Data Driven Service Orchestration for Vehicular Networks

Anestis Dalgkitsis, *Student Member, IEEE,* Prodromos-Vasileios Mekikis, *Member, IEEE,* Angelos Antonopoulos, *Senior Member, IEEE,* Christos Verikoukis, *Senior Member, IEEE,*

***Abstract*—As technology progresses, cars can not only be considered as a transportation medium but also as an intelligent part of the cellular network that generates highly valuable data and offers both entertainment and security services to the passengers. Therefore, forthcoming 5G networks are said to enhance Ultra-Reliable Ultra-Low-Latency that will allow for a new breed of services that will disrupt the industry as we know it today. In this work, we devise a unique fusion of Deep Learning based mobility prediction and Genetic Algorithm assisted service orchestration to retain the average service latency minimal by offering personalized service migration, while tightly packing as many services as possible in the edge of the network, for maximizing resource utilization. Through an extensive simulation based on real data, we evaluate the proposed mobility orchestration combination and we find gains in low latency in all examined scenarios.**

***Index Terms*—Mobility Prediction, Deep Learning, Proactive Resource Allocation, 5G Wireless Mobile Networks, Vehicular Networks, Intelligent Automotive**

I. Introduction

FIFTH Generation (5G) networks are *ante portas* and it is expected to bring a whole new ecosystem of innovative applications. They can support the broad expansion of cellular Vehicle-to-Everything (V2X) to enhance both passenger and pedestrian security in advanced automotive scenarios. Intelligent autonomous driving vehicles can benefit from the existing cellular network architecture to enable data-intensive and latency-sensitive abilities such as situational awareness, proactive collision warning, intersection management, high-definition maps, real-time alerts, Augmented Reality and even lower the cost of autonomous driving [1].

The modern vehicles are equipped with various sensors that generate hundreds to thousands of megabytes per second [2]. Nowadays, location-based services are flourishing, providing an unprecedented opportunity to collect fine-grained spatiotemporal data about the places users visit [3]. Accumulating user location data is a gateway to the future location prediction and movement behavior of the user. [4]. Human daily mobility presents a high degree of temporal and spatial regularity. Each individual is characterized by a time-independent travel distance and a significant probability to return to a few highly frequented locations [5]. It is easy to record the movement as a sequence of time-stamped locations, which can be analyzed and used to predict the future movement [6]. Practically, human mobility prediction is of great importance in a wide range of applications, ranging from personalized recommendation systems to intelligent transportation, urban planning, and mobility management in the 5G mobile communication systems [7], [8].

Centre Tecnologic de Telecomunicacions de Catalunya (CTTC), Avinguda Del Canal Olimpic S/N, Parc Mediterrani de la Tecnologia, 08860 Castelldefels, Barcelona, Spain (e-mail: aantonopoulos@cttc.es; cveri@cttc.es).

Iquadrat Informatica, Carrer del Dr. Rizal 10, 08006 Barcelona, Spain (e-mail: a.dalgkitsis@iquadrat.com; vmekikis@iquadrat.com).

The expansion from Cloud Data-Centers (DCs) to Edge Computing overcomes many obstacles, the greater of all is the distance, in terms of latency or hops, between the end-user and the services. The concept of Multi-Access Edge Computing (MEC), pushes the boundaries of cloud computing technologies advancements, such as Software-Defined Networking (SDN) and Network Function Virtualization (NFV), to the proximity of the user [9]. Services should now be capable of following the user geographically in the network and offer the best Quality of Service (QoS), without hindering the back-end network performance.

By deploying MEC servers throughout the city, it is possible to move cloud services to the edge of the mobile network and introduce intelligence and connectivity in the field, which is crucial for the success of delay-critical V2X use cases. The ETSI Standardization covers numerous use-cases, such as Intersection Movement Assist, Advanced Driving Assistance or Vulnerable Road User [10]. It also defines the standards to make such scenarios attainable.

In this article, we propose a method to decrease the latency between the users and their services by relocating them to the nearest edge at any given time. Our implementation is based on Convolutional Neural Networks (CNNs) and Genetic Algorithms (GAs). It can foresee the next step of each user according to their *a priori* mobility habits and live migrate the services between the MEC servers in a way that retains the End-to-End (E2E) latency as low as possible. Our contribution is twofold by:

- Using a CNN to determine *a posteriori* user mobility trajectory in conjunction with existing street network data and Live Migrate the services proactively,
- Utilizing a GA to tightly pack as many services as possible close to the users, with respect to their priority.

The remainder of the paper is organized as follows. Section II shows various related works currently available in the literature. Section III offers an overview of the System Model and defines all entities that make it apart. Section IV presents the problem formulation and analyzes in detail the background needed to describe the work. Section V showcases the simulation setup and the results produced. Finally, in Section VI we conclude our work and reveal our future intentions.

II. Literature Review

Technologies such as SDN and NFV can be exploited to further reduce the service latency between the user and the applications [11]. Most of the existing research works try to accomplish this by dynamically placing Virtual Machine (VM) service instances, that be transferred in an instant with Live Migration (LM) and allow them to follow the user geographically in the network [12].

Many works in recent literature have studied the idea of proactive service migration in edge computing environments, such as [13]. They provide an extensive analysis in proactive Service Instance Migration at the mobile network edge to support the viability of the idea based in ETSI MEC specifications [10]. Ultra-Dense networks in conjunction with edge computing can offer such a solution, by adding computational resources that can host services in the edge. There is a need for an efficient algorithm that can orchestrate and control the entire virtual infrastructure with such a colossal number of users. The system needs to track and predict the user movement so it can proactively calculate the placement of user services, as the user moves. Ivan Farris et al. [14], formulate the problem of proactive service migration in multi-edge cellular 5G networks by leveraging on the prediction of user mobility patterns. They consider the deployment of multiple service replicas to guarantee fast relocation of user services. By taking advantage information about the user position and orientation, they perform a simple user path prediction and replicate services only at the edges towards the user mobility direction. As stated in their work, the proposed approach introduces additional cost per service However, a more advanced prediction could be developed, that can exploit mobile service usage cartography [15], road map data for urban environments or geographical information, and leverage big data techniques on past measurements [16]. Such an intelligent scheme could deliver superior performance. In our work, we propose a system based on Deep Learning (DL). Not only it can predict the movement direction, but also actually learn user mobility preferences by training on user historical data as well. We utilize LM to avoid data overhead introduced by instance replication and migrate the service based on personalized predictions.

