Time-dependent Decentralized Routing using Federated Learning

Michael Wilbur*, Chinmaya Samal*, Jose Paolo Talusan†, Keiichi Yasumoto† Abhishek Dubey*
*Vanderbilt University
†NAIST

***Abstract*—Recent advancements in cloud computing have driven rapid development in data-intensive smart city applications by providing near real time processing and storage scalability. This has resulted in efficient centralized route planning services such as Google Maps, upon which millions of users rely. Route planning algorithms have progressed in line with the cloud environments in which they run. Current state of the art solutions assume a shared memory model, hence deployment is limited to multiprocessing environments in data centers. By centralizing these services, latency has become the limiting parameter in the technologies of the future, such as autonomous cars. Additionally, these services require access to outside networks, raising availability concerns in disaster scenarios. Therefore, this paper provides a decentralized route planning approach for private fog networks. We leverage recent advances in federated learning to collaboratively learn shared prediction models online and investigate our approach with a simulated case study from a mid-size U.S. city.**

***Index Terms*—Urban Mobility; Routing; Federated Learning; Fog Computing**

I. Introduction

Cities are evolving at a rapid pace. Over half the world's population currently lives in urban areas [1]. Along with a growing population, new challenges are emerging as an increase in housing density, population, and traffic strain city services. To meet these demands, both cities and private companies have turned to data-intensive applications to maximize efficiency of existing resources.

Cloud environments provide near real-time scalability of processing and storage resources, making cloud deployment the standard model for data-intensive applications. One such case is route planning services, which process millions of queries a day to guide vehicles from point A to point B. These services take into account the global transportation infrastructure and current traffic conditions to return the shortest route to the end user. In this context, routing is done on a time-dependent graph where the shortest path depends on the departure time and the current location of the end user.

Current approaches consist of implementing classic shortest path algorithms on either a single server or using a parallel approach in which a full road network is partitioned into multiple processes on a centralized cloud. There are numerous disadvantages in deploying route planning services to the cloud. First, the end user must send each request to a distant data center through a WAN network, inducing significant latency. This is satisfactory for current routing applications in which it is assumed that latency can be sustained for each request. However, latency demands of future technologies such as autonomous vehicles are already showing the limitations of centralized cloud-base route planning models. Additionally, cities typically use third party services [2] for dispatching emergency services. By relying on centralized services in remote data centers, cities risk service availability issues during disaster scenarios, precisely when such services are of most importance.

One potential solution is to deploy routing services on a private city owned sub-network of decentralized fog nodes called road side units (RSUs). In this scenario RSUs are low powered devices similar to Raspberry Pis [3] scattered throughout a city along roads and highways. By linking these devices together in a private sub-network, a reliable network can be created for smart city route planning services that can remain in operation without connection to outside cloud services. Additionally, moving processing to the edge allows end users to connect to nearby RSUs and potentially reduce latency.

The primary concern in moving route services to a fog computing model is that memory is not shared between processes and communication between fog nodes is a primary source of latency. Therefore, we aim to provide a decentralized route planning approach that accounts for communication latency between processes and is well-suited for deployment entirely on private fog networks with intermittent access to remote cloud services. We use prediction models to reduce the search space and limit communication between RSUs at inference time.

Typically, model training occurs offline in centralized cloud environments. However, this process can induce significant bandwidth demands as raw data is transferred to the cloud for training. Therefore we use federated learning [4] to collaboratively learn shared prediction models online for the route planning problem. In our approach, all training occurs at the RSUs and only the model weights are shared between processes and with the central cloud, therefore reducing communication overhead during training.

Contributions: The contributions of this work are as follows:

1) We provide a decentralized approach for route planning on time-dependent transportation networks. All routing occurs at the network edge within a private RSU fog network where access to outside cloud services is assumed to be intermittent.

2) Prediction models are used to condense the search space and limit communication between fog nodes.
3) We apply federated learning to collaboratively learn shared prediction models online. All training occurs on the RSUs and by sharing model weights our system avoids costly transfer of raw data between processes.
4) We apply our approach using a simulated case study from a mid-size U.S. city and compare it to current state of the art methods.

We find that our approach reduces latency and memory requirements compared to current state of the art parallel route planning approaches. The outline of this work is as follows: fundamental notation and background is provided in Section II and related research is provided in Section III. Section IV covers the system model and deployment while V outlines our decentralized route planning approach. Lastly a simulated case study is provided in Section VI.

II. Problem Statement

In this section we provide necessary background on modeling transportation networks. Table I summarizes the symbols used throughout this paper and Figure 1 represents available resources.

A. Transportation Networks as a Graph

Transportation networks are naturally modelled as time-dependent graphs [5]. Let $G_\tau = (V_\tau, E_\tau)$ be the time-dependent directed graph where V_τ is the set of vertices and $E_\tau \subseteq V_\tau \times V_\tau$ is the set of edges of a road network at time interval τ.

Since the graph G_τ is time-dependent, the travel time on an edge $e \in E_\tau$ varies with time. The edges in the network are weighted by a periodic time-dependent travel time function $T(e, \tau) : \Pi \to \mathbb{N}_0$ where Π depicts a set of time points or time-periods.

The function $T(e, \tau)$ is dependent on the network state and can be modeled as a latency function [6] which can be learned from historical states of the network $\{G_0, G_1, ..., G_{\tau-1}\}$. Given the current state of the network we can find the expected travel time for intervals $\{\tau + 1, \tau + 2, \cdots, \tau + f\}$ where f is the number of time intervals in the future. To differentiate this from the actual travel time function, we denote the learned travel time function as $\hat{T}(e, \tau)$.

