GreenDrive: A Smartphone-based Intelligent Speed Adaptation System with Real-time Traffic Signal Prediction

Yiran Zhao
University of Illinois at Urbana-Champaign
zhao97@illinois.edu

Shen Li
University of Illinois at Urbana-Champaign
shenli3@illinois.edu

Shaohan Hu
IBM Research
shaohan.hu@ibm.com

Lu Su
State University of New York at Buffalo
lusu@buffalo.edu

Shuochao Yao
University of Illinois at Urbana-Champaign
syao9@illinois.edu

Huajie Shao
University of Illinois at Urbana-Champaign
hshao5@illinois.edu

Hongwei Wang
University of Illinois at Urbana-Champaign
hwang172@illinois.edu

Tarek Abdelzaher
University of Illinois at Urbana-Champaign
zaher@illinois.edu

ABSTRACT

This paper presents the design and evaluation of *GreenDrive*, a smartphone-based system that helps drivers save fuel by judiciously advising on driving speed to match the *signal phase and timing (SPAT)* of upcoming signalized traffic intersections. In the absence of such advice, the default driver behavior is usually to accelerate to (near) the maximum legally allowable speed, traffic conditions permitting. This behavior is suboptimal if the traffic light ahead will turn red just before the vehicle arrives at the intersection. *GreenDrive* uses collected real-time vehicle mobility data to predict exact signal timing a few tens of seconds ahead, which allows it to offer advice on speed that saves fuel by avoiding unnecessary acceleration that leads to arriving too soon and stopping at red lights. Our work differs from previous work in three respects. First and most importantly, we tackle the more challenging scenario, where some phases (such as left-turn arrows) are added or skipped dynamically, in accordance with real-time traffic demand. Second, our approach can accommodate a low system penetration rate and low vehicle density. Third, *GreenDrive* treats user-specified travel time requirements as soft deadlines and chooses appropriate speed adaptation strategies according to the user time budget. Using SUMO traffic simulator with real and large-scale road network, we show that *GreenDrive* learns phase durations with an average error below 2s, and reduces fuel consumption by up to 23.9%. Real-world experiments confirm 31.2% fuel saving and the ability to meet end-to-end travel time requirements.

ICCPS 2017, Pittsburgh, PA USA

DOI: http://dx.doi.org/10.1145/3055004.3055009

CCS CONCEPTS

•Applied computing →Transportation; •Computing methodologies →Machine learning;

KEYWORDS

Traffic signal prediction, smartphone sensing, optimal speed advisory

ACM Reference format:
Yiran Zhao, Shen Li, Shaohan Hu, Lu Su, Shuochao Yao, Huajie Shao, Hongwei Wang, and Tarek Abdelzaher. 2017. GreenDrive: A Smartphone-based Intelligent Speed Adaptation System with Real-time Traffic Signal Prediction. In *Proceedings of The 8th ACM/IEEE International Conference on Cyber-Physical Systems, Pittsburgh, PA USA, April 2017 (ICCPS 2017),* 10 pages.
DOI: http://dx.doi.org/10.1145/3055004.3055009

1 INTRODUCTION

Drivers often exhibit unnecessary "hurry up and wait" behavior when vehicles accelerate too much from a previous intersection only to arrive at a red signal. This behavior leads to higher fuel consumption because the energy is lost when the vehicle stops completely. *GreenDrive* significantly reduces complete halts at red lights and saves fuel by learning the traffic signal schedule and suggesting optimal driving speed. In general, there are two kinds of approaches, namely, infrastructure-centric and vehicle-centric. The infrastructure-centric approach installs intelligent traffic signal control systems to adapt signal schedule to real-time traffic. It can be complemented by vehicle-centric solutions that offer intelligent speed adaptation advice to drivers, assisting them in choosing the right speed based on a predicted real-time traffic signal schedule.