Recent works in dynamic Virtual Network Function (VNF) placement include [17], which advocates allocating services to a distributed edge infrastructure, minimizing E2E latency from all users to their associated services. Unlike similar works, they offer a smart scheduler that re-calculates the optimal placement automatically, in such a way that keeps the services migration cost as low as possible. The proposed solution is based on an Integer Linear Programming algorithm and identifies the appropriate allocation for all services that have the least E2E latency from all users. Although the results seem promising at first sight, the comparison with the cloud-only deployment is weak. Another major challenge towards efficient service placement in edge computing environments is to understand and predict the mobility of users. The abundance of data in modern networks can be beneficial for trajectory prediction in urban environments. Choi et al. in [18] introduce a DL approach to learn and predict network-wide vehicle movement patterns in urban networks. Specifically, their study employs a Recurrent Neural Network (RNN) that is capable of predicting the next locations in a vehicle's trajectory, given the previous locations. They take advantage of the similarity between trajectory sequence prediction and language modeling, in which RNNs proved a great success. On the other hand, Lv et al. in [19], introduce an algorithm which adopts CNNs to model vehicle trajectories as images and achieve precise destination prediction. To study the problem of precise destination prediction, they utilize a lengthy taxi dataset. Unlike many prediction approaches, they capture the diverse two-dimensional patterns of trajectories in different spatial scales.

Regarding the studies around movement prediction, the unpredictable movement of the users makes it difficult to maintain low latency and service performance in edge computing scenarios. Ouyang et al. in [20], design a novel mobility-aware online service placement framework to achieve the desired balance between user-perceived latency and migration cost. Tackling this issue requires to find a trade-off since frequent service migrations can increase the operational cost and degrade QoS. The authors utilize a Lyapunov optimization technique to incorporate the long-term cost budget constraints into a series of real-time optimization problems, they develop two efficient heuristic schemes based on the Markov approximation and best response update techniques to approach a near-optimal solution. Further, Yu et al. in [21] investigate the problem of service migration in multiple V2N services with different QoS requirements. The main focus is the optimization of long-term service latency in MEC. They propose a partial dynamic optimization algorithm that utilizes priority queue. They evaluate the performance of the proposed algorithm by simulating the real world taxi trajectory in Rome and they prove that the proposed algorithm can keep a stable service latency and approximate the optimal solution of the above-mentioned problem. The proposed algorithm has a time-complexity with an upper bound, which means the algorithm can be finished in each slot.

Inspired by these successful researches, we develop an algorithm that can offer both personalized and low latency service to multiple users with high mobility. Throughout the next sections, we extend these works by combining a CNN for personalized mobility prediction and a GA based recurrent algorithm responsible for resource allocation in the network edge to tackle the latency issue.

III. System Model

We consider a city-wide vehicular network divided into c *Cells* organized together into groups that form a *Region* r. We assume that every region has a *Regional Server*. The role of the regional server is to provide SDN signaling to the underlying part of the network. Optionally, it can also host user services, in case of severe network congestion in the edge of the network. Every cell c includes a *MEC Server* m with limited computational resources, collocated with one Base Station (BS) whose coverage area defines the size and shape of the cell. Furthermore, there are several remote DCs with abundant computational resources for vehicular service hosting, along with an orchestrator that connects to the SDN controllers of the regional servers. Additionally, we assume a set of self-driving vehicles $\mathcal{V}$ that cruise through the city cells. They communicate with each BS through a millimeter Wave (mm-Wave) 5G transceiver and require a specific amount of computational resources from the network to host their services S_v. A graphical representation can be seen in Fig. 1 and the notations of this work are summarized in Table II. The proposed architecture is composed of several interconnected entities, all discussed in detail in the following subsections.

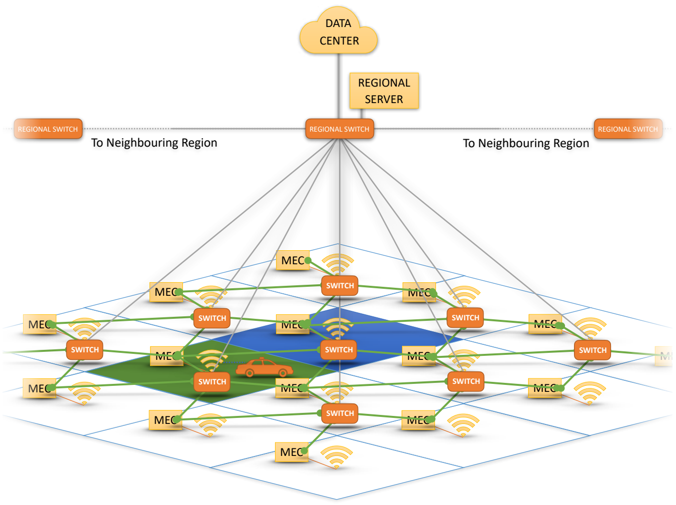


Fig. 1. 3D Orthographic representation example of the proposed architecture with MEC computing servers connected their respective BS.

A. System Model Entities

1) Self-Driving Vehicles: We assume that each *Self-Driving Vehicle* $v \in \mathcal{V}$, requires access to external processing power and storage to provide smart functionalities, such as proactive driving assistance, remote monitoring, security, real-time communication, and entertainment to the passengers. Every vehicle is equipped with a 5G transceiver capable of mm-Wave connectivity and requires an uninterruptible connection with an assigned set of services hosted in the network.

2) Services: A *Service* consists of several VMs or Containers, interconnected in a way that all collectively respond to a request made from a *Self-Driving Vehicle* v, with an end goal to enhance the passenger safety and experience while commuting. As $s \in \mathcal{S}$ and $s \in S_v$, we denote a service, where $\mathcal{S}$ is the set of all network services and S_v a subset of $\mathcal{S}$ assigned to a specific vehicle v. Services can be considered as a software package requires a specific amount of computational resources to operate properly. Regarding their requirements, a service s can be summarized in the terms R_s^{proc}, R_s^{mem} and R_s^{stor}, where R_s^{proc} is the number of cores required for the operation of the service, R_s^{mem} and R_s^{stor} the Gigabytes of memory and storage required respectively. Every service, depending on its functionality, has a *priority index* that defines its Quality-of-Service (QoS) requirements. The QoS requirements are essentially the round-trip latency between any vehicle v and the services s_v assigned to it. It is expressed with the variable R_s^{qos} and is measured in milliseconds. This scheme is agnostic to the type of services hosted and it categorizes them by their QoS requirements as it can be seen in Table I.

TABLE I
SERVICE CATEGORIES

Category	Application Examples
Critical	Auto-breaking, remote control, accident prevention
High Priority	Navigation, Information Services
Low Priority	Multimedia, Entertainment, AR, Cloud Gaming

3) Data-Centers: Typically, the *DCs* host all kinds of services, both user and networking. In this approach, we also consider a unique networking service that is capable of exploiting user location data and modelling user behaviour that the orchestrator can utilize to spatially distribute user services in the network. We use $d \in \mathcal{D}$, where $\mathcal{D}$ is a set of DCs. The latency from the edge of the network to the DC is denoted as D_d^{lat}. The DCs are located in a remote area far from the access network. That suggests a dramatic rise of the round-trip delay, making it impossible to satisfy the QoS of critical and even some high priority services that require instant communication.

