud processing power near the edge by hosting services next to users. Instability within fog federations is considered as a serious problem that degrades the performance of the provided services [36]. With the increasing number of IoT devices, fog computing has emerged, providing processing resources at the edge for the tremendous amount of sensed data and IoT computation [5]. For example the challenge of object detection for autonomous vehicles in snowy weather environments is the high communication cost and delay needed to transfer the data [45]. Therefore, fog devices play a major role in improving the Quality of Service by decreasing latency and increasing processing power. Road side units offer secure and high-speed direct connection between vehicles and the surrounding infrastructure because they employ Dedicated Short Range Communication (DSRC), which is an 802.11p-based wireless communication technology. Vehicular Fog computing (VFC) can benefit from the processing power of the moving and parked

* Corresponding author.
E-mail address: azzam.mourad@lau.edu.lb (A. Mourad).

https://doi.org/10.1016/j.vehcom.2023.100571
2214-2096/

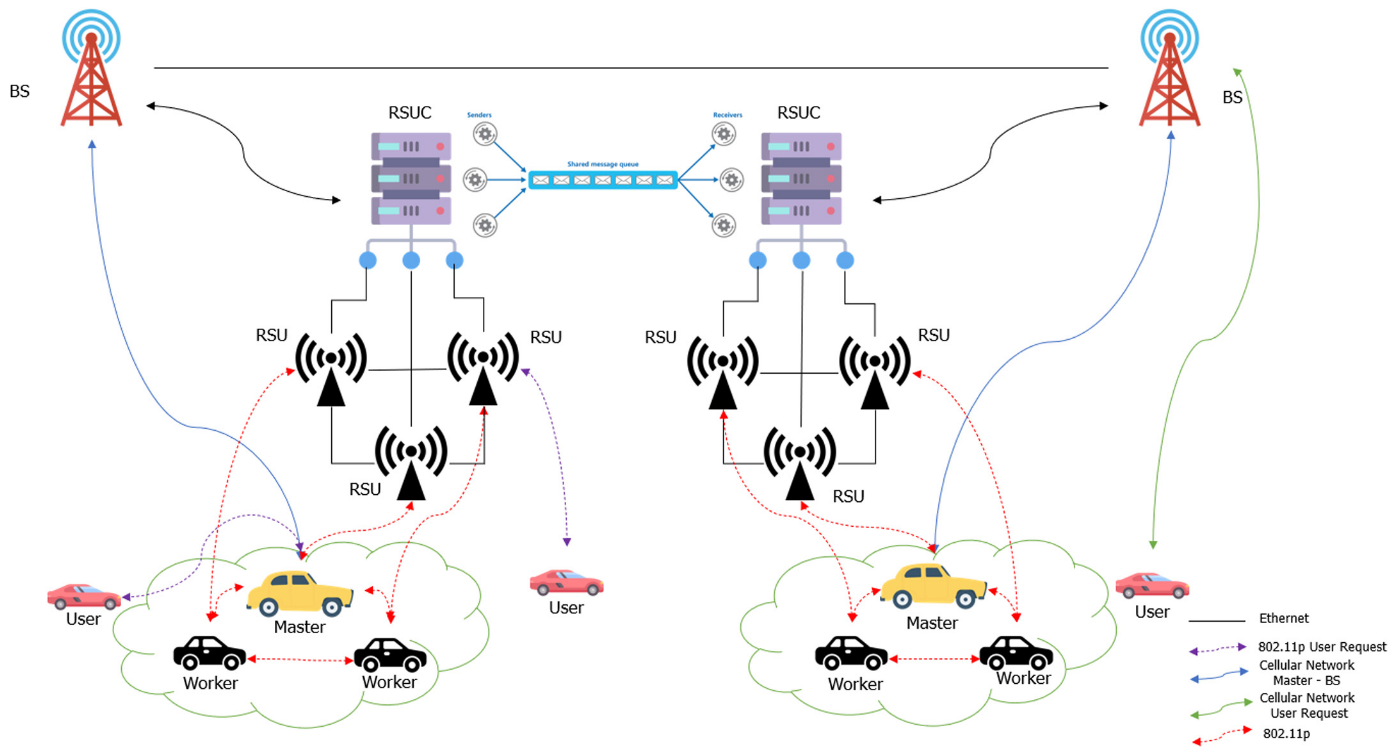

Fig. 1. Proposed architecture.

vehicles through the on-boarding units (OBUs) and any other device nearby [14].

In this context, many researchers have looked into the advantages of adding a fog layer to the edge to help users. For example, [19] [20] and [21] assisted vehicular users by using pre-defined fog spots such as road side units (RSU). Similarly, [16] [17] and [11] used static locations of fogs with certain services functioning on them to target a specific group of IoT devices. Virtual Machines (VM) are used in various methods to move and host fog services based on user requests. Authors also offer, a paradigm for outsourcing vehicular computations to RSUs [22]. Despite the benefits that RSUs provided for supporting vehicles, there are still some significant restrictions that must be addressed. Geographically fixed fogs (RSUs), for example, have restricted network coverage and thus availability to serve vehicles. Furthermore, using a VM to host services on fogs takes a long time to deploy and consumes a lot of resources. When upgrades are needed, these methods have low Quality of Service (QoS) and restricted service flexibility. Efficient resource utilization on fog nodes are quite challenging [9]. In addition, when RSUs are overburdened with requests, their performance can suffer, causing response time to suffer.

Containers are more lightweight than VMs. VMs run every VM on separate copy of operating system that impose a large performance penalty, they are time-consuming to download and start, and they require large physical disk space allocated for every VM. However, single operating system can support multiple containers reducing the processing overhead, containers require less time to download and few seconds to start, and they are stateless so storage space is created or occupied when the container is created and discarded when it's deleted.

In order to serve demanding applications and benefit from VFC, we shed the light on the concept of deploying micro-services on nearby vehicles. The main problem lies in the dynamic movement of vehicles which make the network not easily manageable. In addition, pushing on-demand fogs to volunteer devices all over the place has its drawbacks. Having to push services everywhere can put the service provider's security at danger, especially if the services are controlled by volunteers [23]. All of these considerations need a well managed architecture to optimize resources required. These are the primary motives for restricting the number of fog placements by the cloud. This creates a new difficulty for the cloud, which is deciding when to deploy fog in order to optimize profit by maximizing fog resource utilization and reducing the number of pushed fogs. In [2], for deploying containerized microservices on on-demand fogs, the authors developed a resource and context aware technique where RSU is aware of the master location to recover kubeadm fog, and distribute microservices based on the resources available using a memetic algorithm. To the best of our knowledge, no work has targeted the usage of reinforcement learning for on-demand vehicular fog formation to deploy microservices in service mesh architecture. In this work, we adopt a similar architecture as the one proposed in [2] (Fig. 1), and we enhance it to improve user experience using reinforcement learning to decrease latency, response time and address the inter-service communication aspect.

We first addressed the aforementioned problems by introducing in [1] a caching system for the purpose of achieving scalable micro-service applications which will be located at the RSU level. We also introduced a message bus that is responsible for the communication between the different micro-services (inter service communication). The message Bus is able to determine whether a nearby cluster can serve the user. When deploying a new service, it is a lot more efficient to get the service from a nearby cluster as opposed to getting it from the cloud. In order to minimize the delay in deploying the requested services, we propose an approach based on reinforcement learning where the master node predicts beforehand the services each cluster might need, as opposed to getting the service when it is requested. We extend our approach in this paper to determine the appropriate assignment of microservices on available vehicles in the cluster in order to optimize serving time while maintaining reachability. Therefore, we attempt to formulate the VCP as an assignment problem that can be solved efficiently using the Hungarian Algorithm. The authors in [3] proposed a Fog Scheduling based on the notion of average reward, a decision model, based on reinforcement R-learning, studies the behavior of service requesters and generates a suitable fog placement schedule. Their suggested R-learning model has been proved experimentally to achieve promising result on the Fog nodes. However, our main challenge is to use the model on a dynamic vehicular fogs. In order to solve the VCP problem, [2] defined the problem as multi-objective optimization problem that should be solved in a short amount of time, where they used the Memetic genetic Algorithm. However, their solution requires high computation and long time while the memetic algorithm is based on random sampling and does not find the optimal solution. Therefore, in case the requested service was not available on the nearby vehicles, we reformulate the VCP problem into an assignment problem which can be resolved in a reasonable time using the Hungarian Algorithm. The paper's primary contributions are summarized as follows:

1) Offering efficient orchestration between the different clusters in the vehicular mesh network to serve the user requests and tackle inter-service communication.
2) Formulating the container placement problem as an assignment problem and solving it using the Hungarian algorithm.
3) Using Reinforcement learning to predict the on-demand placement of the containerized micro-services within a cluster, and caching container files at the RSU level to decrease latency.

The rest of this paper is organized as follows. In section 2, we go over the related work and discuss their limitations. In section 3, we discuss our proposed architecture and methodology. In section 4, we expand on our methodology by defining and providing a solution for the VCP problem. The testing of our approach is presented in Sections 5. Finally, in section 6, we discuss future directions.