Infrastructure-centric systems such as SCATS [23], SCOOT [12], and RHODES [10] have been smoothing traffic flows for years in more than a hundred cities around the globe. In academia, intelligent traffic signal control systems are often based on job scheduling algorithms [25], adjacent junction coordination [11], or vehicle arrival time estimation [21]. These systems require the infrastructure to collect real-time vehicle speed and position data via vehicular ad-hoc networks (VANETs) [20], vehicle to infrastructure

(V2I) communication [22] or a combination of both [8]. However, such infrastructure-centric systems are expensive to install, which greatly hinders wide adoption.

In light of this problem, an increasing amount of effort on reducing fuel consumption shifted to vehicle-centric approaches that help adapt driving speed to the traffic signal schedule. Some previous work [2, 4, 24] assumed that the signal schedule is readily available to navigation devices. However, in reality such information is managed by the transportation department, and cannot be accessed in real-time. We observe that signal schedule can be learned from spatial and temporal information conveyed by vehicle movements at intersections. Thanks to the ubiquity of powerful embedded systems such as wearable devices [32] and smartphones, vehicle movements can be captured by sensors and collected in a crowd-sourced manner. Moreover, it is easier to achieve higher system penetration by deploying intelligent speed adaptation systems on smartphones than on built-in vehicular navigational devices. This motivates the development of *GreenDrive*.

GreenDrive uses crowdsourced vehicle movements to predict real-time signal schedules a few tens of seconds ahead and share that information with drivers in the form of speed advice that reduces unnecessary accelerations and stops. We show that the system results in more than 20% fuel savings and can meet end-to-end travel time requirements. The contributions of our work are thus threefold:

(1) To the best of our knowledge, our system is the first to learn the adaptive schedules of intelligent traffic control systems using only smartphone sensors and GPS.
(2) Our system chooses different speed adaptation strategies in real-time according to the predicted schedule and driver's time budget, achieving fuel savings while meeting travel time requirements.
(3) We conduct large-scale realistic simulation using SUMO [3] with real road network to test system performance under low penetration rate. We also conduct real-world experiments to confirm fuel saving.

The rest of this paper is organized as follows. In Section 2, we discuss related work. Section 3 details the system architecture and methodologies. Section 4 presents simulation results. The real-world experiment results are in Section 5. Finally, Section 6 concludes the paper.

2 RELATED WORK

Learning traffic light timing and phase information without a dedicated infrastructure has attracted increasing attention from urban sensing researches. A number of papers propose digital image processing techniques to capture real-time traffic light states [7, 19]. The vision-based approach has its limitations. For example, at an urban intersection with a high traffic volume, the line-of-sight from the signal to the car-mounted camera may be obstructed. It may also be difficult to distinguish traffic lights in cluttered scenes. But with careful system setup and fine-tuning traffic light detection parameters, SignalGuru [16] successfully tracks and predicts traffic signal states by leveraging smartphone cameras and ad-hoc networks. SignalGuru requires smartphones to be mounted on the windshield with some tilting angle, while our system allows arbitrary smartphone positioning and user fiddling. In addition, we employ a central backend server to collect information and enable optimal speed advisories even when the traffic lights of the next intersection are yet to be seen. Compared to the results presented in the SignalGuru paper, our system achieves improved fuel savings and a similar phase duration learning accuracy.

Alternative to the vision-based approach, traffic signal schedule can be inferred from vehicle spatial and temporal movements. Prior work [14] shows that collective velocity profiles can be used to infer signal schedules with either a fixed or slowly changing phase duration. Other papers [18, 26] suggest that historical timing data combined with partial real-time signal phase data can be used to reduce idling in red lights and save fuel. Based on that information, multiple approaches [24, 28] feature algorithms for optimal speed planning. However, they (i) do not target intelligent traffic control systems, (ii) only focus on theory or small-scale simulation, or (iii) do not validate their results in real-world experiments. We compare our simulation results to the aforementioned work in terms of fuel saving.

More recently, CityDrive [33] implemented a smartphone-based system that learns fixed traffic signal schedules and enables optimal speed advisory services. Different from CityDrive, we target traffic signals with intelligent traffic control where the phase schedule is adapted dynamically to traffic conditions. We also conduct large-scale and more realistic simulations to comprehensively evaluate our system.