4) Multi-Access Edge Computing Servers: In an effort to reduce the round-trip communication delay between the users and their services in DC, we employ MEC servers with limited processing power next to every BS [10]. The MEC servers are denoted with $m \in \mathcal{M}$. We consider that each MEC server has available computational resources to host both user and network services and the BSs equipped with 5G mm-Wave capable equipment directly connected to their respective MEC m. Regarding the computational capacity, we refer to C_m^{proc}, C_m^{mem} and C_m^{stor} to the capacity of every MEC m for cores, Gigabytes of memory and storage apiece. There is a specific utilization threshold that if exceeded the MEC server enters a critical mode and new services can no longer be instantiated. New requests to instantiate or migrate a service will be rejected and automatically redirected to a neighbouring MEC server. This threshold is called $T_m^{critical}$ and it depends on the capacity percentage of each server to prevent unstable behaviour or crashing. The MEC servers of each cell are interconnected with their adjacent cells by local, cell border switches. These connections are used to transfer user services and data at high speeds and extremely low delays during operation. They all have access to DC via a network of multi-mode optical fibres and switches.

5) Regional Servers: The *Regional Servers* are intermediary servers between the DCs and the MEC Servers. They are used to host the SDN controller that is managed by the orchestrator, controls and collects statistics from all local cell border switches. Also, it serves passenger services from vehicles with high velocities, when their velocity is high enough that service migration time is enough to move the services between adjacent cells. Similar to MEC Servers, the computational capacity, we refer to C_r^{proc}, C_r^{mem} and C_r^{stor} to the capacity of every Regional DC $r \in \mathcal{R}$ for cores, Gigabytes of memory and storage each.

6) Network Switches: The network *Switches* are hardware devices that receive, process and forward data between devices within the network. They are denoted with $w \in \mathcal{W}$, where $\mathcal{W}$ is the total number of switches in the network. Every switch needs a specific amount of processing time from receiving to re-transmitting the packet from a different port, we note this delay as D_w^{comp} and measured in milliseconds. As D_n^{port} we name the transmission delay of its ports.

7) Network Links: We think of network *Links* that directly connect the two network entities. We express all links as $l \in \mathcal{L}$, where $\mathcal{L}$ is the total number of links. Every link has a propagation delay noted as D_l^{pd} and measured in microseconds.

B. Connectivity

To have an overview of the entire architecture, it is important to recognize the connections between all network entities discussed in the previous subsection.

As the self-driving vehicles v enter a new cell c while cruising in the city, they connect to the BS of the current cell. The BS is connected with the local MEC server that processes the services of the current cell c, to minimize the round-trip service delay with the vehicle. This offers the opportunity for extremely fast response times that can be proven useful for critical applications. In some circumstances, it could even save a life with a smarter auto-braking assistant. The MEC servers of adjacent cells are connected to the switches that are located in every intersection of the border of each adjacent cell, as represented in Fig. 2. That type of connectivity between the MEC servers and the switches provides one-hop communication to the MEC server of the adjacent cell. This way, services of vehicles that transition between two adjacent cells can be moved fast and reliable. Every switch between the cells is also directly connected to the regional switch that covers the entire region r. In the same region is also located a regional server dedicated to fast users and times where MEC servers are heavily congested.

The complete connectivity plan can be examined in detail in the top view diagram of Fig. 2.

IV. Deep Learning based Mobility Prediction

This section presents the proposed method for predicting the future cell of the users in a personalized manner. By predicting the next cell the user is going to be located shortly, we can migrate the services upfront, before the user handover to the next cell and increase the latency. It is clear that if we want to minimize the perceived latency, the user should be served by the nearest edge [20]. The proposed orchestration algorithm attempts to minimize round-trip latency between the users and their services. All notations required are shown in Table. II.

All vehicles show distinct mobility patterns based on the most significant locations of their users, such as home or work. To make predictions for every vehicle, we have to generate a mobility model that represents the vehicle's mobility pattern from its historical trajectory. New vehicles are handled by a generic model that has generalized the mobility of all vehicles in that region.

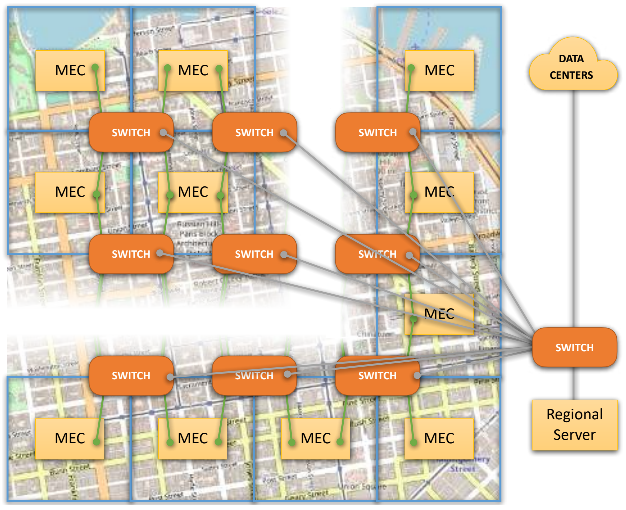


Fig. 2. Network top view layout representation. It is easy to distinct the direct connections between the switches and the MEC servers for fast, 1-hop service migration between neighbouring nodes without limiting the throughput of the back-end links. This layout can be achieved also in arbitrary shaped cell structures by using a switch in every intersection between 3 or more adjacent areas.

TABLE II
NOMENCLATURE

Services	
$\mathcal{S}$	Set of services
R_s^{proc}	Service computing cores requirement
R_s^{mem}	Service memory requirement
R_s^{stor}	Service storage Requirement
R_s^{qos}	Service QoS requirement
Self-Driving Vehicles	
$\mathcal{V}$	Total number of vehicles
S_v	Subset of services assigned to vehicle v
Data-Centers	
$\mathcal{D}$	Set of DCs
D_d^{lat}	Total network edge to DC delay
MEC Servers	
$\mathcal{M}$	Set of MEC Servers
C_m^{proc}	MEC computing cores capacity
C_m^{mem}	MEC memory Capacity
C_m^{stor}	MEC storage capacity
$T_m^{critical}$	MEC critical threshold
Regional Servers	
$\mathcal{R}$	Set of Regional Servers
C_r^{proc}	Regional Server computing cores capacity
C_r^{mem}	Regional Server memory capacity
C_r^{stor}	Regional Server storage capacity
$T_r^{critical}$	Regional Server critical threshold
Network Switches	
$\mathcal{W}$	Set of switches
D_w^{comp}	computational delay of the switch
D_n^{port}	Port transmission delay of the switch
Network Links	
$\mathcal{L}$	Set of links
D_l^{pd}	Link propagation delay

A. Vehicle Trajectory Data Representation

The trajectory of every vehicle can be considered as tuples of coordinates that correspond to a two-dimensional plane. Formally, given a self-driving vehicle v, it's trajectory T_v can be represented by duets of Global Positioning System (GPS) coordinates:

$$T_v = [(x_1, y_1), (x_2, y_2), ..., (x_t, y_t)], \tag{1}$$

where (x_t, y_t) represents the longitude and latitude coordinates at the timeslot t.