2. Related work

The future of smart city depends on the IoT devices and the performance of the network. The architecture plays a major role in future applications such as auto piloting where a huge amount of processing, in short time, is required. Applications designed on micro-services architecture can range from IoT services support, passenger's safety, auto-piloting, real time map updates to infotainment and entertainment services. Therefore, researchers got motivated to introduce an enhanced network where the DSRC integrated into the RSU play the major role in the network performance. In this section, we review the contributions of several related literary works and discuss their limitations in this section.

The authors in [6] formed a vehicular cloud on-demand counting on RSUs present on the roads. The main concept in this work relies on finding a star that offer computation power and the storage required when moving on the road. The RSU receives the information and share them to allow users to host their requested services. The star component here are static and they ask for money in return. In case the star move out of the RSU range, the user loses the service connection. The architecture focuses on finding the best match of a star node to the user requirement taking into consideration available resources, distance, and time to serve inside this VANET. This work lacks the concept of preparing and distributing the services before a user issues a request which could delay service specially for inter-service communication.

The work done in [12] presented a vehicular fog infrastructure showing an improvement in the communication and computational power. Making use of vehicles cloudlet helped in offering high computational capacity. However, its main focus was on urban areas and within limited support, unlike taking into consideration OBUs and personal devices to build a continuous support to nearby users. Furthermore, the authors in [14] worked on benefiting from the parked vehicles in a shopping mall through VFC to augment the resources. Unfortunately, this work was limited to the use of parked vehicles computation power only, and the distribution of services and how to assign services to new vehicles was not well studied.

To the best of our knowledge, no existing research addresses the use of reinforcement learning technology that determines on-demand micro-services placement hosted on vehicles. Many others, on the other hand, utilized a model that organizes activities and determines where fog should be pushed in the form of containers. In recent surveys, authors have emphasized the need for a model that schedules activities and services in the form of containers on available fogs' resources [27][28][29], highlighting its critical impact on any future studies in this field. This schedule is split into two halves. The first is to make an informed choice about when and where the service should be pushed, which we will focus on in our work. Second, there's the method for distributing services over many heterogeneous resources. Various techniques in the literature have addressed tackling the problem of task placement on resources [30][31][32], where heuristics have been shown to make correct selection and placement of containers.

Using containerization technology, the authors of [37] were able to demonstrate the ability to push services to pre-selected fogs. The operating services are managed via an orchestration layer. The work contribution was limited to presenting a method for placing services on fogs, without considering the possibility that the number of services or tasks to be completed will require additional resources, and that not all of them can be planned for placement. Moreover, it disregards volunteering devices and service priority.

Authors in [13] suggest a cooperative vehicular architecture, with network function virtualization (NFV) enabled RSU inside the mobile edge computing unit. A machine learning algorithm is deployed on the edge unit to predict the traffic flow and assist the RSU in improving its coverage range. However, the packet delivery ratio (PDR) did not show a reliable network infrastructure because it falls below 65% when the number of vehicles reached 150. Additionally, they don't manage the resources that are available in the NFV effectively or decide which services should be deployed there.

Using a hybrid network architecture, in [2] the authors proposed the Vehicular OBUs-As-On-Demand-Fogs to provide on-demand fog and service placement on vehicles to maintain the connection between the requesting vehicle, and create vehicular fog cluster. His layered architecture aids in the escalation of a fog-to-user connection failure. The use of kubeadm cluster and electing a master facilitate the management of the cluster and the deployment of micro-services. However, the proposed architecture lacks a caching system that can reduce the network congestion and enhance the architecture performance. Deploying the micro-service when the user requests a specific service makes it more costly than having them deployed and ready for use. Moreover, the proposed VCP solution used the Memetic algorithm to solve the optimization problem which makes the solution not guaranteed. Note that inter-service communication have not been taken into consideration which leads to more latency due to the deployment of multiple micro-services consecutively.

In [3], Reinforcement learning technology is used to present an intelligent model capable of fog placement scheduling. It operates by analyzing service requester behavior and generating a fog placement plan based on the average reward. It also shows that the model may improve over time by responding to changing user demands. Adding to this the concept of creating fogs on demand to create kubeadm cluster containing volunteering nodes will provide storage and processing resources to host services in the location predicted by the model. The experiments showed promising results where the requests processed by the cloud decreased from 100% to 30%. However, the model mentioned does not study what

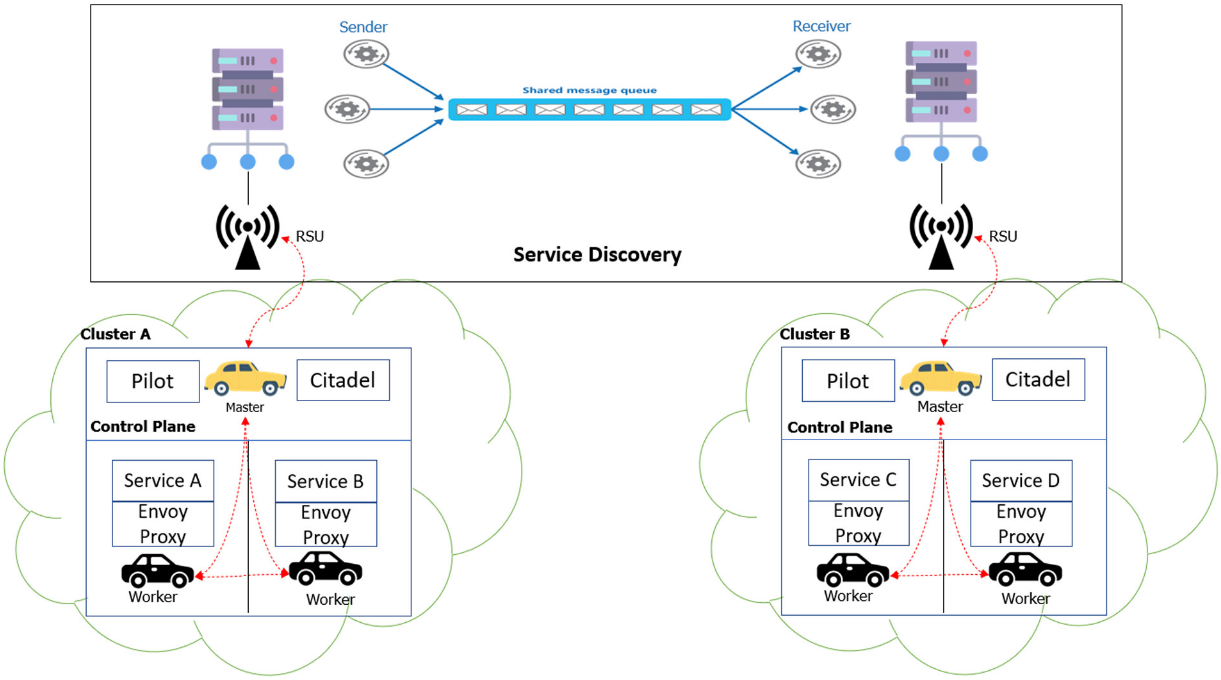

Fig. 2. Cluster communication.

services to push particularly. In vehicular context, the location is pre-selected by the RSU and the concept can be extended to predict the services that need to be pushed to the RSU location.

3. Architecture and methodology

Deploying micro-services on a kubeadm cluster when the user requests a specific service is more costly than having them deployed and ready for use beforehand. In the case of highway where vehicles have high speed compared to other having low speed, a cluster lifetime and the RSU range are too small, so services won't be available for too long and the system won't be stable and reliable. Given that the download and boot time of a container image from the cloud can take around 5 s [2], and the Road Side Unit Controller (RSUC) may have limited resources compared to a cloud, adding a caching layer at the RSU level can serve in decreasing latency of download and boot time when a micro-service is needed. In order to distribute containers over the different cluster components upon solving the VCP problem using optimization techniques, may lower its QoS in such a time critical system. Reformulating the VCP as an assignment problem and solving it using the Hungarian algorithm will make it much more efficient. Moreover, a cluster might not have enough resources to deploy all containers of a specific service. Efficient orchestration between the different clusters to serve the user through a message bus at the RSUC to facilitate the communication between different clusters at different location can help in serving users faster. In this paper, we offer our proposed architecture to address the aforementioned difficulties. For each tier, the role of each component in the architecture is then described. Finally, we discuss a hypothetical interaction between the components using a test example.

3.1. Architecture overview

The architecture shown in Fig. 1, is an expected infrastructure of the next IOV which is formed of Base station (BS), RSUCs, RSUs and smart vehicles having OBUs, and it has three type of communication (Fig. 2). Infrastructure to infrastructure (I2I) communication through Ethernet wires in order for the RSU, RSUC and BS to communicate faster. Vehicle to infrastructure (V2I) communication through DSRC for secure and high speed connection is used by the vehicles to request any service from the RSU. The communication between vehicles using 802.11p in a Kubeadm cluster is called vehicle to vehicle (V2V).

A user has the possibility to request from RSU to host a cluster as a master node, and the RSU is responsible of assigning a master that will initialize a cluster with workers. The reinforcement learning model will be responsible of determining the microservices to be pushed to the cluster. User will request microservices from the master in case there is a nearby cluster, otherwise he will directly request it using the cellular network from the BS. The master node will serve the user either from the previously hosted microservices or will follow the service discovery in Fig. 3. In case of any disconnection from the RSU, a cellular network with the BS is always available.