3 SYSTEM DESIGN

3.1 System overview

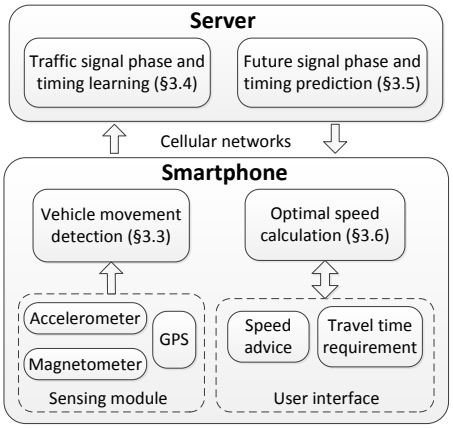

Figure 1: System architecture.

GreenDrive is an infrastructure-less and easy-to-deploy Intelligent Speed Adaptation system using only smartphones and Internet servers. The overall system architecture is shown in Figure 1. Sensing capabilities on smartphones are exploited to detect vehicle movements at intersections, which are then sent to the server through cellular networks. The server first collects vehicle movement data for a sufficiently long period of time to learn the Signal Phase and Timing (SPAT) of each intersection. Then the server predicts future signal timing using the learned SPAT and real-time vehicle movements. The smartphone requests the predicted signal schedule of the intersection ahead and estimates remaining travel time, and then adaptively chooses between a fuel-saving mode and a time-saving mode according to the time budget.

3.2 Traffic signal control system background and notations

Nowadays, many major intersections are installed with intelligent traffic control system, where vehicles waiting on the left-turn lane trigger the system to insert a left-turn phase to meet the actual traffic demand [15]. This on-demand system is more complicated than fixed-time control system. In this case, although each phase has fixed duration, the phase sequence of intelligent traffic control system dynamically changes to adapt to real-time traffic demand.

CityDrive [33] targets traffic signals with fixed-time control systems that contain only 4 fixed phases, and therefore their schedules are mush easier to learn and predict. However, we tackle the more complex 8-phased intelligent traffic control system, where the phase sequence is adaptively changing to accommodate the need of left-turn vehicles.

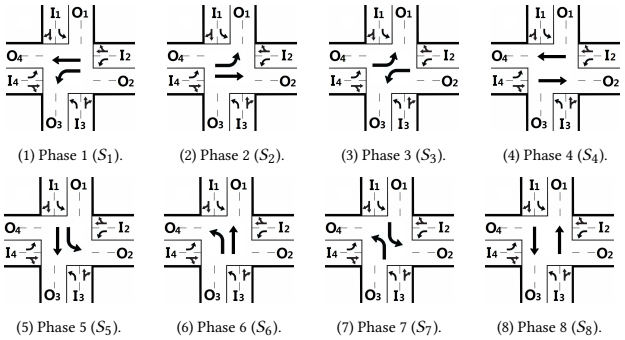

Figure 2: Signal phases of intelligent traffic control.

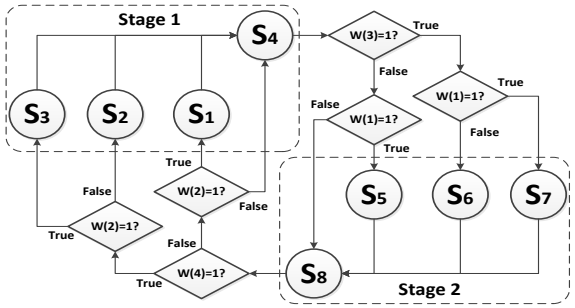

Figure 3: Phase transition of intelligent traffic control.

The 8 phases for a 4-arm intersection are shown in Figure 2 according to [27], where arrows show the vehicle acceleration upon the start of the phase. To facilitate our description, we make the following notations. Shown in Figure 2, let S_i ($1 \le i \le 8$) denote the i-th phase, whose duration is T_{p_i}. Suppose the arm indices start from North and increase clockwise. We name each incoming arm I_i and outgoing arm O_i, where i is the arm index. Then the phase transition logic according to traffic signal design guidelines [27] is shown in Figure 3. Function $w(i) = 1$ means that there are vehicles waiting on the left-turn lane of arm I_i, otherwise $w(i) = 0$. For example, S_1 occurs if at the end of the phase S_8, there are vehicles waiting on the left-turn lane of I_2 while no vehicle is waiting on the left-turn lane of I_4. In this case, the intelligent control system inserts the left-turn phase S_1 before transitioning to S_4.

