Received September 1, 2020, accepted September 14, 2020, date of publication September 25, 2020, date of current version October 7, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3026677

Route Planning Through Distributed Computing by Road Side Units

JOSE PAOLO V. TALUSAN[1], (Member, IEEE), MICHAEL WILBUR[2], ABHISHEK DUBEY[2], (Senior Member, IEEE), AND KEIICHI YASUMOTO[1], (Member, IEEE)
[1]Graduate School of Information Science, Nara Institute of Science and Technology, Nara 630-0192, Japan
[2]Institute for Software-Integrated Systems, Vanderbilt University, Nashville, TN 37235, USA

Corresponding author: Jose Paolo V. Talusan (talusan.jose_paolo.tg3@is.naist.jp)

This work was supported in part by the Japan Society for the Promotion of Science (JSPS) KAKENHI under Grant 19H05665, and in part by the Research and Development for Trustworthy Networking for Smart and Connected Communities, Commissioned Research of the National Institute of Information and Communications Technology (NICT) and the National Science Foundation under Award 1818901 and Award 1647015.

ABSTRACT Cities are embracing data-intensive applications to maximize their constrained transportation networks. Platforms such as Google offer route planning services to mitigate the effect of traffic congestion. These use remote servers that require an Internet connection, which exposes data to increased risk of network failures and latency issues. Edge computing, an alternative to centralized architectures, offers computational power at the edge that could be used for similar services. Road side units (RSU), Internet of Things (IoT) devices within a city, offer an opportunity to offload computation to the edge. To provide an environment for processing on RSUs, we introduce RSU-Edge, a distributed edge computing system for RSUs. We design and develop a decentralized route planning service over RSU-Edge. In the service, the city is divided into grids and assigned an RSU. Users send trip queries to the service and obtain routes. For maximum accuracy, tasks must be allocated to optimal RSUs. However, this overloads RSUs, increasing delay. To reduce delays, tasks may be reallocated from overloaded RSUs to its neighbors. The distance between the optimal and actual allocation causes accuracy loss due to stale data. The problem is identifying the most efficient allocation of tasks such that response constraints are met while maintaining acceptable accuracy. We created the system and present an analysis of a case study in Nashville, Tennessee that shows the effect of our algorithm on route accuracy and query response, given varying neighbor levels. We find that our system can respond to 1000 queries up to 57.17% faster, with only a model accuracy loss of 5.57% to 7.25% compared to using only optimal grid allocation.

INDEX TERMS Distributed computing, middleware, transportation, vehicle routing, road side units.

I. INTRODUCTION

Urban cities are growing at a rapid pace. This growth, bolstered by businesses, commerce, and opportunities, is bringing more and more people into these urban landscapes, putting a strain on the cities' limited infrastructure. For example, roads are faced with an influx of motorists resulting in heavy traffic, which makes living in cities unbearable. To meet the growing demands, cities and private companies are turning to data-intensive applications to make the most out of the limited resources. Companies such as Google, Apple, and Waze[1] created traffic-aware navigation apps that leverage crowd-sourced information and cloud environments. These applications allow users access to up-to-date routing information for their daily commutes.

Cloud environments running on centralized architectures can process data effectively for existing services, however, future use cases such as autonomous vehicles have their own challenges [1], such as requiring ultra-reliable low-latency communications [2]. These requirements expose the limitations of the cloud and call for a new architecture that better fits the requirements for real-time analysis and processing of city-level data.

The associate editor coordinating the review of this manuscript and approving it for publication was Keli Xiao.

[1]https://www.waze.com/

One way to resolve the issue is to use edge computing. Sub-networks of Internet of Things (IoT) devices [3], [4] is one approach to implementing edge computing. One example of these is the road side unit (RSU). RSUs are resource-constrained devices, are placed along city roads and highways. RSUs are readily available and are more robust under communication constraints.

Their location at the edge, and their proximity to users, can help decrease latency. Also, these networks are capable of performing tasks in the absence of the cloud. These characteristics of edge computing make it very attractive for cities, especially those in developing countries, with insufficient investments in infrastructure. Leveraging devices at the edge, where there is little to no cost of utilizing them, is a sustainable solution for smart city services.

Prior efforts to harness these untapped computational resources include various Platform-as-a-Service (PaaS) approaches such as Arkessa,[2] Axeda,[3] and Xively.[4] These approaches suffer from two limitations that prevent it from broader utility. First, these rely on support from the cloud to deploy them over IoT devices. Second, these have not yet achieved the context-based decentralized stream processing which is needed for real-time, city-wide data processing and analysis.

Harnessing the untapped resources of IoT devices would enable these devices to perform tasks without the presence of the cloud. To accomplish this Edge and Fog computing paradigms execute processes on the devices at the edge. However, these paradigms do not consider the utilization of computational resources at the edge. Due to the inherent heterogeneity of the devices, a middleware for interoperability and to maximize the available resources is needed.

Harnessing these untapped resources typically means performing tasks, meant for cloud-based and centralized platforms, on the edge. However, utilizing a decentralized system for centralized architectures is another challenge. One such task, which leverages the resources and city-wide data on the edge, is route planning. Route planning is a well-researched field [5], however, most state-of-the-art algorithms [6], [7], [8] are typically developed and used with centralized approaches. In these scenarios, data is shared and paralleled, allowing for typical search algorithms access to shared memory and direct communication between multiple processors. Efforts have been made to utilize these algorithms in a distributed setting. We discuss these existing solutions in detail in Section II.

Contributions: The key problem, in designing route planning algorithms that take advantage of decentralized networks such as the RSU sub-network, is the efficient distribution and processing of routing tasks. This article introduces *Road Side Units for Edge Computing* (RSU-Edge), a middleware that enables route planning services on road side units.

[2]https://www.arkessa.com/

[3]https://www.immixgroup.com/Axeda/

[4]https://xively.com/

Our approach assumes a smart city where RSUs are deployed along roads and highways. The city is divided into grids which are then assigned RSUs. Each RSU has its own local speed data, models for route planning, and awareness about its neighbors. The goal of the service is to be able to respond to the user's route planning queries with the highest accuracy. This is obtained by using the most up-to-date data available in each RSU as weights to generate routes for trip queries. Tasks must be allocated to the optimal RSUs specified by the system. However, forcing all tasks to only this optimal allocation causes a load imbalance which causes a delay in the overall response time of the system. Conversely, allocating tasks to sub-optimal, less utilized RSUs results in stale data being used, which gives a route based on old traffic information. This is because each RSUs local speed data takes time to propagate to neighboring nodes.

The problem thus, how to allocate tasks in an efficient manner over all the RSUs such that we can respond in an acceptable time with acceptable model accuracy. To fully utilize the decentralized nature of the sub-network, we developed a task allocation algorithm that distributes tasks based on a region of interest heuristic. To evaluate this system, we conducted an experiment based on real-world speed data of Nashville. After the experiment, we examine the potential trade-off between query processing delay and model accuracy based on various parameters of the task allocation algorithm.

The specific contributions of this article are as follows:

- We introduce a middleware which utilizes available resources from decentralized edge devices to process and analyze city-wide data.
- We design and develop a decentralized route planning service over a distributed network of road side units. We design a task allocation algorithm for handling queries and allocating tasks efficiently and to decrease response time while maintaining a high model accuracy. Our code is available online.[5]
- We provide a proof-of-concept implementation of the middleware and service. We evaluate this approach using actual data from Nashville, Tennessee. Summarizing the results of the present study suggests that a decentralized route planning service can feasibly work on distributed nodes given a trade-off between accuracy and speed. We found that we can improve query response time by up to 57.17% with only a 7.25% decrease in accuracy by using neighbor RSUs.

Outline: The rest of the paper is organized as follows. Section II details fundamental notation and related work related to this article. We introduce RSU-Edge in Section III. Section IV defines the problem and discusses our task allocation algorithm. We evaluate the system and show the results in Section V followed by a discussion of its performance in Section VI. Finally Section VII concludes the paper.

[5]https://github.com/linusmotu/mqtt-dist-routing

II. RELATED WORK

A. INTERNET OF THINGS AND TASK ASSIGNMENT

Internet of Things (IoT) allows data to be generated at such a high volume and many services have been created to utilize such influx of data. However, processing of these data is typically handed off to centralized clouds due to IoT devices' resource constraints [9]. Cloud computing complements IoT's data generation with near-unlimited processing and storage for use in smart applications. Despite the known advantages of deploying services over the cloud, the distance between the user and the cloud as well as the unpredictability of end-to-end networks, introduce latency [10]. These latencies can be detrimental to response time requirements of services, affecting its quality of service (QoS). To address this, efforts turn to edge and fog networks as methods to bring processing and computation nearer to the network's edge, reducing latency [4], [11]. Ren *et al.* [12] utilized the collaborative effort of edge nodes and the cloud to minimize latency. Assigning latency-sensitive tasks closer to the data while still being able to assign computationally intensive tasks to the cloud. Yu *et al.* [13] showed that even though the edge computing servers have less computation power than the cloud, they still provide better QoS (Quality of Service) and lower latency to the end-users.

The main challenges associated with edge and fog computing relates to their usability, coordination, and task assignment. Zeng *et al.* [14] worked on task scheduling on the fog, to optimize constrained resources to minimize task execution time. Moschitta *et al.* [15] studied managing and utilizing fog resources to improve the performance of IoT services, such as response time, energy consumption, and cost reduction. Plenty of research has gone into the design and development of urban middleware with the intention of coordinating large systems of heterogeneous edge or fog networks [16], [17] and [18].

A middleware is a software that provides interoperability among incompatible devices and applications. It is one of the technologies that enables IoT solutions [19], [20]. The middleware allows developers to work with heterogeneous objects without having to invest significant amounts of time to learn specific hardware platforms. Additionally, the middleware can receive, process, and distribute data over the network.

