Cooperative System for Free Parking Assignment

Abeer Hakeem*, Narain Gehani†, Reza Curtmola‡, Xiaoning Ding§, Cristian Borcea¶
Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102, USA
Email:{*amh38, †gehani, ‡reza.curtmola, §xiaoning.ding, ¶borcea}@njit.edu

***Abstract*—This paper presents Distributed Free Parking System (DFPS), a decentralized system for assigning free curbside parking spaces. DFPS optimizes a system-wide social welfare objective: the total travel time to destinations for all drivers. DFPS uses the smart phones of the drivers for parking request assignment, and a centralized dispatcher to receive and distribute parking requests. The parked drivers in DFPS are structured in a K-D tree, which is used to serve new parking requests in a distributed fashion. DFPS solves the scalability problem associated with a centralized parking assignment system by removing the computation from the dispatcher and substantially reducing the communication handled by the dispatcher. At the same time, DFPS achieves similar travel time performance with a centralized system. Compared to a naive algorithm that assumes a breadth-first-search for parking spaces around the destinations, DFPS reduces the travel time for over 97% of the drivers.**

***Index Terms*—Parking assignment; cooperative system; mobile app.**

I. Introduction

Drivers searching for free curbside parking waste valuable time and fuel, create traffic congestion, and pollute the air. Existing solutions to this problem employ a sensing infrastructure [1] that feeds data to mobile apps to show a map of the available parking spaces to drivers. These solutions have two problems. First, the sensing infrastructure is costly and may not be deployed in everywhere. Second, they are not designed to provide individual guidance to a specific parking space for each driver. Since they show the same map to all drivers, they can lead to parking space contention and traffic congestion.

To solve these problems, in prior work we proposed a centralized parking assignment algorithm [2] that relies on driver cooperation to maintain the parking availability map. This algorithm assigns drivers to available parking spaces close to their destinations and reduces the total travel time (i.e., the sum of the driving time to the parking space and the walking time from the parking space to the destination) of all drivers. However, the algorithm is centralized and requires substantial computation and communication resources at the server.

This paper presents the Distributed Free Parking System (DFPS), a decentralized system for assigning free curbside parking spaces to drivers. DFPS has two components: a mobile app running on drivers' smart phones and a dispatcher running at a server that enables cooperation among phones. DFPS solves the scalability problem associated with the centralized parking assignment system by substantially reducing the communication and computation costs at the centralized dispatcher. The mobile apps on the phones form the distributed system that enables the parking assignment algorithm.

DFPS provides a distributed version of our previous centralized parking assignment algorithm, which optimizes the total travel time for all drivers as a system-wide social welfare objective. In order to increase the processing efficiency of new parking requests, DFPS structures the distributed system as a K-D tree [3]. This structure allows DFPS to increase parallel processing and decrease the response time. Thus, DFPS is able to scale with an increasing number of drivers by performing localized computations over smart phones of drivers parked in proximity of each other.

DFPS does not assume that all drivers use our system. It relies on subscribed drivers to submit observation reports regarding the parking spaces occupied by unsubscribed drivers. DFPS avoids allocating the reported spaces for a period of time proportional with the age of the observation reports; then, it reconsiders these spaces.

We have evaluated DFPS through extensive simulations using SUMO [4] and PeerSim [5], a real map of a part of New York City with 1024 parking spaces, and as many as 768 drivers looking for parking. The results show that DFPS scales well. In particular, it eliminates all computation from the centralized dispatcher and reduces its communication load by a factor of two. At the same time, the DFPS distributed assignment algorithm achieves similar travel time performance with the centralized version of the algorithm. Compared to a naive algorithm that assumes a breadth-first-search for parking spaces around the destinations, DFPS reduces the travel time for over 97% of the drivers.

II. System Overview

This section presents an overview of DFPS, focusing on how the mobile drivers are organized in DFPS to work cooperatively on assigning free parking spaces. The major design principle of DFPS is to execute as much work as possible on the phones of drivers, instead of relying on a central server. This is feasible with the resources available in today's phones. By executing most work on the phones, the system avoids the performance bottleneck at the central server, and thus remains scalable.

Figure 1 shows the architecture of DFPS. Drivers cooperate to assign parking spaces with the assistance of a dispatcher. The DFPS software runs on the dispatcher and the phones of the drivers. The dispatcher receives parking requests and forwards them to the system formed by the cooperative drivers. This system, without involvement from the dispatcher, performs parking assignment. The communication in the system is done over the Internet (e.g., LTE).

The dispatcher also serves as a bootstrapping unit. During the system adoption period when few drivers have the DFPS

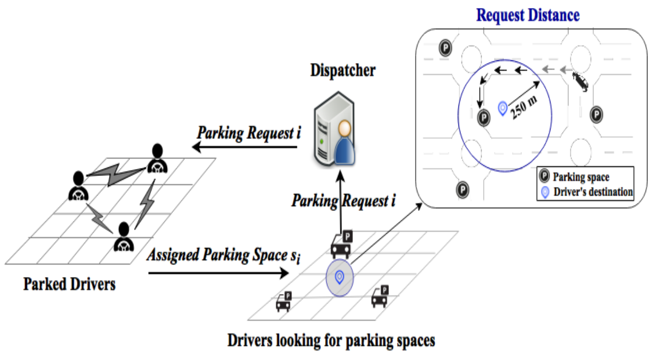

Fig. 1: System architecture

app, the dispatcher could share with the drivers parking availability information derived from historical parking statistics or real-time sources (e.g., street images from video cameras).

