Dynamic Service Entity Placement for Latency Sensitive Applications in Transportation Systems

Yangming Zhao, Xin Liu, Lai Tu, Chen Tian and Chunming Qiao, *Fellow, IEEE*

Abstract—With the development of applications on end devices, such as cell phones and tablets, more and more passengers would like to have entertainment on these end devices when they are cruising on vehicles. Due to the limited computation ability of the end devices, some of these applications have back-end components on the edge clouds, which are realized by Service Entities (SEs). In this work, we propose a system named DSEP to Dynamically determine the SE Placement, such that the maximum latency experienced by the passengers can be minimized. To this end, we first train two sequential neural networks to predict the position of each individual vehicle, and propose an efficient algorithm based on optimization relaxation and Lagrange decomposition to determine the SE placement. Through extensive real-data driven simulations, we find that with the two sequential neural networks proposed in this paper, there are less than 1% errors on estimating where the passengers will access the edge cloud system. When the computation resources in the edge cloud are limited, DSEP can reduce the response latency by up to 43% compared with the nearest placement scheme. Even averaging the performance improvement over all simulation settings, DSEP can reduce the response latency by 16%.

Index Terms—Transportation systems, vehicle position forecasting, service entity placement

1 Introduction

With the development of end devices, such as cell phone, tablet, etc., people can run various of applications on them. With these applications, people can watch movies, listen music and play games when they are traveling on vehicle as passengers. Some of these applications are computation intensive, and hence, they consist of front-end components running on the end devices, and back-end components running on the Cloud [1, 2], where the Service Entities (SEs) are hosted to provide additional computation capabilities.

Running back-end components on remote cloud introduces large communication overhead, which hurts the performance of delay sensitive applications, such Virtual Reality (VR) and Augmented Reality (AR). To solve this problem, some of the researchers proposed the concept of Mobile Edge Cloud (MEC). The key idea of MEC is to move computation closer to users. In MEC, small servers or data-centers that can host cloud applications are distributed across the network and connected directly to Access Points (APs), such as cellular base stations, at the network edge. With this method, the users can put their back-end components or outsource their applications to the edge clouds, such that the communication overhead can be reduced, thus improving the user experience.

- *Corresponding author: Lai Tu (tulai@hust.edu.cn)*
- *Yangming Zhao, Xin Liu and Chunming Qiao are with Department of Computer Science and Engineering, Univerisity at Buffalo, the State University of New York.*
- *Lai Tu is with the School of Electronic and Information and Communications, Huazhong University of Science and Technology.*
- *Chen Tian is with State Key Laboratory for Novel Software Technology, Nanjing University.*

The idea of distributing cloud servers at the network edge is also leveraged by cloudlets [3], edge computing [4] and fog computing [5], etc.. In all these techniques, edge servers are collaborating with each other to provide services to the end users. A significant characteristic of MECs is that the edge clouds are not as powerful as conventional clouds, and hence they cannot provide enough dedicated computation resources to the SEs. The number of SEs placed on the same edge cloud may greatly impact the computation latency. On the other hand, always pursuing the load balance to reduce the computation latency may increase the communication overhead. Accordingly, how to place the SEs among the edge clouds is a big challenge in the edge cloud systems.

Another characteristic of MEC is to support the user mobility. When a user moves from the area covered by an AP to another, the edge cloud hosting his/her SE should be corresponding changed, i.e. the SE should be migrated from one edge cloud to another. Previous works mainly focus on 1-D movement [6], which is not the case when the users are cruising in transportation systems as passengers. In addition, the vehicles are moving much faster than the conventional walking users. Accordingly, we should predict the vehicle position in the near future to guide the SE migration, rather than starting the SE migration until the users reach the border of the area covered by their current APs. This brings more challenges to determine the SE placement and migration for the passengers in transportation system.

In this work, we design a system named DSEP to Dynamically determine the SE Placement for latency sensitive applications in transportation systems when the passengers are cruising on vehicles. The objective is to minimize the maximum response latency experienced by arbitrary passengers, which can provide the worst case performance guarantee

to the entire system. To pursue this objective, we first train two sequential neural networks; the first one is a revised convolutional neural network hosted by the remote powerful cloud to forecast the traffic velocity, while the second one is a traditional full connected neural network hosted by each vehicle to estimate the velocity of itself in the near future. Combined with the vehicle route information, we can predict the position of each vehicle. Based on this vehicle position prediction, a joint optimization model is formulated to calculate the SE placement. Since the SE placement is NP-hard and difficult to solve in a timely manner, an efficient algorithm based on relaxation and Lagrange decomposition is proposed. Through extensive real-data driven simulations, we find that for the passengers on more than 99% of the vehicles, DSEP can estimate exactly the APs from which they will access the edge cloud system. For the latency issue, averaging over all the simulation settings, DSEP can reduce the maximum response latency by 16% compared with the nearest placement scheme. In the scenario where the edge clouds have limited computation resources, DSEP can reduce the maximum response latency by up to 43%.

The main technique contributions of our work can be summarized as follows:

- A system named DSEP to dynamically determine the SE placement for latency sensitive application in transportation systems (Section 3)
- Two sequential neural networks to predict the vehicle position (Section 4)
- An efficient algorithm to calculate the SE placement based on the vehicle position prediction (Section 5)
- Extensive real-data driven simulations to show the effectiveness of DSEP (Section 6)

2 Related Work

There are a number of related works on vehicle position forecast and SE placement. We review the most closely related ones.

Vehicle position forecast: Traffic forecasting in transportation systems is a topic that has been studied for several decades. Most of the works on traffic forecasting is to predict the traffic congestion [7]–[9] or traffic flow [10]–[12], i.e. the number of vehicles passing through a given road segment during a unit of time, rather than the exact individual vehicle position which is required in our work. With the traffic congestion or flow information, it is difficult to get the vehicle position. However, if we know the position of every vehicle, we can forecast the traffic congestion and flow condition in the near future. Another category of work related to our work is the traffic speed/velocity forecasting [13, 14]. These works focused on the group velocity of vehicles on the road, rather than the velocity of each individual vehicle as in our work.

SE placement: It can be treated as job scheduling in an edge cloud system to determine the SE placement. In this area, there are a lot of existing works. OnDisc [15] is a system to dispatch and schedule the jobs in edge clouds and minimize the weighted response time. The objective is similar to our work, but it does not allow the SE migration among edge clouds. ITEM [16] studies how to allocate the SEs among the edge clouds such that a series of costs, such as activation cost, placement cost, etc., can be minimized. [17] proposes a system to allocate the resources in edge cloud systems without the user mobility information. However, this is not suitable to the transportation system as we can accurately forecast the vehicle position in the near future. In addition, [18] and [19] focus on the service migration in edge cloud systems. Both of them are based on stochastic model, and loss the optimization space when the user movement can be forecast as in transportation systems.

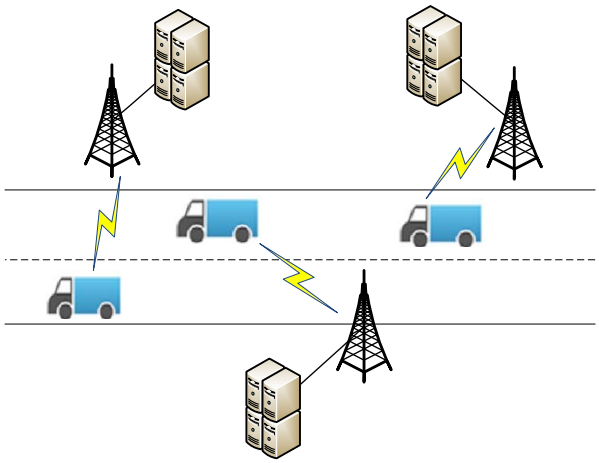


Fig. 1. System Model.

3 Problem Definition

In this section, we define the problem we are to study in this work. At first, we present the system model investigated in Section 3.1. Then, we discuss the latency cost we should consider in Section 3.2. At last, we state the problems to solve in our work and present an overview of DSEP in Section 3.3.

3.1 System Model

We consider a metropolitan-area transportation system. In this system, there are a set of edge clouds dispersed along the road. Each edge cloud is accompanied by an Access Point (AP), which can be a base station and allows the user to connect to the platform.