One of the main objectives of middleware is to maximize available resources. A way to achieve this is to identify the most efficient way of dividing computation tasks into manageable pieces and assigning them to the available nodes. We refer to this problem as the task allocation problem [21]. This problem has been extensively studied in the cloud architecture [22]–[24]. The main objective of these works is to efficiently utilize processing resources and still be to meet all job deadlines. However, the task allocation problem in cloud computing does not take into account parameters such as data transfer delays, typically the networking between machines in a cloud environment is negligible. Another key component of urban middleware is resource discovery, a vital step especially considering the heterogeneity of devices within the edge and fog networks [25], [26].

Research in the provisioning of resources in edge and fog networks have increasingly become important. Skarlat *et al.* [27], [28] proposed a platform centered around the idea of fog colonies (sets of fog nodes) with a centralized cloud for additional resources when needed. Xu *et al.* [29] proposed a platform for location-based and latency-sensitive applications that use micro data centers on the network edge or large centralized cloud for processing. This research includes a cloud component for additional processing and storage. Research on In-situ edge IoT devices looks at task assignment without relying on cloud resources [30], [31].

There is no shortage in the number of available IoT devices currently deployed in the wild. Computational resources are abundant, the problem is that most of it exist on often idle devices. Existing solutions for harnessing this power, such as volunteer computing (e.g., BOINC), are centralized platforms in which an entity can control participation and pricing. MODiCuM [32], a distributed-ledger based platform for decentralized computation, offers a system for an open market of computational resources. It offers a smart contract-based protocol, incentivizing creators, and resource providers to join an open-market, allowing users to harness the computational power at the edge.

B. CENTRALIZED AND DECENTRALIZED ROUTING

Dijkstra [5], Bellman [33] and Ford [34] proposed some of the first routing planning algorithms. Routing algorithms such as A* [35] use heuristics to guide the shortest path search while contraction hierarchies [6] simplify the graph for faster search.

Current state of the art route planning is typically deployed in centralized cloud systems [6]–[8]. In this architecture, the routing algorithms are deployed in a central location from which it serves user queries. Within this context, QoS improvements (e.g., in terms of query response time) have been made by parallelizing shortest path algorithms [36]–[38]. These parallelized algorithms split processing over multiple nodes. These approaches provide high scalability, optimal for cloud-based services. However, these models assume a shared memory and do not take into account network latency between nodes, and therefore are not easily adaptable to edge or fog-centric architectures.

C. PRELIMINARY WORK

We designed a task allocation algorithm intended for decentralized route planning. We evaluated this design only using simulations, without devices-in-the-loop. All processing and delay computations were synthetic and processes were run in a single thread. In this article, we further develop this design and test them with devices-in-the-loop. All route and speed data are based on real-world data. The results and discussions in this study validate the results from the prior study in [39].

III. SYSTEM ARCHITECTURE

In [40] we proposed a middleware for heterogeneous IoT devices (sensors and actuators) that is capable of processing data streams in real-time and in a distributed manner. In this article, we introduce RSU-Edge, a realization of such a middleware for RSU devices. The primary goal of the RSU-Edge is task distribution and the distributed execution of the tasks over multiple IoT devices. In this section, we define what we assume to be a modern smart city. We then outline the RSU-Edge architecture and methods for data collection used for the distributed route planning service. Table 1 summarizes all the symbols used in this article.

TABLE 1. List of symbols.

Symbol	Description
G_r	Grids making up the target area
RSU_i	Road Side Unit i
R	Road Side Unit assigned to a grid
N	Network graph, $N = (V, E)$
V	Road intersections
E	Set of roads
d_i	Local speed data generated by sensors in a grid
Q_p	Set of queries sent within in the same time period p
G^i	Subgraph whose edge and vertex maps to grid i
	Query
q	Query sent by the user
s	User source location
d	User destination location
τ	User desired travel time
	Route Planning
SG_q	Sequence of grids (s to d) generated per query q
k_q	Multiplier based on q parameters
T_q	Sequence of tasks $t_{q,i}$ for each SG_q
$t_{q,i}$	Route planning tasks assigned to each grid in SG
	Task Allocation
T	Set of all tasks T_q for all queries Q_p
$x_{t,r}$	Assignment variable, 1 if task is assigned, 0 otherwise
$ST(t), ET(t)$	Task execution start and end times
D_{th}	Delay threshold for query responses
$TQ(th)$	Threshold for assigned queries per RSU
$IT(q)$	Time when q was issued
$ct(t, r)$	Computation time of task t executed at RSU r
MA	Model accuracy of the route planning model
$MD(g, g')$	Manhattan Distance between two different grids

A. SMART CITY

We assume that smart cities are equipped with sensors and computational resources that allow it to monitor and optimize their resources to maximize its services to its citizens. Fig. 1 shows our idea of a smart city. We assume that traffic lights and lamp posts, equipped with road side units, exist along the city roads and highways. We assume that mobility data, generated by passing vehicles, are sent to sensors nearby or within the RSUs.

The main components are as follows:

1) *Road Side Unit (RSU):* Low power computational nodes [41] located near roads and highways in the transport network. They are resource constrained, low-power devices. Each RSU is able to gather information such as speed, congestion, and flow of traffic from sensors and vehicles within their area of responsibility. Together, all RSUs form a fog network that is more private and reliable than traditional networks [42].
2) *Broker:* The primary role of the broker is as the central administrator of the RSUs. It generates, distributes, and aggregates messages to and from the RSUs. A broker can be deployed on computational nodes such as an RSU. These brokers can be deployed as the sole process of the RSU or co-exist with route planning processes due to their low computational needs. It is assumed that RSUs connect to the brokers via wired/wireless networking.

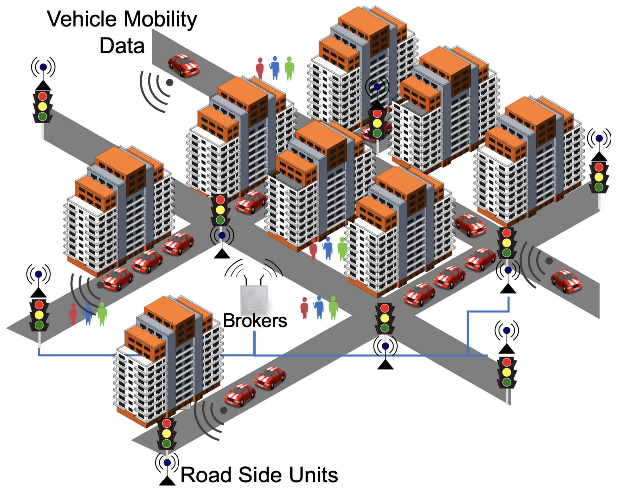

FIGURE 1. Smart city system architecture.

Broker to RSU and RSU to RSU are assumed to have wired/wireless network connectivity.

B. SPATIAL REGION

The target area is a city region converted into a network graph $N = (V, E)$, where the road intersections are its vertices V and its roads are the edges E. This is then divided into equidistant and similar shaped grids $G_r = \{g_1, g_2, \cdots, g_m\}$. Each grid is assumed to have an allocation of RSUs which handle road information and local data within the particular grid. Each RSU is assumed to be both computational and memory resource-constrained thus can only process a finite amount of tasks and data. This collection of RSUs forms a sub-network.

To work around RSU resource constraints, these grids can be further sub-divided into sub-grids. Each resulting sub-grid has to accommodate fewer tasks and sensor data. This process can be repeated as many times as needed until the desired sensor density per RSU is achieved.

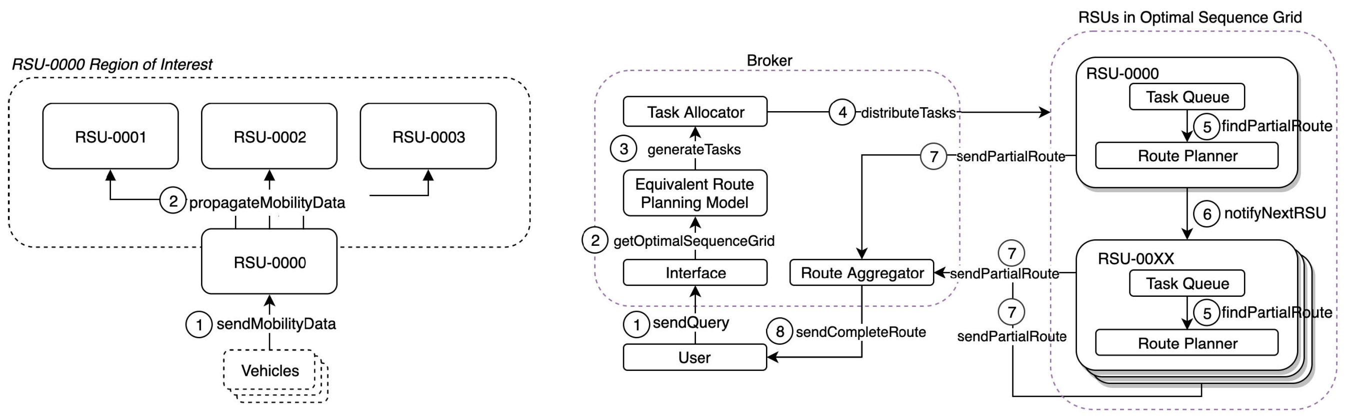

FIGURE 2. **(Left) Data gathering and propagation architecture of RSU-Edge. Each RSU gathers mobility data from vehicles in their area and then propagates them to other RSUs within their search area. (Right) Distributed Route Planning architecture of RSU-Edge and the flow of interaction between components. Once the user sends a query to the Broker,** `getOptimalSequenceGrid` **generates an optimal route for the query. The query is then divided into tasks which are then sent to corresponding RSUs by** `distributeTasks`**. Each** `findPartialRoute` **and** `sendPartialRoute` **is run in a sequential manner. Once all routes have been sent to the broker and aggregated, the user receives the complete route.**

C. DECENTRALIZED ROUTE PLANNING SERVICE

This service runs on the RSU-Edge that utilizes the distributed characteristics of the architecture. This system allows users access to time-dependent and privacy-preserved services for smart transportation. With respect to the problem of consistency, availability and partitioning for distributed systems, RSU-Edge is able to work under partitioning. Since RSUs, each with their local data, form their own sub-network, the service is able to maintain availability even with the loss of cloud services. Services that make use of geospatial data, such as smart grids and smart mobility, can effectively use this system. However, for this article, we focus on implementing a *decentralized route planning service*.