A BS's job in our design 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 over Ethernet through RSUCs. The addition of the RSUC-RSU layer is intended to reduce traffic load and management on BSs [2]. The RSUC distributes requests between RSUs and shares information like the location of master nodes in range, the services provided by each Kubeadm cluster, and the list of Docker images accessible. In addition it plays a major role in the inter-service communication process when one micro-service of a docker image deployed on one cluster calls another micro-service available on another cluster nearby. It's tough to observe user behavior while requesting services using a simple machine learning model. Therefore on the RSU level, docker images requested by the users are cached on the RSU, pushed and removed to the cluster based on the reinforcement learning (RL) decision. We used RL on each RSU to learn user behavior via experience and adjust as the needs of the services alter by time. The RL agent aims to maximize a certain reward or minimize a given cost [18]. Moreover, it keeps track of the position of the serving kubeadm master in order to link users to services.

The kubeadm master node plays a major role in clustering fog components. It is elected based on the vehicle history, connects to other vehicles using 802.11p, and decides on the optimal service

Table 1
Node Architecture.

Layer	Component
Base Station	Master Manager
RSUC	Master Manager Service Discovery
RSU	Master Manager Cache Container Registry Manager Reinforcement learning Model
Kubeadm Master	Containerization Required Modules Fog/Micro-Service Manager VCP Hungarian Control Plane Profiler
Fog	Containerization Required Modules Kubeadm Cluster Initializer Data Plane Profiler

distribution by solving the VCP problem throughout a collection of specified vehicular fogs, and it keeps track of the state of the containers. Using 802.11p, a device in the fog layer can interact with a neighboring car or the RSU. The master node assigns microservices to the vehicles in the fog, and its profile and availability are likewise updated by the fog.

Before the user makes a request to host services, RSU is running a reinforcement learning model that takes a decision for each service whether to cache at RSU, push or remove a specific service from the cluster. Once the set of services is determined, they are sent to the kubeadm master to run the VCP algorithm to assign each vehicle available in the cluster to host a service. Therefore, once a user requests the service, it will be available on the nearby cluster. However, in case the service was not available on the cluster, we will follow our previous approach in [2].

3.2. Node architecture

This section discusses the functionality of each component per layer in our suggested strategy. The BS, RSUC, and RSU communicate with each other to keep the user linked to the master node by using the Master Manager. [2]. The RSUC works as a service discovery component within this mesh network. While driving, the RSU collaborates with the Kubeadm Master and Fog nodes to establish a dependable Kubeadm cluster with high service availability and long user support periods. Through the Fog/Micro-Service Manager and Control Plane, the master supervises fogs and containers that run on them, and decides on the optimal service distribution among the available vehicles in its cluster. Table 1 lists the architecture components by layer, where RSU is aware of the master location to recover kubeadm fog, and distribute microservices based on the resources available using a memetic algorithm. Service mesh and Software Defined Wireless Networks (SDWN) use a software layer to manage and control the network infrastructure. This will allow for the service mesh to benefit from the advantage of Software Defined Networks (SDN) architecture. The main advantage of using service mesh is that it focuses on providing networking intelligence between microservices on container-based infrastructure, and it provides fast and reliable communication between microservices. It can include load balancing, authentication, and encryption.

The network topology was inspired by Istio service mesh that provides security, service management and flexibility [44]. The functionalities inside each component are described in depth as follows.

1) *Data Plane*: The data plane is made up of sidecars that are intelligent envoy proxies injected to each microservice deployed on the fog. For all the microservices in the service mesh, Envoy proxies mediate all inbound and outgoing traffic. They supplement services with various functionality such as TLS termination. In addition, Fog/Micro-Service Manager and Control Plane allow specific policy decisions to be enforced.
2) *Control Plane*: The main purpose of the control plane is to manage and configure proxies for routing traffic that we get from the data plane through interaction with the injected proxies. It consists of two components, pilot and citadel. Pilot provides service discovery for envoy sidecars on the level of the cluster and traffic management for intelligent routing. In order to control traffic, it converts high level routing rules into envoy configuration and sends them to sidecars. The citadel component is responsible of security and service communication enforcing policies based on service identity. Its built-in identity and credential management allows for strong microservice-to-microservice and end-user authentication, and it adds an encryption layer to the traffic of the service mesh.
3) *Master Manager*: This component is extracted from the work done in [2]. The RSU, RSUC, and BS all have access to the master node information in range. When a master node moves into the range of a new RSU, the old RSU notifies its neighbors of the joining master node. The new RSU sends "Hello" packets to the master to establish that there is a connection. The master profile is contained in the Hello reply messages. The master manager's role is to connect users to master nodes.
4) *Service Discovery*: The service discovery system is deployed on the RSUC service register in order to know where all the endpoints are and which services they belong for guiding traffic within the mesh. Each RSUC is responsible to keep track of the microservices deployed in the clusters that it operates on. RSUC are connected to other RSUCs through ethernet wires which allows a fast communication between RSUCs. The service discovery plays a major role in the service mesh to determine the service namespace. In case the microservice needs to communicate with another microservice deployed on a cluster nearby, the request follows the communication flow shown in Fig. 3. The worker asks the envoy proxy of its microservice A to look for microservice B. It shares the request across the same cluster to serve the worker. Once found, it directly builds a connection between the two microservices to communicate. However, when the envoy can't find the microservice requested on the same cluster, it asks the master to contact RSU and the RSUC to broadcast the request across RSUCs. The RSUC broadcasts the request, but if for certain threshold time no body responds, the microservice will need to be fetched and deployed. Otherwise, an RSUC directly responds with the namespace of the microservice, which will be returned to the worker to know how to communicate with microservice B. The communication follows the Transport Layer Security (TLS) protocol which is an encryption algorithm that is usually followed between servers to provide privacy and data integrity between the two communicating vehicles.
5) *Cache Container Registry Manager*: The container registry is stored in the cloud to push and pull images from it. In order to shorten the time it takes to extract the required microservices, we offer a new cached storage available on the RSU to bring these pictures closer to the user in our architecture. The RSU contains a cache for container image registry where users may find microservices that are commonly requested. These microservices are determined by the reinforcement learning model deployed on the RSU, which decides whether a specific service should be pushed to a cluster, cached on RSU or pulled back from the cluster. All RSUs communicate informa-

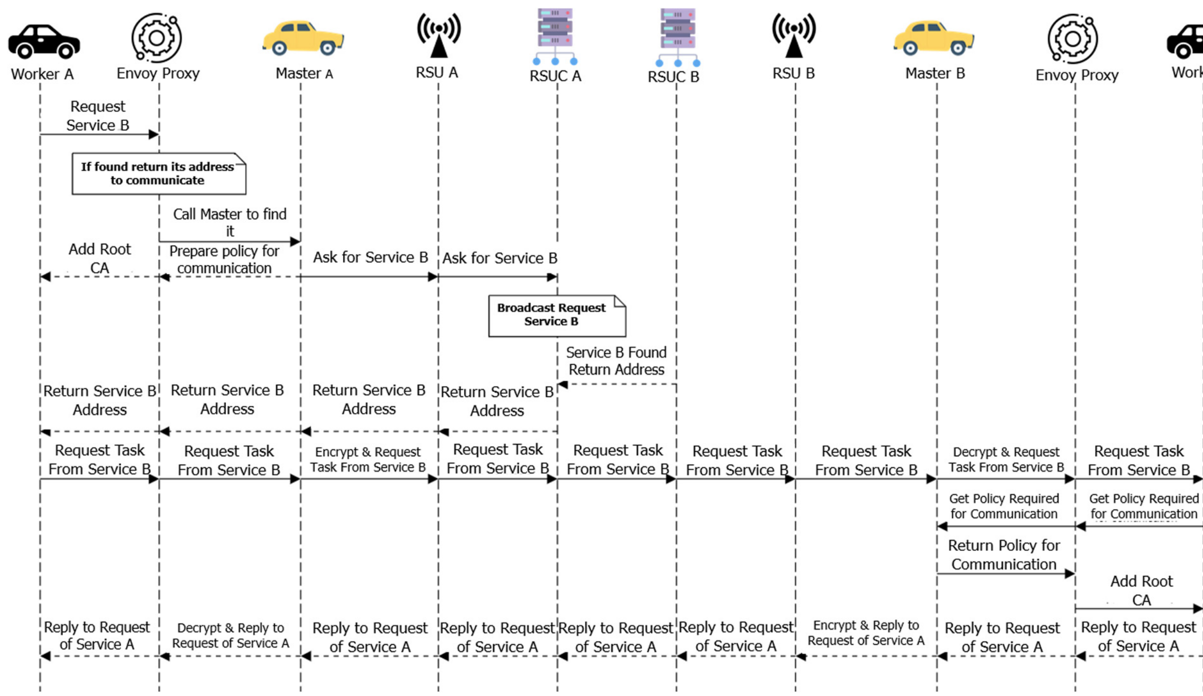

Fig. 3. Microservice-to-microservice communication across clusters.

tion about services they host with their RSUC by replying to the container registry manager's request. If a fog node requests an image from the RSU that it does not have, the RSU calls the RSUC to find out where the service is located so that it may retrieve it and install it on the cluster.