Using this model, routing is a problem of finding the shortest path between two nodes. In time-dependent graphs, the shortest path depends on the departure time at the source node. Hence the route query is defined by a tuple (s, d, τ_s), where $s \in V_{\tau_s}$ is the source, $d \in V_{\tau_d}$ is the destination, τ_s is the departure time from s and τ_d is the arrival time at destination d. In contrast to the time-independent graph, here the directed route R for the query (s, d, τ_s) involves finding a sequence of edges along the time $[(e_1, \tau_1), (e_2, \tau_2), \cdots, (e_n, \tau_n)]$ from source s at time τ_s to reach destination d at time τ_d. The cost of the route $len(R)$ is defined as the sum of the time-dependent weights from the function $T(e, \tau)$ for each edge in route such that $len(R) = \sum_{(e,\tau)\in R} T(e, \tau)$.

TABLE I: List of symbols

Symbol	Description
$\mathbb{R}$	Real Numbers
$\mathbb{N}_0$	Natural numbers
G	Static graph, $G = (V, E)$
V	Set of network vertices
E	Set of network edges
τ_i	Actual time interval i of a day
$\hat{\tau}_i$	Estimated time interval i of a day
Π	Set of time points or time-period (seconds, minutes or hours of a day)
RSU_i	Road Side Unit i
R	Directed path from source vertex $s \in V$ to destination vertex $d \in V$, at time interval from source τ_s
R_d^s	Partial route from source vertex $s \in V$ to destination vertex $d \in V$, at time interval from source τ_s
$len(R)$	Travel time of the route R
G_τ	State of time-dependent graph (V, E) at time τ
V_τ	Set of vertices at at time interval τ
E_τ	Set of edges at at time interval τ
T	Travel time function
$\hat{T}$	Travel time predictor
$\hat{E}$	Equivalent Grid Routing predictor
$GetRoute$	Shortest path algorithm that uses Travel time predictor $\hat{T}$ to find route with minimum travel time
g_i	Grid i
G^i	Subgraph whose each vertices and edges maps to grid i
$\hat{t}_v^u$	Estimated travel time from vertex $u \in V$ to vertex $v \in V$ using Travel time predictor $\hat{T}$

B. The Decentralized Routing Problem

We are primarily concerned with routing on decentralized fog networks. We assume there is a set of fog nodes (RSUs) located along highways and roadways which are scattered throughout the transportation network. We also assume that the connection between RSUs and central cloud services is intermittent, therefore all routing occurs at the RSUs.

Ultimately we must divide the transportation graph $G = (V, E)$ into subgraphs $G^1, G^2, \cdots, G^k$ which can be mapped to an RSU. We first divide the city into a set of grids $g_1, g_2, \cdots, g^k$ where each grid represents a small square region of the city. Each grid g_i corresponds to a subgraph G^i which is the portion of the transportation graph G within this region. Then one or more subgraphs $G^1, G^2, \cdots, G^k$ are mapped to a single RSU.

This results in a hierarchical structure as shown in Figure 1. At the base level there are the grids. At the next level each RSU is responsible for a set of grids in the network. At the top level there is a centralized cloud responsible for coordination and deployment of the network.

III. Related Work

In this section we cover related research and limitations of existing solutions.

A. Current State of the Art Routing

Current state of the art route planning is typically deployed in centralized cloud-based systems [7], [8], [9]. In this architecture, data is stored in distributed databases while a travel

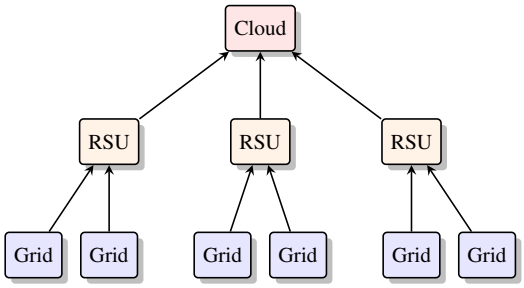

Fig. 1: Hierarchical representation of decentralized routing architecture. The city is divided into grids which are mapped to an RSU. All routing occurs at the RSU level. RSUs can intermittently communicate with a central cloud.

time model represents the current state of the transportation network. Vehicles query a central router for routes.

We draw on two bodies of work in routing algorithms that heavily influence the central architecture for route planning. The first is single server routing where it is assumed that the routing network resides entirely in a single physical node or server which handles all routing queries. The second is parallel routing where the network is partitioned between multiple nodes and routing queries are processed concurrently using multiple processes.

The single server approach relies on work from Dijkstra [10] and Bellman and Ford [11], [12], who proposed some of the first algorithms to solve the routing problem in a single server. Many advanced route planning algorithms that exist today are variants of these works. While these algorithms compute optimal shortest paths, they are too slow to process real-world data sets such as those derived from large-scale road networks. To address this issue, there are many techniques aimed at speeding up these algorithms. Such techniques often are based on clever heuristics that accelerate the basic shortest paths algorithms by reducing their search space. Bi-directional search [13], [9] not only computes the shortest path from the source s to the target t, but simultaneously computes the shortest path from t to s on the backward graph. Guided search approaches such as A* [14] use heuristics to guide the search and limit the search space. Goldberg et al. proposed the ALT approach in which they enhance A* by introducing landmarks to compute feasible potential functions using the triangle inequality [9], [15]. In other work, contraction techniques are used to speed-up the shortest path computation. This includes highway hierarchies [16], [7] which exploits the hierarchical in road networks, while contraction hierarchies [8] contract the graph in a pre-processing stage.