We divide the phases into two stages of perpendicular flows. As shown in Figure 3, stage 1 corresponds to east or west approaching directions, which is composed of S_4 alone or a left-turn phase (S_1, S_2 or S_3) followed by S_4. Stage 2 corresponds to north or south approaching directions, which is composed of S_8 alone or a left-turn phase (S_5, S_6 or S_7) followed by S_8. Although the duration of each phase is fixed, the actual phase sequence in each stage depends on real-time traffic demand, therefore the total duration of a stage can change every traffic signal cycle.

Let T_{d_i} ($1 \le i \le 8$) denote the total duration of the stage starting with S_i. The total duration can be the duration of a through phase alone or the sum of a left turn phase and a through phase. With reference to Figure 3, for stage 1 there are four possible total durations: $T_{d_1} = T_{p_1} + T_{p_4}$, $T_{d_2} = T_{p_2} + T_{p_4}$, $T_{d_3} = T_{p_3} + T_{p_4}$, $T_{d_4} = T_{p_4}$. Similarly, for stage 2, $T_{d_5} = T_{p_5} + T_{p_8}$, $T_{d_6} = T_{p_6} + T_{p_8}$, $T_{d_7} = T_{p_7} + T_{p_8}$, $T_{d_8} = T_{p_8}$. Therefore, learning SPAT means to learn all 8 total durations.

3.3 Vehicle movement detection

Shown in Figure 1, the first step is to detect vehicle movements and send the movement data to the server. We use smartphone's accelerometer, magnetometer and GPS to reliably and accurately detect vehicle movements at intersections [30, 33]. The movement data should include spacial and temporal information of the vehicle either when it accelerates after waiting in red light, or when it directly passes the intersection without a stop. The format of the vehicle movement is $(n_{in}, n_{out}, t_{acc}, is_{acc})$, where n_{in} is the incoming arm index, n_{out} is the outgoing arm index, and is_{acc} indicates whether the movement is an acceleration or a direct pass. t_{acc} is the timestamp of acceleration if $is_{acc} = 1$, or the timestamp of the direct pass if $is_{acc} = 0$. The arm indices help to infer stage and phase number, and timestamp conveys phase transition time.

To detect acceleration, the smartphone feeds the accelerometer and magnetometer data into the Android API function *getRotationMatrix()* to get the acceleration vector in the earth's North-East-Down coordinate system. Then the acceleration vector is projected onto the vehicle's heading direction (given by GPS) so that the acceleration along the driving direction can be calculated. This acceleration is cumulated over a 1-second sliding window to filter out jitters caused by user fiddling. Preliminary test shows that on average the acceleration detection false positive rate is 1.9%, the false negative rate is 3.6%, and the average detection delay is 1.014s, with standard deviation 0.334s.

To detect a direct pass, the smartphone monitors the transition of road segments using GPS and local map. If the transition occurs at an intersection, the n_{in} and n_{out} are recorded, and the timestamp of the GPS point closest to the intersection center is chosen to be the movement timestamp. A direct pass movement is helpful to infer traffic signal schedule, because it tells that the stage containing the movement is in green phase.

3.4 Learning traffic signal phase and timing

For each intersection, the server collects the movement data from vehicles going through that intersection to learn its traffic signal schedule. To describe the learning process, we first make several additional notations.

3.4.1 *Notations.*

Let $Mov(i, j)$ denote the movement from arm I_i to arm O_j. Each movement $Mov(i, j)$ can be categorized into two types. One is $Acc(i, j)$ meaning that the vehicle has stopped, waited and then

accelerated. The other is $Pas(i,j)$ which means the movement is a direct pass without a stop.