6) *Reinforcement Learning Model for On-Demand Microservices*: Given the requested micro-services history, expressed as the number of requests generated from the location of the RSU at specific times on specific days, and a list of previously pushed micro-services, the RSU benefits from this information to make appropriate decisions about whether or not to push, retrieve or cache a service based on the hours of the day. The creation of fog takes place before the push of any micro-service, and every volunteering device is welcome to join. As a consequence, fog resource use would be maximized, serving more users, implying improved QoS and satisfaction for consumers. Before a user request, the reinforcement learning model determines which microservices are truly required to be hosted on the fog to enhance performance and QoS. Since our reinforcement model has to predict a huge amount of micro-services whether to cache, push or remove it from the cluster, the usage of a deep q-learning approach, is more feasible in our scenario. Given that our model will always face new micro-services, it would be more effective to get the help from deep neural network. In the deep q-learning approach the q-value function is computed using a deep neural network (DNN). DNN solutions require time to learn by interacting with the environment in two cases: (1) learning the environment from scratch through trial and error, and (2) facing unseen patterns of services' demand or available resources in the studied data [10]. In order to train DNN, the replayed memory technique is used to save all its experience in a replayed memory buffer. The experience in the buffer is a tuple formed of the environment, the reward, the chosen action and next state of the environment. Following each episode, the agent selects a set of memories from the replayed memory to train the DNN. Q-Learning is a value-based reinforcement learning method that finds the optimum action-selection strategy using a Q function. The Q-Learning algorithm uses the Bellman Optimality Equation to iteratively learn the optimal Q-value function [38]. Therefore, all the Q-values are recorded in a table, and will be updated using the Q-Learning iteration at each time step. Saving every state-action pair and storing all of them would require a large amount of memory and many more iterations for the Q-table to converge. Hence this approach works better for finite states and action spaces [39].

The *Max-Boltzman Policy* problem is solved using the Max-Boltzmann exploration approach, which assigns a probability to each action and ranks them from best to worst. One disadvantage of the various Greedy approaches is that all exploratory activities are picked at random, meaning that the second best action has the same chance of being chosen as the worst action. ϵ-Greedy and Boltzmann exploration are combined in Max-Boltzmann exploration [25]. It selects the greedy action with probability 1; otherwise, it selects the action using the Boltzmann distribution. A Boltzmann distribution function is used to assign probability. In state s, the probability $\pi(s,a)$ of selecting action a is:

$$\pi(s,a) = \frac{e^{Q(s,a)/T}}{\sum_i^{|A|} e^{Q(s,a^i)/T}} \tag{1}$$

where $|A|$ denotes the number of potential actions and T denotes the temperature. A high T indicates that you will be doing a lot of exploring.

7) *Containerization Required Modules*: This module is taken from the architecture in [2]. Kubeadm is an appropriate orchestration tool for our design since it allows us to employ a variety of devices to build the cluster. Docker is the de-facto containerization technology used for Kubeadm orchestration. In order for the device to engage with the master node and for multiple pods to connect with one another, Kubelet must be installed on it. Kubectl is used to run Kubeadm commands on the master and worker nodes. Docker and Kubeadm, as well as the essential needs, must be installed on cars in order to establish clusters.

8) *Fog/Micro-Service Manager*: This component runs on the master node and is responsible for managing the Kubeadm cluster's worker nodes as well as container execution. It is the responsibility of this component to keep the fog nodes operational and to address any physical or service issues that may arise.
9) *VCP Hungarian*: The VCP solution is responsible of assigning microservices to vehicles. The RSU determines the microservices that should be pushed to the master of the cluster, and the master of the kubeadm cluster is responsible of resolving the VCP problem in order to assign the microservices. This is a time-critical problem since the cluster lifetime might not last for too long and it is essential to distribute the microservices on vehicles as fast as possible. We elaborate more on this problem in Section 4.
10) *Profiler*: This module collects from the vehicles the available CPU, memory, disk space and all other information required to be shared with the master node. These information are required to be able to determine the micro-services distribution and cluster resources. In addition, any other relevant information can be shared in terms of time availability, speed, etc.

4. Vehicular container placement model

Solving the VCP problem plays a major role in determining the efficiency of the network topology. In a real life scenario, vehicles may change their speed and path randomly, yet in our approach we are not taking into consideration the different paths of each vehicle in the network. In our topology, the RSU run the reinforcement learning model to determine what micro-services should be pushed to the cluster. The master node resolves the VCP problem in order to assign for each fog node the required micro-services.

In [2] VCP problem have been formulated as an optimization problem and resolved it using meta heuristic technique called the Memetic Algorithm [2]. Although memetic algorithm is a combination of evolutionary algorithms with local search and proved promising results in such type of problems, the time complexity depends on many factor which makes it unpredictable and unreliable. In addition, notice that [2] had to distribute the service images first randomly and then start moving them between the fog nodes to avoid violating constraints. In this way a micro-service will stop running the container and then move the image to the new host and start running the container. This leads to intermittent availability of micro-services before the convergence of the algorithm, add to it that most genetic algorithms tend to get stuck to a suboptimal solution.

In general, the initial solution selection affects the convergence of genetic algorithms to an optimal result. Most of them are unable to converge quickly enough to reach an optimal solution, or they become trapped in sub-optimal solutions. Generational advancement may be prevented by fitness functions that converge to local optimums and may diverge into nothingness. Moreover, complex problems are increasingly harder to compute and require more processing power. However, the Hungarian method is defined as a combinatorial optimization technique that solves the assignment problems in polynomial time with minimal processing power.

Combinatorial optimization is the process of looking for maxima in a discrete but vast configuration space for an objective function F. Because it can be reduced to the bin packing issue, VCP may be classified as a combinatorial optimization problem. Given a set of N objects each with a specified size s_i, fit them into as few bins, each of size B, as possible. In order to solve this problem, we can reformulate it as weighted bipartite matching problem where on one side we have nodes u_i representing the vehicles of the cluster and the other side v_j are the services and let each edge uv has weight $w_{i,j}$.

In order to determine the weight $w_{i,j}$ of each edge in the weighted bipartite graph, we reformulate the multi-objective constraint optimization problem as one objective function. First, the equality constraints are shown in the following:

$$\begin{aligned} &\min_x \quad f_0(x) \\ &\text{such that} \quad f_1(x) = 0 \end{aligned} \tag{2}$$

In order to approximate the problem, we move the hard constraint $f_1(x) = 0$ into the cost function square it and multiply it by α which is called a penalty parameter. A minimizer of the new approximate function, x, is supposed to be an appropriate solution to the original issue. The problem will be approximated to the following:

$$\min_x \quad f_0 + \alpha f_1(x)^2 \tag{3}$$

Important to note that $f_1(x)^2 \geq 0$ where $f_1(x) \neq 0$ and $f_1(x)^2 = 0$ if and only if $f_1(x) = 0$, hence this condition takes place when the original constraint $f_1(x) = 0$ is satisfied. Squaring $f_1(x)$ is essential to avoid negative numbers that may affect the overall optimization problem since our main concern is to minimize the final cost function. Through this approximation we can ensure that $f_0(x)$ is minimal when $f_0(x) + \alpha f_1(x)^2$ is minimal.

Second, when the optimization problem has an inequality constraint as shown in following:

$$\begin{aligned} &\min_x \quad f_0(x) \\ &\text{such that} \quad f_2(x) \leq 0 \end{aligned} \tag{4}$$

Our aim is to move the constraint to the objective function where we have the added term $\phi(x)$ as 0 when the constraint is satisfied $f_2(x) \leq 0$ or penalize the function and have $\phi(x) \geq 0$ when the constraint is not satisfied, hence $f_2(x) \geq 0$. This can be achieved by setting $\phi(x) = max[0, f_2(x)]^2$. Squaring the penalty value can make it more severe, which is known as quadratic loss function. Therefore to approximate the optimization problem, we reformulate it as follows:

$$\begin{aligned} &\min_x \quad f_0(x) + \alpha\phi(x) \\ &\text{where} \quad \phi(x) = max[0, f_2(x)]^2 \end{aligned} \tag{5}$$

The Penalty function can create abrupt slope changes in the graph near the border, which can make normal reduction methods difficult to apply. Therefore multiplying it by α control the severeness of the penalty when violating the constraint. The higher penalty increases accuracy, and as we narrow in on the solution, the sharpness of the graph is less of a problem. To determine the value of α, we need to keep increasing the penalty weight, hence the closer the approximation x to a solution of the original issue.

In order to determine the cost of assigning a micro-service x to vehicle y, we aim to optimize the cluster lifetime and the number of pushed micro-services under specific cluster resources constraint. The input data will be the resources required for each micro-service and each vehicle resources. The output data will be a mapping of micro-services to the vehicle nodes inside the cluster that satisfies the minimal cost to assign the micro-services predicted to be pushed to vehicles (Table 2).

- *Resources Limit:* On a vehicle, the hosted service's total CPU, memory, and disk resources should be less than the vehicle's available resources. For each service x, we will determine the cost in case we assign it to each vehicle y. We make sure through Equation (9), (10) and (11) that the CPU, Memory

Table 2
Table of Notations.