The drivers in the system are divided in three categories: (1) the drivers who are looking for parking spaces, (2) the parked drivers, and (3) departing drivers. The drivers in the first category submit their requests and then wait to receive parking spaces while heading to their destinations. Once the parking assignment is done, they drive to the parking spaces. When a driver parks, it transitions from the first category into the second. Parked drivers cooperate to handle the requests for parking spaces. Departing drivers report when they leave the system, such that DFPS can update the status of the parking spaces.

The lifecycle of a request in DFPS, from generation to completion, is described as follows. When a driver is about to set off for her destination and needs a parking space near the destination, the DFPS software on her phone generates a parking request and computes a *Request Distance*. This distance determines when the parking request is sent to the system. If the request is sent when the drivers are far away from the destination, non-subscribing drivers have a high likelihood of taking the assigned space. If the request is sent when the driver is too close to the destination, the system may not be able to find a parking space close-enough to the destination. DFPS on the requester's phone computes the request distance based on road conditions within the region of the destination, which are learned from the dispatcher. The distance defines a circle centered around the destination, and the request is submitted to the dispatcher as soon as the driver intersects this circle.

The dispatcher accepts requests and forwards them to the cooperative system of parked drivers (i.e., to the DFPS instance running on their phones). All the work requried to serve a request is done by the parked drivers. This minimizes the workload on the dispatcher and allows it to serve a larger number of drivers.

To maximize scalability, the workload of assigning parking spaces is shared by the parked drivers. We use a structured network to organize parked drivers, partition the whole area into regions, and assign a parked driver to manage a region and allocate parking spaces in the managed region.

III. Organization of Parked Drivers and Roles

DFPS uses an overlay network to organize parked drivers and the regions that the parked drivers manage. The organization of parked drivers needs to satisfy the following requirements:

- *Scalability*: The drivers must be organized in a scalable way to share the workload effectively.
- *Fast Routing*: Given a request, DFPS must identify quickly the driver managing the region where the parking space is requested.
- *Adaptability*: The overlay network must adapt quickly and with low overhead to high churn (i.e., parked drivers coming and going frequently).

A. K-D Tree in DFPS

To meet these requirements, DFPS organizes the overlay network of the parked drivers as a K-D tree. A K-D tree is a k-dimensional binary search tree for partitioning and spatially indexing data distributed in a k-dimensional space [3], [6]. A node in the K-D tree is associated with three types of information: a value, a rectangular representation (i.e., a region) containing a set of data points, and the coordinates of these data points.

Each node in the K-D tree represents a region. The region corresponding to a parent node is divided into sub-regions corresponding to the children of that node. The locations of the parking spaces in a region are represented as data points associated with the node for that region. The node's value is the location of the first driver parked in the corresponding region when the region and the node are created. For brevity, we call this value the location of the node.

DFPS associates a parked driver with each tree node. The tasks of forwarding parking requests and allocating parking spaces, as well as the data structures required to manage these tasks are associated with nodes. Since the nodes follow a tree structure, DFPS can manage the tasks and data structures in a hierarchical way, which leads to good scalability.

There are two roles that may be associated with a node in the K-D tree as shown in Figure 2: 1) *region manager* which forwards parking requests, divides a region into two sub-regions when necessary, and assigns sub-regions to drivers that park in these sub-regions; 2) *parking manager* which assigns parking spaces within the region associated with the node. Depending on its position in the tree, a node may have one or both of these roles. A leaf node acts only as parking manager for its region (i.e., nodes C, F, G, and H). An internal node that has two children acts only as region manager (i.e., nodes A and E). An internal node that has only one child acts as parking manager for the sub-region that is not covered by the child node, and it acts as a region manager by forwarding requests to its child or by assigning the sub-region not covered by the child to a driver that parks in that sub-region (i.e., nodes B and D).