Considering the massive amounts of data produced by the vehicles, there are infinite possible data points used to describe each trajectory due to the longitude and latitude coordinates being continuous in space [18]. A commonly used mechanism to overcome such an issue is to convert the continuous space to a finite set of locations or cells to combine adjoining spatial points into one divisible unit. Various practices have been used in the literature such as *Voronoi Polygons* [22], [23] or *Dense Grids* [4], [18], [19]. Any implementation can be used with our solution as long as it assigns a unique id to every cell.

We adopt the *Dense Grid* approach in our work for the following reasons:

- It retains only the global mobility trend of every vehicle we need to extract and removes unnecessary moves.
- It enables the mapping of two-dimensional patterns into one-dimensional sequences.
- Utilizing number sequences instead of two-dimensional matrices enables using proven approaches for sequence classification and prediction.
- Avoiding learning matrices with long dimensions that only contain a trace, helps avoid sparse input for our neural network and prevents low accuracy fitting.
- It notably reduces the amount of data for storage and processing and makes the prediction time possible in microseconds.

First, we divide the city plane into a square grid of c cells, where K represents the length of each side. Starting from left to right, bottom to top, we give every cell a unique identification code to perform the conversion as depicted in Fig. 3.

The resolution of the Dense Grid is a *quid pro quo* between feature retention and information reduction. In general, such trajectories contain information that do not directly influence the prediction accuracy and can be abstracted, like corners or curves. Keeping such features may cause sparsity problem because it requires much greater resolution and less cells now contain useful information. In contrast, lower resolutions tend to keep general information about the mobility.

Considering the above, we group the set T_v from the Equation 1 to a cell sequence form:

$$D_v = [c_1, c_2, ..., c_t], \tag{2}$$

where v is the self-driving vehicle and D_v its trajectory described in *cells* c_t.

B. Convolutional Neural Networks personalized prediction

For the personalized user mobility prediction, we utilize CNNs. They gained great success not only in research but also in everyday life because of their performance and agility. They are mostly used for feature extraction from raw-pixel images and pattern recognition in various dimensional spaces, for audio, image or 3D object representations.

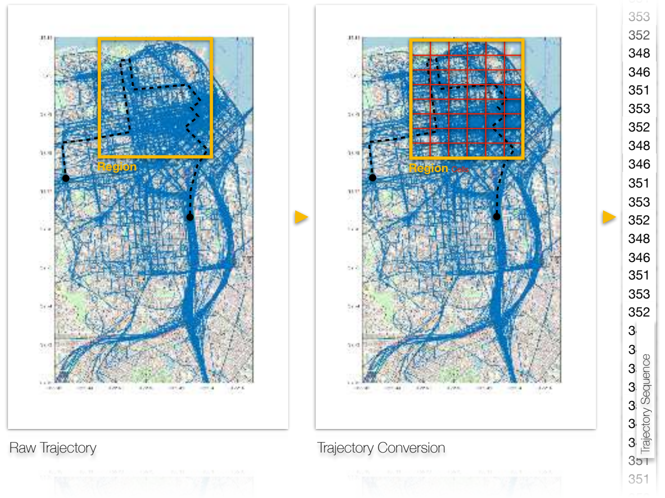


Fig. 3. Illustration of the trajectory conversion.

Similar to the pattern recognition in a one-dimensional problem, such as word recognition in an audio stream, we identify mobility patterns by assigning probabilities to specific sequences from data to the next cell. That way we can make predictions by producing a measurable number of *confidence* for each prediction.

Formulating the problem as a time-slotted model offers a way to describe every transformed trajectory D_v as a uni-variable time-series. Each time-series expresses as a sequence of cells.

An abstract representation of the model is presented below in Fig 4.

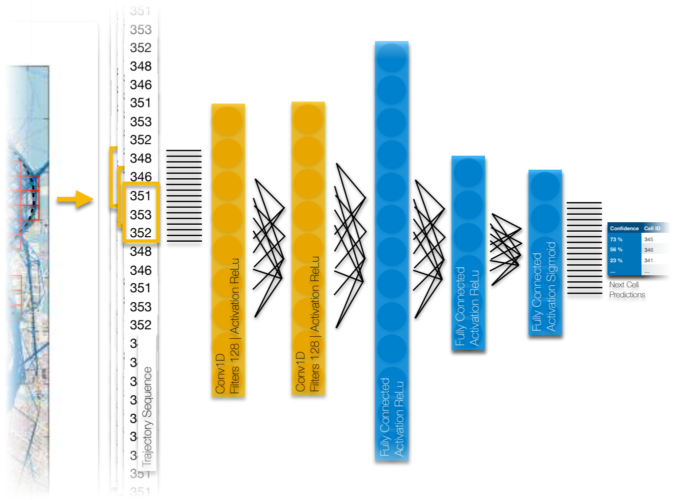


Fig. 4. Sequence mobility prediction Neural Network architecture. The inputs are three sets of the last three positions with a lag factor of one and the output is a vector with all the adjacent cells that are most likely to be the next cell in the trajectory of the vehicle.

At first, we feed the input sequences into the CNN model. The CNN model is a compound of convolution, max-pooling and Fully Connected (FC) layers.

The first layer L_1 convolves D_t to generate a feature map $F_{C \times L^2}$ with C channels.

Each channel can be expressed by the following equation:

$$F_{C\times L^2} = \sigma(\sum_{h=-[k/2]}^{[k/2]} \theta_{c,h} * (t+h)) + b_c), \tag{3}$$

where $\theta_{c,h}$ is the convolution parameter, k is the length of the sliding window, b_c the bias parameter and $\sigma(x)$ the activation function. These layers are typically rectified by a *ReLu* activation function, that can also be represented as $\sigma(x) = max(x, 0)$. The objective of the first layer is to extract the spacial dependency of the cell sequences we used as input.

In the following a max-pooling layer L_2 is used to down-sample the extracted input feature maps and generate high-level features on a larger scale. It reduces the input sequence D_v by keeping only the maximum value between as sliding window. The reduced feature maps are now described as $FC \times L/p \times L/p$, with each element:

$$F_{c,t} = max_0 <= h, F_{c,t*p+h}, \tag{4}$$

where p indicates the pooling size.

The output is then flattened into a vector S and exposed into consecutive FC layers L_3 L_4, that each output element of every layer is calculated as following:

$$e_i = tanh(\sum_j W_{i,j} \times S_j + b_i), \tag{5}$$

where $W_{i,j}$, $W'_{i,j}$ and b_i are the parameters of these layers and need to be learned.

Finally in the output layer L_5, we calculate the probabilities set P, as the weighted sum of the frequently significant visited locations for each (cell) to be the next one that will serve the user v. In particular, for every neuron of the output layer we have:

$$e_i = \phi(\sum_j W_{i,j} \times S_j + b_i), \tag{6}$$

where $\phi(x)$ is the *Sigmoid* activation function $\phi(x) = \frac{1}{1+e^{-z}}$ and $\phi(x) \in (0, 1)$.