For each intersection, the server stores the collected movement data in a table sorted by t_{acc}. Using the movement data format in Section 3.3, each movement (n_{in}, n_{out}, t_{acc}, is_{acc}) is denoted by $Pas(n_{in},n_{out})$ if $is_{acc} = 0$, or otherwise by $Acc(n_{in},n_{out})$ if it is an acceleration. An example of movement data of a particular intersection is shown in Table 1, where "Sample" column will be explained later. We say two movements happen simultaneously if their t_{acc} difference is less than 5s (e.g. row 3, 4), otherwise they're time-separated.

In our work we assume that the left-turn traffic signal is symmetric for traffic from opposite directions (180°), which means the added left-turn phases for opposite directions have the same duration. However, we don't assume symmetry for perpendicular directions (90°). Using the notations in Section 3.2, this implies $T_{p_1} = T_{p_2}$, $T_{p_5} = T_{p_6}$, $T_{d_1} = T_{d_2}$, and $T_{d_5} = T_{d_6}$. We have 3 reasons for this assumption: 1) in our field observation, this symmetry is prevalent; 2) making it non-symmetric would be a trivial extension, since the methodology remains the same; 3) assuming non-symmetry significantly complicates the writing and notations.

The server groups consecutive movements that appear to be in the same stage into one sample, denoted by $x^{(i)}$, where i means the i-th sample. For example, in Table 1, row 1 and 2 are movements from west and east, respectively, so they both belong to stage 1, and form the first sample $x^{(1)}$. We define the time duration of sample $x^{(i)}$ be the earliest acceleration timestamp in $x^{(i+1)}$ minus the earliest acceleration timestamp in $x^{(i)}$. For example, the duration of $x^{(1)}$ is the absolute time difference of t_{acc} between row 3 and row 1.

3.4.2 Validating samples on servers.

Ideally, each sample represents a stage, and the sample's duration equals to the total duration of this stage. In reality, since our system allows low vehicle density and low penetration rate, the stage transition may not be captured by vehicle movements. If so, the samples' duration is not the true total duration of that stage and thus should be invalidated.

First, we check whether a sample's earliest movement is an acceleration event. If not, this sample itself and its previous sample are marked as invalid, because the phase transition timestamp between the two stages (samples) is unclear. For example, in Table 1, row 8 ($Pas(1,2)$) is not an acceleration movement, so it invalidates $x^{(4)}$ and $x^{(3)}$.

Second, we check if some samples contain multiple traffic signal cycles. If so, they usually span multiple stages and should have much larger total durations. Analyzing what phases are contained in such samples can be difficult, especially when the duration of each phase is unknown yet. Instead, we simply filter out the samples with durations larger than twice the median sample duration.

Table 1: Example movement data table.

Row #	Movement	n_{in}	n_{out}	t_{acc}	is_{acc}	Sample
1	$Acc(4,2)$	4	2	484	1	$x^{(1)}$
2	$Pas(2,3)$	2	3	500	0	
3	$Acc(1,2)$	1	2	509	1	$x^{(2)}$
4	$Acc(3,4)$	3	4	510	1	
5	$Acc(3,1)$	3	1	522	1	
6	$Acc(2,4)$	2	4	546	1	$x^{(3)}$
7	$Acc(4,2)$	4	2	549	1	
8	$Pas(1,2)$	1	2	566	0	$x^{(4)}$
9	$Pas(1,3)$	1	3	680	0	

Similarly, false positive acceleration movements, though rare, usually result in samples with much smaller durations, as they tend to split up a stage incorrectly. Therefore, we filter out samples with duration smaller than half of the median sample duration. After the filtering, the valid samples are fed to Expectation-Maximization algorithm to learn the phase durations.

3.4.3 Expectation-Maximization algorithm.

The goal of this algorithm is to find out T_{d_i} ($1 \le i \le 8$). It would be a trivial task if we know the phases contained in each sample. However, each sample does not come with a label saying whether it is composed of one through phase only, or a left-turn phase plus a through phase. So we use a latent variable z to represent which case the sample belongs to, and try to maximize the likelihood of observed samples. As mentioned in Section 3.4.1, we consider symmetric left-turn signals, so there are 3 total duration values for each stage. This means the duration of each valid sample should be one of the 3 total durations of its stage, and z for each sample should take one of the three values (1, 2, or 3).