Since parking space allocation is handled by the phones of parked drivers, we also refer to the parked drivers as the region

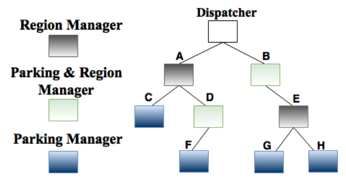

Fig. 2: The roles associated with nodes in the K-D tree

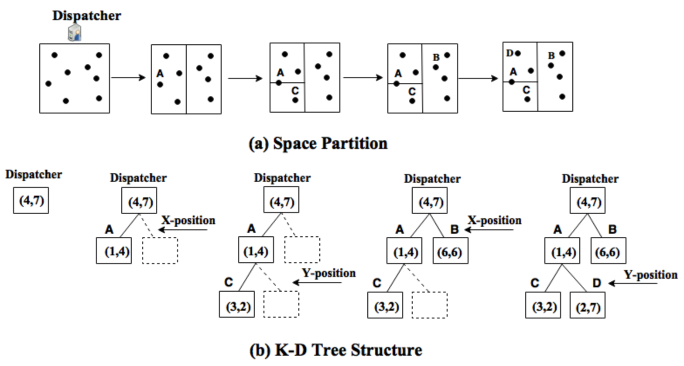

Fig. 3: Example showing how a K-D tree grows when drivers A, C, B, D park in a 8x8 area. Dots represent parking spaces and letters represent parked drivers. The numbers in each box of sub-figure (b) are the 2-D coordinates of the parking space of the corresponding driver.

manager or the parking manager of the region's corresponding to the node (depending on the node's role).

B. Joining and Departing K-D Tree

The K-D tree grows dynamically when more drivers park. When a driver informs its parking manager that it had parked, the parking manager creates a sub-region and a new node for the sub-region. Then, it attaches the new node as the child of the node it manages and assigns the newly parked driver to manage the new node and the parking spaces in the corresponding sub-region. Thus, the newly parked driver becomes the parking manager for these parking spaces. Over time, it also becomes a region manager when it has to divide this sub-region.

When a parking manager creates sub-regions, it divides its region based on the location of its parking space. This design has two advantages over evenly splitting the entire region among all the parked drivers. First, it helps to evenly distribute parking requests to parking managers. Due to hot spots, the destinations of drivers are not distributed evenly in the region. If the entire region is evenly partitioned among parking managers, the drivers managing hot areas might be overloaded. In contrast, the method employed by DFPS guarantees that more parking managers are assigned to hot areas than cold areas. Second, sub-regions are created dynamically within a small region where the driver parks. Other regions are not affected. This minimizes the changes to regions and the associated overhead, such as exchanged messages.

Figure 3 shows an example to illustrate how a K-D tree grows, in which four drivers (A, C, B, D) park sequentially in a region of size 8x8. Figure 3-a shows how the sub-regions are created when these drivers park, and Figure 3-b shows how each new node is created and added to the K-D tree. Initially, before any driver parks, there is only one node (i.e., the dispatcher) in the tree, managing the entire region. The location of the dispatcher ((4,7) in the figure) is chosen such that its x coordinate is in the middle of the region, whereas its y coordinate is chosen at random.

When driver A parks at coordinate $(1, 4)$, the whole region is split into two sub-regions. Sub-regions are created by splitting the parent region along alternating dimensions depending on the K-D tree level of the node managing the parent region. At the root level, the x dimension is used. In our example, the first splitting is along the x coordinate of the dispatcher. The sub-region where driver A parks is assigned to A. Though the dispatcher is still the region manager of the entire region, node A is the parking manager of the sub-region assigned to it. The dispatcher remains the parking manager for the other sub-region.

Driver C requests a parking space in the region whose parking manager is A. Assume that A assigns parking space (3,2) for driver C. When C parks, the new node created for C is added as a child node of A. Thus, A splits its region into two sub-regions based on the y coordinate of A's parking space location. Since the parking space of C is in the bottom sub-region, C becomes the parking manager of this sub-region. Though A is the region manager of the region consisting of the two (top and bottom) sub-regions, it acts as the parking manager only for the top sub-region.

Driver B requests a parking space in the region managed by the dispatcher, and the dispatcher allocates a parking space at (6, 6) to B. When B parks, the node created for B is added as a child of the root. It then becomes the parking manager of the dispatcher's second sub-region (the right half of the whole region). Since A and B handle the parking space allocation of both dispatcher's sub-regions, the dispatcher does not handle parking space allocation any more.

When parked drivers leave their parking spaces, the tree nodes associated to those drivers must be updated. For each node managed by a leaving driver, if the node is a leaf, the node is deleted from the K-D tree and its parent node (i.e., the corresponding driver) takes over the parking space allocation task associated with the node. If the node is an internal node, one option is to apply existing solutions for deleting K-D tree internal nodes [6]. However, because the sub-trees rooted at the internal node's children must be re-organized to form another subtree, these solutions can be very expensive and may cause considerable overhead, especially when the sub-trees are large.

Instead of deleting an internal node, DFPS assigns the node to another parked driver. DFPS considers the following two situations:

- The node is the parking manager of the region containing its location (the node value). In this case, the node has only one child node. DFPS will find the driver who is managing the child node, and assign the node to this driver.
- The location of the node is managed by another node, i.e., another node is the parking manager of the region containing its location. DFPS will first locate the parking

manager. Then, it assigns the node to the driver of the parking manager. In the example shown in Figure 3, when A leaves, driver D will be asked to take care of node of A, since D is in charge of the parking space allocation in the region where A parked. Thus, later on, when the parking space is re-allocated by D to another driver E, E can be assigned to manage the node.

C. Request Forwarding

Each parking request is forwarded down the tree from the root until it reaches the corresponding parking manager, which will assign a parking space in its region. This process is described in Algorithm 1.