We assume that each grid contains several road segments and each road is assigned an RSU that handles all incoming speed data.[6] Each RSU $r_i \in R$ receives mobility data from vehicles on roads within their designated grids g_i.

Each RSU stores up-to-date speed data only for the roads of the grid it is assigned to. Each RSU propagates its aggregated data to neighboring RSUs at a set delay interval. Data shared with neighbors is assumed to not be as up-to-date as data found at the data source. This historical data distribution alleviates communication costs between RSUs while allowing the system to maintain a snapshot of the whole target area's speed information for route planning.

Our system consists of the following types of data:

1) *Mobility data:* Periodically collected up-to-date speed data from vehicles that pass along roads within the RSU sensor's range. These are stored as time-series data with both road speed and location information. These are used as the weights of the network data.
2) *Historical data:* These are assumed to be data that has been averaged and propagated to neighboring RSUs within a region of interest. This is not up-to-date data and are snapshots of neighboring RSUs data at a previous time frame.
3) *Network data:* Each RSU maintains a list of subgraphs, G^i, extracted from the global routing graph $N = (V, E)$.

[6] These RSUs are all connected but have one RSU as the representative of the grid, which we refer to in the paper.

Figure 2 (left) shows the methods used to propagate data between RSUs. Passing vehicles call ① `sendMobilityData` to the nearest RSU on the road. At some time interval or delay, each RSU will call ② `propagateMobilityData` to all the RSUs within its region of interest. All RSUs within the target area perform this.

D. QUERIES AND TASKS

Each RSU $r \in R$ is able to receive a query q with parameters $(id,\ s,\ d,\ \tau)$. id is used to differentiate queries while s and d are the user's start location and trip destination respectively. Finally τ is the departure time for the planned trip. Queries can be received asynchronously by any RSU. Queries received within the same time window p are denoted as $Q_p = \{q_{p,1}, q_{p,2}, \cdots, q_{p,n_i}\}$.

We assume that for every query, q there exists an optimal sequence of grids, SG, from s to d. We assume that each RSU has a trained model for identifying the next best grid given a set of inputs: current grid, source, destination, and time. By recursively utilizing this model from the source until the destination, we generate the optimal sequence grid, SG [43].

For every sequence grid, SG, a sequence of tasks $T_q = \langle t_{q,1}, t_{q,2}, \ldots, t_{q,k_q} \rangle$ is generated, where $t_{q,i}$ is the route planning task that passes through $g_{q,i} \in SG$. Task $t_{q,i}$ executes the heuristic generation of routes within a single grid, $g_{q,i}$, using various path searching algorithms such as Dijkstra's, to obtain the fastest time to traverse the particular grid.

Assigning tasks $t_{q,i}$ to the corresponding RSU of grid $g_{q,i}$ for all instances of i, yields the highest model accuracy since we assume this combination provides path searching algorithms access to the grid's up-to-date local data, to be used as weights.

IV. DISTRIBUTED ROUTE PLANNING

The distributed route planning service needs to provide a *shortest route* from a source to a destination for a given departure time. User's queries can be sent to any RSU which then serves as the broker for this particular query. The architecture is seen in Fig. 2, describes all major components of the platform. Each edge includes a circled number, i.e, ⊕.

A. DEFINITION OF THE PROBLEM

For a target area that is populated by a set of RSUs R, a set of user queries q is sent within a period p. For each q, an optimal route is generated by ② `getOptimalSequenceGrid`. The route is broken down to its corresponding grids and divided into separate tasks by ③ `generateTasks`. The set of all tasks T for all queries Q must be allocated to RSUs R for processing. The problem thus, identifying the most efficient and optimal allocation of tasks T to RSUs R. All task assignments should satisfy the constraints on query response delay while maximizing the overall accuracy of the result.

1) DELAY

We set Eq. 1 as the first constraint. We define T as the set of all tasks T_q of all queries $q \in Q_p$, sent at time period p.

$$T \triangleq \bigcup_{q \in Q_i} T_q \tag{1}$$

For every pair of task $t \in T$ and RSU $r \in R$, we define a variable $x_{t,r}$ which is 1 if t is assigned to r and 0 otherwise.

We assume that every task $t \in T$ is assigned to a single RSU, so the following condition must hold.

$$\forall t \in T, \sum_{r \in R} x_{t,r} = 1 \tag{2}$$

In each sequence of tasks T_q for query q, tasks must be sequentially executed. Hence the following equation must hold. Here, $ST(t)$ and $ET(t)$ represent task execution start and end times, respectively. Here $t_{q,i}$ is the current task being executed at grid i and $t_{q,i+1}$ is the next task in the sequence T_q, to be executed in the next grid $i+1$ in sequence. Here k_q is the last grid in the sequence for query q.

$$\forall T_q (q \in Q_i) \forall i (1 \leq i \leq k_q - 1)\, ET(t_{q,i}) < ST(t_{q,i+1}) \tag{3}$$

Upon assignment of all tasks in T, we define that the overall service delay for queries Q_i, should not exceed some delay threshold D_{th}. Here, $IT(q)$ is the time when the query is issued.

$$\forall q \in Q_i, ET(t_{k_q}) - IT(q) \leq D_{th} \tag{4}$$

Also, we define that the number of tasks t that can be queued on any single RSU r should not exceed the queue length threshold TQ_{th}. Here $TQ(r)$ is the number of tasks queued on RSU r.

$$\forall r \in R, TQ(r) \leq TQ_{th} \tag{5}$$

However, in cases where all RSUs have reached the TQ_{th}, the task is allocated to the least utilized available neighbor RSU. The purpose of Eq. 5 is to facilitate task load distribution over all the RSUs.

Each RSU r must first execute all tasks, within its task queue, ahead of task $t_{q,i}$. The worst-case execution time of task $t_{q,i}$ will be the sum of worst-case execution times of all tasks ahead of it in the queue.

Then, task execution start and end times of each task $t_{q,i}$ can be defined as follows.

$$ST(t_{q,i}) \overset{def}{=} IT(q) + \sum_{j=1}^{i-1} \sum_{r \in R} \sum_{t' \in T} ct(t', r) \cdot x_{t',r} \cdot x_{t_{q,j},r} \tag{6}$$

$$ET(t_{q,i}) \overset{def}{=} ST(t_{q,i}) + \sum_{r \in R} ct(t_{q,i}, r) \cdot x_{t,r} \tag{7}$$

where $ct(t, r)$ represents the computation time of task t executed at RSU r. These are given in advance.

Equation 6 defines that task, $t_{q,i}$ can only be executed after all prior tasks from the same query, $(t_{q,1}, \ldots, t_{q,i-1})$, have been processed. This means that the result of these tasks, such as *travel time* and *shortest paths*, has been generated.

2) ACCURACY

For every query, Q_i, a set of tasks T_q is generated based on the optimal sequence grid, $SG(q)$. Assigning tasks to the grids in SG produces the highest accuracy for route planning. However, due to the computational and memory constraints of the RSUs, tasks often take longer to be processed than to be assigned. As the task queue increases, the execution time of any additional tasks will increase as denoted by Equation 6. This increase in total execution time results in query response delays. A solution would be to allocate certain tasks onto less utilized but sub-optimal neighbor RSUs.

We assume that all RSUs only have up-to-date access to their local data and access to only stale data from neighboring ones. We assume that RSUs propagate their data to nodes within their region of interest, using gossip-based protocols [44]. The neighboring RSUs pass the received information, as well as their local data, to their neighbors. The size and rate of propagated data decreases the farther the receiving RSUs is from the data source. Thus tasks allocated to sub-optimal RSUs produce less up-to-date, but nonetheless correct route from s to d. The staleness of this data is given by the differences between the optimal and actual assignment locations as follows:

$$MA(g, g') \triangleq 1 - MD(g, g') \tag{8}$$

where $MD(g, g')$ is a factor of the Manhattan Distance between the optimal and actual grid assignments, while $MA(g, g')$ is the estimated model accuracy difference due to task allocation.

3) IMPACT OF ACCURACY AND DELAY

To demonstrate the impact of delay and accuracy on the route planning output, we use a sample route, Fig. 3, with 5 RSUs

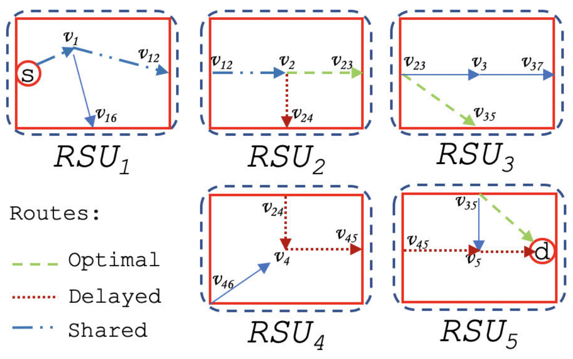

FIGURE 3. Decentralized Route Planning example to show the effects of accuracy and delay.

and a trip query from source s to destination d. An optimal route will have all its tasks allocated to optimal RSUs. However, this optimal route might exceed the D_{th} shown in Eq.4. If all the tasks are forced to optimal RSUs, up-to-date data is used resulting in high accuracy at the cost of processing time.