To parallelize the routing problem, a full road network is typically partitioned into multiple processes and the edge expansion proceeds similarly to Dijkstra. In parallel versions of Dijkstra, the priority queue is based on a shared memory model where it is assumed that communication between processors is constant [17], [18]. The parallel priority queue supports simultaneous insertion and deletion of an arbitrary sequence of elements ordered according to key, in addition to find-minimum and single element delete operations [17], [19]. Techniques to parallelize advanced routing algorithms such as contraction hierarchies are only limited to the pre-processing step where the contraction of nodes can be done in parallel [20].

B. Limitations of Centralized Route Planning

Cloud based route planning models assume near unlimited processing and network availability. This makes current approaches poorly suited for deploying route planning services on private fog networks where resources are constrained and access to outside cloud resources can be intermittent, particularly in disaster scenarios.

Many approaches to parallel route planning [17], [18], [19] assume that the graph network has static weights which doesn't hold in real transportation networks where traffic congestion changes with time. In a time-dependent network, edge expansion depends on arrival/departure time at each edge, hence processing is sequential. There are some approaches [20] which model the time-dependent nature of the network but the parallelization is only limited to the pre-processing phase and not during real-time query.

All of these approaches use a parallel shared memory model where an assumption is made that the shared memory allows constant time direct communication between each pair of processors. This holds in a multiprocessing system and possibly in a data center, but this assumption is not realistic in a decentralized setting where communication between processes can add significant latency. While previous work of ours has focused on data integrity in distributed RSU networks [21], limited work has been done on routing in such networks.

IV. System Model and Deployment

In this section we outline the system architecture, data collection and deployment for decentralized route planning.

A. Architecture

Figure 2 shows our decentralized architecture for route planning. The fundamental components are outlined as follows:

1) *Road Side Unit (RSU)*: low-powered compute nodes [3] located near roads and highways throughout the transportation network. These nodes are assumed to have computational resources similar to Raspberry Pis. Linked together, the RSUs form a private fog network.
2) *Central Server*: The central server is assumed to be a cluster of compute and storage nodes with horizontal on demand scaling. The primary role of the central server is as the central administrator for the RSU network metadata and resources. It is assumed that access to the central server is intermittent and can fail in disaster scenarios.
3) *Vehicle*: Vehicles are assumed to be GPS equipped and network enabled. They provide two functions. First, they periodically send location data and travel speed to the RSU network. Second, vehicles can query the network for routes from their current location to a destination.
4) *Admin*: Maintains the global transportation network graph $G = (V, E)$ and helps divide the network into sub-graphs as outlined in Section IV-C.

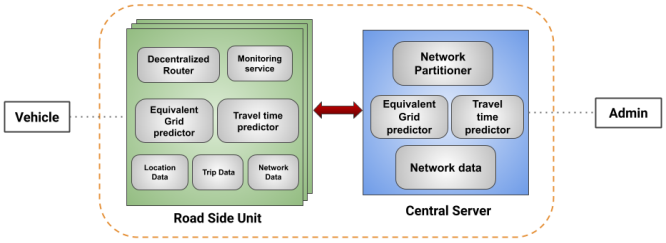

Fig. 2: Decentralized architecture and services for route planning.

B. Data collection

Our system consists of the following types of data:

1) *Location data*: This data is periodically collected from vehicles and is stored as a set $\{(e_1, \tau_1, t_1), \cdots, (e_n, \tau_n, t_n)\}$, where e_i is the edge traversed, τ_i is the time interval of the day.
2) *Trip data*: This data consists of a set of route plans $\{R_1, \cdots, R_n\}$ where each route is a sequence of edges $R_i = [(e_1, \tau_1), (e_2, \tau_2), \cdots, (e_p, \tau_p)]$ from the location of the vehicle to its destination.
3) *Network data*: Each RSU maintains a list of subgraphs of the global routing graph $G = (V, E)$. Location data and trip data from vehicles travelling on this subgraph is maintained at that RSU.

C. System Deployment

In this section we outline the procedure for deploying the system using Algorithm 1.

1) Central Server Setup: First, Admin sends the full graph $G = (V, E)$ and a geohash precision $prec$ to the Network Partitioner in Central Server. The geohash precision value determines the resolution or area of the desired grids. For this we use geohash encoding [22] to encode the geographical coordinates.

The Network Partitioner receives the graph $G = (V, E)$ and $prec$ from the Admin and partitions the network into grids $\{g_1, g_2, \cdots, g_k\}$ using Algorithm 1. The algorithm proceeds by first annotating each vertex v with the grid they belong to using the geohash encoding function $gh.encode(v, prec)$, where $prec$ represents the precision of the encoding. Then for each edge $e_{u,v}$ it checks if both the vertices u and v are in the same grid. If they are in different grids a boundary vertex w is found at the intersection between the two grids through which the edge passes using the function $intersection(e_{u,v}, u.grid, v.grid)$. Vertex w splits edge $e_{u,v}$ into edges $e_{u,w}$ and $e_{w,v}$. We then add vertex w and edges $e_{u,w}$ and $e_{w,v}$ to the graph G and remove edge $e_{u,v}$.

2) Deployment to RSUs: Each RSU requests the subgraphs for a set of grids $\{g_1, g_2, \cdots, g_k\}$ from the Central Server. The Central Server receives the set of grids $\{g_1, g_2, \cdots, g_k\}$ and returns the set of subgraphs $\{G^1, G^2, \cdots, G^k\}$ to the requesting RSU to be used for decentralized routing in Section V.