Algorithm 1 Parking request forwarding procedure

```
 1: Upon node $n$ receiving a parking request ($v$, $D_v$)
    // $v$ refers to a driver's identity and $D_v$ represents the driver's destination
    // Region(n) is the region managed by n when acting as parking manager
 2: if ($n$ is not a parking manager) then
 3:     Forward the request to the child whose region contains $D_v$.
 4: else if ($n$ is a parking manager that has no children) and ($D_v \in$
    $Region(n)$) then
 5:     Accept the request.
 6: else if ($n$ is a parking manager that has one child) then
 7:     if ($D_v \in Region(n)$) then
 8:         Accept the request.
 9:     else
10:         Forward the request to the child node.
11:     end if
12: end if
```

D. Load Balancing

Each parking manager receives requests for its region. However, the distribution of destinations and requests coupled with the tree-structure of the network can cause heavy load on certain managers. Heavy load leads to slow downs and significant battery consumption on the impacted phones. To avoid this problem, DFPS applies a simple load balancing mechanism. An overload threshold is determined by each parking manager as the difference between the local load δ (i.e., the number of requests that have been processed by the phone) and a load threshold α. If $\delta > \alpha$, an imbalance is detected and the phone removes itself from the system. As described in Section III-B, a phone of another parked driver will replace this phone in the overlay network and will handle any pending requests inherited from the removed phone.

E. Failure Recovery

DFPS employs the mechanism proposed in [6] to ensure that the system continues to function in the presence of failures or disconnections of the phones of parked drivers. For example, the phones may fail without warning if the drivers decide to turn them off. Failure or disconnection of the phones is detected by periodic gossip messages from their neighbors (see Table I). Each neighbor knows the region boundary of the failing node w and maintains a replica of the data it stores (e.g., the number of available spaces and total number of spaces) in order to restore the data and improve availability. In addition, a parent maintains a list of requests forwarded to w and requests assigned by w in case of failure. To avoid consistency problems, a disconnected parking manager will not attempt to reconnect to the system when the wireless connection becomes available again. Finally, let us note that the overload threshold at parking managers (used for load balancing as described in Section III-D) also reduces the effect of node failures or disconnections.

IV. Parking Space Assignment

In DFPS, each parking manager periodically runs the same parking space assignment algorithm to satisfy the requests that are forwarded to it and are still outstanding. The algorithm computes an assignment for these requests, aiming to optimize the total travel time of the drivers.

A. Assumptions and Problem Statement

The set of curbside parking spaces in the region is denoted $S = \{s_1, s_2, ..., s_m\}$. The set of drivers whose requests have been forwarded to the parking manager because their destinations are within its region is denoted $V = \{v_1, v_2, ..., v_n\}$. To the extent possible, each driver will be assigned a parking space in the region that contains her destination. If this is not possible, the driver will receive a parking space in a nearby region.

The destinations of the drivers are geographical locations, such as shops, bank, houses, hotels, etc. Their addresses are converted into latitude and longitude coordinates, similar to the locations of parking spaces. The drivers are assumed to be moving independently based on legal speeds and the congestion levels on different road segments. We also assume that the parking request is submitted to the dispatcher when the driver is within its request distance of the destination. Then, the request is forwarded to the corresponding parking manager.

A parking assignment of spaces to drivers is defined as Y: $V \rightarrow S$, where y_{ij} is the assignment of a driver $v_i \in V$ to a parking space $s_j \in S$:

$$y_{ij} = \begin{cases} 1, & \text{if } v_i \text{ is assigned to } s_j \\ 0, & \text{otherwise} \end{cases} \quad 1 \le i \le n, 1 \le j \le m \tag{1}$$

$$\sum_{i=1}^{n} y_{ij} \le 1, \quad 1 \le j \le m \ (i.e., s_j \in S) \tag{2a}$$

$$\sum_{j=1}^{m} y_{ij} = 1, \quad 1 \le i \le n \ (i.e., v_i \in V) \tag{2b}$$

Constrains 2a and 2b ensure, respectively, that a driver receives at most one space and that a space is not assigned to more than one driver.

For a set of drivers and a set of parking spaces, there may exist a large number of assignments. The algorithm is to find an assignment that can minimize the total travel time of the drivers. The travel time $TC(v_i)$ of a driver v_i is the time period from the moment when the driver submits her request until she arrives at her destination. It consists of two parts, the driving time and the walking time:

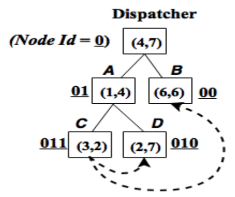

Fig. 4: Nodes in the maximal sibling subtrees of node C

- T_d (O_{v_i}, s_j) is the driving time of driver v_i from the moment she submits her request from location O_{v_i} until she parks at the parking space s_j.
- T_w (s_j, D_{v_i}) is the walking time of the driver from the moment she parks until the moment she arrives at her destination D_{v_i}.