Along the road, there are sensors installed to measure the vehicle velocity and count the number of vehicles passing through. These data will be collected and sent to a remote cloud for further analysis. The analysis results can be pulled by the vehicles on the road. In addition, each vehicle can measure the velocity of other vehicles on its road through the Vehicle-to-Vehicle (V2V) communication. Furthermore, we also assume that the route of each vehicle is fixed in the near future. The route of vehicles may change due to some unpredictable factors, such as traffic congestion, however, the route in the near future, such as 5 to 10 minutes, can be treated as fixed.

On the vehicles, passengers are running applications like Virtual Reality (VR) and Augmented Reality (AR). Since these applications are resource-hungry and delay-sensitive, they should outsource their back-end components to the edge cloud system. To host these back-end components, Service Entities (SE) are set up on the edge cloud to record the users personal data and the processing logics on the data, take care of the user state and computation-intensive tasks such as scene rendering, object recognition and tracking. For simplicity, we say an SE is placed on an AP instead of it is placed on the edge cloud accompanied by the AP hereafter.

Passengers (in fact, their end devices) always connect to the nearest AP, and hence, we assume all the passengers on the same vehicle are connecting to the same AP. However, the SEs for their applications may not be placed on the AP they are connecting to. For example, when the vehicle is cruising on the road, the passengers may connect to an AP far from the one that is originally hosting their SEs. In this case, there will be larger latency to transmit the data. To ensure the high Quality of Services (QoS), we should dynamically migrate the SEs among the edge clouds. In addition, this SE migration should be done in advance in order to keep the continuous services, rather than start migration when an application suffers from high latency.

The resources in the edge clouds are virtualized using container-based technologies, and hence they can be allocated and shared flexibly. When a container hosts multiple SEs, the docker engine schedules the resources to execute these SEs based on round-robin. Based on these facts, a remote powerful cloud centrally plans the SE placement such that the maximum latency experienced by arbitrary passenger can be minimized.

There are two terms, *vehicle velocity* and *traffic velocity*, should be defined. Vehicle velocity is the velocity of an individual vehicle, while traffic velocity is the group velocity of all the vehicles on a given road segment. The traffic velocity can be calculated as the average vehicle velocity on such road segment. Apparently, the vehicle velocity is tightly coupled with the traffic velocity.

3.2 Latency Models

To allocate back-end component on edge clouds, an application would suffer from two types of latency: data transmission latency and computation latency.

Transmission latency: When the SE of an application is not placed on the edge cloud accompanying with the AP that the corresponding passenger connects to, the application data should be delivered among APs, which incurs transmission latency. The transmission latency increases with the distance on the topology between the AP from which the passenger accesses the edge cloud system and the AP on which the SE of his/her application is placed [20]. Say d_{ij} is the distance, i.e. hop number, between the AP i and AP j, a_i^u is a binary parameter to denote if passenger u connects to AP i, and $y_i^u \in \{0, 1\}$ denotes if the SE for passenger u's application is placed at the edge cloud accompanying with AP j, the transmission delay for passenger u's application is

$$T_u = \sum_{i,j} a_i^u d_{ij} y_j^u \tag{1}$$

Computation latency: Since the edge clouds are not intentionally designed for large-scale resource multiplexing, and performance isolation is typically difficult with light-weight virtualization, the SEs allocated on the same edge cloud should compete for the computation resources [16]. When multiple SEs are placed on the same CPU core, they will be scheduled based on round-robin, and thus, the stretch on execution time. If there are K SEs allocated on a specific CPU core, the execution time can be modeled as $\alpha K + \beta$, where α and β are CPU specific parameters. Smaller α indicates a more powerful edge cloud. If all the SEs in an edge cloud are properly distributed among all the CPU cores, the computation latency on the edge cloud connecting with AP i can be formulated as

$$C_i = \alpha_i m_i + \beta_i \tag{2}$$

where α_i and β_i are edge cloud specific parameters, and m_i is the number of SEs placed on edge cloud i.

It is worth noting that in DSEP, the SEs would migrate among edge clouds when the vehicles are cruising on the road. Accordingly, there should be migration delay experienced by the passengers. However, in DSEP, this migration is executed in advance based on the vehicle position forecasting. Therefore, the SE migration latency experienced by the passengers is only the time to invoke the SE instance on the edge cloud with container technology. Such latency is usually hundreds of microsecond, and hence can be ignored.

3.3 Problem Statement and Nutshell of DSEP

In DSEP, the time horizon is divided into time slots. At any time slot, DSEP should determine the placement of SE for every application in the next time slot such that the maximum application latency experienced by arbitrary passenger can be minimized. To achieve this goal, DSEP should first forecast the vehicle positions in the next time slot. There is a simple way to forecast the vehicle position. Say the vehicle position at time slot s is x_s and the velocity is v_s, then we simply forecast the vehicle position in next time slot as $x_{s+1} = x_s + v_s \Delta t$. This method works in small cities with light traffic load and few traffic lights. However, in a metropolitan city, such as New York city, this method does not work, since there are plenty of blocks, and the vehicle velocity is greatly impacted by other vehicles and traffic lights on the road.

To solve this problem, DSEP divides the entire area covered by the edge cloud system into a grid of squares, and then adopts two sequential Neural Networks (NN) to forecast the vehicle velocity in each square. The first NN forecasts the traffic velocity in every square based on the traffic and environment conditions in the nearby squares. This NN is hosted by a remote powerful cloud since it needs the traffic information of the entire area. In addition, it is a large computation task to forecast the traffic velocity of all the squares.

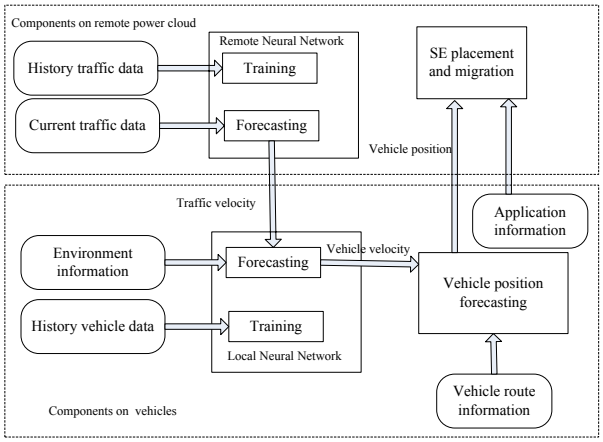


Fig. 2. Overview of DSEP.

Based on the traffic velocity prediction derived by the NN hosted remote powerful cloud, every vehicle maintains a local NN to forecast its vehicle velocity. Such NN should be maintained by each vehicle itself since the vehicle velocity depends on many individual information, such as vehicle type, driver's behavior, etc.. In addition, this NN only needs to predict the vehicle velocity in a very few number of squares along its route. Accordingly, it can be done by a device with very few computation resources, such as on the GPS device carried by the vehicle.

With the route and velocity forecast of each vehicle, the position of each vehicle in the next time slot can be estimated. Then, a vehicle will push following information to the remote powerful cloud: 1) its position estimation in the next time slot; 2) the SE identifications of all its passenger's applications. When collecting all these information from vehicles in the area served by the edge cloud system, the remote powerful cloud calculates the new SE placement, and starts the SE migration in order to provide low latency services.

According to above discussions, the workflow of DSEP can be summarized in Fig. 2. With the history traffic data, the remote powerful cloud trains an NN, which will be used to forecast the traffic velocity based on the latest traffic information. Meanwhile, each vehicle trains a local NN that can predict its vehicle velocity according to the traffic velocity and environment information. Based on the traffic velocity derived by the remote powerful cloud and the environment information sensed by the vehicle itself or through the V2V communication, the local NN hosted by each vehicle can estimate its own velocity in the near future. Combining with its route information, it can estimate the vehicle position in the next time slot. This position information will be pushed to the remote powerful cloud with the application information. Then, the remote powerful cloud optimizes the SE placement for the next time slot and invokes the SE migration procedure.

In the following two sections, we will discuss how to forecast the vehicle position and how to optimize the SE placement in detail, respectively.

4 Vehicle Position Prediction

As we discussed in last section, there are two main parts in DSEP: vehicle position prediction and SE placement. At first, we present how to predict the vehicle position in this section.