Algorithm 1: Partition Network

Data: A graph $G = (V, E)$, $g = \{g_1, g_2, \cdots, g_n\}$, $prec$ = geohash precision value.

```
begin
    foreach v ∈ V do
        v.grid = gh.encode(v, prec);
    foreach e_{u,v} ∈ E do
        e_{u,v} = edge between vertices u and v;
        if u.grid ≠ v.grid then
            w = intersection(e_{u,v}, u.grid, v.grid)
            w.grid = (u.grid, v.grid)
            G.add(w, e_{u,w}, e_{w,v})
            G.remove(e_{u,v})
```

V. Decentralized Routing

Our system needs to provide a shortest route from an origin location to destination. Any RSU can receive a route query between two points, which may be within one of the RSU's subgraphs or require communication with neighboring RSUs.

In this section we outline prediction models for estimating both travel times throughout the network as well as predicting which next RSU should be contacted to find the current shortest path for that query. We then provide the full decentralized routing algorithm and an example execution procedure on a single routing query.

A. Training With Federated Learning

Model training occurs on the RSUs. We achieve this by leveraging recent advances in federated learning [23] which enable the RSUs to collaboratively learn a shared prediction model without transferring raw data from the devices to the centralized cloud.

The goal of federated learning is to learn a model with parameters embodied in a real matrix $\mathbf{W} \in \mathbb{R}^{d_1 \times d_2}$ from vehicle data stored across the RSUs. Here $\mathbf{W}$ is a 2D matrix representing the parameters of each layer in a fully-connected feed-forward network [24]. d_1 and d_2 represents the output and input dimensions respectively. The tasks proceed in rounds and each round alternates between local and global model updates. The general procedure for model training is outlined as follows:

1) *Distribute global model*: Admin randomly initializes the weights $\mathbf{W_0}$ of the prediction model and stores it in Central Server. In round $t \geq 0$, the Central Server distributes the current model $\mathbf{W}_t$ to a subset S_t of n_t RSUs.
2) *Local update*: Each RSU then independently updates the model based on its local data. Let the updated local models be $\mathbf{W_t^1}, \mathbf{W_t^2}, \cdots, \mathbf{W_t^{n_t}}$, so the update of each RSU i can be written as $\mathbf{H_t^i} := \mathbf{W_t^i} - \mathbf{W_t}$, for $i \in S_t$. For this update we use stochastic gradient descent (SGD) [25]. Each RSU then sends the update back to the Central Server.

TABLE II: Input features for Travel Time and Equivalent Grid predictors.

Feature	Dim	Description
From location	42	Geohash encoded binary of start loc
To location	42	Geohash encoded binary indication of To coordinate of an edge
Week of year	52	One-hot encoded binary indication of Week of year used to sample travel time data
Day of week	7	One-hot encoded binary indication of Day of week used to sample travel time data
Hour of day	24	One-hot encoded binary indication of Hour of day used to sample travel time data
Minutes	60	One-hot encoded binary of Minutes of hour used to sample travel time data

TABLE III: Target features for Travel Time and Equivalent Grid predictors.

Feature	Dim	Description
Travel time	1	One-hot encoded binary indication of the true travel time data collected from HERE API
Next Grid	28	Geohash encoded binary indication of a Grid

3) *Global update*: aggregation of local updates.

$$\mathbf{H_t} := \frac{1}{n_t} \sum_{i \in S_t} \mathbf{H_t^i}, \mathbf{W_{t+1}} := \mathbf{W_t} + \eta_t \mathbf{H_t}.$$

Here η_t is learning rate. For simplicity we can choose $\eta_t = 1$.

B. Prediction Models

Our decentralized routing approach requires two prediction models described as follows.

1) Travel Time Predictor: A travel time predictor model estimates travel time on an edge e_i in time interval τ_i. Let edge e_i define a directed edge from v_i to v_{i+1}, then the travel time function is defined as $T(v_i, v_{i+1}, \tau_i)$ where τ_i refers to the departure time at vertex v_i.

For training, we build an input feature set described in Table II. The resulting feature space has 228 dimensions. We used one-hot encoding to map the travel time τ_i to one-hot encoded binary for Week of Year, Day of Week, Hour of Day and Minutes of Hour. The output of this model is travel time as shown in Table III which is a scalar value representing the travel time on an edge.

Geohash encoding is used to map v_i and v_{i+1} to the From location and To location respectively. An important aspect of the feature set is the geohash resolution. We used a resolution of 9.5m which matches the width of most major road segments in the United States. We found that reducing resolution to greater than 9.5m caused some vertices to belong to the same geohash while increasing resolution added complexity to the model and did not noticeably improve model performance. A resolution of 9.5m was represented by 42 bits.

2) Equivalent Grid Routing Predictor: The search procedure can extend to multiple RSUs. As more RSU are included in the search, it can incur huge delays from communication costs. Hence, the goal is to minimize the number of message exchanges required during the search. Therefore the equivalent grid routing model predicts the best neighboring grid through which the shortest path likely resides for a particular route. By iteratively finding the next grid the optimal route will pass through, the search space is reduced and communication is optimized. Hence we learn an Equivalent Grid Routing Predictor $\hat{E}$ such that $\hat{E}(s, d, \tau_s)$ gives the next best possible grid to travel to destination d and τ_s is the departure time from s. For this model, we use the same feature set as shown in Table II. The output of this model is the next grid as described in Table III which represents the next grid in the route.