However, we cannot determine z solely by the sample's duration, since different phase sequences could have similar total duration. And we also cannot determine z just by analyzing the movements in each sample, because low vehicle density and low system penetration rate may cause missing movement reports. For example, in Table 1, row 3 and 4 are two simultaneous left-turn accelerations from opposite directions, it is easy to see that the sample $x^{(2)}$ starts with S_7. However, if row 3 is not reported (missing), $x^{(2)}$ can also start with S_6. But if $x^{(2)}$ has duration closer to T_{d_7}, then $x^{(2)}$ is still more likely to start with S_7. So we need some parameters describing the movement characteristics in each sample, and then find the maximum a posteriori (MAP) estimates of the parameters (including z).

Therefore, we apply EM algorithm to samples of the same stage of an intersection. Assume there are m samples in the sample set $\boldsymbol{X} = \{x^{(i)},\ i = 1,2,\ldots m\}$, we expand $x^{(i)}$ into a vector of 5 elements, each describing an observed feature:

$$x^{(i)} = \left[x_1^{(i)}\ x_2^{(i)}\ x_3^{(i)}\ x_4^{(i)}\ x_5^{(i)}\right]^T \tag{1}$$

where each element takes the following value:

(1) $x_1^{(i)} = 1$ if $x^{(i)}$ contains left-turn acceleration, otherwise $x_1^{(i)} = 0$.
(2) $x_2^{(i)} = 1$ if $x^{(i)}$ contains 2 phase transitions captured by time-separated accelerations, $x_2^{(i)} = 0$ if there's only 1.
(3) $x_3^{(i)} = 1$ if $x^{(i)}$ contains simultaneous left-turn and straight-through accelerations, otherwise $x_3^{(i)} = 0$.
(4) $x_4^{(i)} = 1$ if $x^{(i)}$ contains two simultaneous left-turn accelerations from opposite directions, otherwise $x_4^{(i)} = 0$.
(5) $x_5^{(i)}$ is the duration of $x^{(i)}$, which we assume to follow Gaussian distribution $N(\mu_5, \Sigma_j)$ (if $z = j$ for $x^{(i)}$).

The value of each element is easy to obtain by checking the movements contained in that sample. Take $x^{(2)}$ in Table 1 as an example. It has two simultaneous left-turn accelerations (row 3, 4), so $x_1^{(2)} = 1$, and $x_4^{(2)} = 1$. It also has 2 time-separated phase transition movements (row 3, 5) so $x_2^{(2)} = 1$. But $x_3^{(2)} = 0$ since

straight-through $Acc(3,1)$ does not happen simultaneously with its previous left-turn $Acc(3,4)$. Finally $x_5^{(2)} = 546-509 = 37$ seconds.

Suppose when z=j, the expected mean values of the 5 elements in vector form is $\boldsymbol{\mu}_j$=$[\mu_{j1}\ \mu_{j2}\ \mu_{j3}\ \mu_{j4}\ \mu_{j5}]^T$. Let $\boldsymbol{\mu} = \{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\mu}_3\}$. Let ϕ_j be the prior probability of z=j, $\boldsymbol{Z}$ be the set of latent variables, $\boldsymbol{\Phi} = \{\phi_1, \phi_2, \phi_3\}$, and $\Sigma = \{\Sigma_1, \Sigma_2, \Sigma_3\}$. Then, given a set of m observed data samples $\boldsymbol{X}$, a set of latent variables $\boldsymbol{Z}$, and unknown parameters $\boldsymbol{\theta} = \{\boldsymbol{\Phi}, \boldsymbol{\mu}, \Sigma\}$, we are trying to maximize the likelihood function:

$$\begin{aligned} L(\boldsymbol{\theta};\boldsymbol{X},\boldsymbol{Z}) &= p(\boldsymbol{X},\boldsymbol{Z}|\boldsymbol{\theta}) \\ &= \prod_{i=1}^{m} p(x^{(i)};\boldsymbol{\theta}) = \prod_{i=1}^{m}\sum_{z^{(i)}} p(x^{(i)},z^{(i)};\boldsymbol{\theta}) \\ &= \prod_{i=1}^{m}\sum_{j=1}^{3} p(x^{(i)}|z^{(i)}{=}j;\boldsymbol{\mu},\Sigma)p(z^{(i)}{=}j;\boldsymbol{\Phi}) \end{aligned} \tag{2}$$