In DSEP, two sequential Neural Network (NN) is leveraged to predict the vehicle position. The first one is a revised Convolutional Neural Network (CNN) hosted by the remote powerful cloud to predict the traffic velocity, we call it *Remote Neural Network (RNN)*, while the second one is a conventional full mesh neural network hosted by each vehicle itself, we call it *Local Neural Network (LNN)*. In the following, we present how to design these two NNs in detail.

4.1 Neural Network on Remote Powerful Cloud

Structure of Remote Neural Network: On the remote powerful cloud, a revised CNN as shown in Fig. 3 is hosted as the RNN to predict the traffic velocity. It should be noted that the structure of the RNN is different from the conventional CNN.

In RNN, there are two categories of information that should be input into the convolutional layer, the traffic load information and the traffic velocity information. Different from conventional CNN in which the convolutional kernels (a.k.a filter or feature detector) are used to deal with all the input data, the traffic load information and the traffic velocity information have their own convolutional kernels, respectively. We make such change due to two reasons: 1) the kernels are used to distill the characteristics of the input data, and different kernels should catch different characteristics of the input data. The traffic load and the traffic velocity are two categories of information and hence have their specific characteristics; hereby we train different convolutional kernels for them; 2) since each convolutional kernel does not connect to all the input data, the computation complexity to train the RNN can be reduced.

The other difference between conventional CNN and the RNN is that in the full connected layer, more data, such as the weather, the time and some identifier to indicate if there is an event nearby, etc., are input into the RNN. We do not input these information into the convolutional layer because they are independent to each other, and hence, we do not need kernels to distill the interrelationship among them. Again, this is to reduce the computation complexity of training the RNN.

Similar to most of the conventional CNN, in the sub-sampling layer (a.k.a. spatial pooling step), RNN adopts the function *MAX* to reduce the dimensionality of each feature map but retain the most important information. This is only a choice based on experience. We have tried multiple functions, such and MIN, AVERAGE etc., and found that the MAX function can derive the best results.

Input of Remote Neural Network: To capture the traffic information (both of the traffic load and the traffic velocity) of each time slot, we first divide the entire area covered by the edge cloud system into a grid of squares as in Fig. 4(a) (the size of the squares will be discussed based on simulations in

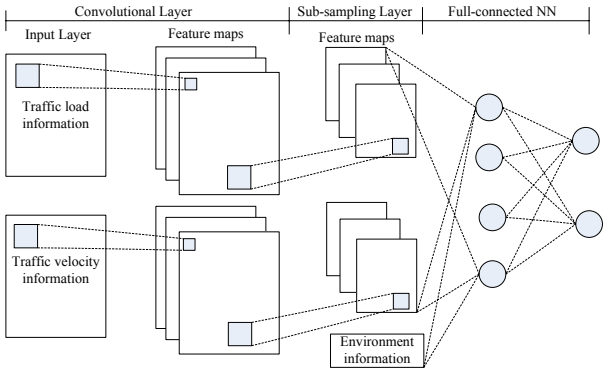


Fig. 3. Structure of remote neural network.

Section 6.3). Then, we can calculate the number of vehicles appear in each square in each time slot, which is the traffic load information input into the RNN. In the meanwhile, the vehicle velocity when it passes through each square is also recorded. By averaging the velocity of all the vehicles passing through each square in each time slot, we can derive the traffic velocity information. A sample of the traffic velocity input is shown in Fig. 4(b).

An alternative and more intuitive method to divide the area covered by the edge cloud system is to perform the division along the road line. However, it is not necessary in our work, since the convolutional kernels can distill the road line information. Also, using squares with dynamic size may also improve the prediction performance. As we will see in Section 6.3, dividing the area with the same-size square can achieve a good performance. Accordingly, we do not adopt the dynamic size squares which would increase the problem complexity.

Output of Remote Neural Network: The ideal output of RNN is the exact traffic velocity in each square. However, through extensive experiments, we found that such modeling method cannot achieve a good performance. Consider we only need to estimate from which AP that a vehicle will access the edge cloud system, rather than the exact position, alternatively, we quantize the traffic velocity. For example, say the maximum velocity in an area is 50 mph, we can quantize the vehicle velocity every 5 mph. Then, we can leverage the quantized traffic velocity in Fig. 4(c) to denote the traffic velocity shown in Fig. 4(b). With such limited number of possible outputs for each square (say there are N possible outputs), we leverage N-bit binary value to encode the quantized traffic velocity and set up N output perceptrons for each square. The k^{th} perceptron for a specific square outputs the likelihood that the traffic velocity in this square is from $5(k-1)$ mph to $5k$ mph.

Based on above discussions, leveraging the traffic and environment information in each time slot t, and the quantized traffic velocity information in time slot $t+\Delta t$, we can train the RNN to forecast the traffic velocity of each square in Δt time slots later.

(a) Map division.

45	48	47	38	40
30	32	48	30	25
30	0	0	45	49
28	0	40	0	49
0	35	0	15	0

(b) Traffic velocity data for input

6	7	7	5	5
3	4	7	3	2
3	0	0	6	7
3	0	5	0	7
0	4	0	1	0

(c) Traffic velocity data for output

Fig. 4. How to generate traffic information.

4.2 Neural Network on Local Vehicles

The neural network on each local vehicle, i.e. LNN, is a conventional multi-stage full-connected neural network. The input of this LNN is the output of RNN, the velocity of the host vehicle and some of the vehicles nearby, the identifier to indicate the vehicle route, and other environment information that is also used by the RNN. The LNN should be maintained by each vehicle itself since the vehicle velocity depends on much individual information, such as vehicle type, the driver's behavior, etc..

The only issue that needs more discussions is how to generate the training data to integrate the route and vehicle velocity. At first, We estimate how many squares a vehicle may traverse in a time slot. Suppose this value is S, we should input identifiers of the next S squares along the vehicle route, and use the velocity of this vehicle in these squares as the desired output. For example, a vehicle can traverse at most 3 squares in a time slot, and the next 3 squares on vehicle V's route are squares 1, 2 and 3. In this case, the identifiers of squares 1, 2 and 3 should be used as part of the input to train the LNN. In the meanwhile, the vehicle velocity in these three squares in the following time slot is used as the output in the training sample. If vehicle V does not arrive in square 3 in the following time slot, the desired vehicle velocity in square 3 is 0.

During the training phase, the real traffic velocity is used to train the LNN. However, in the inference phase, each LNN first pulls the traffic velocity forecast output from the RNN, and set it as part of its input. The output of an LNN is also the quantized vehicle velocity as in RNN. We use the median value of the output velocity interval to estimate the vehicle position. For example, if an LNN indicates its quantized vehicle velocity in an area is between 40 mph and 45 mph, we use 42.5 mph to calculate the vehicle position prediction.

It is worth noting that by putting more output perceptrons in to RNN, it can be used to forecast the traffic velocity in multiple time slots later. With this forecast, LNN can provide a longer-term vehicle velocity and position prediction.

4.3 Discussions

Communication cost: The communication cost in DSEP contains three parts: RNN collects the traffic load and velocity from each square, vehicles pull the traffic velocity forecasting from the RNN, and the local V2V communication to detect the velocity of nearby vehicles. Say there are K squares covered by the edge cloud system, and we need B bytes to encode the

traffic information in each square, the RNN should collect KB bytes data in each time slot. Usually, we only need several bytes to encode the traffic information in a square; even if there are millions of squares in the area covered by the edge cloud system, the RNN needs to collect several millions of bytes data for the traffic velocity forecasting.

To pull the traffic velocity information from the RNN, the data amount received by each vehicle should be less than the amount received by RNN for traffic velocity forecasting, since the output of RNN is quantized and can be encoded with fewer bits. To reduce the amount of data sent by the remote powerful cloud, the traffic velocity forecasting results can be first pushed to the edge cloud, and each vehicle pulls these forecasting results from the edge cloud.

In addition, the V2V communication cost should be much less than that to pull the traffic velocity forecasting from RNN. Since each vehicle only asks for velocity data from several nearby vehicles, such communication cost should be on the dozens of bytes level. Accordingly, DSEP suffers from very light communication cost.