C. Confidence & Decisions

We define *Confidence* as an integer that is used for the decision making of the service placement and it shows the level of sureness for a specific prediction. If the output is not accurate enough, the orchestration and placement algorithm has to take that into account to be able to guarantee low latency to as many services as possible.

The output vector of the neural network $O \in (0, 1)$ because of the *Sigmoid* activation function. We set a threshold t_{output} that defines when the term a_o becomes 1, so that $a_o \in \{0, 1\}$.

We sort the output vector O descending and pass it through the following function:

$$Q_{confidence} = \sum_{s=1}^{S} a_o, \tag{7}$$

where $a_o \in \{0, 1\}$ is whether the output accuracy for every service surpassed the confidence threshold $t_{confidence}$ and:

$$Q_{case} = \begin{cases} No\ Confidence,\ if\ Q_{confidence} = 0 \\ Confident,\ if\ Q_{confidence} = 1 \\ Dilemma,\ if\ Q_{confidence} > 1 \end{cases}, \tag{8}$$

The placement actions taken for each specific Q_{case} are shown below:

1) No confidence, Flood Service: When the algorithm does not pose any confidence, if the service is critical we create duplicated in all neighbouring cells and keep only the one that the user uses after the handover.

2) Confident, Migrate Service: If the algorithm poses great confidence in only one cell, we migrate directly the service upfront.

3) Dilemma, Duplicate Service: If the confidence level is high for more than one MEC servers, we create multiple copies of the original service and keep on only the one that the user uses after the handover.

V. Dynamic Service Orchestration with Genetic Algorithms

In the following paragraphs, we tackle the orchestration of user services inside a region, by designing a recursive algorithm that is based on a GA solution of the famous 1/0 Multidimensional Knapsack Problem (MKP).

A. Knapsack Problem Preliminaries

The Knapsack Problem (KP) is an optimization problem. Its goal is to determine which items should be included in a collection, so that the total weight is less or equal to a given limit, given a set of items that each item has weight and a value. It is a known NP-Hard problem, often used for resource allocation, where there is a fixed budget or constraint.

1) 0/1 Knapsack Problem Definition: Given n objects with value v_i and m knapsacks, each with a capacity constraint c_j, maximize the value to satisfy each m constraint. Each of the m constraints have i weights w associated with it.

We want to maximize the term:

$$\sum_{i=1}^{n} x_i v_i, \tag{9}$$

such that:

$$\sum_{i=1}^{n} x_i w_{i,j} \leq c_j, j : 1...m. \tag{10}$$

Here $x_i \in 0, 1$ represents the number of instances of weight i to include in the knapsack j.

2) Multi-dimensional Knapsack Problem Definition: In continuation of the previous definition, the main major difference is that the weight of a knapsack item is given by a D-dimensional vector $\overline{w_i} = (w_{i1}, w_{i2}, ..., w_{iD})$ and the knapsack has a D-dimensional capacity vector $(W_1, W_2, ..., W_D)$. The target is to maximize the sum of the values of the items in the knapsack so that the sum of weights in each dimension d does not exceed W_d. MKP is very computationally intensive and for that reason use will use a GA to approach its solution and avoid NP-hard complexity.

B. Multi-dimensional Knapsack Problem solution with Genetic Algorithms

Solving the aforementioned multidimensional knapsack problem using dynamic programming is difficult. We leverage a solution studied multiple times in literature and it involves the use of a GA [24]. It is based on [24], that uses greedy crossover for the 0/1 knapsack problem and it shares similarities with the work [25] by also using surrogate multipliers. This algorithm, according to this work [24], is able to exceed the greedy estimate and find an optimal solution.

We leverage *Lagrangian multipliers* to augment the utility ratio for the multidimensional knapsack problem. It can be summarized with the following steps:

- For each object and constraint for the given object, the constraint value is multiplied with the corresponding Lagrangian multiplier l_j and the sum of these values is obtained.
- This sum is then divided by the number of constraints.
- The ratio of the profit and the divided sum is obtained and forms the profit-to-weight ratio for the specific object.

In mathematical form, the steps can be summarized in the following equation

$$ratio_i = \frac{v_i}{\frac{\sum_{j=1}^{m} l_j * W_{i,j}}{m}}, \tag{11}$$

where l_j is the j^{th} Lagrangian multiplier and m is the number of constraints. The greedy crossover takes objects from the parents in a non-increasing order of the ratio and constructs one offspring such that it satisfies all constraints.

C. Service Orchestration and Recursive Placement

Based on the future predictions of the CNN, we devise a recurrent algorithm that generates the placement for the next timeslot recursively. This algorithm determines the optimal service placement for all vehicles and their services in a region, by taking into account the predicted vehicle location, the formed confidence, and the resources of the next timeslot.

The algorithm works by generating and assigning a value for every service s according to its latency requirement R_s^{qos}.

1) Initialization Sequence: The sequence begins by attempting a placement in a cell MEC. If there are multiple services expected to be served by the specific MEC according to the CNN predictions and confidence values, it performs a selection with the MKP algorithm to prioritize the placement of services with higher prices, such as critical. After the successful placement of these services, the algorithm proceeds to place the rest services.

2) Recursive Sequence: The recursive part of the algorithm starts by examining the total service latency in different placement scenarios. If the total service latency of the given placement decision and user position is greater than placing the service in the closest DC, it directly migrates the service that DC. If the previous statement is false, it proceeds to investigate if there are any available resources to host the service in question in this MEC. Given that there are available computational resources, the algorithm assigns the service to the specific MEC and returns the function. In case there are no available resources, it sets a flag variable as false and receives a list of all neighbouring cells. If the flag variable is still false, it calls itself pointing at the next neighbouring cell of the first list. When the placement occurs, the function returns. In case that the placement could not be performed in any of the neighbouring cells of the list, the flag variable is still false and can access the next neighbouring cell from the initial call in greater depth. The operation repeats itself until there is a placement with total service latency lower than placing the service on the closest DC. The visual result is a clockwise search for resources, each time in a longer distance from the desired MEC until the service is placed. A graphical example of the recursive operation can be seen in Fig. 5 below.

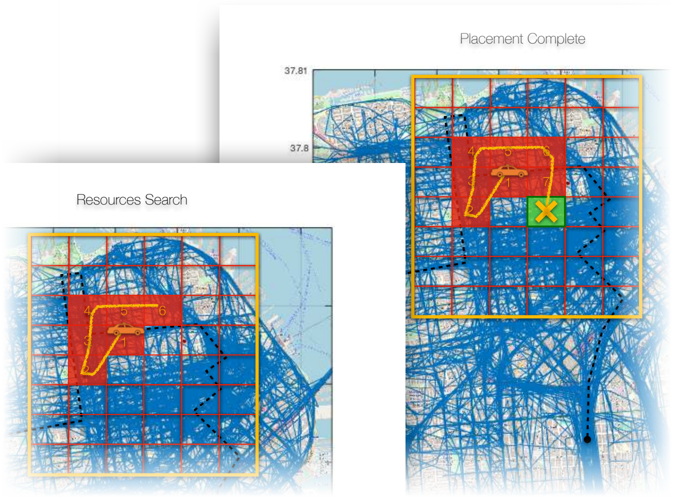


Fig. 5. On the left side of the figure, we can see a snapshot of the algorithm during the recursive search for resources, while on the right side we can see the search completed.