C. Decentralized Route Planning Algorithm

The goal of the decentralized route planning is to distribute the query among different RSUs. One of the problems we discussed earlier is that state of the art solutions for parallelizing the query fails in a time-dependent network. We mitigate this problem by using Travel Time Predictor $\hat{T}$. To minimize communication between RSUs during the search we use Equivalent Grid Routing Predictor $\hat{E}$.

Algorithm 2 handles decentralized routing queries from vehicles as well as from other RSUs as the query is propagated forward through the network. Algorithm 3 iteratively builds the route as the results propagates back to the RSU from which the query started.

We first list some utility functions that are used in the algorithm and then discuss the algorithm in detail. The utility functions are:

1) $gh.encode(v, prec)$: Uses geohash encoding to find the grid to which the vertex belongs to with geohash precision $prec$.
2) $GetRSU(g_i)$: Finds the RSU mapping for any grid i.
3) $GetRoute(G, s, d, \tau_s)$: This function uses network G, Dijkstra [10] and Travel time Predictor $\hat{T}$ to find the route from source s at departure time τ_s to destination d with minimum travel time.
4) $msg(type, RSU_i, val)$: An async call for communicating the type of message ($type$) and actual message (val) to an RSU i. There are two types of messages:
   a) *query*: Upon receiving this message, Algorithm 2 is executed. The message, represented by (val) is passed as an argument to this function.
   b) *partial path*: Upon receiving this message, Algorithm 3 is executed, with the message represented by (val) as input. If the final route plan is returned as a response, it is communicated to the client which made the routing request.

D. Decentralized Route Planning Example

To demonstrate execution, Figure 3 shows an example where a network is partitioned into 4 RSUs. Figure 4 shows a sequence diagram for this example.

Algorithm 2: Handle Query

Data: A graph $G = (V, E), s \in V, d \in V$, τ_s = departure time from vertex s, RSU_o = RSU from where the query origins, id: Unique identifier to identify this query.

begin
  $save(id, (s, d, \tau_s))$;
  $g_s = gh.encode(s)$;
  $g_d = gh.encode(d)$;
  if $g_s \neq g_d$ **then**
    $g_{\text{next}} = \hat{E}(s, d, \tau_s)$;
    RSU_{next} = GetRSU(g_{next});
    $\{v_1, v_2, \cdots, v_b\} = g_s.intersect(g_d)$;
    foreach $v \in \{v_1, v_2, \cdots, v_b\}$ **do**
      $\hat{t}_v^s = \hat{T}(s, v, \tau_s)$;
      $msg(\text{“query”}, RSU_{\text{next}}, \{id, v, d, \tau_s + \hat{t}_v^s, RSU_o\})$;
    foreach $v \in \{v_1, v_2, \cdots, v_b\}$ **do**
      R_p = $GetRoute(G, s, v, \tau_s)$;
      $msg(\text{“partial path”}, RSU_o, \{id, s, v, \tau_s, R_v^s\})$;
  else
    R_p = $SP(G, s, d, \tau_s)$;
    $msg(\text{“partial path”}, RSU_o, \{id, s, d, \tau_s, R_d^s\})$;

Algorithm 3: Handle Partial Path

Data: A graph $G = (V, E), u \in V, v \in V$, τ_u = departure time from vertex u, R = Route from u to v, starting at τ_u, id: Unique identifier to identify this query.

Result: Final Route plan (id, R_{final}) or NULL

begin
  (s, d, τ_s) = $get(id)$;
  if $v == d$ **then**
    R_{final} = $GetRoute(G_{id}, s, d, \tau_s)$;
    return (id, R_{final});
  else
    G_{id} = GetGraph(id);
    if $G_{id} == NULL$ **then**
      Initialize Graph G_{id};
    foreach $(e_i, \tau_i) \in R$ **do**
      $G_{id}.add(e_i)$;
      $G_{id}[e_i] = (\tau_i, \tau_{i+1} - \tau_i)$;
    SaveGraph(id, G_{id});
    return $NULL$;

In this example, RSU_1 receives a route query (id, s, d, τ_s) from a client and calls Algorithm 2 to find the route. Since the source s and destination d do not belong to the same grid, Equivalent Grid Predictor is called to find the next best possible grid to reach destination d. Then the algorithm finds the nodes at the intersection of the current grid and the next best possible grid. After getting all the boundary vertices, for each boundary vertex Travel Time Predictor estimates the time it will take to reach that vertex. An asynchronous message $(\text{“}query\text{”}, id, v_{12}, d, \hat{\tau}_{v_{12}}, RSU_1)$ is sent to RSU_2.

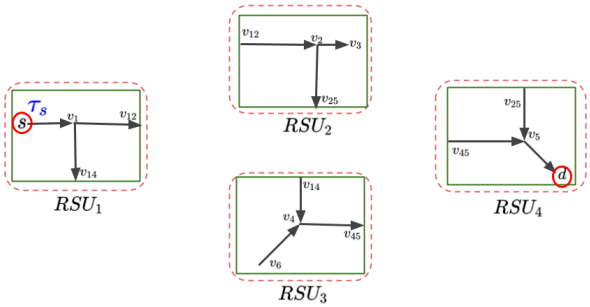

Fig. 3: Decentralized Route Planning example setup.