Variable	Description
i	Micro-service i
j	Vehicle j
n	Number of requested micro-services
m	Number of available vehicles
S	Set of micro-services
V	Set of vehicles
S_{c_i}	CPU usage of S_i
S_{m_i}	Memory usage of S_i
S_{d_i}	Disk usage of S_i
S_{t_i}	Minimum time to host S_i
V_{c_j}	CPU available on V_j
V_{m_j}	Memory available on V_j
V_{d_j}	Disk space available on V_j
U_d	Maximum allowed distance between the user and a cluster
C_t	Cluster time availability
C_d	End distance between user and cluster

and Disk usage of the micro-service are satisfied by the vehicle when c_1, c_2, c_3 are zeros.
The CPU usage of the service should be less than the CPU power of the vehicle

$$S_{cx} \leq V_{cy} \tag{6}$$

The memory usage of the service should be less than the available memory on the vehicle

$$S_{mx} \leq V_{my} \tag{7}$$

The disk usage of the service should be less than the disk space available on the vehicle

$$S_{dx} \leq V_{dy} \tag{8}$$

Equation (6), (7), and (8) will transformed to the constraints c_1, c_2, c_3 respectively, and added to the cost function

$$c_1 = max(0, S_{cx} - V_{cy})^2 \tag{9}$$
$$c_2 = max(0, S_{mx} - V_{my})^2 \tag{10}$$
$$c_3 = max(0, S_{dx} - V_{dy})^2 \tag{11}$$

- *Minimum Cluster Serving Time:* To make sure that the micro-service can be hosted on the cluster and to ensure that the user can be served by the cluster within a reasonable time, we compare the cluster lifetime to the amount of time required to host the micro-service. When the value of $c_4 \leq 0$, then the constraint is satisfied. Therefore, the minimum time to host a service x should be less than the cluster lifetime availability.

$$S_{tx} \leq C_t \tag{12}$$

Equation (12) will transformed to the constraint c_4 and added to the cost function

$$c_4 = max(0, S_{tx} - C_t)^2 \tag{13}$$

- *Distance Threshold To User:* To avoid networking delays and improve the quality of service, we determine a maximum threshold U_d that should be less than the distance between the user and the cluster. If a low-latency application requires the fog to be located close to the user in order to prevent networking delays, VCP makes sure that the distance between the cluster and the user is reasonable. This can be expressed through the cost function by the following:

$$C_d \leq U_d \tag{14}$$

Equation (14) will transformed to the constraint $c5$ and added to the cost function

$$c_5 = max(0, C_d - U_d)^2 \tag{15}$$

- *Objective:* In [2], the authors included two objective functions to Maximize micro-service connection time and Minimize the number of active vehicles. However, in our architecture, we are only deploying micro-services needed on the specific cluster. There is communication across clusters in the vehicle mesh network, especially in the case where the resources required to deploy all micro-services of a specific service are not met. Adding to this, minimizing the number of active vehicles that host micro-services might lead to high utilization in the CPU, hence the response time goes up abruptly [26]. In order to deploy the micro-services safely and host the user request, we maximize kubeadm cluster lifetime as much as possible. We define this function by maximizing the shortest time it takes for one fog to depart the cluster, as maximizing cluster longevity maximizes user serving time:

$$F_1 = min(-C_t \times W_{f1}) \tag{16}$$

Our second aim is to maximize the number of pushed services K_{ij} on the cluster and fulfill as many requests as feasible. In order to achieve this constraint, we tackle it in Algorithm 1 during the assignment of microservices to vehicles. It is further described below in *Algorithm to solve Vehicular Container Placement problem*
In a penalty method, the unconstrained optimization problem are created by adding a component, to the objective function. w is called the overall cost, and it consists of the summation of objective function with a penalty formed of parameter α_i multiplied by a measure of constraint violation. When a constraint is violated, its penalty is strictly positive, and 0 otherwise.
The final cost to be minimized is:

$$\begin{aligned} w = F_1 + \alpha_1 c_1 + \alpha_2 c_2 + \\ \alpha_3 c_3 + \alpha_4 c_4 + \alpha_5 c_5 \end{aligned} \tag{17}$$

- *Algorithm to solve Vehicular Container Placement problem*

Algorithm 1 VCP Hungarian Algorithm.

```
Input: Vehicles and Micro-services Resources
Output: A complete matching M between Vehicles and Micro-Services
while not all microservices are pushed do
    Perform initialization:
     Begin with an empty matching, M_0 = Ø
     Assign feasible values to the dual variables α_i and β_j as follows:
      – ∀v ∈ V,   α_i = 0
      – ∀u ∈ U,   β_j = min_i(c_ij)

    costMatrix = Calculate the cost value for each vehicle i when mi-
     croservice j is assigned to it
     if available vehicles ≥ microservices required to be pushed then
        Add dummy microservice rows
        Perform the Hungarian Algorithm steps
        Output the matching after the n^th stage: M = M_n.
        Microservices required to be pushed = Ø
    else
        Select n micro-services
        Perform the Hungarian Algorithm steps
        Remove microservices assigned from microservice to be
        pushed
        Update Vehicle resources capacity
    end
end
```

VCP is the problem of distributing micro-services to vehicles based on the resources of each vehicle. To resolve such type of a problem, a well known approach is the Hungarian algorithm [33] [34]. The Hungarian method is a combinatorial optimization algorithm that solves the assignment problem in polynomial time $O(n^3)$. To reformulate the VCP as a problem that can be solved using the Hungarian algorithm, we determine the cost of assigning a vehicle to a micro-service and solve the problem in $O(\frac{\textit{number of micro-services}}{\textit{number of vehicles}} n^3)$ to minimize the overall cost. In order to approach this problem, Algorithm 1 takes over it by resolving the Hungarian algorithm to determine the least possible cost when assigning n micro-services to m vehicles. When $n \geq m$, it will assign m micro-services out of n at first iteration while choosing the least cost possible. Then, we deduct the resources required for the assigned micro-services from the vehicles chosen. Now, we repeat the same process on the remaining $n - m$ microservices, but with m' vehicles. m' is the new list of vehicles having new resources reduced according to what we assigned in the previous step and filtering the ones busy with a threshold of more than 85%. This approach will help us achieve our second aim of maximizing the number of pushed micro-services. We repeat the procedure to finish when all the micro-services assigned or when all vehicles become busy.

5. Experiments and results

In this section, we provide experiments to show how the proposed architecture outperforms existing models by predicting in advance the microservices to be pushed to the cluster. We also provoke the importance and efficiency of microservice-to-microservice using Istio architecture where the network topology plays as mesh network and allows communication between vehicles across different clusters.

5.1. Results and discussions

To simulate the overall scheme, we employ the Mininet-Wifi emulator. For vehicle connections, Mininet-Wifi allows to run a comprehensive network on a single workstation using wireless technology where processes play as hosts. It provides tools to represent VANET and run scenarios to simulate behavior of real vehicles. The whole network topology from RSU to RSUC, as well as moving vehicles traveling at predetermined speeds and routes, may be depicted. To use Docker containers as hosts in the emulated network topologies, we use Containernet which is built on top of Mininet-Wifi to simulate the vehicle behavior using a container.

Istio is a service mesh, a modernized service networking layer that allows you to dynamically and quickly automate the application network operations in a transparent and language-independent way. It's typically used to manage many microservices that make up a cloud-native application. Through Istio service mesh, microservices can communicate and share data between each other securely, mainly through peer authentication by mutual TLS. It works with Envoy and Kubernetes to make a microservices-based containerized environment operates smoothly. Kubernetes automates and orchestrates the deployment of containerized applications, while Envoy controls all inbound and outgoing traffic in the Istio service mesh.

The conducted experiments show how our approach combine the advantage of existing approaches and resolve their limitations. First, some services might require high amount of resources to deploy all microservices, and the cluster might have insufficient resources. Given that the problem is time critical, we show the improvement in the time it takes to deploy 1 microservice instead of deploying the whole microservices of a service. Moreover, the VCP problem is classified as NP-hard [2], so it takes long time to resolve the VCP optimization problem. Unfortunately, constant updates of services are time consuming in terms of environment setup, especially when required services and available fog nodes are changing [24]. The memetic algorithm approach followed by [2] take a long amount of time since it has to run it 10 times to get the output. In [2], the algorithm randomly assigns images then the constraint violations are repaired by moving containers from vehicle to another. This may cause an interruption in the connection with the microservice.

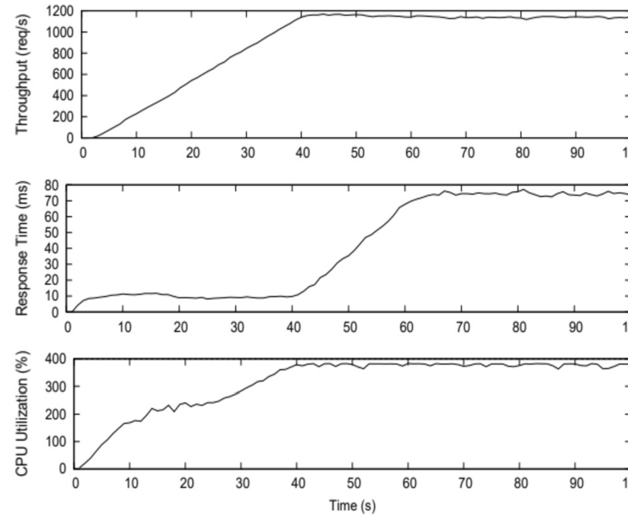

Fig. 4. Relationship between CPU utilization, response time and throughput [41].