B. Parking Space Assignment Algorithm

Each parking manager in DFPS assigns parking spaces in its region in the same way as the dispatcher assigns parking spaces in our centralized solution. The detailed algorithm can be found in [2]. A brief description is included below for convenience.

A parking manager assigns parking spaces periodically to outstanding requests. In our simulations, we experimentally determined that the time interval of 2s provides a good trade-off between performance and overhead. In each period, it first pre-allocates to the driver of each outstanding request the closest available parking space to her destination. The pre-allocation adapts the solution to the flow problem described in [7] to minimize the total walking time of these drivers. The actual assignment of parking spaces takes place based on the urgency of the demands for parking spaces, which is measured by how close the corresponding drivers are to their destinations or their pre-allocated parking spaces. Specifically, in each period, the drivers with the most urgent demand are selected. Their pre-allocated parking spaces are officially allocated to them.

The algorithm described above is named FPA. It assumes that subscribed drivers are generally representative of the entire driving population and all spaces in the region are available to them. To consider the cases in which spaces may be taken silently by unsubscribed drivers, the algorithm is enhanced to track the spaces taken by unsubscribed drivers and avoid assigning these spaces. This algorithm, described in [2], is named FPA-1.

FPA and FPA-1 are used under the assumption that there are still available parking spaces in the region. However, in DFPS, the assignment of parking spaces is done by multiple parking managers. It is possible that a parking manager runs out of parking spaces, but still has outstanding requests. In such a case, DFPS allows a parking manager to acquire parking spaces from nearby parking managers temporarily to satisfy her outstanding requests, as explained next.

C. Finding Best Available Spaces from Neighboring Regions

DFPS uses an indexing scheme based on the multi-indexing technique in [6] to locate nearby regions. The scheme assigns a binary identifier to each node as its index. The binary format of the index reflects the path from the root to the node, and thus reflects roughly the location of the corresponding region. For example, as shown in Figure 4, the index of the root node is 0 and the indexes of its children nodes (A and B) are 01 and 00. The first bit of the indexes (i.e., 0) reflects that they are in the region of the root node (i.e., index 0). The second bit (i.e., 1 or 0) reflects the corresponding sub-region created by the root node. Nodes C and D are the children of node A, and the first two bits (i.e., 01) reflect that they are in the region of node A (i.e., index 01), and the last bit reflects the sub-region.

TABLE I: Neighbor relation table

Node	Neighbors		
C(011)	D(010)	B(00)	
D(010)	C(011)	B(00)	
B(00)	A (01)	C(011)	D(010)

In DFPS, each parking manager maintains a neighbor table, as shown in Table I, which includes the nodes managing the neighboring regions, named neighboring nodes. For any two nodes (N_1 and N_2) with indexes of lengths L_1 and L_2 respectively, the two nodes are neighboring nodes if one of the following two conditions is met: 1) If $L_1 \leq L_2$, the first $L_1 - 1$ bits of the two indexes are the same, and the ${L_1}^{th}$ bits are different. 2) If $L_1 > L_2$, the first $L_2 - 1$ bits of the two indexes are the same, and the ${L_2}^{th}$ bits are different. The neighbor table is built by broadcasting the index of each newly-created node.

The best parking spaces are those close to the destinations of the requests. To find such spaces, a parking manager that runs out of spaces first needs to contact her neighboring nodes to get their lists of available parking spaces. Note that a neighboring node may not be a parking manager, which has first-hand information on available parking spaces. If a neighboring node is not a parking manager, to obtain a list of available parking spaces in its region, the node needs space availability reports from all its offsprings. Then, the parking manager short of spaces examines the parking spaces in the lists received from its neighbors, calculates the distances between the spaces and the destinations of the pending requests, and selects the parking space with the shortest distance for each of these requests.

V. Experimental Evaluation

Our simulation-based evaluation has three goals. First, it assesses the scalability and load balancing of DFPS. Second, it measures the benefits of DFPS in terms of travel time when compared to a Naive parking solution, which resembles what drivers normally do. Third, it compares the performance of DFPS with that of the centralized system (FPS) under two scenarios: (i) all drivers subscribe to our systems; (ii) a fraction of drivers does not subscribe to our systems.

A. Simulation Setup

Our experiments use SUMO [4] and PeerSim [5]. The road network and the locations of curbside parking spaces are imported into the simulator based on the real map of a business

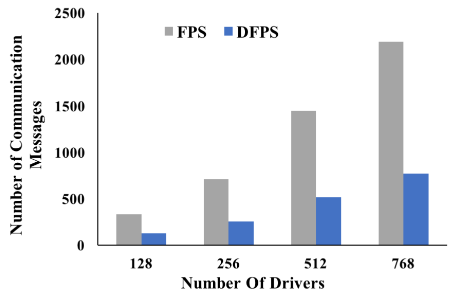

Fig. 5: Number of messages handled by the dispatcher/server in DFPS and FPS. The number of destinations is 8

district in Manhattan, New York City. The total number of parking spaces is 1024, and the total number of travel destinations is 400. SUMO simulates the traffic and sends commands to drivers to guide them to the assigned spaces. PeerSim simulates the communication in the distributed system.