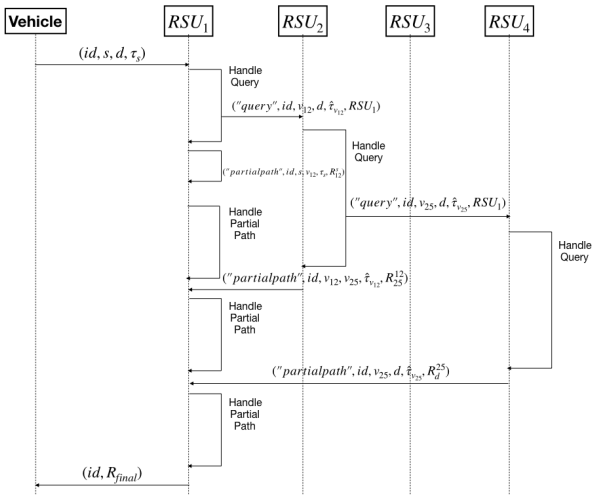

Fig. 4: Sequence Diagram of Decentralized Route Planning example.

After sending the message, RSU_1 proceeds to find the actual route from s to boundary vertex v_{12}. After getting the route, denoted by R_{12}^s, a message $(\text{“}partialpath\text{”}, id, s, v_{12}, \tau_s, R_{12}^s)$ is prepared and sent to the requesting RSU informing the starting RSU that the operation resulted in a partial path, i.e. the destination is not located in that RSU.

This message is meant to be sent to the RSU to which the client sent the request. Since its RSU_1, which is itself, a function call is made to handle this message where the function arguments are the same as the message. This function implements Algorithm 3 which handles the partial routes or paths. We call it partial route because this is still not the final route that needs to be given to the client.

The goal of Algorithm 3 is to create a new graph with the id of the request or if it already exists, add all partial routes to the graph and finally, do a simple shortest path routing on it.

At this step, RSU_1 waits for partial routes from the other RSUs for this request, identified by its id, and executes Algorithm 3 if it receives a message with the partial route in it. This process continues until RSU_1 gets a partial route which has the destination in it. When a partial route contains the destination vertex Algorithm 3 executes $GetRoute(G_{id}, s, d, \tau_s)$ to obtain the final route.

E. Decentralized Route Planning Properties

As discussed in Section III, A^* is a classic algorithm for informed search, which relies on a heuristic function to guide the search procedure. In this context, our approach for route planning is an informed search procedure where Equivalent Grid Routing Predictor acts as a heuristic function.

It is well established that if a graph G is finite and edge weights are non-negative, then A^* is guaranteed to terminate and is complete, i.e. it will always find a route from source to destination if one exists. Our approach is similar to A^*, but it cannot give guarantees on its termination and hence may not be complete. This is due to our use of learned models in the Equivalent Grid Routing Predictor. Therefore as with all machine learning models, the accuracy of our approach is tied to the accuracy of the learned models.

VI. Experiments and Results

In this section, we evaluate our decentralized architecture for route planning with a case study from a mid-size U.S. city.

A. Experiment Setup

1) *RSUs*: Cluster of 5 RSUs simulated by Docker [26] containers. RSUs are static as their location does not change with time.
2) *Global Network Graph*: We use OpenStreetMap to generate the underlying routing graph $G = (V, E)$. For the region in this study there are a total of $233,123$ nodes and $474,213$ edges.
3) *Graph Partitioning*: We used a geohash precision value of 28 bits, which results in a grid area of $1.44km^2$. A total of 1034 grids are generated as a result of the partition. These grids are assigned to the 5 RSUs as shown in Figure 5.
4) *Location data*: To simulate vehicle locations in the region, we use historical traffic data collected at an interval of 1 minute from the HERE API [27] for the region. Traffic data from January 1 to January 31, 2018 was used for training and data from Feb 1 to Feb 7, 2018 was used for testing.
5) *Trip data*: To simulate routing queries from vehicles, we synthetically generate 1,000,000 source and destination pairs which are chosen randomly within the region. For the source-destination pairs, departure times were chosen uniformly from 9am-5pm.

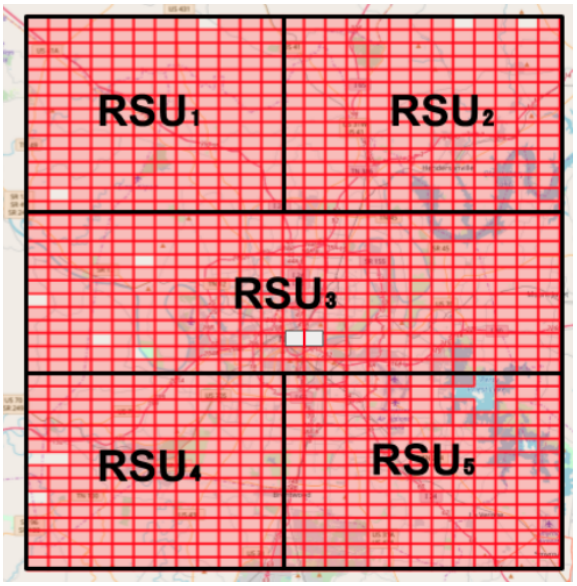

Fig. 5: Partition of city into grids of area $1.44km^2$ and placement of grids in RSUs. Grids are represented by the 1034 square grids while the RSU regions are represented by bolded black lines.