LNN for new vehicles: Since the LNN may capture the characteristics of drivers' behaviors, the LNN on a new vehicle may not work. To provide a compensation for this, each company can train an initial LNN for each brand of vehicle. When the driver sells the vehicle or move to another place, he/she can reset the LNN and train it with new driving data.

Migration cost: There is cost to migrate the SEs among the edge clouds. If we migrate the SEs too frequently, it would incur too much communication cost and even result in network congestion at the network edge. Consider the migration procedure is only invoked once in each time slot, we can control the migration cost by properly set the duration of one time slot.

5 Service Entity Placement

In last section, we discussed how to forecast the vehicle position. Based on the vehicle position forecast, we can dynamically adjust the SE placement to improve the application latency experienced by the passengers. In this section, we first formulate the problem to minimize the largest latency experienced by passengers through SE placement and analyze the problem complexity in Section 5.1. Since the SE placement problem is NP-hard, we propose an efficient algorithm to solve it in Section 5.2.

5.1 Problem Formulation and Analysis

To minimize the largest latency experienced by arbitrary passenger in a specific time slot through SE placement, we can formulate following SE Placement Problem (SEPP):

$$\text{minimize} \quad T \tag{3}$$

Subject to:

$$\sum_{i,j} a_i^u d_{ij} y_j^u + \sum_j y_j^u(\alpha_j m_j + \beta_j) \le T, \quad \forall u \tag{3a}$$

$$\sum_j y_j^u = 1, \quad \forall u \tag{3b}$$

$$\sum_u y_j^u = m_j \quad \forall j \tag{3c}$$

$$y_j^u \in \{0,1\}, \quad \forall j,u \tag{3d}$$

In this formulation, a_i^u is a binary parameter to indicate if passenger u will access the edge cloud system through AP i, which can be derived based on the position forecast in last section; d_{ij} is the distance between AP i and AP j on the topology; and y_j^u is the decision variable to indicate if the SE of passenger u is placed on the edge cloud connecting with AP j. The objective of SEPP is to minimize the maximum latency experienced by arbitrary passenger, T. The constraint ($3a$) says the latency experienced by every passenger u should be less than the maximum latency. On the left hand of this constraint, the first term is the data transmission delay which is associated with the distance on the topology, while the second term is the computation latency which is determined by m_j, i.e. the number of SEs that are placed on the same edge cloud. Constraint ($3b$) indicates that the SE of every passengers should be placed in one and only one edge cloud. Constraint ($3c$) is used to calculate the number of SEs placed on each edge cloud.

Due to the binary variable y_j^u in this formulation, the SEPP model may be difficult to solve directly. Accordingly, we first study the complexity of SEPP. To this end, we only consider a special case of SEPP, i.e. $d_{ij} = 0$ and $\beta_j = 0$. In this case, the SEPP can be modified as

$$\text{minimize} \quad T \tag{4}$$

Subject to:

$$\sum_j \alpha_j y_j^u \sum_v y_j^v \le T, \quad \forall u \tag{4a}$$

$$\sum_j y_j^u = 1, \quad \forall u \tag{4b}$$

$$y_j^u \in \{0,1\}, \quad \forall j,u \tag{4c}$$

Theorem 1. *The special case of SEPP* (4) *is equivalent to following unrelated machine scheduling problem*

$$\text{minimize} \quad T \tag{5}$$

Subject to*:*

$$\sum_u \alpha_j y_j^u \le T, \quad \forall j \tag{5a}$$

$$\sum_j y_j^u = 1, \quad \forall u \tag{5b}$$

$$y_j^u \in \{0,1\}, \quad \forall j,u \tag{5c}$$

Proof: To prove this theorem, we should show constraint ($4a$) is equivalent to constraint ($5a$).

At first, if we have $\sum_u \alpha_j y_j^u \leq T$,

$$\sum_j \alpha_j y_j^u \sum_v y_j^v = \sum_j y_j^u (\sum_v \alpha_j y_j^v) \leq T \sum_j y_j^u = T$$

Contrarily, if we have $\sum_j \alpha_j y_j^u \sum_v y_j^v \leq T$ for any u, suppose there is some j^* such that $\sum_u \alpha_{j^*} y_{j^*}^u > T > 0$, to ensure

$$\sum_j \alpha_j y_j^u \sum_v y_j^v = \sum_j y_j^u (\sum_v \alpha_j y_j^v) \leq T$$

there must be

$$y_{j^*}^u = 0 \quad \forall u$$

It means

$$\sum_u \alpha_{j^*} y_{j^*}^u = \alpha_{j^*} \sum_u y_{j^*}^u = 0$$

which contradicts to our assumption. Accordingly, there must be $\sum_u \alpha_{j*} y_{j^*}^u \leq T$. □

Since a special case of SEPP is equivalent to unrelated machine scheduling problem, which is NP-hard [21], we have following theorem:

Theorem 2. *SEPP is NP-hard.*

Due to the hardness of SEPP, we need an efficient heuristic to solve it.

5.2 Algorithm Details

Since there are two types of latency in SEPP, i.e. the data transmission latency and the computation latency, we can consider to decompose the SEPP into two subproblems and optimize them separately. To this end, we observe that constraint $(3a)$ is the only constraint that couples these two types of latency. Therefore, we reformulate it as

$$\sum_{i,j} a_i^u d_{ij} y_j^u + \sum_j y_j^u (\alpha_j m_j + \beta_j) = \sum_j (\sum_i a_i^u d_{ij} + \beta_j) y_j^u + \sum_j y_j^u \alpha_j m_j$$

If we associate the first term to T_1, while the second term to T_2, and introduce another variable $z_j^u = y_j^u$ to substitute the y_j^u in the second term, SEPP can be reformed as

$$\text{minimize} \quad T_1 + T_2 \tag{6}$$

Subject to:

$$\sum_j (\sum_i a_i^u d_{ij} + \beta_j) y_j^u \leq T_1 \quad \forall u \tag{6a}$$

$$\sum_j y_j^u \alpha_j \sum_u z_j^u \leq T_2, \quad \forall u \tag{6b}$$

$$\sum_j y_j^u = 1, \quad \forall u \tag{6c}$$

$$y_j^u = z_j^u, \quad \forall j, u \tag{6d}$$

$$y_j^u \in \{0, 1\}, \quad \forall j, u \tag{6e}$$

It should be noted that in $(6b)$, we replace m_j by $\sum_u z_j^u$ due to $(3c)$. By relaxing constraint $(6d)$ with Lagrange multiplexer $\boldsymbol{\lambda} = \{\lambda_j^u\}$, we obtain

$$\text{minimize} \quad T_1 + T_2 + \sum_{j,u} \lambda_j^u (y_j^u - z_j^u) \tag{7}$$

Subject to:

$$(6a),\ (6b),\ (6c),\ (6e)$$

$$\sum_j z_j^u = 1, \quad \forall u \tag{7c'}$$

$$z_j^u \in \{0, 1\}, \quad \forall j, u \tag{7e'}$$

Then, (7) can be decomposed into two subproblems. One is associated with the data transmission, named SEPP-T:

$$L_1(\boldsymbol{\lambda}) = \text{minimize} \quad T_1 + \sum_{j,u} \lambda_j^u y_j^u \tag{8}$$

Subject to:

$$(6a),\ (6c),\ (6e)$$

and the other is associated with the computation, named SEPP-C:

$$L_2(\boldsymbol{\lambda}) = \text{minimize} \quad T_2 - \sum_{j,u} \lambda_j^u z_j^u \tag{9}$$

Subject to:

$$(6b),\ (7c'),\ (7e')$$

Following the the same thought to prove Theorem 1, SEPP-C can be simplified as

$$L_2(\boldsymbol{\lambda}) = \text{minimize} \quad T_2 - \sum_{j,u} \lambda_j^u z_j^u \tag{10}$$

Subject to:

$$\alpha_j \sum_u z_j^u \leq T_2 \quad \forall j \tag{10a}$$

$$(7c'),\ (7e')$$

Since $L_1(\lambda) + L_2(\lambda)$ is a lower bound of the objective value of SEPP, we should pursue:

$$T = \text{maximize} \quad L_1(\boldsymbol{\lambda}) + L_2(\boldsymbol{\lambda}) \tag{11}$$