In our work, we are assuming that the RSU knows the trajectory of the vehicle, RSUs cover the whole area where vehicles move, and the network coverage of two RSU next to each other has common area when they are under the same RSUC. In the common area, the RSU-1 notifies RSU-2 that the vehicle is moving to its coverage to send hello packets and receive the master profile.

First, to show the importance of using Hungarian algorithm in distributing services on available vehicles equally, we use the work done in [41]. The limited CPU power and insufficient memory and battery, threaten personas performance and viability at any time being [35]. As shown in Fig. 4, when the CPU utilization of an application decreases, we notice decrease in the average response time. This is to be expected because no time is wasted waiting queuing to the existence of other queries, but as CPU utilization approaches 400 percent, the response time increases rapidly.

When using an evolutionary algorithm to solve a multi-objective optimization problem, the convergence of the best solution is dependent on the initial solution chosen, and most algorithms tend to become stuck on a suboptimal solution [42]. Therefore, we reformulate the problem by changing the constraints into a set of weights that build up the cost function to be minimized.

We utilize in our experiments the Google trace 2011 the dataset shown in Table 3 and Table 4 to create our scenario. The Google cluster trace dataset [43] contains data on microservices' resource requirements in terms of CPU, memory, and disk space, as well as data on the cluster's available resources, and each node is represented by a vehicle in our example. In our experiment, when the vehicles used are V4, V5, V6 and V7, while the service to be pushed are S5, S6 and S7. Our VCP algorithm assigns S5 to V7, S6 to V5 and S7 to V4. We notice that vehicle CPU is not fully busy, but the services are distributed equally on the volunteering devices which causes less traffic on one node of the cluster. Moreover, the

Table 3
Vehicles Dataset Combining Google Cluster Trace 2011-2.

	V_p	V_t	V_s	V_c	V_m	V_d	Departure	V_o
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 4
Services dataset.

Service	ID	S_c	S_m	S_d
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

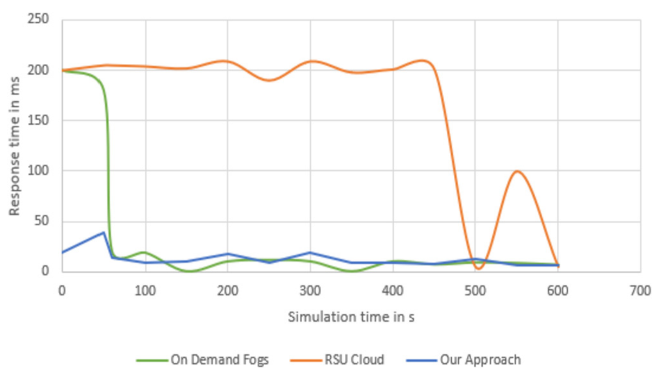

Fig. 5. Performance of RSU cloud in [7,8], on-demand-fog [2] and our approach.

resource limit constraint is respected where the memory, disk and CPU of a service are less than the vehicles capability.

To simulate the environment of an Istio cluster, we use Minikube. Minikube runs a single-node Kubernetes cluster on a single machine. We retrieve from Google Cloud Platform on Github a set of microservices to be deployed on our cluster. First downloading the microservices required around 8 minutes on 9 Mb/s internet connection speed. After the download, the deployment of the 12 microservices on a Minikube took around 57 seconds. However, the deployment of one microservice with an envoy proxy injected will require around 9 seconds. In case the RL model correctly predicted the microservice, the user won't notice any delay. However, in case it did not, deploying one microservice instead of the whole service will improve QoS.

To show that connection across clusters is better than deploying the whole project on the same cluster, we create a network topology that imitates our architecture. We fix the position of RSU1, RSU2, RSUC1 and RSUC2, and we assume that each RSU is in different area. Therefore RSU1 is connected to RSUC1, RSU2 is connected to RSUC2, but RSUC1 and RSUC2 are connected. We initiate a scenario with four vehicles having 300 m wireless coverage moving on a determined path. RSUs are aligned in sequence but a gap of 200 m between RSU1 and RSU2. V1, V2, V3 and V4 start moving at time 0 s, 2 s, 25 s and 25 s with a speed of 19 m/s, 22.3 m/s, 25 m/s and 5 m/s respectively. Note that V4 will remain in the RSU1 range at all time. We measure the time it takes to send a request and receive the response. In case the vehicle requesting lies in the same cluster on average it takes 8.5 ms. However,

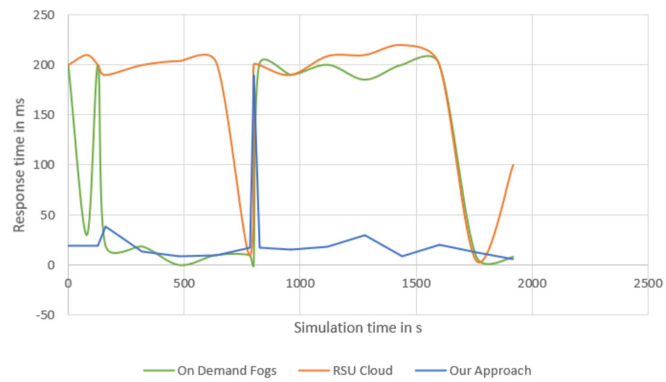

Fig. 6. Performance of RSU cloud in [7,8], on-demand-fog [2] and our approach on high load.

when the microservice is deployed on another cluster, the discovery time may take up to 1 s and then any other communication will happen within 9 ms on average. Although we had an increase in the communication between microservices, but this time is invested against the time we saved during the deployment process. Moreover, it will resolve the cluster resource limitation problem.

In Fig. 5, we show that the usage of containerized microservices is better than VM due to the size of VM [7] [8] compared to docker containers. Moreover, the improvement of our approach over [2] is illustrated in the start up scenario when the reinforcement learning determine accurately the microservices before any user request. Note that, the on-demand fogs approach have show failure for a small amount of time due to the usage of memetic algorithm for VCP. Given that [2] runs the containers randomly at first and then starts moving containers between host to solve the VCP, the move of a docker container requires to stop, move it to another host and run it again. Although running a container from a host to another differ from a container to another, in our experiment we notice that this took around 0.4 s.

In Fig. 6, we used the same setup that we used in our previous experiment by setting V3 as a user, but we increased the number of requests to the cluster to include all 7 services in Table 4 during the simulation in order to host multiple different services at the same time. Therefore, the number of containers required to download in our approach and [2] have increased. Moreover, for [7] and [8] more virtual machines were required to host services on the cluster. Due to the need of deploying more services to serve these requests in this experiment, more resources were required at some point that exceed the cluster available resources.

For [7] [8] we keep on noticing a long time of serving requests through the cloud due to the long download time of virtual machines. However, [2] the containers were downloaded very fast, the cluster hosted the containers and it started serving requests. Due to the high load of microservices sent to the cluster, few other microservices were not yet downloaded and hosted. Thus, we notice another increase at t=50 s. In our approach, because we had a reinforcement learning that correctly predicted the microservices to be hosted on the fog, we do not notice any increase in response time at the beginning of the scenario. However, at t=800 s, when inter-service communication took place, [7] another VM has to be downloaded and it started downloading and serving through the cloud causing an increase in response time. Because of lack in resources, [2] approach was not able deploy the other service on the cluster causing an increase also in response time when communicating with the other service. However, our approach did not deploy all microservices of the service and was communicating with other microservices hosted on nearby clusters. At t=800 s the request did not find the requested microservice nearby, causing an increase in response time to fetch, download and host its container. Once the new microservice is deployed, the response time decreased and inter-service communication across clusters took place.

5.2. Reinforcement learning model for on-demand microservices

These experiments have two major goals, which are mentioned below:

1) The first objective of our reinforcement learning model is to prove that it was able to learn by experience. This can be achieved by showing that the model prioritizes pushing microservice to cluster instead of caching them on the RSU or keep it in the cloud.
2) The second objective is to show the reduction in the number of requests sent to the cloud and the importance of serving them through the fog.

In this experiment, we use python as a programming language and Keras-RL as a library to build the reinforcement learning model. The data used to simulate the experiment contains a single month of server logs of the NASA.gov website [40]. The data is dated August 1995, and it was retrieved from [40]. Due to resource limitation, only a chunk of it is used in this experiment.

The model was deployed on the RSU which determines the action to be taken for each request. The model is expected to start without any previous information and keep track of the requests that are taking place from this location. When new microservice is requested, it will be added to the RSU database and set its status. When an on-demand request takes place, the RSU runs the model to determine the status of each of the microservices previously seen. Once the status is set to push, the RSU initializes the cluster and requests the microservice then pushes it to the cluster. In case the status was cache, it pulls it from the cloud and keeps it on the RSU. If its status was "remove", it removes it from the cluster in case found.

In order to simulate the process, we used the following:

1) *Cloud*: It has all the micro-services
2) *Micro-Service*: It needs an ID and size of the container
3) *Vehicle*: It is determined by its memory capacity, the services deployed on this vehicle, and its status whether it belong to a cluster or not.
4) *RSU*: It is similar to the vehicle where it has a memory capacity and services cached.
5) *Cluster*: It includes a set of vehicles that belongs to this cluster, and the RSU that created it.