We generate a set of random trips for the drivers. While the starting locations of the vehicles are randomly chosen over the entire road network, the destinations are chosen from a small region in the center of the map to ensure enough contention for parking spaces. Once a vehicle parks, we calculate the driving time and the walking time; for walking time, we consider a speed of 1.4 m/s, which is reasonable for adults [8]. DFPS starts each test with 1024 vacant parking spaces. The arrival rate of the requests falls within the range of 1 to 5 requests per second. For each experiment, we collected results from 5 runs and averaged them.

The parameters of the assignment algorithms (i.e., the scheduling period and the request distance) are set to the values of the centralized system [2] to have a fair comparison.

B. Results and Analysis

Scalability and Load Balancing. In DFPS, the computation bottleneck at the central server (i.e., dispatcher) is removed, as the parking assignment is computed in a distributed fashion by the phones of the drivers. Therefore, DFPS scales better from a computation point of view.

The total computation time consumed in each period for the parking assignment algorithm is the sum of (1) finding a valid allocation that minimizes the total walking time to destinations and (2) determining the urgency of pending requests in order to assign spaces to the most urgent requests and minimize the total driving time.

Minimum-cost network flow in a directed bipartite graph is used to generate a valid allocation. Its cost is $O(ve)$, where v is a number of nodes (i.e., m spaces + n drivers in the region) and e is the number of edges (i.e., equal to n, the number of drivers in the region). The cost of computation to determine request urgency and select urgent requests is $O((n+\varphi)^2)$, where n is the number of drivers to be assigned parking spaces and φ is the number of drivers with high urgency. Thus, the total time for each parking manager is $O(n^2+nm)$+$O((n+\varphi)^2)$. Given that each parking manager handles only a limited number of parking spaces and drivers, as described in Section III-D, this computation can easily be done on today's smart phones.

TABLE II: Maximum number of assigned requests by a parked driver's mobile phone for different numbers of destinations and a constant number of drivers (768)

	Destinations		
	2	**4**	**8**
DFPS	13	12	12
DFPS*	268	166	100

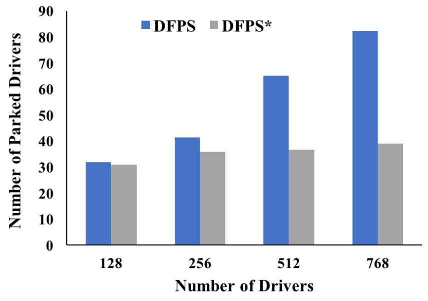

Fig. 6: Number of parked drivers involved in the assignment process. The number of destinations is 8.

Figure 5 compares the number of messages handled by the dispatcher in DFPS and the server in FPS by varying the number of drivers from 128 to 768. The results show that DFPS reduces the number of messages by a factor of 2 when compared to FPS.

We next investigate the effect of load balancing on parking managers. We compare the number of assigned requests by a phone of parked driver when DFPS employs its load balancing mechanism (DFPS) and when it does not (DFPS*). The value of the load threshold α in the load balancing mechanism is experimentally determined to be 10, which provides a good trade off between performance and overhead. Table II compares the maximum number of assigned requests in DFPS and DFPS*. As expected, DFPS scales better due to its load balancing mechanism. The maximum number of requests in DFPS* is about 20 times higher than the maximum number in DFPS. We also observe that the maximum in DFPS is 13, not 10 as expected (the load threshold). This is because of two reasons. First, a parking manager receives requests until it has served α of them (while the others are pending). Second, a parking manager can not depart the network until it completes its set of requests with high urgency.

Figure 6 presents another measure of scalability: the average number of parking managers cooperating to serve the incoming parking requests in DFPS and DFPS*. The results show that the number of managers in DFPS increases by 185% compared to DFPS*. Higher numbers are better because the load is distributed more evenly across the participants, and the system scales better.

Travel Time Performance. We conduct an experiment to find out the gains and losses in travel time for individual drivers using DFPS when comparing to the travel time obtained by a Naive solution (i.e., a baseline assignment algorithm that assumes the driver goes to the destination and, after arriving there, she starts a breadth-first-search for parking spaces along the nearby road segments). To measure the gains/losses, we calculate the ratio between the travel time

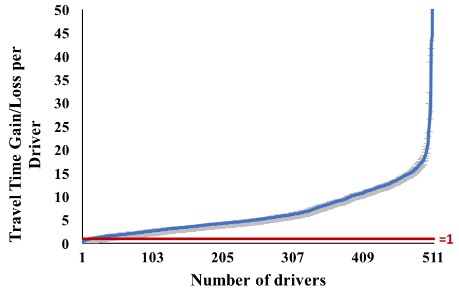

Fig. 7: Distribution of travel time gain/loss for 512 drivers. The number of destinations is 8. Error bars are shown.

obtained by the Naive solution and the travel time obtained by DFPS for each driver. If the ratio is higher than 1, the driver has benefited from DFPS. Otherwise, the driver has not.