Based on above discussions, we design Algorithm 1 to solve the SEPP problem. In this algorithm, we first initialize all the Lagrange multiplexer to be 0, and solve SEPP-T and SEPP-C, respectively. If these two subproblems cannot achieve an agreement on the SE placement, i.e. $\sum_{j,u} |y_j^u - z_j^u| > 0$, we adjust the Lagrange multiplexer in Line 5. When $y_j^u - z_j^u > 0$, Algorithm 1 should increase λ_j^u. This increases the cost to place the SE of passenger u to the edge cloud j in SEPP-T. Meanwhile, it reduces the cost to place the SE of passenger u to the edge cloud j in SEPP-C. Accordingly, increase λ_i^u benefits SEPP-T and SEPP-C to achieve an agreement on the SE placement. For the same reason, Algorithm 1 reduces λ_j^u if $y_j^u - z_j^u < 0$. Combining these considerations, we update the λ_i^u by setting $\lambda_j^u \leftarrow \lambda_j^u + \kappa(y_j^u - z_j^u)$, where κ is a positive number

Algorithm 1: Service entity placement algorithm

Input: The access point of each passenger $\{a_i^u\}$ and the latency between arbitrary AP pair $\{d_{ij}\}$, and the computation capability of each edge cloud $\{\alpha_j\}$ and $\{\beta_j\}$

Output: The service entity placement $\{y_i^u\}$

```
Initialize λ_j^u ← 0  ∀j, u
Solve (8) and obtain the solution {y_j^u}
Solve (9) and obtain the solution {z_j^u}
while Σ_{j,u} |y_j^u − z_j^u| > 0 do
    λ_j^u ← λ_j^u + κ(y_j^u − z_j^u)
    Update {y_j^u} to be the solution of (8) associated with parameter λ_j^u
    Update {z_j^u} to be the solution of (9) associated with parameter λ_j^u
end while
return {y_j^u}
```

as the iteration step size. Another reason behind this Lagrange multiplexer updating method is that $y_i^u - z_i^u$ is the gradient of λ_j^u. To pursue the objective of (11), we should update λ_j^u in its gradient direction. After updating the Lagrange multiplexer, we solve problems (8) and (9) once again. Such iteration ends when these two subproblems achieve the SE placement agreement.

So far, we have derived an algorithm framework to solve SEPP based on Lagrange relaxation and optimization decomposition. However, problems (8) and (9) are also Integer Linear Programming (ILP) models for given $\boldsymbol{\lambda}$. We need efficient methods to solve them. In the following, we discuss how to solve SEPP-T and SEPP-C in detail.

5.3 Efficient solutions to SEPP-T and SEPP-C

At first, we consider SEPP-T, which can be reformulated as

$$\text{minimize} \quad M \tag{12}$$

Subject to:

$$\sum_j (\sum_i a_i^u d_{ij} + \beta_j) y_j^u + \sum_{j,u} \lambda_j^u y_j^u \le M \quad \forall j, u$$

$$(6c), (6e)$$

Since

$$\sum_j (\sum_i a_i^u d_{ij} + \beta_j) y_j^u + \sum_{j,u} \lambda_j^u y_j^u$$
$$= \sum_j (\sum_i a_i^u d_{ij} + \beta_j + \sum_u \lambda_j^u) y_j^u$$

we have

Theorem 3. *Suppose j^* is one of the locations, such that*

$$j^* = \arg\min_j (\sum_i a_i^u d_{ij} + \beta_j + \sum_u \lambda_j^u)$$

The optimal solution of (12) *is*

$$y_j^u = \begin{cases} 1 & \text{if } j = j^* \\ 0 & \text{otherwise} \end{cases}$$

Proof: At first, the solution proposed in this theorem clearly satisfies the constraints of (12). Suppose there is j' such that

$$\sum_i a_i^u d_{ij'} + \beta_{j'} + \sum_u \lambda_{j'}^u > \sum_i a_i^u d_{ij^*} + \beta_{j^*} + \sum_u \lambda_{j^*}^u$$

and $y_{j'}^u = 1$, we can modify the solution by setting $y_{j^*}^u = 1$ and $y_{j'}^u = 0$. This generates another feasible solution without increasing the objective value. □

As to SEPP-C (10), we can also modify it to be

$$\text{minimize} \quad M \tag{13}$$

Subject to:

$$\sum_u (\alpha_j - \sum_j \lambda_j^u) z_j^u \le M \quad \forall j$$

$$(7c'), (7e')$$

This is clearly a classic parallel unrelated machine scheduling problem, which is NP-hard [21]. Fortunately, we have 2-approximation algorithm to solve it efficiently based on relaxation and rounding [21].

6 Performance Evaluation

In this section, we evaluate the performance of DSEP through extensive real-data driven simulations.

6.1 Data Description and Preprocessing

To evaluate the performance of DSEP, we leveraged the trace data of all the Electronic Taxis (ETs) in Shenzhen, Guangdong province, China from March 1^{st}, 2016 to March 31^{st}, 2016, which was collected by Shenzhen Institute of Beidou Applied Technology (SIBAT). During the data collection procedure, the staffs of SIBAT equipped a location tracing device on every ET in Shenzhen, and the device would report ET trace data to the controlling center every 20 seconds. The trace data of each ET includes following information: its plate ID, time stamp and its location (i.e. longitude and latitude). According to these messages, we can easily construct the route and calculate the velocity of each ET. In addition, we can also count how many ETs pass through each square, i.e. the traffic load information, in each time slot. The same data set has been used in some of the previous works [22]–[25].

During the simulation, the data on the first 21 days are used for the training purpose, while the remaining data are used for testing purpose. Before we generate the samples for training the RNN and LNNs, we remove some of the data indicating an ET stops on the road. If an ET stops for more than 3 minutes, and at least one of the following conditions for the square in which this ET stops holds, all its data from the time it stops to the time it starts to move again are removed:

- There is a charge station in 20 m from the location where the ET stops
- There are ETs stopping for more than 20 minutes in its nearby squares almost everyday
- The ET stops in this square or its nearby squares for more than 20 minutes almost everyday

The first item implies the ET is waiting for the charging services. The second item is proposed in case that the ET enters a parking area and waiting for passengers. The last item occurs because a driver would park the ET at a place when he/she has a rest everyday. Under these conditions, the ET would not impact the traffic condition, and hence we do not use related data to train the NNs. In other cases, an ET may stop on the road due to the traffic signal (< 3 min) or traffic congestion (> 3 min). In addition, our simulations are only based on the data in the daytime, i.e. 7:00am – 9:00pm. During the night, since the drivers go off work and the ETs stop in the parking lots, most of the data are filtered out and there are not enough data for us to train the NNs.

Since ETs seldom appear in some areas, we select a $10\ km \times 10\ km$ area, in which ETs are usually cruising. There were 697 ETs that appear in this area everyday during the period we collected data. We only use these ETs in the simulation.

6.2 Simulation Settings

Edge cloud system setup: Because we only have the ET trace information, but not the edge cloud system information, we randomly generate longitudes and latitudes following uniform distribution to allocate edge servers in the area we study to host SEs. In Shenzhen, a metropolis in China, busy roads are overcrowded in our studying area, and hence we randomly allocate the edge servers over the entire studying area, rather than along the roads. Considering the coverage radius of each edge cloud is several hundreds of meters [26, 27], we allocate 200 edge servers in the area. In this case, the coverage radius of each edge server is about 400 meters. It should be noted that by changing computation latency parameter, α_i, we can derive similar simulation results with different number of edge servers. For example, if we increase the α_i by 10 times, we can derive similar results with $\frac{1}{10}$ of edge servers. Therefore, we fix the number of edge servers in our simulations. For simplicity, we set $\beta_i = 0$ and assume $\alpha_i = \alpha$ for all the edge cloud i.