To Simulate on going scenarios of vehicles moving inside and outside the range of cluster, first we define four moving vehicles where their availability flags are set randomly. In our simulation, we are assuming the worst case scenario where a vehicle can exit cluster at any time and no recovery approach is applied. Because the vehicles can move randomly and we are not taking into consideration the predefined path of each vehicle, we used the random flags to determine whether a vehicle is currently part of this cluster or not, which changes constantly.

Our reinforcement learning model has 3 actions to choose between them either to push a microservice to the cluster, or to cache the microservice, or pull a microservice. Each state of the reinforcement learning model is defined by the time and service ID. Once a request is sent to the cluster, we calculate the reward first assuming that the model has pre-determined the microservices that are pushed. After that we take an action for each microservice the model has already seen. Once an action is taken to push to cluster, we make sure a storage capacity is available otherwise we save it on the RSU cache. The RSU can take over the work of the cluster in case the vehicle storage is minimal or the number of vehicles participating in the cluster is low. In case the RSU was full of storage, it will no more be pushed. This penalizes the model by -1 point. When a user requests specific service, the reward is calculated based on a basic function that is used to penalize the model where the model receives 2 points in case the microservice was found on cluster, 1 point in case the microservice was found on the RSU, and -1 for each request the cloud serves.

In Fig. 7, more than 90% of total requests are served by the cluster and the RSU. Reducing the number of requests served by the cloud would increase the performance of the overall scheme and reduce the traffic on the cloud, which in return can improve the cloud performance.

In Fig. 8, we compare the number of request served by the cloud to the number of requests served by the cluster and the number of requests served by the RSU. It shows the number of requests served by each component per hour. The main goal of the model is achieved when the most number of requests is served by the cluster, then fewer requests are served by the RSU and finally the remaining labeled in red are served through the cloud.

The Reinforcement learning model reduced massively the number of requests served by the cloud as shown in Fig. 7, and the importance of the caching action that reduced requests from the cloud when there is not enough space on the cluster to host was reflected Fig. 8.

After taking the decision foraching action each microservice using our reinforcement learning model, the table of status for each microservice is created. We initialize the cluster with the microservices that the reinforcement learning predicted. We face the following scenarios:

1) The service is not available and we can follow the approach provided in [2].
2) The service is set to be pushed to the cluster, then we need to resolve the VCP problem to assign microservices to vehicles.
3) The service is on the RSU cached, so we directly deploy it on the available vehicles.

5.3. Comparative study over existing approaches

Comparing our approach to containerized microservices [2], RSU cloud [7] and Follow-me cloud (FMC) [8], our approach offers the best solution to serve vehicular applications using RSUs, VM and docker containers. Our approach overcomes [2], [7] and [8] in terms of time to download a service when our reinforcement learning model determines the correct microservices to be pushed and cached. [2] initializes the cluster and pushes the microservices required after the user request, and [7] [8] need to download VMs. Hence, the user need to wait some time before his request is served unlike our approach. Note that the microservice is always updated to the latest in short amount of time in [2] and our approach, unlike in [7] [8], because docker containers are much less in size and can work on any OS, unlike VM, which are bigger in size and you need to download them each time you

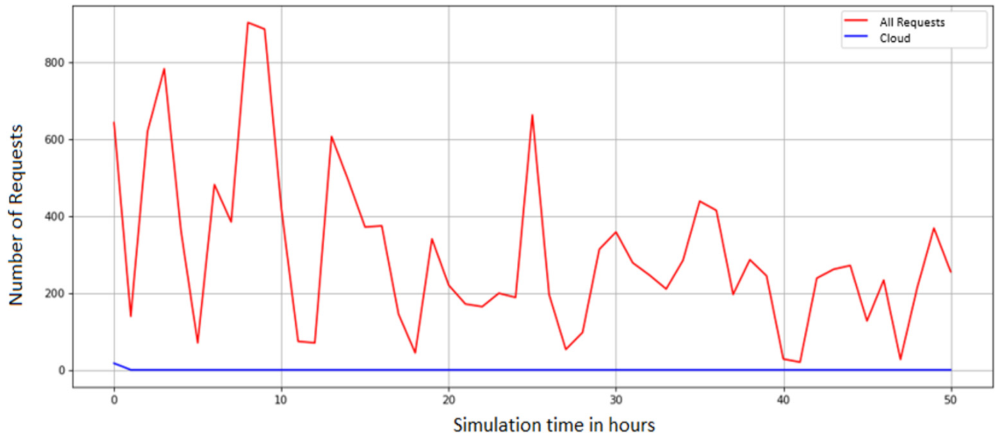

Fig. 7. Number of requests served by the nearby fog compared to cloud.

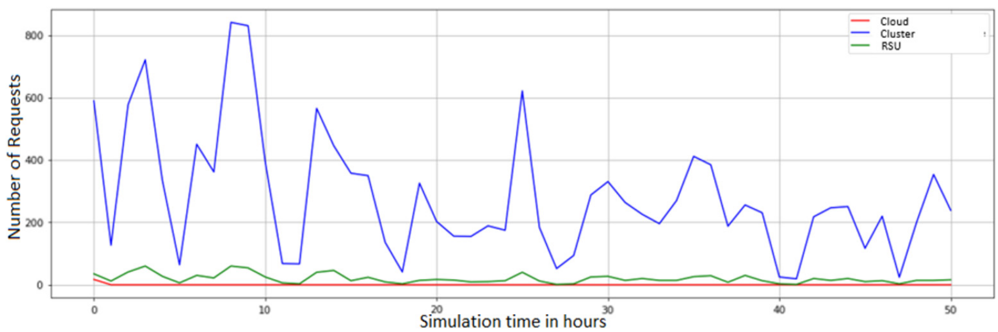

Fig. 8. Our approach showing the number of requests served by the cluster, cloud and RSU.

push a service. In [2], all microservices of a service are pushed to the cluster when a user request a specific microservice. In our approach, we take into consideration cluster having low storage capacity where only microservices needed are pushed, and we allow to deploy on-demand microservices. Moreover, the communication allowed across the infrastructure between vehicles forming a mesh network will reduce the need to deploy additional microservices. The communication across clusters may require longer time sometimes as shown in Fig. 5, yet it seems minimal compared to the waiting time required to download and start the container.

6. Conclusion

In this paper we explored that number of volunteering vehicles may limit the cluster capability of deploying services because one or two OBUs are not enough to handle the processing with insufficient resources. Moreover, deploying microservices on vehicles may be time critical because cars may not be able to keep an alive connection with the user to provide longtime support. Therefore, the main goal was to optimize the time of deployment of microservice to make it available the soonest for the user. Hence, we proposed deploying reinforcement learning model on RSU's to be able to determine which microservice have to be deployed and cached on the edge beforehand. To the best of our knowledge, no work in the literature has used service mesh in vehicular network to benefit from the advantage of providing networking intelligence on container-based microservices infrastructure with fast and reliable communication between them. Another issue that emerges in this context is determining the appropriate assignment of micro-services on available vehicles in order to optimize serving time while maintaining reachability. We suggested to resolve the VCP problem using Hungarian Algorithm to benefit from its performance and running time. The deployment of a microservice needed compared to the full service have shown an improvement of 84%. Our experiment showed that the number of processed requests by the cloud have massively reduced after hosting the reinforcement learning model on the RSU, and we showed how cache have served in reducing that amount. As future work, it is interesting to take into consideration each vehicle path to keep the longest alive connection for support. To secure the network, a lightweight intrusion detection system can be integrated [46] to autonomously detect suspicious activities, because the communication links between these vehicles might be susceptible to different security threats such as eavesdropping, Denial of Service (DoS), black-hole attack, and other attacks that compromise the operational safety and data reliability [47]. Future cellular network technologies appear to be highly promising in terms of supporting real-time vehicle applications. As a result, as networking technology develop, our design and methodology may be changed and improved.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] A. Nsouli, A. Mourad, W. El-Hajj, Reinforcement learning based scheme for on-demand vehicular fog formation and micro services placement, in: Proceedings of the 18th IEEE International Wireless Communications and Mobile Computing Conference (IWCMC 2022), IEEE, 2022, pp. 1244–1249.

[2] H. Sami, A. Mourad, W. El-Hajj, Vehicular-OBUs-as-on-demand-fogs: resource and context aware deployment of containerized micro-services, IEEE/ACM

Trans. Netw. 28 (2) (April 2020) 778–790, https://doi.org/10.1109/TNET.2020.2973800.

[3] P. Farhat, H. Sami, A. Mourad, Reinforcement R-learning model for time scheduling of on-demand fog placement, J. Supercomput. 76 (2020) 1–23, https://doi.org/10.1007/s11227-019-03032-z.

[4] A. Hammoud, H. Sami, A. Mourad, H. Otrok, R. Mizouni, J. Bentahar, AI blockchain and vehicular edge computing for smart and secure IoV: challenges and directions, IEEE Int. Things Mag. 3 (2) (June 2020) 68–73, https://doi.org/10.1109/IOTM.0001.1900109.