VI. Simulation Results & Evaluation

In this section, we conduct a simulation study to evaluate the latency and QoS for the critical services of the proposed algorithm. The performance evaluation takes place in six distinct traffic scenarios in an urban environment to verify the theoretical results in a more realistic manner.

A. Dataset

For the mobility of the vehicles, we leveraged real data from taxis roaming the San Francisco Bay Area [26]. This dataset contains the GPS coordinates and timestamps of 536 taxis, collected for over 30 days.

The motivation and criteria for the dataset selection were the following:

- The dataset should contain GPS coordinates and timestamps to derive the vehicle trajectories and extract features such as speed and direction.
- It should include enough data to train our model appropriately. More data can help the algorithm generalize better the movement of the vehicles in the city.

- The vehicle trajectories should present hot-spots to emulate the different points of interest individual users have in their daily lives, such as homes, workplaces, malls, etc. In our case, the hot-spots are tourist attractions and taxi pickup areas.
- It should cover the entire investigation area at least 2 times per day to emulate a typical work-home mobility sequence with distinct patterns.

Real-life user location data were not available at the time of this research, mainly due to privacy and security reasons. The dataset can be found available online in the community of Crawdad.

B. *Simulation Setup*

The simulation of both physical and virtualized network entities was simulated with *Python* language. For the construction of the CNN model, we used *TensorFlow* and the high level *Keras API* [27]. We utilized the *pyeasyga* library for the implementation of the Genetic Algorithm solution of the 0/1 Multi-dimensional Knapsack Problem.

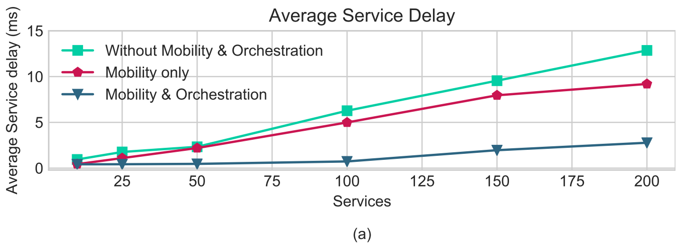
For the simulation results presented in the following subsection, we performed simulation scenarios from groups of $\mathcal{V} = \{5, 25, 50, 100, 150, 200\}$ vehicle trajectories. We presume that each vehicle v carries one passenger and moves at a variable speed, never exceeding 65 kilometers per hour. We simulated the mobility of every vehicle for 65 timeslots, in all studied traffic scenarios. The $2/3$ of the total length of all traces was used for training and mobility pattern extraction. We used *Dense Grids* to divide the area into small cells. This simulation covers one region $r = 1$ with a *Dense Grid* of $K = 16$ slices that correspond to $3km \times 3km$, approximately 0.1875 kilometers per side of every cell and a total of $c = 256$ cells. The physical network is composed of one remote DC $D = 1$ with overall latency $D_1^{lat} = 22$ milliseconds as well as a Regional Server with $C_1^{proc} = 28$ cores, $C_1^{mem} = 1.5$ TB RAM and $C_1^{stor} = 2$ TB storage. In addition, all MEC Servers $m \in \mathcal{M}$ are designed with $C_m^{proc} = 4$ cores, $C_m^{mem} = 8$ GB RAM, $C_m^{stor} = 32$ GB storage with a co-located BS. The *Service Categories* include:

- Critical Services such as Object Recognition, with the requirement of $R_s^{proc} = \{2, 3\}$ cores, $R_s^{mem} = \{1, 2\}$ GBs RAM, $R_s^{stor} = \{1, 2\}$ GB Storage and $R_s^{qos} = 1$ ms strict limit.
- High Priority Services, such as AR Navigation, with the requirement of $R_s^{proc} = \{1, 2\}$ cores, $R_s^{mem} = \{2, 3, 4\}$ GBs RAM, $R_s^{stor} = \{2, 3, 4\}$ GB Storage and $R_s^{qos} = 10$ ms limit.
- Low Priority Services, such as Caching Database or Cloud Gaming, with the requirement of $R_s^{proc} = \{2, 3, 4\}$ cores, $R_s^{mem} = \{2, 3, 4\}$ GBs RAM, $R_s^{stor} = \{4, 5, 6, 7, 8\}$ GB Storage and $R_s^{qos} = 20$ ms limit.

Every vehicle has at least one Critical Service that we consider as mandatory for the self-driving capability and safety of the passengers. The rest of the Critical, High or Low Priority services are generated and assigned to the vehicles in a random manner with Gaussian distribution.

C. *Performance Evaluation*

In our simulations, we examine the performance of the proposed algorithm by comparing both the average service latency from the vehicles and the average number of QoS violations that occur during every mobility scenario.



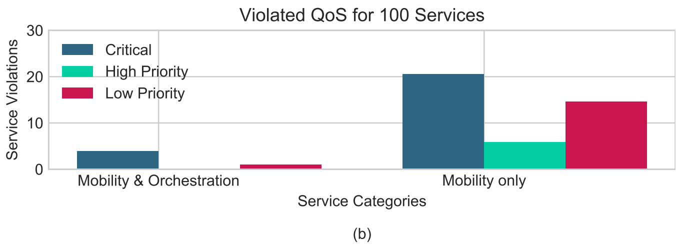


Fig. 6. (a) average service latency between the vehicles and their services. (b) average QoS violations that took place in a scenario with 100 active services.

As we can observe in Fig. 6, increasing the number of vehicles in the network increases at the same time the average service latency due to insufficient MEC resources. Specifically, we can observe that our algorithm in Fig. 6a, outperforms the option without mobility and orchestration, by offering considerably lower average service latency, especially for a high number of vehicles. Moreover, according to Fig. 6b, on average only 3.9 of the critical services exceeded their latency QoS compared to the 20.2, which showcases that our proposed GA based MKP orchestration can tightly fit more services per cell in average without having to reject the requests. As demonstrated, with proper service distribution in the same space, there is a noticeable improvement in the average service latency, since our proposed DL algorithm migrates proactively the services in the network one step ahead of the vehicle. It is evident that our proposed orchestration algorithm can keep more services close to the vehicles, even when the nearest MEC is under full load. This is possible by recursively searching for available resources in the adjacent cells that are one network hop away.

In Fig. 7, we examine a simulation snapshot with 100 vehicles that shows the number of occupied CPU cores in all cells of a region. We can clearly observe a high traffic spot in the middle of the grid due to the vast lack of unoccupied CPU cores in that area. As we can see, spreading the services with lower priority across the network can help to avoid overload. It also prevents the congestion of the back-end links by performing most of the computation at the edge of the network, near the vehicles.