Among the edge clouds, we assume there is a wired connection between any two edge clouds with a probability reversely proportional to the physical distance between them. The transmission latency between two directly connected edge clouds is set to be 2 ms, and the application data are delivered through the wired path with minimum number of hops among edge clouds. This setting is based on the measurement with traceroute or ping command. We can see that usually the latency from our hosts to a nearby server is hundreds of microseconds to 2 ms. Accordingly, we conservatively assume the transmission latency between two directly connected edge clouds is 2 ms, since the smaller the transmission latency is, the larger performance improvement DSEP can achieve. Actually, we can also assume it is hundreds of microseconds or tens of mini-seconds. Regardless of which setting we adopt, almost the same conclusion should be derived in the simulations: DSEP can achieve a good trade-off between the transmission latency and the computation latency, such that the entire latency experienced by the passengers will be minimized.

Baselines and Metrics: For the vehicle velocity forecast, the baseline is the ground truth which is derived based on the collected ET location data. We leverage Average Root Mean Square Error (ARMSE) to evaluate the performance of the vehicle velocity forecast method, which is defined as:

$$ARMSE = \frac{1}{E}\sum_{e}\sqrt{\frac{\sum_t(\hat{v}_e(t) - v_e(t))^2}{T}} \tag{14}$$

where E is the number of ETs, T is the number of time slots, while $\hat{v}_e(t)$ and $v_e(t)$ are the forecast and real velocity of ET e at time slot t, respectively. To evaluate the performance of vehicle position prediction, we test the average number of ETs whose passengers connect to the APs different from the estimated one over all the time slots.

As to the SE placement part, we have three baselines: 1) place SEs on the nearest edge cloud based on the predicted ET positions; 2) place SEs on the nearest edge cloud based on the real ET positions; 3) place SEs with Algorithm 1 based on the real ET positions. It should be noted that, in fact, DSEP places SEs with Algorithm 1 based on the predicted ET positions. In this part, the evaluation metric is the performance improvement. We define the performance improvement of scheme 1 compared with scheme 2 as $\frac{L_2 - L_1}{L_2}$, where L_1 and L_2 are the maximum latency experienced by arbitrary passenger under scheme 1 and scheme 2, respectively.

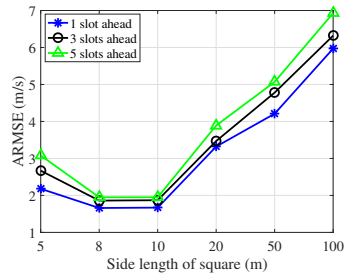
Since the locations of edge clouds are randomly selected, all the points in following simulations are averaged by 20 tries.

6.3 Performance of Vehicle velocity/Position Prediction

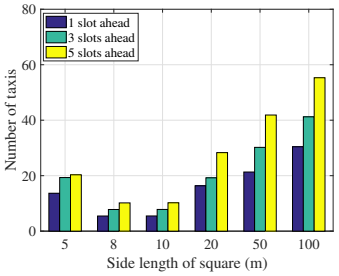
There are mainly three parameters that greatly impact the performance to predict the vehicle position: 1) the size of squares to divide the studying area; 2) the size of the convolutional kernels in RNN; and 3) the number of hidden layers in LNNs. In this subsection, we investigate the impact of these parameters through extensive simulations, which also provides us guidelines on how to set the NN parameters.

Impact of square size: To study how the square size impacts the performance to predict the vehicle position, we assume the convolutional kernel of the RNN is 50×50 and each ET hosts a LNN with only 1 hidden layer. Then, we change the size of the squares and test the vehicle position prediction performance. The simulation results are shown in Fig. 5. From this figure, we can make following observations.

Firstly, when we divide the area into a grid of squares with side length $8 \sim 10\ m$, we can derive the best traffic velocity prediction performance (the performance is slightly worse

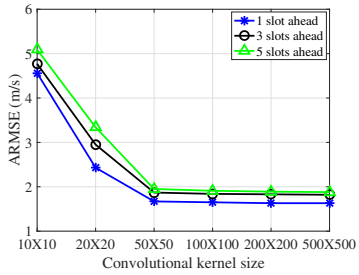


(a) Traffic velocity ARMSE.

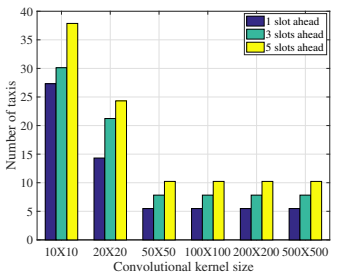


(b) Wrong AP estimation.

Fig. 5. The impact of square side length.

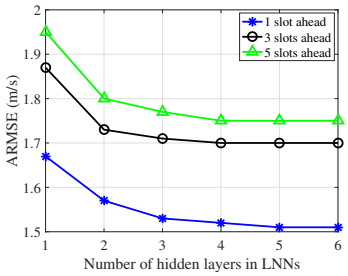


(a) Traffic velocity ARMSE.

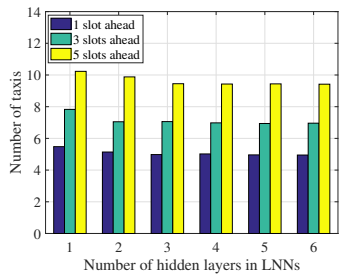


(b) Wrong AP estimation.

Fig. 6. The impact of convolutional kernel size in RNN.



(a) Traffic velocity ARMSE.



(b) Wrong AP estimation.

Fig. 7. The impact of the number of hidden in LNNs.

when the side length is $8\ m$ than that when the side length is $10\ m$). No matter the square size goes larger or smaller, the vehicle velocity prediction performance goes worse. By carefully studying the road condition of the metropolis where the traffic data were collected, we found that the width of most of the roads in one direction is about $8 \sim 9$ meters. The squares with side length $8 \sim 10\ m$ can best match such characteristic. Because we cannot ensure the squares used to divide the studying area starting from the bound of the roads, both of these two types of squares may incur some errors (interference between two directions or cannot cover one direction). Even if we can set the square side length to be the road width in one direction (in fact, we cannot since the road width is changing over the studying area), such error still exists and we cannot expect a much better performance. However, both of these two settings can derive almost the same vehicle velocity prediction performance and outperform other settings, which gives us some hints to tune the parameters. On the other hand, the squares with larger size may include too many lanes in a square, while the smaller squares may divide the traffic information in one direction into multiple squares, therefore, larger or smaller size of the squares used to divide the studying area incurs larger error and hurts the performance of traffic velocity prediction.

Secondly, as shown in Fig. 5(a), by dividing the entire area into a grid of $8\ m \times 8\ m$ or $10\ m \times 10\ m$ squares, the ARMSE of vehicle velocity prediction is about 1.8 mph, which is close to the average quantizing error 1.25 mph. It shows the good performance of our vehicle velocity prediction method.

Thirdly, it performs better to predict the vehicle velocity in a time slot closer to the current one. This is a very intuitive observation. However, it is worth noting that even to predict the vehicle velocity in 5 time slots later, the performance is very close to that only predicts the vehicle velocity in 1 time slot later. This enables DSEP to predict the vehicle velocity in several minutes later, which is enough for the SE placement optimization and migration.

Lastly, from Fig. 5(b), we can observe that, with improper square size, many passengers would connect to the APs different from our estimation. However, by tuning the parameters, there are on average less than 1% of the ETs (about 6 ETs) suffer from the wrong AP estimation in 1 time slot later.

Impact of convolutional kernel size: Now, we assume there are 10 convolutional kernels for both of the traffic load information and traffic velocity information; we divide the entire area into a grid of squares, each of which is $10\ m \times 10\ m$. Then, we change the size of the convolutional kernels and derive the simulation results as shown in Fig. 6.

In addition to the observations we made when we studied the impact of square size, we can see that the vehicle velocity prediction performance enhances with the increase of the convolutional kernel size. This is also a straight forward observation as the larger convolutional kernel introduces more decision variables to figure out the traffic characteristics. However, we can also observe that when the convolutional kernel size achieves 50×50, the vehicle velocity prediction performance does not significantly improve even we continue to enlarge the convolutional kernel size. This indicates that the velocity of a vehicle is impacted by the traffic condition within a 500 m radius. Consider that larger convolutional kernel incurs larger computation overhead which hurts the performance of online systems, we believe 50×50 should be the best choice of the convolutional kernel size in our simulation scenario.