Figure 7 plots the distribution of individual travel time gains/losses for all drivers in the experiment. We observe that the system manages to improve the travel time for a large majority of drivers (over 97%). Many drivers reduce their travel times by more than an order of magnitude. These results are possible due to the high parking contention generated in the experiment, which leads to high traffic congestion and thus to very high travel times for the Naive solution. The error bars in the figure demonstrate that these results are consistent across different simulation runs.

Comparison with Centralized System. While the previous results show that DFPS outperforms the Naive solution, one may ask whether DFPS can achieve performance similar to the centralized system (FPS) [2]. We use the *average travel time* as a metric to compare their performance. The experiments simulate two different scenarios: *subscribed-driver-only*, which assumes that all drivers in the system use DFPS; *unsubscribed-driver-interference*, which assumes that there are a number of drivers who have not subscribed to DFPS. In the second scenario, the unsubscribed drivers may occupy, without notification, parking spaces known to the system as available.

Figure 8 shows the average travel time for DFPS vs. FPS in the *subscribed-driver-only* scenario. We observe a difference of 1% fluctuating between the two systems. For the same scenario, Figure 9 shows the average travel time when the number of destinations is varied from 2 to 8. The figure also plots the contribution of walking time and driving time in the total time. Similar results are obtained for DFPS and FPS. Therefore, DFPS is a better solution because it reduces the computation and communication bottleneck at the server/dispatcher, while obtaining similar travel time with FPS.

The final set of experiments evaluates the performance of DFPS and FPS in the *unsubscribed-driver-interference* scenario. Each system, has two parking assignment algorithms. One (DFPS/FPA and FPS/FPA) just keeps trying to find another space if the assigned parking space is found to be taken by an unsubscribed driver. The other (DFPS/FPA-1 and FPS/FPA-1) keeps track of the spaces found to be taken by unsubscribed drivers and tries them later. We call the spaces taken by unsubscribed drivers “hidden” spaces. The interested reader can refer to [2] for more details on how we model the

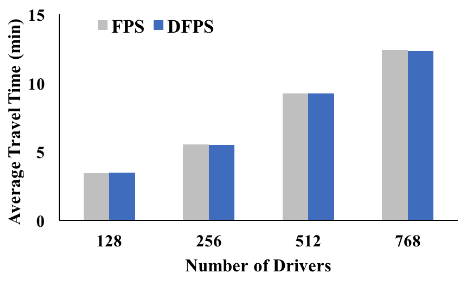

Fig. 8: Average travel time with different number of drivers and 8 destinations

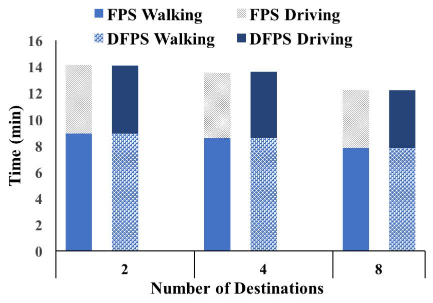

Fig. 9: Walking and driving time of DFPS and FPS for different number of destinations and 768 drivers

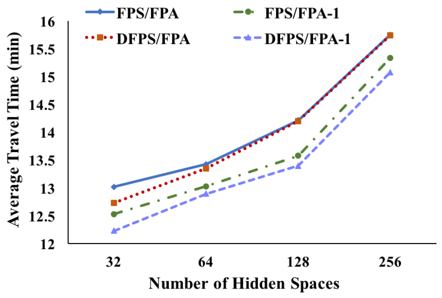

Fig. 10: Average travel time when varying the number of hidden spaces; 768 drivers and 8 destinations.

behavior of unsubscribed drivers.

Figure 10 compares the performance difference between DFPS and FPS when the number of hidden spaces is varied from 32 to 256. We observe that DFPS outperforms FPS consistently. We also notice that DFPS/FPA-1 achieves the lowest average travel time. These results demonstrate that DFPS/FPA-1 adapts very well to the interference caused by unsubscribed drivers.

VI. Related Work

Several solutions have been developed in order to assist drivers with parking, and they focus on implementing a wireless sensor network to detect parking information and provide real-time parking services. For example, SFpark [1] installs sensors, each costing $500, in the asphalt to detect the parking availability on the streets. ParkNet [9] uses ultrasonic sensor technology on a vehicle door, each costing $400, to detect and collect parking availability information, and then build a real-time map for drivers. Although these solutions increase the probability of finding vacant spaces, they have several shortcomings. First, the cost involved in deploying and maintaining the sensor infrastructure is extremely high. Second, all drivers see the same map at any given time, and many of them will compete for the same spaces. This parking contention problem leads to traffic congestion and frustrating driving/parking experience. DFPS does not rely on expensive infrastructure and guides drivers to their assigned spaces. It relies on cooperative mobile phones, which is a cheaper, more convenient, and more flexible alternative.