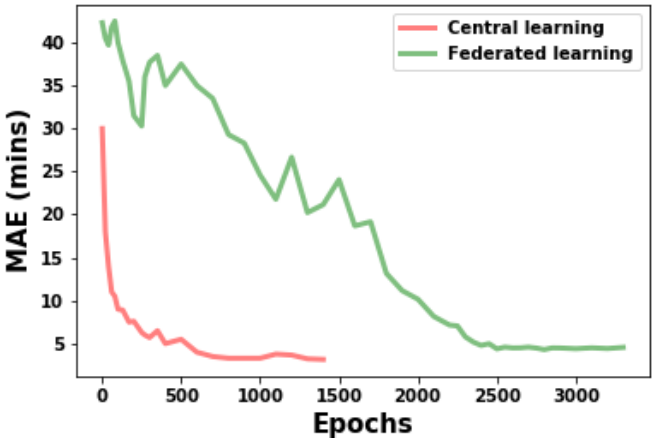

Fig. 6: MAE vs Epoch curve during training of Travel time predictor.

B. Evaluation of Prediction Models

1) Travel Time Predictor - Training Evaluation: For the travel time predictor we used a deep feed-forward neural network (DNN) [28] for regression to estimate travel time of an edge. We used SGD [25] as the optimizer and a hidden layer configuration of $[200, 190, 170, 150, 100, 50, 20, 10]$. Early stopping criteria was implemented to avoid over-fitting. Fig. 6 shows the change in validation Mean Absolute Error (MAE) during the training. We found that the federated learning model took longer to train than the central learning model.

Table IV evaluates the resource consumption of the model during training for federated learning and central learning. As the central learning model was trained on a single large server, we divided the resource consumption of the central model by the number of fog nodes for a direct comparison with the

TABLE IV: Resource consumption for Travel time predictor.

	CPU (%/RSU)	Memory (MB/RSU)	# Messages
Central Learning	78% (median) 97% (max)	191 (median) 307 (max)	N/A
Federated Learning	67% (median) 84% (max)	51 (median) 88 (max)	6255

TABLE V: Resource consumption for Equivalent Grid Routing predictor.

	CPU (%/RSU)	Memory (MB/RSU)	# Messages
Central Learning	81% (median) 93% (max)	217 (median) 336 (max)	N/A
Federated Learning	74% (median) 97% (max)	64 (median) 91 (max)	9543

federated training model. Results show that federated learning used less CPU per node as well as 3.4 - 3.7 times less memory per node than central learning. Lastly, federated learning sent 6255 messages while training.

2) Equivalent Grid Routing Predictor - Training Evaluation: We use a deep feed-forward neural network (DNN) [28] for a binary classification that gives the next best possible grid for a given pair of source, destination along with the time interval. For binary classification, we used a sigmoid function [29] for the output layer and an Adam optimizer was used as the optimizer for the model. The configuration for hidden layers was $[250, 200, 170, 100, 50, 20, 10]$. Fig. 7 shows the loss vs epoch curve during the training phase for this predictor. Federated learning took more time to train than central learning. The loss for a model trained with central learning was 0.32 which is less than the model trained from federated learning which was 0.37.

Table V evaluates the resource consumption of the federated learning and central learning models during training. We find that federated learning used less CPU than Central learning per node on average. Additionally, federated learning used 3.3 - 3.6 times less memory per node than central learning. Federated learning sent 9543 messages while training.

C. Evaluation of Decentralized Route Planner

During testing we monitored CPU and memory consumption per node compared to current state of the art solutions. The resource requirements of each method is presented in Table VI. To evaluate performance of our approach we measured the query response time per request as well as the accuracy of the returned routes compared to the optimal route.

1) Resource Consumption: We find that our approach uses more CPU than single server Dijkstra or parallel Dijkstra because in our approach shortest paths are calculated between boundary nodes in parallel when the request is received. In terms of memory, we find that our approach uses less memory per node than single server Dijkstra and slightly more memory than parallel Dijkstra. We find that contraction hierarchies and parallel contraction hierarchies use the most memory due to caching shortcut edges.

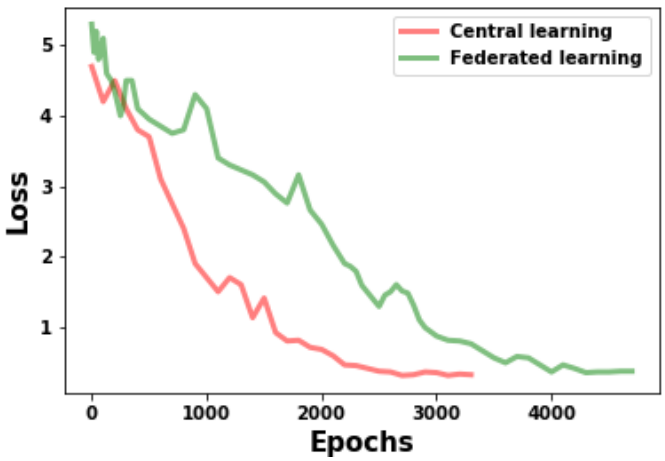

Fig. 7: Loss vs Epoch curve during training of Equivalent Grid Routing predictor

TABLE VI: Evaluation of routing algorithms.

Algorithm	CPU per RSU (% used)	Memory per RSU (MB)	Query time per trip request (s)
Single server Dijkstra	23% (median) 31% (max)	5.78	0.97 (median)
Parallel Dijkstra	27% (median) 36% (max)	0.76 (median) 1.14 (max)	9.2 (median) 19.13 (max)
Contraction Hierarchies	18% (median) 23% (max)	13.36	0.016 (median)
Parallel Contraction Hierarchies	13% (median) 21% (max)	3.31 (median) 5.79 (max)	5.78 (median) 10.21 (max)
Our approach	52% (median) 67% (max)	0.94 (median) 1.31 (max)	2.43 (median) 5.81 (max)

2) Query Response Time: Single server Dijkstra and contraction hierarchies result in the lowest query response times as expected since this simulation was done on one machine. It is expected that in production cloud environments the latency between vehicles and the cloud would factor into this result. Parallel algorithms such as parallel Dijkstra and parallel contraction hierarchies have higher query times than our approach since their search proceeds sequentially.