Next, with Fig. 8 we measure the spread of computational load in the region under investigation. We support that the variance with our implementation remains lower in the case of 100 services, even with an ongoing congestion in this region.

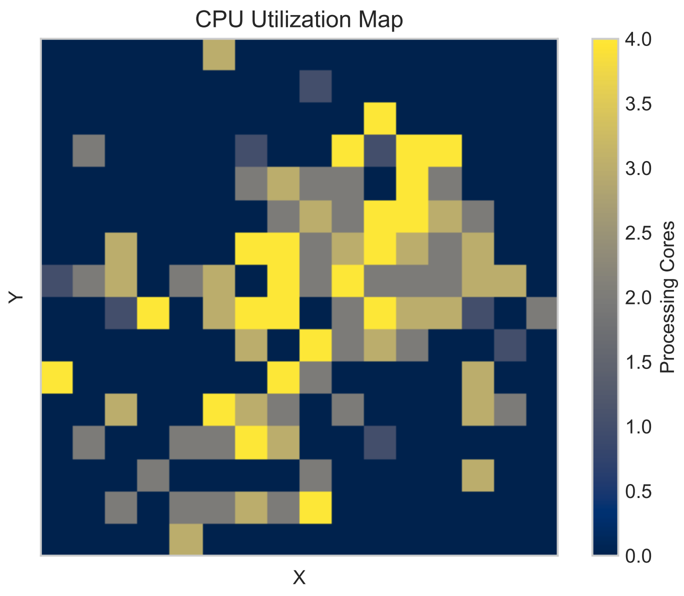


Fig. 7. CPU Utilization snapshot from a scenario with 100 active services, that corresponds to the QoS Violation presented in the previous figure.

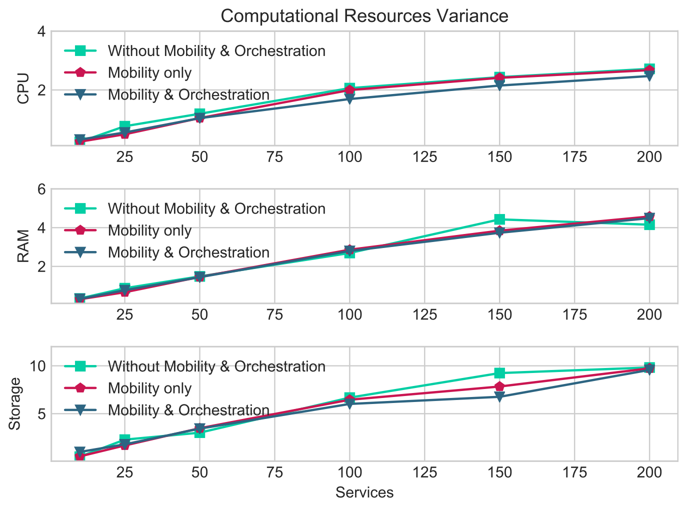


Fig. 8. Variance of computational resources with 100 active services.

The results from Fig. 8 infer to evenly spread computational and network load throughout the entire examined region, sharing and retaining additional room to support new users with critical services. This form of spatial load balancing keeps the latency uniform throughout the network and thus, offering consistent user Quality of Experience while commuting.

Finally, in Fig. 9 we present the results of our study for the optimal computational resources required to cover all simulated traffic configurations. Each bar in the figure shows the minimum number of CPU cores needed on average to avoid migrating services to neighbouring cells. Keeping services to the nearest MEC, while never exceeding their capacity, can deliver service with lower latency. Even in the highest traffic scenario of 200 vehicles, our proposed implementation can retain minimal latency when the available MEC capacity is equipped with 12 cores.

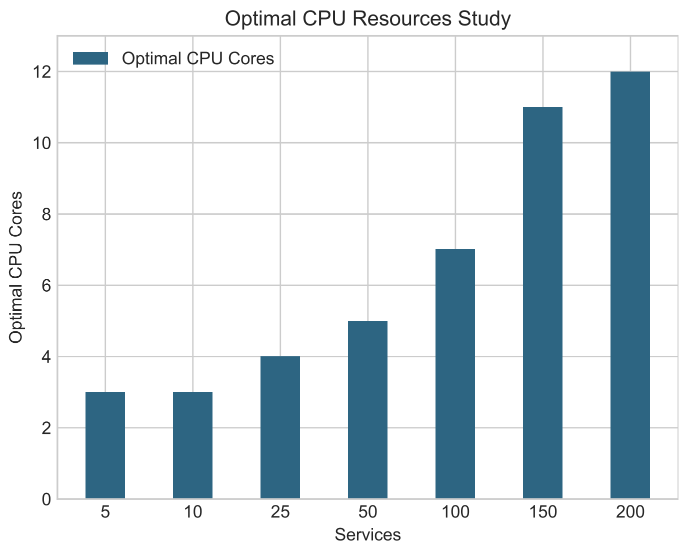


Fig. 9. Optimal CPU Cores for every traffic scenario studied.

VII. Conclusions & Future Work

It is well studied in the literature that using location data to place the services at the edge of the network reduces the latency of the services. However, we go a step further to predict the next move of the vehicle and proactively rearrange the services according to their requirements. This kind of awareness enables a whole new ecosystem of innovative applications, such as a life saving proactive collision warning. Motivated by the potential benefits, the contribution of this paper focuses on employing a CNN to exploit the past location data of vehicles to estimate their next position and developing a GA based recurrent orchestration algorithm to organize services at the edge of the network. We have evaluated the performance of the proposed algorithm by simulating six traffic scenarios with real location data from taxis in San Francisco Bay Area. The simulation results show that our algorithm was able to reduce the average rejection rate of critical services from 20.2 to 3.9. We proved that the GA based recursive service orchestration algorithm we designed was able to search for available resources close to the vehicles and prioritize the placement in a scenario with congestion. The results also supported that our algorithm performs a form of spatial load balancing that keeps the latency uniform everywhere in the network. Finally, we conducted a simulation study to identify the optimal number of CPU cores needed to satisfy each traffic scenario.

As future work, we intend to use Deep Reinforcement Learning to substitute the MKP in the orchestration part of our work. By approximating the optimal placement, we will decrease further the time required for orchestration.

References

[1] A. Mahmood, B. Butler, and B. Jennings, *Potential of Augmented Reality for Intelligent Transportation Systems*. Cham: Springer International Publishing, 2018, pp. 1–7. [Online]. Available: https://doi.org/10.1007/978-3-319-08234-9_274-1

[2] A. Alhilal, T. Braud, and P. Hui, "Distributed vehicular computing at the dawn of 5g: a survey," 01 2020.