Impact of hidden layer number: To study how the number of hidden layers in LNNs impacts the vehicle velocity prediction, we divide the entire area into a grid of squares of $10\ m \times 10\ m$, and set the size of the convolutional kernels to be 50×50. Then, we try different number of hidden layers in LNNs and obtain the simulation results shown in Fig. 7. From this figure, we can see that increasing the number of hidden layers in LNNs can only slightly improve the vehicle velocity prediction performance. Consider the vehicle position prediction should be completed in a timely manner and the scarcity of computation resources carried by each vehicle, we can maintain the LNNs on each vehicle with only 1 hidden layer.

Another observation from Fig. 7 is that even the ARMSE of vehicle velocity prediction keeps the same, the number of ETs connecting to the APs different from the estimation may slightly change. This is because even if the velocity ARMSE keeps the same, the vehicle position estimation can be different, and hence there are different number of ETs connecting to the APs different from our estimation.

6.4 Performance of DSEP

In this section, we explore how the performance of DSEP is impacted by the computation latency parameters, and the workload in the entire system. The parameters of the RNN and LNNs are set based on the simulation in last subsection. Because these simulations are based on real traces where the vehicle velocity is continuously changing, it also shows that the DSEP has the ability to deal with the dynamic characteristic of the real transportation systems. It should be noted that under every specific simulation setting, we not only show the maximum latency averaged among all the simulation time slots, but also present the 95% confidence interval of the maximum latency with the error bar.

Impact of computation resource power: Consider that the capacity of an ET is 5 or 7, besides the driver, there would be 0 – 6 passengers on an ET. Accordingly, we first assume that in each ET, the passengers runs 0 – 6 and on average 2 applications having SEs on edge clouds. Then, we test how the maximum latency experienced by all the passengers changes with the computation latency parameter α. Fig. 8(a) shows the simulation results.

From this figure, we can see that regardless of the nearest placement or the Algorithm 1 is adopted, place the SE with the real ET position leads to a lower latency since it eliminates the unnecessary data transmission among edge clouds. However, such performance improvement is very slight since the vehicle position prediction in DSEP is very close to the ground truth.

When the nearest placement scheme is adopted, we cannot balance the workload among the edge clouds to reduce the computation latency, and thus, the latency experienced by passengers increasing linearly with the computation latency parameter α. In DSEP, we can balance the workload among edge clouds, and hence the maximum latency increases slower than that derived by nearest placement. When the α is set to be 1, DSEP outperforms the nearest placement with real ET positions by 43% when averaging the maximum latency among all the time slots. Even averaging all six cases we tested in the evaluations, DSEP can outperform the nearest placement scheme by 16%.

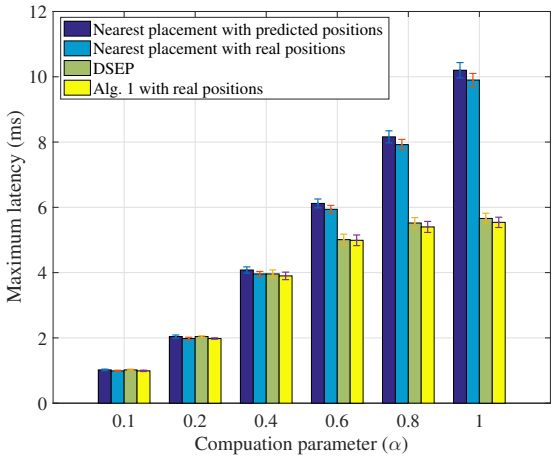
When the α is small (less than 0.3 in our simulation), the performance achieved by nearest placement is the same as that achieved by Algorithm 1. This is because that when the α is small, i.e. the edge clouds are very powerful, the transmission latency dominates the entire latency experienced by the passengers. In this case, we should minimize the transmission latency, which is exactly what the nearest placement does.

The 95% confidence interval of maximum latency increases with the degradation of the computation power, i.e. increase of α. This is only because the computation latency increases when we do not have powerful computation resources. Regardless of which scheme is adopted, the maximum latency is determined by the edge servers located at the hot spot, and hence the system performance is stable for most of the time. Accordingly, we can observe that the 95% confidence interval of maximum latency is narrow in all the cases.

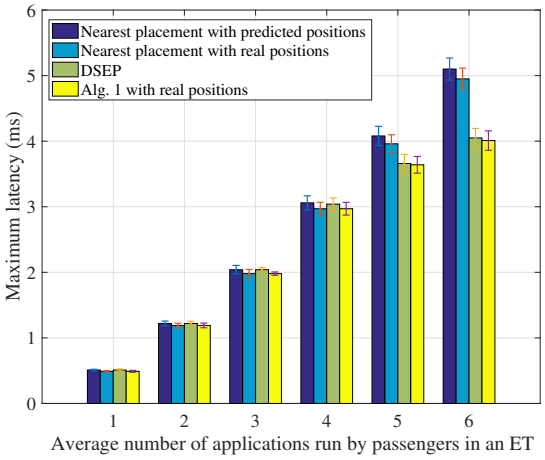
It is worth noting that during the simulations when the computation parameter α is set to be 1, we can observe in some time slots that the maximum latency under nearest placement scheme may reach about 17 ms, while the worst maximum latency in DSEP is only about 7 ms. This case seldom happens and hence is not covered by the 95% confidence interval, but it really results in latency "jitter" in the system. In DSEP, we can forecast the vehicle locations and know there would be a hot spot, and then distribute the workload to other edge clouds, which reduces the worst maximum latency experienced by the passengers. Accordingly, DSEP can reduce the latency "jitter" in the system.

Impact of workload: To see how DSEP adapts to different workload, we have two different ways to change the workload. First, change the number of vehicles, i.e. the vehicle density in the system. However, this scheme is difficult to implement in our simulations since in our real trace driven simulations, we cannot ensure the vehicles we add into the system follows the same characteristics in the real system, especially how their velocity changes with the environment condition. It may degrade the forecasting performance of the nerual networks in DSEP. Accordingly, we adopt the second way, changing the number of applications run by the passengers in each ET. It should be noted that in the workload perspective, i.e. the number of SEs that should be placed on the edge clouds, changing the vehicle density and the number of applications run by the passengers in the ETs achieves the same effect. In this simulation, we set the computation parameter α to be 0.1. The simulation results are shown in Fig. 8(b).

We can see that we derive a group of curves similar to those in Fig. 8(a). When the workload is light, the performance of DSEP is similar to that derived by nearest placement scheme since the computation latency is smaller than the transmission latency. With the increase of workload, the computation la-



(a) Impact of computation parameter.



(b) Impact of workload.

Fig. 8. Performance of DSEP.

tency on some edge clouds exceeds the transmission latency for balancing the workload. In this case, the latency in DSEP increases slower than that with nearest placement since DSEP can trade off the computation and transmission latency, such that the total latency can be minimized. When there are on average 6 applications run by passengers in each ET, DSEP can reduce the maximum latency experienced by passengers by 21% on average. When the workload increases, though the 95% confidence interval of maximum latency experienced by passengers also increases, it is always narrow, which shows the stability of the system. Again, we can also observe extreme large maximum latency experienced by passengers in some time slots when the workload is large in the system under the nearest placement scheme, which is incurred by the hot spot problem. With DSEP, we can solve the hot spot problem. Accordingly, such "jitter" can be reduced and better experience can be provided to the passengers.

7 Conclusions

This paper proposed DSEP to dynamically place Service Entities (SEs) of passengers' applications in edge clouds. To this end, we first designed two sequential neural networks to predict the position of each vehicle in the near future; and then proposed an efficient algorithm based on Lagrange relaxation and decomposition to calculate the SE placement. Extensive real-data driven simulations showed that DSEP can not only predict the vehicle position accurately, but also greatly reduce the maximum latency experienced by arbitrary passenger compared with the nearest placement scheme.

Acknowledgment

Many thanks to the Shenzhen Institute of Beidou Applied Technology (SIBAT), who provides us with the electronic taxi location data in Shenzhen. This research is partially supported by the National Key R&D Program of China 2018YFB1003202; NSF grants CNS-1626374, US Department of Transportation grant DTRT13-G-UTC48, NSFC grants 61671130, 61671124 and 61602194; the Fundamental Research Funds for the Central Universities-HUST 2016YXMS304.