Several works explicitly consider parking space contention when assigning parking spaces to drivers. Parking assignment is studied in two contexts. The first is centralized, in which a central authority makes parking decisions and assigns each driver to a specific parking space. The second is a model with distributed agents that are competing for the parking spaces. The works in [2], [10] are examples of centralized solutions. Ayala et al. [10] developed a pricing model to minimize the system-wide driving distance. However, the proposed approach depends on pricing data and is offline in nature, as the number of drivers and resources are known in advance and do not dynamically change. In [2], we introduced FPS for on-the-fly curbside parking assignment. Unlike other parking systems, FPS adapts on-the-fly to new parking requests and shows good performance in reducing the total travel time for drivers. However, it has a scalability problem, inherent to a centralized solution. DFPS solves this problem, while maintaining the same level of travel time performance.

Among the decentralized solutions, Ayala et al. [7] analyze the parking problem as a competitive game in which individual, selfish drivers are competing for the available spaces. It is assumed that each vehicle has access to the location of other vehicles, which raises privacy concerns and has technical difficulties in real-time. In DFPS, vehicles share location information only with parking managers at the time of request submission. Furthermore, the main objective in Ayala et al. [7] is to prove the existence of an equilibrium for the considered game. However, the authors show that the cost (i.e., travel time) for the whole system can get arbitrarily worse than the cost of the optimal assignment. DFPS, on the other hand, optimizes the system cost (i.e., total travel time). In [11], the parking assignment capitalizes on the ability of a trustworthy central controller to construct a feasible assignment in a distributed fashion via the coordination of drivers. The problem with this solution is that the assignment computation and communication are burdens on the coordinator. In DFPS, we reduce the communication and computation cost by offloading the assignment process to the parked drivers, where each one manages and assigns drivers to spaces in their regions. Thus, DFPS achieves better scalability.

VII. Conclusion and Future Work

This paper presented DFPS, a cost-effective and efficient distributed parking assignment system that can be implemented and deployed in real-life settings. DFPS uses the smart phones of the drivers to offload the computation of parking request assignments, and thus the assignment process becomes scalable in real-time. Parked drivers cooperate to serve parking requests in a distributed fashion while optimizing the social welfare of the whole system, i.e., minimizing the total travel time. DFPS is scalable and achieves performance similar to a centralized system.

So far, DFPS does not protect the driver privacy in terms of hiding her destination from the dispatcher and potentially from other drivers. To solve this problem, we are currently extending DFPS to support privacy-aware parking requests through location cloaking based on k-anonymity. This future work will ensure that the effect of privacy protection on DFPS's performance is minimal.

VIII. Acknowledgment

This research was supported by the National Science Foundation (NSF) under Grants No. CNS 1409523, SHF 1617749, CNS 1054754, and DGE 1565478, and by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under Contract No. A8650-15-C-7521. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF, DARPA, and AFRL. The United States Government is authorized to reproduce and distribute reprints notwithstanding any copyright notice herein.

References

[1] "Sfpark," http://sfpark.org/.

[2] A. Hakeem, N. Gehani, X. Ding, R. Curtmola, and C. Borcea, "On-The-Fly Curbside Parking Assignment," in *Proceedings of the 8th EAI International Conference on Mobile Computing, Applications and Services*, ser. MobiCASE'16, 2016, pp. 1–10.

[3] E. Nardelli, "Distributed K-D Trees," in *Proceedings 16th Conference of Chilean Computer Science Society (SCCC'96)*, 1996, pp. 142–154.

[4] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "SUMO - Simulation of Urban MObility: An overview," in *Proceedings of the Third International Conference on Advances in System Simulation (SIMUL)*, 2011, pp. 63–68.

[5] A. Montresor and M. Jelasity, "PeerSim: A Scalable P2P Simulator," in *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*, 2009, pp. 99–100.

[6] G. Tsatsanifos, D. Sacharidis, and T. Sellis, "MIDAS: Multi-Attribute Indexing for Distributed Architecture Systems," in *Advances in Spatial and Temporal Databases*. Springer, 2011, pp. 168–185.

[7] D. Ayala, O. Wolfson, B. Xu, B. Dasgupta, and J. Lin, "Parking Slot Assignment Games," in *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2011, pp. 299–308.

[8] R. W. Bohannon, A. W. Andrews, and M. W. Thomas, "Walking Speed: Reference Values and Correlates for Older Adults," *Journal of Orthopaedic & Sports Physical Therapy*, vol. 24, no. 2, pp. 86–90, 1996.

[9] S. Mathur, T. Jin, N. Kasturirangan, J. Chandrasekaran, W. Xue, M. Gruteser, and W. Trappe, "ParkNet: Drive-by Sensing of Road-side Parking Statistics," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2010, pp. 123–136.

[10] D. Ayala, O. Wolfson, B. Xu, B. DasGupta, and J. Lin, "Pricing of Parking for Congestion Reduction," in *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, 2012, pp. 43–51.

[11] E. Alfonsetti, P. C. Weeraddana, and C. Fischione, "A Semi Distributed Approach for Min-Max Fair Car-Parking Slot Assignment Problem," *arXiv preprint arXiv:1401.6210*, 2014.