[3] A. Noulas, S. Scellato, N. Lathia, and C. Mascolo, “Mining user mobility features for next place prediction in location-based services,” in *2012 IEEE 12th International Conference on Data Mining*, 2012, pp. 1038–1043.
[4] L. M. Chen, L. and G. Chen, “A system for destination and future route prediction based on trajectory mining,” *Pervasive and Mobile Computing*, vol. 6, no. 6, pp. 657–676, 2010.
[5] M. C. González, C. A. Hidalgo, and A.-L. Barabási, “Understanding individual human mobility patterns,” *Nature*, vol. 453, no. 7196, pp. 779–782, Jun 2008. [Online]. Available: https://doi.org/10.1038/nature06958
[6] C. Wang, L. Ma, R. Li, T. S. Durrani, and H. Zhang, “Exploring trajectory prediction through machine learning methods,” *IEEE Access*, vol. 7, pp. 101 441–101 452, 2019.
[7] K. Zhao, S. Tarkoma, S. Liu, and H. Vo, “Urban human mobility data mining: An overview,” in *2016 IEEE International Conference on Big Data (Big Data)*, 2016, pp. 1911–1920.
[8] C. Yao, J. Guo, and C. Yang, “Achieving high throughput with predictive resource allocation,” in *2016 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, 2016, pp. 768–772.
[9] F. Giust, V. Sciancalepore, D. Sabella, M. C. Filippou, S. Mangiante, W. Featherstone, and D. Munaretto, “Multi-access edge computing: The driver behind the wheel of 5g-connected cars,” 2018.
[10] “Multi-access edge computing (mec); study on mec support for v2x use cases,” European Telecommunications Standards Institute, Sophia Antipolis Cedex, FRANCE, Standard, Mar. 2019.
[11] M. F. Bari, S. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, “Orchestrating virtualized network functions,” *IEEE Transactions on Network and Service Management*, vol. 13, 05 2016.
[12] A. Machen, S. Wang, K. Leung, B. Ko, and T. Salonidis, “Live service migration in mobile edge clouds,” *IEEE Wireless Communications*, vol. 25, 02 2018.
[13] I. Farris, T. Taleb, H. Flinck, and A. Iera, “Providing ultra-short latency to user-centric 5g applications at the mobile network edge,” *Transactions on Emerging Telecommunications Technologies*, 03 2017.
[14] I. Farris, T. Taleb, M. Bagaa, and H. Flinck, “Optimizing service replication for mobile delay-sensitive applications in 5g edge network,” 05 2017.
[15] A. Nadembega, T. Taleb, and A. Senhaji Hafid, “A destination prediction model based on historical data, contextual knowledge and spatial conceptual maps,” 06 2012, pp. 1416–1420.
[16] G. Xu, S. Gao, M. Daneshmand, C. Wang, and Y. Liu, “A survey for mobility big data analytics for geolocation prediction,” *IEEE Wireless Communications*, vol. PP, pp. 2–10, 10 2016.
[17] R. Cziva, C. Anagnostopoulos, and D. Pezaros, “Dynamic, latency-optimal vnf placement at the network edge,” 04 2018, pp. 693–701.
[18] S. Choi, H. Yeo, and J. Kim, “Network-wide vehicle trajectory prediction in urban traffic networks using deep learning,” *Transportation Research Record: Journal of the Transportation Research Board*, p. 036119811879473, 09 2018.
[19] J. Lv, Q. Li, Q. Sun, and X. Wang, “T-conv: A convolutional neural network for multi-scale taxi trajectory prediction,” 01 2018, pp. 82–89.
[20] T. Ouyang, Z. Zhou, and X. Chen, “Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing,” 09 2018.
[21] X. Yu, M. Guan, M. Liao, and X. Fan, “Pre-migration of vehicle to network services based on priority in mobile edge computing,” *IEEE Access*, vol. 7, pp. 3722–3730, 2019.
[22] F. Alhasoun, “City scale next place prediction from sparse data through similar strangers,” 2017.
[23] X. Ouyang, C. Zhang, P. Zhou, and H. Jiang, “Deepspace: An online deep learning framework for mobile big data to understand human mobility patterns,” 10 2016.
[24] S. Shah, “Genetic algorithm for the 0/1 multidimensional knapsack problem,” 2019.
[25] A. Guler, U. Nuriyev, and M. Berberler, “A genetic algorithm to solve the multidimensional knapsack problem,” *Mathematical and Computational Applications*, vol. 18, pp. 486–494, 12 2013.
[26] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser, “CRAWDAD dataset epfl/mobility (v. 2009-02-24),” Downloaded from https://crawdad.org/epfl/mobility/20090224/cab, Feb. 2009, traceset: cab.
[27] F. Chollet *et al.*, “Keras,” https://keras.io, 2015.

Anestis Dalgkitsis received the Diploma degree (5 years) in Informatics & Telecommunications Engineering from University of Western Macedonia (UoWM), Kozani, Greece in 2018. Since October 2018, he works as a Research Engineer at Iquadrat and is involved in EU-funded research projects, while pursuing his Ph.D. degree at the Technical University of Catalonia (UPC), Barcelona, Spain His main research interest are Network Function Virtualization Management and Orchestration, with Machine Learning algorithms and techniques.

Prodromos-Vasileios Mekikis Dr. Prodromos-Vasileios Mekikis is a Senior Researcher at Iquadrat Informatica S.L since January 2018. He has received his PhD degree from the Department of Signal theory and Communications of the Technical University of Catalonia (UPC), Spain, in 2017. Also, he holds an Electrical and Computer engineering degree (2010) and a M.Sc. in System-on-Chip design (2012) from Aristotle University of Thessaloniki and Royal Institute of Technology (KTH), respectively. Prodromos won the first prize in the IEEE ComSoc Student competition in 2015. Currently, he is actively involved in EU-funded research projects, while his main research interests include Network Function Virtualisation, Wireless Energy Harvesting, and Connectivity in Aerial and Internet of Things networks.

Angelos Antonopoulos received the Ph.D. degree from UPC in 2012. He is currently a Senior Researcher at CTTC/CERCA. He has authored over 100 publications on various topics, including 5G wireless communications, content caching and dissemination, energy efficient network planning and network economics. He currently serves as an Editor for IEEE Access, IEEE Networking Letters, Computer Networks (Elsevier) and Inventions (MDPI). He has been nominated as Exemplary Reviewer for the IEEE Communications Letters, Reviewer of the Month for the IEEE Access and Outstanding Reviewer for Sensors. He has received the Best Paper Award at IEEE GLOBECOM 2014, the Best Demo Award at IEEE CAMAD 2014, the first prize in the IEEE ComSoc Student Competition (as a Mentor), and the EURACON Best Student Paper Award at EuCNC 2016. He is an IEEE Senior Member since 2015.

Christos Verikoukis received the Ph.D. degree from Technical University of Catalonia (UPC), Barcelona, Spain, in 2000. He is currently the Head of the SMARTECH Department, CTTC, Barcelona, Spain, and an Adjunct Professor with the University of Barcelona (UB), Barcelona, Spain. He has authored 133 journal papers and over 200 conference papers. He is also a coauthor of three books, 14 chapters in other books, and two patents. He has participated in more than 40 competitive projects, and has served as a principal investigator of national projects in Greece and Spain. He is currently the IEEE ComSoc EMEA Director and member-at-large of IEEE ComSoc GITC.