When tasks are re-allocated to sub-optimal RSUs such that the new allocation sequence does not exceed D_{th}, the route from s to d is still obtained. However, the data used may not be up-to-date, given by Eq. 8, resulting in a route of lower quality. A low-quality result does not mean an error in the routing, instead it means that the data being used to generate the route is not up-to-date resulting in a different route (dashed vs dotted line) to the same s to d. This increases the chances of using congested routes. The trade-off is that these sub-optimal RSUs will be able to process the tasks faster since they meet the D_{th} and/or TQ_{th} constraints.

4) UTILITY FUNCTION

Given the example above, we assume users have two requirements from the service. First, to receive a response within a preferable time delay and secondly, to receive it with an acceptable degree of accuracy. Based on these two requirements, we design the utility function $U(q)$ as follows. At every time i, the tasks T_q generated by a query q, should be assigned to RSUs such that it meets the constraints while maximizing the accuracy of the generated route.

$$U(q) = \sum_{j=1}^{k_q} \frac{MA(g_{q,j}, g(t_{q,j}))}{k_q} \tag{9}$$

5) OBJECTIVE FUNCTION

The purpose of distributed task allocation is to find the most optimal assignments of tasks to RSUs that satisfy the given constraints while maximizing the accuracy of the generated routes. The objective therefore is:

$$\textbf{Maximize: } \sum_{q \in Q_i} U(q) \text{ subject to } (2) - (5) \tag{10}$$

Algorithm 1 `getOptimalSequenceGrid`

```
Input: Source $s \in V$, Destination $d \in V$, Time: $\tau$
Output: Optimal sequence grid $OG$
 1: Initialize SeqGrids list
 2: $i \leftarrow 0$
 3: $SeqGrids[i] \leftarrow$ GetGrid($s$)
 4: $g_{final} \leftarrow$ GetGrid($d$)
 5: while $SeqGrids[i] \neq g_{final}$ do
 6:     $currentGrid \leftarrow$ SeqGrids[$i$]
 7:     $SeqGrids$[i+1] $\leftarrow$ GetNextGrid($currentGrid$, $g_{final}$, $\tau$)
 8:     $i \leftarrow i + 1$
 9: end while
10: SeqGrids[$i$] $\leftarrow g_{final}$
11: return SeqGrids
```

Algorithm 2 Get Next Grid

```
Input: Current Grid: $g_{curr}$, Destination: $g_{dest}$, Time: $\tau$
Output: Next Grid: $g_{next}$
 1: Get Equivalent Grid Routing model $\hat{E}$
 2: $g_{next} \leftarrow \hat{E}$.predict($g_{curr}$, $g_{dest}$, $\tau$)
 3: return $g_{next}$
```

B. REGION OF INTEREST HEURISTIC

To meet the objective function, our system needs to divide queries into tasks and then allocate them to RSUs such that accuracy is maximized while constraints are met. Region of interest controls the search area for RSUs that can be used to handle tasks that cannot be allocated to optimal RSUs because of over-utilization. Over-utilization occurs when the RSU exceeds either D_{th} or TQ_{th}, such that further allocation to it will cause delay on the system.

For our approach, we vary the region of interest using the neighbor levels. A level of 0 allocates tasks only to the most optimal RSUs as decided by SG. This prioritizes accuracy over the processing time. Figure 4 shows how the levels affect the region of interest. For levels 1 and 2, we increase the region of interest to the surrounding 8 and 24 RSUs around the optimal RSU respectively. The wider the region of interest, the greater the number of RSUs available for task reallocation. However, the wider region of interest also affects the distance between the optimal RSU and the selected RSU, given by Eq. 8.

C. DECENTRALIZED ROUTE PLANNING EXAMPLE

To demonstrate the overall execution of the route planning service, we use Fig. 3 which shows a network partitioned into 5 RSUs. This network is equipped with the RSU-Edge shown in Fig. 2. The information flow is initiated by a user calling ① `sendQuery` with parameters (id, s, d, τ_s) to RSU_1. The rest of the flow follows the numbered components in Fig. 2.

Upon receiving the query from the user, RSU_1 calls ② `getOptimalSequenceGrid`, which executes Algo. 1.

Algorithm 3 Sequence Grid Task Allocator

```
Input: Set of queries: $Q$
Output: Modified Grid: $MG$
1: for all $q \in Q$ do
2:     $OG_q \leftarrow$ getOptimalSequenceGrid($q.s, q.d, q.\tau$)
3:     if $Delay(OG_q) > D_{th}$ then
4:         $OG_q \leftarrow$ ModGSeq($OG_q$)
5:     end if
6:     distributeTasks($OG_q$)
7: end for
```

Algorithm 4 Modified Sequence Grid Generation

```
Input: Sequence Grid: $SG$, Neighbor Level: $L$
Output: Modified Grid: $MG$
1: $MG \leftarrow SG$
2: while $Delay(MG) > D_{th}$ and $|SG| \neq \emptyset$ do
3:     $g \leftarrow$ a grid randomly selected in $SG$
4:     $ns \leftarrow$ GetGridNeighbors($g, L$)
5:     $bg \leftarrow$ GetLeastUtilized($ns$)
6:     $MG \leftarrow$ modified $MG$ by replacing $g$ with $bg$
7:     $SG \leftarrow SG - \{g\}$
8: end while
9: return $MG$
```

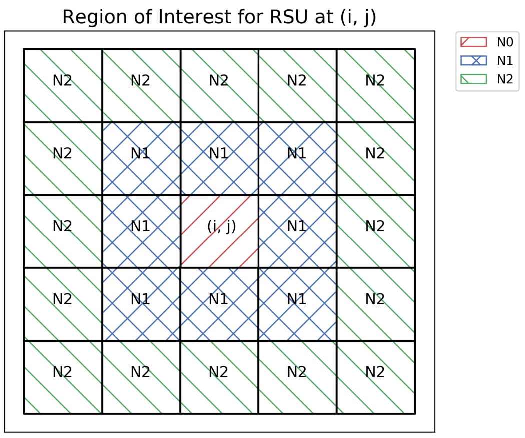

FIGURE 4. Region of interest based on varying Neighbor levels for RSU at the location (i, j). Neighbor level 1 includes the N0 grid at (i, j) and all N1 grids. Level 2 includes all grids N0, N1, and N2. As neighbor level increases, more grids become alternative grids.

This uses the source, destination and time parameter of the query as well as the *Equivalent Grid Routing model* ($\hat{E}$) [43], to recursively identify the optimal grid sequence SG, from the source s to the destination d. $\hat{E}$ is a routing model used by Algo. 2, that predicts the best neighboring grid through which the shortest path likely resides for a particular route. $\hat{E}(s, d, \tau_s)$ returns the next best possible grid to travel to destination d from s at time τ_s.

Here, the optimal sequence grid generated is as follows: $[RSU_1, RSU_2, RSU_3, RSU_5]$. A set of tasks for this sequence is then generated by ③ `generateTasks`. These tasks are then run through the task allocator described by Algo. 3. Algorithm 3 first assigns tasks to the optimal sequence grid, and then it measures the total delay given this configuration. If such a configuration causes a delay greater than D_{th}, we modify the sequence grid using Algorithm 4.

In Algorithm 4, we pick the most utilized RSU in the optimal grid sequence SG, and then we select an RSU within its region of interest. We move this RSU's task assignment to the least utilized RSU in that region of interest, modifying the current sequence of grids in response. The total delay of this new sequence grid is again checked against D_{th}.

This process is repeated until the total response time for the query is less than D_{th}. We then assign the tasks to this sequence of grids, making sure to consider the task queue constraint in Eq. 5, during the allocation.

In this example, we assume that the modified sequence grid is now $[RSU_1, RSU_2, RSU_4, RSU_5]$. The two-fold impact on accuracy and delay of the allocation algorithm was discussed in Section IV-A3. Once the task allocation algorithm has verified that constraints have been met or search has been exhausted, tasks are distributed to the RSUs following the modified sequence grid with a call to ④ `distributeTasks`.

Tasks are distributed in parallel and executed sequentially (Eq. 3). RSU_1 processes the task once it appears at the start of the queue. It generates a partial route with a call to ⑤ `findPartialRoute` and sends it to the broker for aggregation with ⑦ `sendPartialRoute`. At the same time, it notifies the next RSU in sequence, RSU_2, via ⑥ `notifyNextRSU` and the process continues. Once the final RSU, RSU_5 sends its partial route, the broker calls ⑧ `sendCompleteRoute` to send the final route to the requesting user.

V. EXPERIMENT AND RESULTS

In this section, we evaluate our distributed route planning service over RSU-Edge. We evaluate the system in two phases. First, we revisit and update the simulation done in the previous paper which evaluates the system without devices in the loop. This is done to prove the feasibility of the system as well as to identify the parameters that would be used in the next phase of evaluation. In the second phase, we evaluate the approach on real-world data where we implement the RSU-Edge and RSUs on Docker containers. The goal of these experiments is to identify the effects of task allocation algorithm on processing delay, accuracy, and generated routes.

A. PHASE 1: FEASIBILITY TEST AND PARAMETER IDENTIFICATION

In [39], we simulated 600 RSUs, and used 100 trip queries to evaluate it. We revisit that scenario but increase the number of queries and give a clearer view of the effect of neighbor levels which we validate afterward in Phase 2.

Based on the top 10 employers in Nashville which have nearly 140,000 employees [45] and assuming people have variable work shifts with a variance in departure time of one hour, our system needs to be able to process around 12,000 queries every 5 minutes.

We simulate the division of these user queries into tasks and then allocate these tasks to 600 different RSUs. We vary the neighbor levels and identify the effect the distribution of 230,000 tasks (generated from the 12,000 queries) has on overall query response time and model accuracy.