References

[1] Y. Abe, R. Geambasu, K. Joshi, H. A. Lagar-Cavilla, and M. Satyanarayanan, "vtube: Efficient streaming of virtual appliances over last-mile networks," in *Proceedings of the 4th SoCC*, 2013, pp. 1–16.

[2] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, "Towards wearable cognitive assistance," in *Proceedings of the 12th MobiSys*, 2014, pp. 68–81.

[3] M. Satyanarayanan, G. Lewis, E. Morris, S. Simanta, J. Boleng, and K. Ha, "The role of cloudlets in hostile environments," *IEEE Pervasive Computing*, vol. 12, no. 4, pp. 40–49, Oct 2013.

[4] S. Davy, J. Famaey, J. Serrat, J. L. Gorricho, A. Miron, M. Dramitinos, P. M. Neves, S. Latre, and E. Goshen, "Challenges to support edge-as-a-service," *IEEE Communications Magazine*, vol. 52, no. 1, pp. 132–139, 2014.

[5] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the MCC*, 2012, pp. 13–16.

[6] S. Wang, R. Urgaonkar, T. He, M. Zafer, K. Chan, and K. K. Leung, "Mobility-induced service migration in mobile micro-clouds," in *IEEE Military Communications Conference*, 2014, pp. 835–840.

[7] X. Kong, Z. Xu, G. Shen, J. Wang, Q. Yang, and B. Zhang, "Urban traffic congestion estimation and prediction based on floating car trajectory data," *Future Gener. Comput. Syst.*, vol. 61, no. C, pp. 97–107, Aug. 2016.

[8] G. Marfia and M. Roccetti, "Vehicular congestion detection and short-term forecasting: A new model with results," *IEEE Transactions on Vehicular Technology*, vol. 60, no. 7, pp. 2936 – 2948, 2011.

[9] P. Lopez-Garcia, E. Onieva, E. Osaba, A. D. Masegosa, and A. Perallos, "A hybrid method for short-term traffic congestion forecasting using genetic algorithms and cross entropy," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 2, pp. 557 – 569, 2016.

[10] M. Castro-Neto, Y.-S. Jeong, M.-K. Jeong, and L. D. Han, "Online-svr for short-term traffic flow prediction under typical and atypical traffic conditions," *Expert Syst. Appl.*, vol. 36, no. 3, pp. 6164–6173, 2009.

[11] Y. Lv, Y. Duan, W. Kang, Z. Li, and F.-Y. Wang, "Traffic flow prediction with big data: A deep learning approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 865 – 873, 2015.

[12] P. Dell'Acqua, F. Bellotti, R. Berta, and A. D. Gloria, "Time-aware multivariate nearest neighbor regression methods for traffic flow prediction," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 6, pp. 3393 – 3402, 2015.

[13] "Monte carlo simulation-based traffic speed forecasting using historical big data," *Future Generation Computer Systems*, vol. 65, pp. 182 – 195, 2016.
[14] C. Dong, Z. Xiong, C. Shao, and H. Zhang, "A spatialtemporal-based state space approach for freeway network traffic flow modelling and prediction," *Transportmetrica A: Transport Science*, vol. 11, no. 7, pp. 547–560, 2015.
[15] H. Tan, Z. Han, X.-Y. Li, and F. C. M. Lau, "Online job dispatching and scheduling in edge-clouds," in *the IEEE INFOCOM*, April 2017, pp. 1–9.
[16] L. Wang, L. Jiao, T. He, J. Li, and M. Muhlhauser, "Service entity placement for social virtual reality applications in edge computing," in *the IEEE INFOCOM*, April 2018, pp. 1–9.
[17] L. Wang, L. Jiao, J. Li, and M. Mhlhuser, "Online job dispatching and scheduling in edge-clouds," in *the IEEE ICDCS*, 2017, pp. 1281 – 1290.
[18] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge-clouds," in *the IFIP Networking*, 2015, pp. 1 – 9.
[19] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. K. Leung, "Dynamic service migration and workload scheduling in edge-clouds," *Perform. Eval.*, vol. 91, no. C, pp. 205–228, Sep. 2015.
[20] Q. Qin, K. Poularakis, G. Iosifidis, and L. Tassiulas, "Sdn controller placement at the edge: Optimizing delay and overheads," in *the IEEE INFOCOM*, April 2018, pp. 1–9.
[21] J. K. Lenstra, D. B. Shmoys, and E. Tardos, "Approximation algorithms for scheduling unrelated parallel machines," in *IEEE FOCS*, Oct 1987, pp. 217–224.
[22] Z. Tian, L. Tu, Y. Wang, F. Zhang, and C. Tian, "Impact of core charging station's cease operation in the entire charging station system: A case study in shenzhen," in *2017 IEEE Third International Conference on Big Data Computing Service and Applications (BigDataService)*, April 2017, pp. 90–95.
[23] J. Zhang, D. Shen, L. Tu, F. Zhang, C. Xu, Y. Wang, C. Tian, X. Li, B. Huang, and Z. Li, "A real-time passenger flow estimation and prediction method for urban bus transit systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 11, pp. 3168–3178, Nov 2017.
[24] Zhiyong Tian, Yi Wang, Chen Tian, Fan Zhang, Lai Tu, and Chengzhong Xu, "Understanding operational and charging patterns of electric vehicle taxis using gps records," in *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, Oct 2014, pp. 2472–2479.
[25] Z. Tian, T. Jung, Y. Wang, F. Zhang, L. Tu, C. Xu, C. Tian, and X. Li, "Real-time charging station recommendation system for electric-vehicle taxis," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 11, pp. 3098–3109, Nov 2016.
[26] Y. Yu, J. Zhang, and K. B. Letaief, "Joint subcarrier and cpu time allocation for mobile edge computing," in *IEEE Global Communications Conference (GLOBECOM)*, Dec 2016, pp. 1–6.
[27] Y. Zhang, K. Wang, Y. Zhou, and Q. He, "Enhanced adaptive cloudlet placement approach for mobile application on spark," *Security and Communication Networks*, pp. 1–12, 2018.

Yangming Zhao is a research scientist with SUNY Buffalo. He received the B.S. degree in communication engineering and the Ph.D. degree in communication and information system from University of Electronic Science and Technology of China in 2008 and 2015, respectively. His research interests include network optimization, data center networks, edge computing and transportation systems.

Xin Liu is a Ph.D. candidate with SUNY Buffalo. He received his M.S. degree in Computer Science and Engineering from SUNY Buffalo in 2016, and B.S. degree in Electrical and Engineering from Beijing University of Technology in 2014. His research interests include edge computing and transportation systems.

Lai Tu received the B.S. degree in communication engineering and the Ph.D. degree in information and communication engineering from Huazhong University of Science and Technology, China, in 2002 and 2007, respectively. From 2007 to 2008, he was a Post-Doctoral Fellow with the Department of EIE, Huazhong University of Science and Technology. From 2009 to 2010, he was a Post-Doctoral Researcher with the Department of CSIE, Nation Cheng Kung University, Taiwan. He is currently an Associate Professor with the School of Electronic and Information and Communications, Huazhong University of Science and Technology. His research areas include urban computing, human behavior study, intelligent transportation system, mobile computing, and networking.

Chen Tian is an associate professor at State Key Laboratory for Novel Software Technology, Nanjing University, China. He was previously an associate professor at School of Electronics Information and Communications, Huazhong University of Science and Technology, China. Dr. Tian received the BS (2000), MS (2003) and PhD (2008) degrees at Department of Electronics and Information Engineering from Huazhong University of Science and Technology, China. From 2012 to 2013, he was a postdoctoral researcher with the Department of Computer Science, Yale University. His research interests include data center networks, network function virtualization, distributed systems, Internet streaming and urban computing.

Chunming Qiao is a SUNY Distinguished Professor and also the current Chair of the Computer Science and Engineering Department at University at Buffalo. He was elected to IEEE Fellow for his contributions to optical and wireless network architectures and protocols. His current focus is on connected and autonomous vehicles. He has published extensively with an h-index of over 69 (according to Google Scholar). Two of his papers have received the best paper awards from IEEE and Joint ACM/IEEE venues. He also has 7 US patents and served as a consultant for several IT and Telecommunications companies since 2000. His research has been funded by a dozen major IT and telecommunications companies including Cisco and Google, and more than a dozen NSF grants.