3) Accuracy: We found that our approach returned no route for 0.8% of the queries and a sub-optimal route (i.e. longer than the shortest route) for 7.6% of the queries. Therefore we found that 91.6% of routes from our model were the shortest route. As our approach is reliant on trained models, it is expected that our model improves as more data is available for training.

VII. Conclusion and Future Work

In this work we provided a decentralized route planning approach and deployment model for fog networks. Our approach uses prediction models to limit communication between fog nodes and thus improve latency and memory demands over current parallel approaches to route planning. The core of our architecture relies on data-driven models that estimates travel times and guides the search procedure during query time.

Additionally, to limit communication during training we used recent advances in federated learning to train the models.

Through this approach all training occurs on the RSUs and only model weights are shared between nodes. Therefore costly transfer of raw traffic data is avoided, reducing bandwidth stress during training. This work was evaluated through a simulation using real traffic data for a mid-sized U.S. city.

Potential extensions of this work include investigating ways to improve travel time and grid prediction models to mitigate the impact of errors on user trips, as well as expanding the architecture to handle node failures. Additionally, our approach can be extended to allow multiple modes of transportation.

Acknowledgements: This work was supported in part by National Science Foundation through award numbers 1647015 and 1818901 and JSPS KAKENHI Grant Number 16H01721 & 19H05665 and R&D for Trustworthy Networking for Smart and Connected Communities, Commissioned Research of National Institute of Information and Communications Technology (NICT).

REFERENCES

[1] U. Nations, D. Economic, S. Affairs, and P. Division, *World Urbanization Prospects: The 2018 Revision (ST/ESA/SER.A/420*. New York: United Nations.

[2] Omnigo, "911 dispatch software," 2019. [Online]. Available: {https://www.omnigo.com/solutions/computer-aided-dispatch-software}

[3] F. Perry, K. Raboy, E. Leslie, Z. Huang, D. Van Duren *et al.*, "Dedicated short-range communications roadside unit specifications." United States. Dept. of Transportation, Tech. Rep., 2017.

[4] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konecny, S. Mazzocchi, H. B. McMahan *et al.*, "Towards federated learning at scale: System design," *arXiv preprint arXiv:1902.01046*, 2019.

[5] T. Pajor, "Multi-modal route planning," *Universität Karlsruhe*, 2009.

[6] T. A. Manual, "Bureau of public roads," *US Department of Commerce*, 1964.

[7] P. Sanders and D. Schultes, "Engineering highway hierarchies," in *ESA*, vol. 6. Springer, 2006, pp. 804–816.

[8] R. Geisberger, P. Sanders, D. Schultes, and D. Delling, "Contraction hierarchies: Faster and simpler hierarchical routing in road networks," *Experimental Algorithms*, pp. 319–333, 2008.

[9] A. V. Goldberg and C. Harrelson, "Computing the shortest path: A search meets graph theory," in *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2005, pp. 156–165.

[10] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[11] L. R. Ford Jr, "Network flow theory," RAND CORP SANTA MONICA CA, Tech. Rep., 1956.

[12] R. Bellman, "On a routing problem," *Quarterly of applied mathematics*, vol. 16, no. 1, pp. 87–90, 1958.

[13] G. Dantzig, *Linear programming and extensions*. Princeton university press, 2016.

[14] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[15] A. V. Goldberg and R. F. F. Werneck, "Computing point-to-point shortest paths from external memory." in *ALENEX/ANALCO*, 2005, pp. 26–40.

[16] P. Sanders and D. Schultes, "Highway hierarchies hasten exact shortest path queries," in *European Symposium on Algorithms*. Springer, 2005, pp. 568–579.

[17] G. Di Stefano, A. Petricola, and C. Zaroliagis, "On the implementation of parallel shortest path algorithms on a supercomputer," in *International Symposium on Parallel and Distributed Processing and Applications*. Springer, 2006, pp. 406–417.

[18] Y. Tang, Y. Zhang, and H. Chen, "A parallel shortest path algorithm based on graph-partitioning and iterative correcting," in *2008 10th IEEE International Conference on High Performance Computing and Communications*. IEEE, 2008, pp. 155–161.

[19] A. Crauser, K. Mehlhorn, U. Meyer, and P. Sanders, "A parallelization of dijkstra's shortest path algorithm," in *International Symposium on Mathematical Foundations of Computer Science*. Springer, 1998, pp. 722–731.

[20] C. Vetter, "Parallel time-dependent contraction hierarchies," *Student Research Project*, p. 134, 2009.

[21] M. Wilbur, A. Dubey, B. Leão, and S. Bhattacharjee, "A decentralized approach for real time anomaly detection in transportation networks," in *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 2019, pp. 274–282.

[22] T. Vukovic, "Hilbert-geohash-hashing geographical point data using the hilbert space-filling curve," Master's thesis, NTNU, 2016.

[23] H. B. McMahan, E. Moore, D. Ramage, S. Hampson *et al.*, "Communication-efficient learning of deep networks from decentralized data," *arXiv preprint arXiv:1602.05629*, 2016.

[24] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[25] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[26] docker, "docker," 2019. [Online]. Available: {https://www.docker.com/}

[27] [Online]. Available: https://www.here.com

[28] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT Press, 2016, vol. 1.

[29] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.