Processing starts once all tasks have been allocated. Processing time is measured in cycles. Total processing time is based on the number of cycles needed before all tasks in all RSU's task queue have been processed. A task is processed by removing it from the RSU's task queue if it is flagged as done, else it is left there to be checked again in the next cycle.

We define each cycle as one loop across all RSUs. Since tasks must be processed in sequence, task $t_{q,i-1}$ must be executed before task $t_{q,i}$. For every cycle, we loop through each RSU with a non-empty task queue and we get its task at the start of the queue and flag it as done if it is the starting task of the query, $t_{q,0}$ or if the previous tasks from the same query $t_{q,i-1}$, has been done. A task that is done is removed from the task queue. A single cycle is done once all RSUs with non-empty task queues have been checked. The cycle repeats as long as there are RSUs with non-empty task queues.

At the end of every cycle, tasks flagged as done are reviewed. A query q is finished when all of its tasks, $T_q = \langle t_{q,1}, t_{q,2}, \ldots, t_{q,k_q} \rangle$, have been flagged as done.

When forcing tasks to be allocated optimally (level 0), certain RSUs have more than 2000 tasks allocated to it. Increasing the search grid for task allocation allows tasks to be assigned to more RSUs, resulting in a much more balanced distribution of tasks. This is reflected in the total task processing time. Figure 5 shows that using at least neighbor level 1, all tasks finish in less than half the cycles of neighbor level 0.

Due to the increase in the number of possible RSUs to assign tasks to, the average Manhattan distance between the optimal and actual RSUs used increases the higher the neighbor level. Figure 6 shows the relationship between Manhattan distance and average cycles per query completion for different neighbor levels.

B. PHASE 2: REAL-WORLD DATA AND SCALABILITY

The goal is to investigate the feasibility of the approach in Phase 1. We compare the performance of our service's task allocation algorithm to the intuitive case of using only the optimal allocations to have the highest model accuracy. We use total query processing time and route travel time accuracy as performance metrics to show the effect of our algorithm on the service.

We perform simulations and experiments on Docker containers to be able to easily test and adjust the parameters of the system and algorithms while also keeping scalability in mind. The number of grids and queries used in this experiment has been reduced since the actual route generating processes will be done on containers on a single physical machine. Table 2 shows the devices that were used or tested in this section.

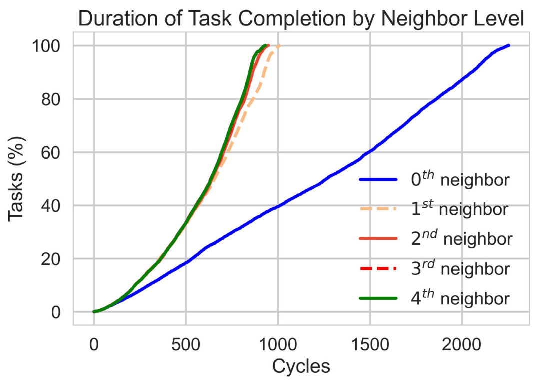

FIGURE 5. Synthetic Processing time for all 12,000 queries. Neighbor Level 0 takes more than 2 times longer than when utilizing neighbor grids.

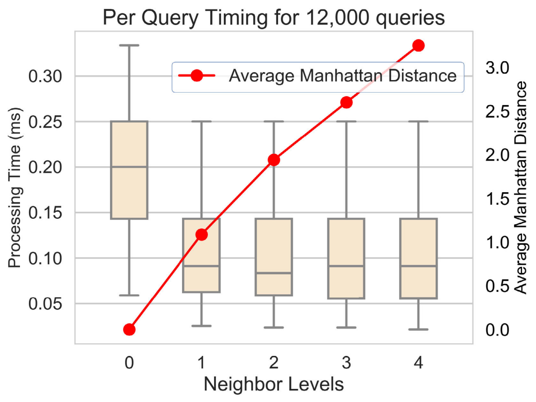

FIGURE 6. The trade-off when utilizing neighboring nodes to decrease query response time is the increase in Manhattan Distance between optimal and allocated RSUs, resulting in a decrease in model accuracy.

TABLE 2. Comparison of computational resources of devices used in the experiment.

Device	CPU	RAM
Mac mini	3 GHz Intel Core i5	64 GB DDR4
Mac pro	2.9 GHz Intel Core i7	16 GB DDR3
RPi 3B [46]	1.5GHz Quad core Cortex-A72	1GB LPDDR4-3200
RSU [47]	800 MHz Dual core iMX6	1GB

A sample specification for road side units is included here.

From 600 RSUs, we choose a subset of 49 RSUs. We match this reduction in the number of processing nodes to the number of queries. We pick 1,000 queries from a total of 12,000 queries. This allows us to focus more on understanding how communication between RSUs occurs while maintaining the query to RSU ratio we used in the simulation experiments. For this experiment, we assume that RSUs have the similar computational capacity to Raspberry Pi-like devices.

1) CONTAINER BENCHMARKING

Since we assume the RSU to be similar to a Raspberry Pi level device, we verify the difference in computational capacities of our Docker containers and a single Raspberry Pi. For the following experiments, we use a single Mac mini hosting all 49 RSUs via Docker containers is used. This was used without any limitations on its computational and memory resources. We run a route planning benchmark on all 49 RSUs simultaneously and then on a single Raspberry Pi. The benchmark runs 1,000 queries on the devices and we get the average time it takes to complete all queries as a measurement of the device's computational capacity.

We tested two different devices, a Mac mini, and a Macbook Pro, as the central physical host machines for containers. Figure 7 is the result of running route planning benchmarks on the devices.

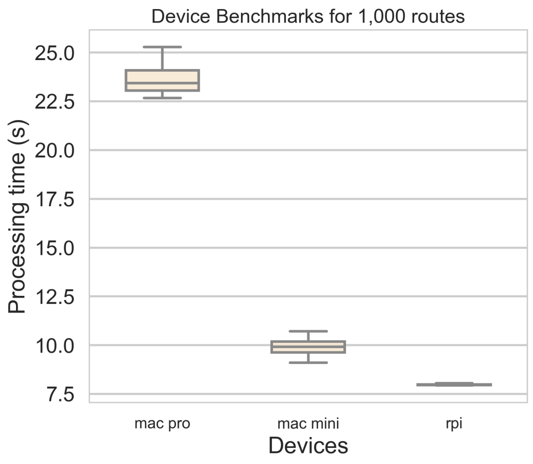

FIGURE 7. **Comparing the overall processing time for 1000 trip queries using Docker containers and Raspberry Pi 3B. Mac mini and Macbook Pro devices are running 49 containers simultaneously and benchmarks are run simultaneously on all containers.**

The results show that the 49 RSUs being emulated on Docker containers on the Mac mini have a similar computational capacity as Raspberry Pi 3B. For all subsequent experiments, a Mac mini hosting all 49 Docker containers emulating RSUs, will be used.

2) EXPERIMENT PARAMETERS

- *Network Graph*: The target area, shown in Figure 8, is a 20 km $\times$ 20 km area in Nashville, TN. A network graph, $N = (V, E)$, is constructed with geometries and data supplied by HERE API [48]. The region used in this study has a total of 1924 nodes and 8522 edges.
- *Graph Partitions*: The network graph is first divided into a $\mathbf{5 \times 5}$ grid. However as Figure 8 shows, some grids still have a higher density of roads and sensors. These grids were then sub-divided into a further $\mathbf{2 \times 2}$ grid. Figure 8 shows the division as well as the density of roads and sensors assigned to a particular grid. Each grid has a size of 15 km^2, while each sub-grid is 3.75 km^2.
- *Road Side Units*: 49 RSUs are used and deployed. RSUs are assumed to be static and connected via a wired network.

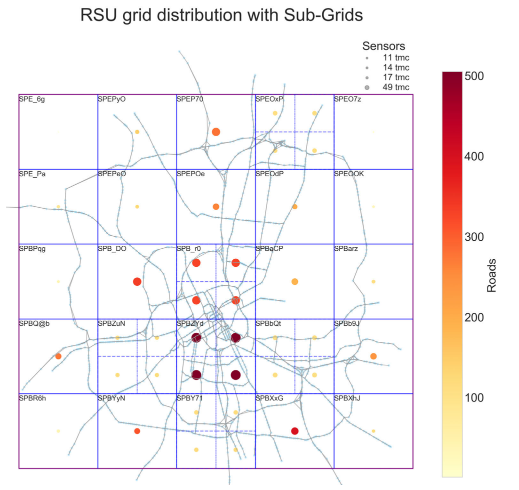

FIGURE 8. **The target area is divided into a 5 $\times$ 5 grid layout. Each over-utilized grid is further divided into sub-grids. The bar shows the density of road segments in the grid.**

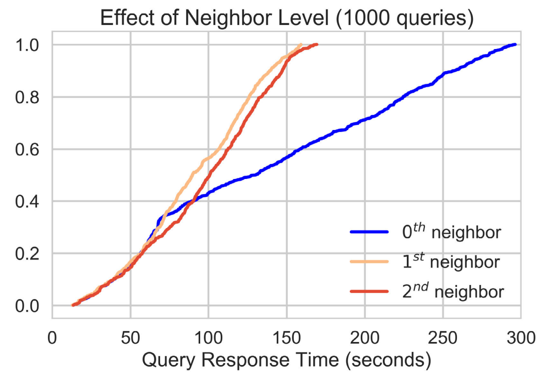

FIGURE 9. **CDF curves showing the effect of Neighbor Levels on the query response times.**

- *Mobility data*: To simulate traffic across roads, data collected by HERE API for the region on March 2018 is used. These data are collected by sensors placed near road segments. We assume these sensors gather data and send them to representative RSUs for aggregation and processing. This data is logged in one-minute intervals which are then averaged into one-hour time windows, resulting in 24-speed data entries per RSU per day. These are used as the edge weights for the network graph above.
- *Trip Query data*: From the network graph, we pick 1,000 uniformly distributed source and destination pairs along with randomly selected departure times. We assume these queries are sent within five minutes, aggregated, and then processed.
- *Constraints*: For all the subsequent experiments, TQ_{th} is set at 100.