[5] H. Sami, A. Mourad, Dynamic on-demand fog formation offering on-the-fly IoT service deployment, IEEE Trans. Netw. Serv. Manag. 17 (2) (June 2020) 1026–1039, https://doi.org/10.1109/TNSM.2019.2963643.

[6] K. Mershad, H. Artail, Finding a star in a vehicular cloud, IEEE Intell. Transp. Syst. Mag. 5 (2) (2013) 55–68.

[7] M.A. Salahuddin, A. Al-Fuqaha, M. Guizani, Software-defined networking for rsu clouds in support of the internet of vehicles, IEEE Int. Things J. 2 (2) (2015) 133–144.

[8] T. Taleb, A. Ksentini, P. Frangoudis, Follow-me cloud: when cloud services follow mobile users, IEEE Trans. Cloud Comput. (2016).

[9] S. Arisdakessian, O. Abdel Wahab, A. Mourad, H. Otrok, N. Kara FoGMatch, An intelligent multi-criteria IoT-fog scheduling approach using game theory, IEEE/ACM Trans. Netw. (2020), IEEE.

[10] H. Sami, H. Otrok, J. Bentahar, A. Mourad, AI-based resource provisioning of IoE services in 6G: a deep reinforcement learning approach, IEEE Trans. Netw. Serv. Manag. (2021), IEEE.

[11] M. Salahuddin, A. Al-Fuqaha, M. Guizani, Software-defined networking for rsu clouds in support of the internet of vehicles, IEEE Int. Things J. 2 (2) (2015) 133–144.

[12] Xueshi Hou, Yong Li, Min Chen, Di Wu, Depeng Jin, Sheng Chen, Vehicular fog computing: a viewpoint of vehicles as the infrastructures, IEEE Trans. Veh. Technol. 65 (2016) 3860–3873, https://doi.org/10.1109/TVT.2016.2532863.

[13] Malik Saad, Muhammad Toaha Raza Khan, Gautam Srivastava, Rutvij Jhaveri, Md Mahmudul Islam, Dongkyun Kim, Cooperative vehicular networks: an optimal and machine learning approach, Comput. Electr. Eng. 103 (2022) 108348, https://doi.org/10.1016/j.compeleceng.2022.108348.

[14] M. Sookhak, F.R. Yu, Y. He, H. Talebian, N.S. Safa, N. Zhao, M.K. Khan, N. Kumar, Fog vehicular computing: augmentation of fog computing using vehicular cloud computing, IEEE Veh. Technol. Mag. 12 (3) (2017) 55–64.

[15] A. Mourad, H. Tout, O. Abdel Wahab, H. Otrok, T. Dbouk, Ad-hoc vehicular fog enabling cooperative low-latency intrusion detection, IEEE Int. Things J. (2020), IEEE.

[16] F. Bonomi, R. Milito, P. Natarajan, J. Zhu, Fog Computing: A Platform for Internet of Things and Analytics. Big Data and Internet of Things: A Roadmap for Smart Environments, Springer, 2014, pp. 169–186.

[17] O. Salman, I. Elhajj, A. Kayssi, A. Chehab, Edge computing enabling the internet of things, in: Proceedings of the 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT), IEEE, 2015, pp. 603–608.

[18] H. Sami, A. Mourad, H. Otrok, J. Bentahar, FScaler: resource scaling of containers in fog clusters using reinforcement learning, in: Proceedings of the 16th IEEE International Wireless Communications and Mobile Computing Conference (IWCMC 2020), IEEE, June 15–19, 2020.

[19] J. Li, J. Jin, D. Yuan, H. Zhang, Virtual fog: a virtualization enabled fog computing framework for internet of things, IEEE Int. Things J. 5 (1) (2017) 121–131.

[20] G. Lee, Software defined networking-based vehicular adhoc network with fog computing, in: Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), IEEE, 2015, pp. 1202–1207.

[21] M.A. Salahuddin, A. Al-Fuqaha, M. Guizani, Software-defined networking for rsu clouds in support of the internet of vehicles, IEEE Int. Things J. 2 (2) (2015) 133–144.

[22] I. Sorkhoh, D. Ebrahimi, R. Atallah, C. Assi, Workload scheduling in vehicular networks with edge cloud capabilities, IEEE Trans. Veh. Technol. 68 (9) (2019) 8472–8486.

[23] R. Roman, J. Lopez, M. Mambo, Mobile edge computing, fog et al.: a survey and analysis of security threats and challenges, Future Gener. Comput. Syst. 78 (2018) 680–698.

[24] H. Sami, A. Mourad, H. Otrok, J. Bentahar, Demand-driven deep reinforcement learning for scalable fog and service placement, in: IEEE Transactions on Services Computing, IEEE, 2021.

[25] M. Wiering, Explorations in Efficient Reinforcement Learning, 1999.

[26] K. Deb, Multi-Objective Optimization Using Evolutionary Algorithms, Wiley, 2001.

[27] Z. Hao, E. Novak, S. Yi, Q. Li, Challenges and software architecture for fog computing, IEEE Internet Comput. 21 (2) (2017) 44–53.

[28] R. Mahmud, R. Kotagiri, R. Buyya, Fog Computing: A Taxonomy, Survey and Future Directions. Internet of Everything, Springer, 2018, pp. 103–130.

[29] S. Yi, C. Li, Q. Li, A survey of fog computing: concepts, applications and issues, in: Proceedings of the 2015 Workshop on Mobile Big Data, ACM, 2015, pp. 37–42.

[30] H. Tout, C. Talhi, N. Kara, A. Mourad, Smart mobile computation offloading: centralized selective and multi-objective approach, Expert Syst. Appl. 80 (2017) 1–13.

[31] W. Marrouche, H.M. Harmanani, Heuristic approaches for the open-shop scheduling problem, in: Information Technology-New Generations, Springer, 2018, pp. 691–699.

[32] T. Dbouk, A. Mourad, H. Otrok, H. Tout, C. Talhi, A novel ad-hoc mobile edge cloud offering security services through intelligent resource-aware offloading, IEEE Trans. Netw. Serv. Manag. 16 (4) (2019) 1665–1680.

[33] S. Ge, M. Cheng, X. He, X. Zhou, A two-stage service migration algorithm in parked vehicle edge computing for Internet of Things, Sensors 20 (10) (May 2020) 2786.

[34] T.S. Chin, M.S. Shokri, N.Z. Zaibidi, Application of assignment model in resources utilization of University Service Centre, in: Proceedings of the 4TH Innovation and Analytics Conference & Exhibition (IACE), 2019.

[35] H. Tout, C. Talhi, N. Kara, A. Mourad, Selective mobile cloud offloading to augment multi-persona performance and viability, IEEE Trans. Cloud Comput. 7 (2) (2019) 314–328, IEEE.

[36] A. Hammoud, H. Otrok, A. Mourad, Z. Dziong, Stable federated fog formation: an evolutionary game theoretical approach, Future Gen. Comput. Syst. J. 104 (2021) 92–104, Elsevier.

[37] W. Felter, A. Ferreira, R. Rajamony, J. Rubio, An updated performance comparison of virtual machines and linux containers, in: Proceedings of the 2015 IEEE International Symposium Performance on Analysis of Systems and Software (ISPASS), IEEE, 2015.

[38] A. Amine, Q-learning algorithm: from explanation to implementation, towards data science, https://towardsdatascience.com/q-learning-algorithm-from-explanation-to-implementation-cdbeda2ea187, 2020.

[39] V. Phy, Reinforcement learning concept on cart-pole with DQN, towards data science, https://towardsdatascience.com/reinforcement-learning-concept-on-cart-pole-with-dqn-799105ca670, 2019.

[40] https://www.kaggle.com/souhagaa/nasa-access-log-dataset-1995.

[41] Carlos Oliveira, Vinicius Petrucci, Orlando Loques, Impact of Server Dynamic Allocation on the Response Time for Energy-Efficient Virtualized Web Clusters, 2010.

[42] K. Deb, Multi-Objective Optimization Using Evolutionary Algorithms, Wiley, Chichester, 2001.

[43] C. Reiss, J. Wilkes, J.L. Hellerstein, Google Cluster-Usage Traces: Format+Schema, Google Inc., White Paper, 2011, pp. 1–14.

[44] A. Koschel, M. Bertram, R. Bischof, K. Schulze, M. Schaaf, I. Astrova, A look at service meshes, in: Proceedings of the 12th International Conference on Information, Intelligence, Systems & Applications (IISA), 2021, pp. 1–8.

[45] G. Rjoub, O.A. Wahab, J. Bentahar, A.S. Bataineh, Improving autonomous vehicles safety in snow weather using federated YOLO CNN learning, in: Proceedings of the Mobile Web and Intelligent Information Systems, MobiWIS, in: Lecture Notes in Computer Science, vol. 12814, 2021.

[46] S. Zafar, S. Jangsher, O. Bouachir, M. Aloqaily, J. Othman, QoS enhancement with deep learning-based interference prediction in mobile IoT, Comput. Commun. (2019).

[47] M. Aloqaily, R. Hussain, D. Khalaf, D. Slehat, A. Oracevic, On the role of futuristic technologies in securing UAV-supported autonomous vehicles, IEEE Consum. Electron. Mag. (2022).