C. EXPERIMENT EVALUATION

Processing time, $ct(t', r)$, is measured as the time it takes for a sent query to be responded to with the final route. These

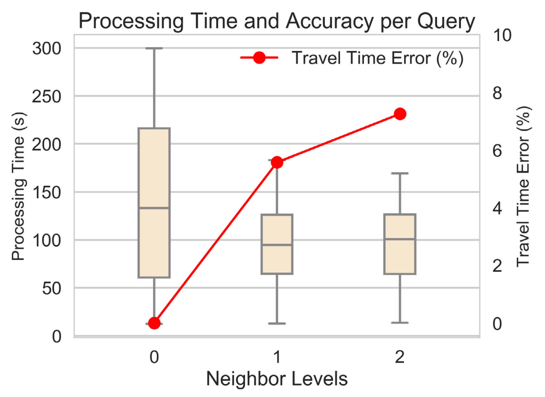

FIGURE 10. Effect of utilizing neighbor nodes during task allocation on total query time.

include the generation of the optimal sequence grid, task allocation, and task execution. Accuracy is measured in terms of total travel time per route.

To quantify the effects of varying neighbor levels on task allocation, processing time, and accuracy, we send 1,000 trip queries to the broker. All queries are sent to a single broker and it performs all sequence grid (*SG*), generation as well as task allocations.

1) TASK ALLOCATION

Task allocation time is negligible when compared to the total processing time as shown in Table 3. Total task allocation for 1000 queries, is 1.7 seconds (the execution time of Algorithms 1 and 3) while total processing time takes 145 seconds.

TABLE 3. Effect of number of queries on allocation and processing (neighbor level 1).

Query Count	Allocation Time	Total Processing Time
100	0.4 s	37 s
200	0.7 s	54 s
500	2 s	111 s
1000	1.7 s	145 s
2000	8.6 s	360 s

Figure 11 shows the breakdown of how sub-tasks are allocated to the different RSUs. Certain high usage RSUs that have sub-grids have their tasks distributed internally. With neighbor level 0, these high usage RSUs have five times the number of tasks allocated to it while some RSUs such as SPE_{6g} have little to no allocated tasks. In neighbor level 0, there is a direct correlation between the number of tasks and the number of roads in an RSU.

2) PROCESSING TIME

Figure 9 shows how much faster processing times are when utilizing neighbor grids. Without using neighbor grids, the service gets overloaded when around 35% of the queries are being processed. On the other hand, using neighbor grids allow the service to comfortably handle subsequent tasks. The similarities between Neighbor levels 1 and 2 are due to the computational limitations of the host device.

The processing time for each query includes the delay it takes for messages to be sent between a broker and RSUs. An average message between broker and RSU has 0.5KB of information, while a response to user queries has 0.7KB.

Messages sent from broker to RSUs contain the following information:

- Unique Task ID
- RSU ID allocation
- Next RSU in sequence
- Source
- Destination
- Departure time

In their response to Brokers, RSUs add the following to the message:

- Partial Route
- Partial route travel time

Query response messages contain:

- Unique Query id
- Final route
- Final route travel time

Assuming that RSUs and brokers have a wired/wireless connection such as LTE, this message will be sent in milliseconds and thus negligible. Similarly, processed responses will be sent back in milliseconds using Dedicated Short Range Communication (DSRC) or LTE connectivity.

3) ACCURACY

Tasks are processed faster when using neighbor levels 1 and 2. This is because tasks are allocated to less utilized but less optimal RSUs, resulting in a decrease in model accuracy. We measure this accuracy as the overall travel time for a generated route. We assume that routes generated by with neighbor level **0** have 100% accuracy. Each task allocated differently from the most optimal RSU penalizes the data being obtained. This penalty is a function of the Manhattan distance, between optimal and assigned RSU as described in Equation 8.

To simulate this loss of accuracy, we define that the Manhattan distance between optimal and assigned RSU be equivalent to the staleness of the speed data in minutes. In prior experiments, a Manhattan distance of 1 means that only speed data from 60 minutes ago will be available to that assigned RSU to be used in generating routes.

We assume that this staleness of data is an adequate measure of possible inaccuracies due to sub-optimal task allocations. Figure 10 shows that the trade-off for decreasing query response time by around 50% is a 5.5% loss in model accuracy. While Neighbor level 1 is providing an acceptable trade-off, Neighbor level 2 shows no substantial decrease in processing time to justify the additional 2% of loss. This loss in accuracy is primarily due to the Manhattan distance between optimal and assigned RSUs as shown in Figure 12. With neighbor level 2, almost 400 tasks that were assigned

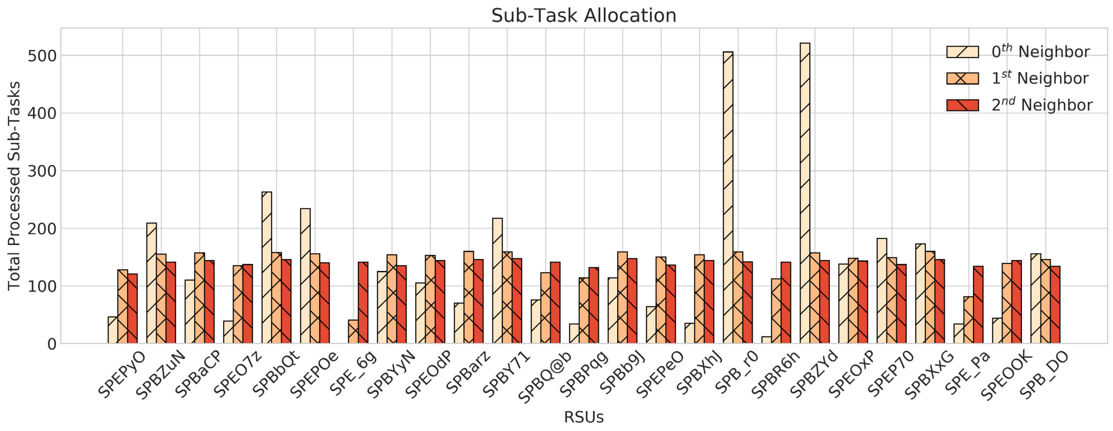

FIGURE 11. Task allocation.

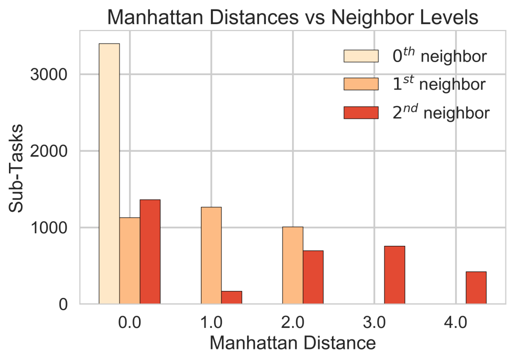

FIGURE 12. Histogram of manhattan distances based on Neighbor Levels.

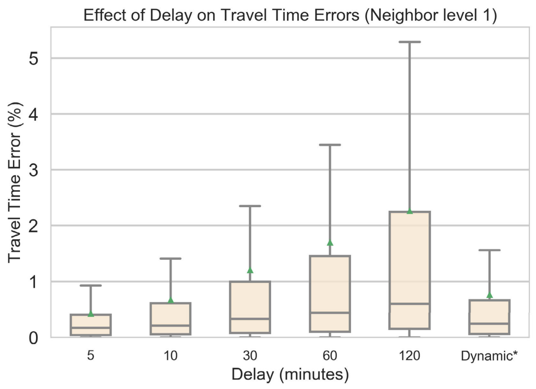

FIGURE 13. Effect of varying delay on average travel time errors. The rightmost point (Dynamic) is based on assigning different delays based on road speed changes. Nodes get speed updates in increments of 5, 10, 30, and 60 minutes based on how often the road speed changes.

to an RSU four Manhattan distances away from its optimal allocation.

The 60-minute delays were based on the assumed speed with which data is propagated to neighbor nodes. We vary the delay of speed updates between neighbor nodes to verify its effect on the overall accuracy of the system. Figure 13 shows that as we increase the delay between speed updates, the average travel time error per trip increases. Also, we defined dynamic speed updates based on how often the road speed changes. The goal for this dynamic method of updating speed data is to obtain a balance between the number of messages between RSUs and the overall travel time accuracy. In this method, we divided the roads into four distinct categories, and based on the frequency of speed changes throughout the day, we assign them varying speed updates ranging from 5, 10, 30, and 60 minutes.

This gives a better description of the effect of road speed changes on the travel time. The drawback would be the communication costs between nodes as updates are sent between them.

4) CONCURRENT QUERY COUNT

We have seen that there is a substantial decrease in processing time when using neighbor levels 1 and 2 over neighbor level 0. However, a median of almost 100 seconds of processing time for neighbor level 1 is still quite large. Here we test varying concurrent queries on the system with a neighbor level 1.

Figure 14 shows how long it takes for different concurrent queries to be completely processed. Processing time increases tenfold when increasing the number of concurrent queries from 200 to 2,000. Table 3 shows the time it takes different numbers of concurrent queries to be queried, allocated completely, and finally processed.

5) ROUTE GENERATION

For 1,000 trip queries, the majority of the trips have the same routes regardless of the neighbor level. This is either due to tasks being allocated optimally or speed data is close enough that there is no need for rerouting. These instances occur in more than half of the trips. For these trips the average difference in trip travel times for neighbor levels 1 and 2 are **3.5**% and **4.5**% respectively.

However, for the remaining trips, the routes differ because the staleness of the data was large enough to affect the route

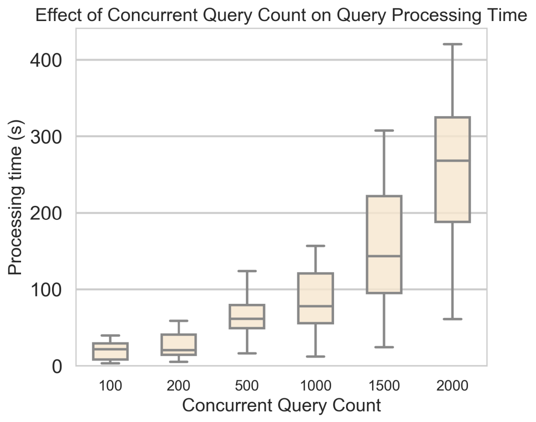

FIGURE 14. Box plots showing the effect of the number of concurrent queries sent on the query response times of the system (Neighbor level 1).

planning. For these trips, users had to be routed through different roads. For these trips the average difference in travel times for neighbor levels 1 and 2 are **12.7%** and **18.11%** respectively. Although higher, these differences occurred only in 81 trips out of 1,000.

Figure 15 shows a worst-case scenario due to data staleness. Each route passes through the same set of grids; however, the route generated by level 2 diverges the most, which is to be expected, especially on roads where traffic changes more drastically. This particular route passes through downtown, which we assume to have very large speed changes throughout the day. Due to the staleness of available data due to task reallocation, the final route differs between neighbor levels. In this scenario level 1 and 2 had a difference in travel times of **7.37%** and **31.81%** respectively.

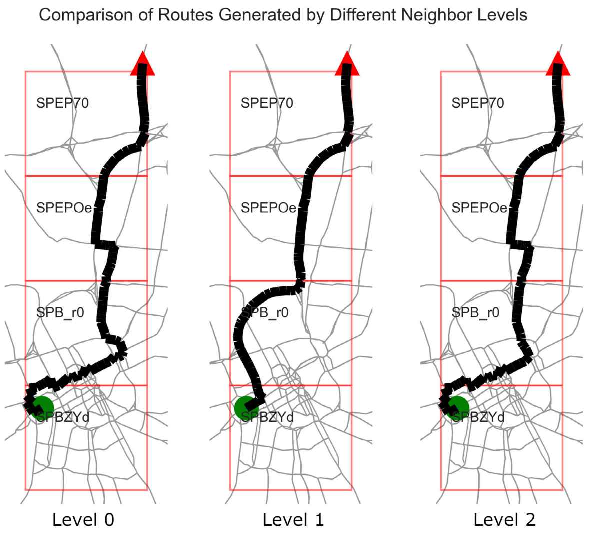

FIGURE 15. Routes generated with varying neighbor levels. Circle and triangle are source and destination respectively. Only the 2nd half of the route is shared by all.

VI. DISCUSSION AND LIMITATIONS

A. DISCUSSION

We showed in Figure 7 that emulating 49 RSUs on a Mac mini gave similar results to Raspberry Pi-like devices. Running all experiments using this configuration, gives us an idea of how a Raspberry Pi or a similar device would perform as an RSU given the same parameters.

We checked how characteristics of a decentralized route planning service would change depending on varying parameters. In Section V-C, we found that by simply increasing the region of interest for available RSUs, we can decrease the processing time. Figure 10 shows that by including neighbor grids as possible reallocation alternatives, the overall processing time for user queries is decreased by **50%**. Section V-C3 shows that while this task allocation algorithm offers the benefit of decreased processing time, there is a trade-off with model accuracy. As tasks get allocated farther and farther away from the most optimal RSU, the available speed data becomes staler. However, Figure 10 shows that for 1,000 queries the total travel time accuracy is decreased by an average of only **7%**.

Finally, we tested the actual route being generated by the service in Section V-C5. We found that utilizing neighbor nodes did not have a large negative effect on the actual route planning algorithm. Out of 1,000 trip queries, more than half was given the same route regardless of the travel time difference which had an average of **4.5%** decrease in model accuracy. Less than 10% of the trips had a route that was affected by the staleness of available data due to the distance between optimal and assigned RSUs. Of these trips, the decrease in accuracy is **12.7%** and **18.11%** for neighbor levels 1 and 2 respectively. The decrease in accuracy does not impact the correctness of the route, only the timeliness of the data being used. Lower quality or less accurate route increases the chances that the route will pass through congested roads.

B. LIMITATIONS

The first limitation of our study was the way we stored local speed data. Depending on the transport network density, speed data could easily reach millions of rows of data. While an updating database of time-series data is preferred, we found that it consumes too much processing time (when accessing stored data files) or waits for too long (database querying) for the response to be sent in close to real-time.

To work around this and still be able to provide speed data for route planning, we averaged data over the month into 1,440 data points (per minute, per day) and stored them as a look-up table. While real-time updating local data provides significant improvements in actual route generation, this is beyond the scope of this study.

The second limitation was the way the optimal sequence grids are generated. We assume that there is only a single optimal route per trip query. For trips with multiple optimal feasible routes, we must update the Equivalent Grid Routing model ($\hat{E}$). Updating the model to handle such trips and routes is beyond the scope of this article.

VII. CONCLUSION AND FUTURE WORK

We proposed a decentralized route planning service that runs on RSU-Edge. This service provides the shortest route from a source to a destination. To achieve the highest accuracy, tasks must be allocated to the most optimal RSUs defined by the optimal grid sequence. However, due to resource constraints, allocating all tasks to their optimal RSUs will result in delays to the overall query response.

The problem is how to ensure that all task assignments satisfy query response constraints while maximizing the overall accuracy of the result. To solve this, we proposed a distributed task allocation algorithm that identifies the most optimal assignments to meet accuracy and time requirements.

Evaluating the system on real-world data, we measured the efficiency of the system based on various parameters such as the region of interest, concurrent queries, and computational capacity. We found potential trade-offs between overall query processing time and model accuracy when using different neighbor levels. By having a wider region of interest for available RSUs, we decrease processing time by **50%** with an average of **7%** decrease in model accuracy.

As future work, we must verify the performance of the middleware and distributed route planning service. To accomplish that, the system must be deployed and validated with real-world test cases. The actual deployment of this system introduces new challenges that must be solved such as geographical distances and communication delays. Thus, the network configuration and architecture must be carefully planned. In Japan, ITS (Intelligent Transport Systems) are steadily expanding with the popularization of ETC (Electronic Toll Collection) systems. With ETC 2.0 [49], vehicles have the ability for V2V and vehicle to RSU communication. We should consider how our architecture will be deployed in such an environment.

REFERENCES

[1] D. Grewe, M. Wagner, M. Arumaithurai, I. Psaras, and D. Kutscher, ''Information-centric mobile edge computing for connected vehicle environments: Challenges and research directions,'' in *Proc. Workshop Mobile Edge Commun. (MECOMM)*. New York, NY, USA: Association for Computing Machinery, 2017, pp. 7–12, doi: 10.1145/3098208.3098210.

[2] X. Ge, ''Ultra-reliable low-latency communications in autonomous vehicular networks,'' *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 5005–5016, May 2019.

[3] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, ''A survey on Internet of Things: Architecture, enabling technologies, security and privacy, and applications,'' *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1125–1142, Oct. 2017.

[4] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, ''Fog computing and its role in the Internet of Things,'' in *Proc. 1st Ed. MCC Workshop Mobile cloud Comput.*, 2012, pp. 13–16.

[5] E. W. Dijkstra, ''A note on two problems in connexion with graphs,'' *Numer. Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[6] P. Sanders and D. Schultes, ''Engineering highway hierarchies,'' in *Proc. Eur. Symp. Algorithms*. Berlin, Germany: Springer, 2006, pp. 804–816.

[7] R. Geisberger, P. Sanders, D. Schultes, and D. Delling, ''Contraction hierarchies: Faster and simpler hierarchical routing in road networks,'' in *Proc. Int. Workshop Exp. Efficient Algorithms*. Berlin, Germany: Springer, 2008, pp. 319–333.

[8] A. V. Goldberg and C. Harrelson, ''Computing the shortest path: A search meets graph theory,'' in *Proc. 16th Annu. ACM-SIAM Symp. Discrete Algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2005, pp. 156–165.

[9] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, ''Internet of Things (IoT): A vision, architectural elements, and future directions,'' *Future Generat. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, 2013.

[10] S. Shekhar, A. D. Chhokra, A. Bhattacharjee, G. Aupy, and A. Gokhale, ''INDICES: Exploiting edge resources for performance-aware cloud-hosted services,'' in *Proc. IEEE 1st Int. Conf. Fog Edge Comput. (ICFEC)*, May 2017, pp. 75–80.

[11] P. G. Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, ''Edge-centric computing: Vision and challenges,'' *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 37–42, 2015.

[12] J. Ren, G. Yu, Y. He, and G. Y. Li, ''Collaborative cloud and edge computing for latency minimization,'' *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 5031–5044, May 2019.

[13] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, ''A survey on the edge computing for the Internet of Things,'' *IEEE Access*, vol. 6, pp. 6900–6919, 2018.

[14] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu, ''Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system,'' *IEEE Trans. Comput.*, vol. 65, no. 12, pp. 3702–3712, Dec. 2016.

[15] A. Moschitta, M.-Q. Tran, D. T. Nguyen, V. A. Le, D. H. Nguyen, and T. V. Pham, ''Task placement on fog computing made efficient for IoT application provision,'' *Wireless Commun. Mobile Comput.*, vol. 2019, Jan. 2019, Art. no. 6215454, doi: 10.1155/2019/6215454.

[16] A. Dubey, S. Pradhan, D. C. Schmidt, S. Rusitschka, and M. Sturm, ''The role of context and resilient middleware in next generation smart grids,'' in *Proc. 3rd Workshop Middleware Context-Aware Appl. IoT (M4IoT)*, 2016, pp. 1–6.

[17] S. Pradhan, A. Dubey, S. Khare, S. Nannapaneni, A. Gokhale, S. Mahadevan, D. C. Schmidt, and M. Lehofer, ''Chariot: Goal-driven orchestration middleware for resilient IoT systems,'' *ACM Trans. Cyber-Phys. Syst.*, vol. 2, no. 3, p. 16, 2018.

[18] S. M. Pradhan, A. Dubey, A. Gokhale, and M. Lehofer, ''CHARIOT: A domain specific language for extensible cyber-physical systems,'' in *Proc. Workshop Domain-Specific Modeling (DSM)*, 2015, pp. 9–16.

[19] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, ''Internet of Things: A survey on enabling technologies, protocols, and applications,'' *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 2347–2376, Jun. 2015.

[20] M. A. A. da Cruz, J. J. P. C. Rodrigues, A. K. Sangaiah, J. Al-Muhtadi, and V. Korotaev, ''Performance evaluation of IoT middleware,'' *J. Netw. Comput. Appl.*, vol. 109, pp. 53–65, May 2018, doi: 10.1016/j.jnca.2018.02.013.

[21] A. Salman, I. Ahmad, and S. Al-Madani, ''Particle swarm optimization for task assignment problem,'' *Microprocess. Microsyst.*, vol. 26, no. 8, pp. 363–371, Nov. 2002.

[22] Q. Zhu and G. Agrawal, ''Resource provisioning with budget constraints for adaptive applications in cloud environments,'' *IEEE Trans. Services Comput.*, vol. 5, no. 4, pp. 497–511, 4th Quart., 2012.

[23] M. Mao and M. Humphrey, ''Auto-scaling to minimize cost and meet application deadlines in cloud workflows,'' in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal. (SC)*, 2011, pp. 1–12.

[24] D. Ardagna, M. Ciavotta, and M. Passacantando, ''Generalized Nash equilibria for the service provisioning problem in multi-cloud systems,'' *IEEE Trans. Services Comput.*, vol. 10, no. 3, pp. 381–395, May 2017.

[25] C. Catlett *et al.*, ''Teragrid: Analysis of organization, system architecture, and middleware enabling new types ofapplications,'' in *High Performance Computing and Grids in Action* (Advances in Parallel Computing), vol. 16. IOS Press, 2008, pp. 225–249.

[26] D. Schafer, J. Edinger, J. M. Paluska, S. VanSyckel, and C. Becker, ''Tasklets: Better than best-effort computing,'' in *Proc. 25th Int. Conf. Comput. Commun. Netw. (ICCCN)*, 2016, pp. 1–11.

[27] O. Skarlat, S. Schulte, M. Borkowski, and P. Leitner, ''Resource provisioning for IoT services in the fog,'' in *Proc. IEEE 9th Int. Conf. Service-Oriented Comput. Appl. (SOCA)*, Nov. 2016, pp. 32–39.

[28] O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar, ''Towards QoS-aware fog service placement,'' in *Proc. IEEE 1st Int. Conf. Fog Edge Comput. (ICFEC)*, May 2017, pp. 89–96.

[29] J. Xu, B. Palanisamy, H. Ludwig, and Q. Wang, "Zenith: Utility-aware resource allocation for edge computing," in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, Jun. 2017, pp. 47–54.

[30] Y. Nakamura, T. Mizumoto, H. Suwa, Y. Arakawa, H. Yamaguchi, and K. Yasumoto, "*In-situ* resource provisioning with adaptive scale-out for regional IoT services," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Oct. 2018, pp. 203–213.

[31] J. P. Talusan, F. Tiausas, K. Yasumoto, M. Wilbur, G. Pettet, A. Dubey, and S. Bhattacharjee, "Smart transportation delay and resiliency testbed based on information flow of things middleware," in *Proc. IEEE Int. Conf. Smart Comput. (SMARTCOMP)*, Jun. 2019, pp. 13–18.

[32] S. Eisele, T. Eghtesad, N. Troutman, A. Laszka, and A. Dubey, "Mechanisms for outsourcing computation via a decentralized market," in *Proc. 14th ACM Int. Conf. Distrib. Event-Based Syst.*, Jul. 2020, pp. 61–72.

[33] R. Bellman, "On a routing problem," *Quart. Appl. Math.*, vol. 16, no. 1, pp. 87–90, 1958.

[34] L. R. Ford, Jr., "Network flow theory," Rand Corp., Santa Monica, CA, USA, Tech. Rep. P-923, 1956.

[35] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. SSC-4, no. 2, pp. 100–107, Jul. 1968.

[36] G. Di Stefano, A. Petricola, and C. Zaroliagis, "On the implementation of parallel shortest path algorithms on a supercomputer," in *Proc. Int. Symp. Parallel Distrib. Process. Appl.* Berlin, Germany: Springer, 2006, pp. 406–417.

[37] Y. Tang, Y. Zhang, and H. Chen, "A parallel shortest path algorithm based on graph-partitioning and iterative correcting," in *Proc. 10th IEEE Int. Conf. High Perform. Comput. Commun.*, Sep. 2008, pp. 155–161.

[38] A. Crauser, K. Mehlhorn, U. Meyer, and P. Sanders, "A parallelization of Dijkstra's shortest path algorithm," in *Proc. Int. Symp. Math. Found. Comput. Sci.* Berlin, Germany: Springer, 1998, pp. 722–731.

[39] J. P. Talusan, M. Wilbur, A. Dubey, and K. Yasumoto, "On decentralized route planning using the road side units as computing resources," in *Proc. IEEE Int. Conf. Fog Comput. (ICFC)*, Apr. 2020, pp. 1–8.

[40] Y. Nakamura, H. Suwa, Y. Arakawa, H. Yamaguchi, and K. Yasumoto, "Design and implementation of middleware for IoT devices toward real-time flow processing," in *Proc. IEEE 36th Int. Conf. Distrib. Comput. Syst. Workshops (ICDCSW)*, Jun. 2016, pp. 162–167.

[41] F. Perry, K. Raboy, E. Leslie, Z. Huang, and D. Van Duren, "Dedicated short-range communications roadside unit specifications," United States Dept. Transp., Washington, DC, USA, Tech. Rep. FHWA-JPO-17-589, 2017.

[42] H. Madsen, B. Burtschy, G. Albeanu, and F. Popentiu-Vladicescu, "Reliability in the utility computing era: Towards reliable fog computing," in *Proc. 20th Int. Conf. Syst., Signals Image Process. (IWSSIP)*, Jul. 2013, pp. 43–46.

[43] M. Wilbur, C. Samal, J. P. Talusan, K. Yasumoto, and A. Dubey, "Time-dependent decentralized routing using federated learning," in *Proc. IEEE 23rd Int. Symp. Real-Time Distrib. Comput. (ISORC)*, May 2020, pp. 56–64.

[44] A.-M. Kermarrec, L. Massoulie, and A. J. Ganesh, "Probabilistic reliable dissemination in large-scale systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 3, pp. 248–258, Mar. 2003.

[45] Department of Finance. (2019). *Comprehensive Annual Financial Report for the Year Ended*. Accessed: May 21, 2020. [Online]. Available: https://www.nashville.gov/Portals/0/SiteContent/Finance/docs/CAFR/CAFR%202019.pdf

[46] Raspberry Pi Foundation. (2019). *Raspberry Pi 3B Tech Specs*. Accessed: May 22, 2020. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[47] Siemens. (2020). *Sitraffic Vehicle2X*. Accessed: May 22, 2020. [Online]. Available: https://www.mobility.siemens.com/global/en/portfolio/road/connected-mobility-solutions/sitraffic-vehicle2x.html

[48] HERE Technology. *Here*. Accessed: Jun. 20, 2019. [Online]. Available: https://www.here.com

[49] Transport Ministry of Land. *Infrastructure and Tourism ITS*. Accessed: Aug. 21, 2020. [Online]. Available: https://www.mlit.go.jp/road/road_e/p1_its.html

JOSE PAOLO V. TALUSAN (Member, IEEE) received the bachelor's and M.S. degrees in electronics engineering from the Ateneo de Manila University, Philippines, in 2011 and 2015, respectively. He is currently a Graduate Student with the Nara Institute of Science and Technology, Japan. His research interests include distributed systems and urban middleware, with a focus on smart transportation networks.

MICHAEL WILBUR received the bachelor's degree in civil engineering from the University of Notre Dame, in 2012, and the M.S. degree in structural engineering from Northwestern University, in 2018. He is currently a Graduate Student with the Department of Electrical Engineering and Computer Science, Vanderbilt University, where he works as a Research Assistant with the Institute for Software Integrated Systems. His research interests include high integrity and secure spatio-temporal decision procedures and software systems for urban systems.

ABHISHEK DUBEY (Senior Member, IEEE) received the bachelor's degree in electrical engineering from IIT (Banaras Hindu University) Varanasi, India, in 2001, and the M.S. and Ph.D. degrees in electrical engineering from Vanderbilt University, in 2005 and 2009, respectively. He is currently an Assistant Professor with the Department of Electrical Engineering and Computer Science, Vanderbilt University. His research interests include resilient system design, performance management, and failure diagnostics of smart and connected community systems, with a focus on transportation networks and smart grid.

KEIICHI YASUMOTO (Member, IEEE) received the B.E., M.E., and Ph.D. degrees in information and computer sciences from Osaka University, Osaka, Japan, in 1991, 1993, and 1996, respectively. He is currently a Professor with the Graduate School of Science and Technology, Nara Institute of Science and Technology. His research interests include distributed systems, mobile computing, and ubiquitous computing. He is a member of ACM, IPSJ, SICE, and IEICE.