Where $p(z^{(i)} = j;\boldsymbol{\Phi}) = \phi_j$, the latent variable of $x^{(i)}$ is $z^{(i)}$, and

$$\begin{aligned} &p(x^{(i)}|\, z^{(i)} = j;\boldsymbol{\mu},\Sigma) \\ &\quad= \prod_{k=1}^{4} p(x_k^{(i)}|z^{(i)} = j;\boldsymbol{\mu}) \cdot p(x_5^{(i)}|z^{(i)} = j;\mu_{j5},\Sigma) \\ &\quad= \prod_{k=1}^{4} \mu_{jk}^{x_k^{(i)}}(1-\mu_{jk})^{(1-x_k^{(i)})} \cdot f_n(x_5^{(i)},\mu_{j5},\sqrt{\Sigma_j}) \end{aligned} \tag{3}$$

In Equ. (3), μ_{jk} $(1 \le k \le 4)$ is the mean value of the k-th element for samples with $z = j$. μ_{j5} is the mean value of $x_5^{(i)}$ with $z = j$, which is also the mean total duration of samples having $z = j$. Σ_j is the variance of $x_5^{(i)}$ when $z = j$. And $f_n(x,\mu,\sigma)$ is the normal distribution function.

a) E-step:

We calculate the expected value of the log likelihood function with respect to $\boldsymbol{Z}$ given $\boldsymbol{X}$ and current estimate of the parameters $\boldsymbol{\theta}^{(t)}$:

$$Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) = E_{\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{\theta}^{(t)}}(log\ L(\boldsymbol{\theta};\boldsymbol{X},\boldsymbol{Z})) \tag{4}$$

Denote $w_j^{(i)}$ as the probability that $z^{(i)} = j$ (j=1,2,3). In E-step we calculate the best values for $\boldsymbol{Z}$ given current parameters, i.e., we calculate $w_j^{(i)}$ given $\boldsymbol{\theta}^{(t)}$ using Equ. (3):

$$w_j^{(i)} = p(z^{(i)} = j|x^{(i)};\boldsymbol{\theta}) = \frac{p(x^{(i)}|z^{(i)} = j;\boldsymbol{\mu},\Sigma) \cdot p(z^{(i)} = j;\boldsymbol{\Phi})}{\sum_{l=1}^{3} p(x^{(i)}|z^{(i)} = l;\boldsymbol{\mu},\Sigma) \cdot p(z^{(i)} = l;\boldsymbol{\Phi})} \tag{5}$$

b) M-step:

In M-step we use the just-computed values of $w_j^{(i)}$ to get a better estimate for the parameters, i.e., we find the parameters that maximize the $Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)})$:

$$\boldsymbol{\theta}^{(t+1)} = \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) \tag{6}$$

Specifically, we update the parameters as follows:

$$\phi_j = \frac{1}{m}\sum_{i=1}^{m} w_j^{(i)},\ j = 1,2,3. \tag{7}$$

$$\mu_{jk} = \frac{\sum_{i=1}^{m} w_j^{(i)} x_k^{(i)}}{\sum_{i=1}^{m} w_j^{(i)}},\ j = 1,2,3;\ 1 \le k \le 5. \tag{8}$$

$$\Sigma_j = \frac{\sum_{i=1}^{m} w_j^{(i)} (x_5^{(i)} - \mu_{j5})^2}{\sum_{i=1}^{m} w_j^{(i)}} \tag{9}$$

The iterations end when the parameters converge. Then we need to determine which z represents which case. Suppose we are processing samples of stage 1, where T_{d_4} is the smallest. Without loss of generality, suppose μ_{15} is the smallest, then $T_{d_4} = \mu_{15}$ and z=1 represents the total duration starting with S_4. Suppose $\mu_{23} > \mu_{33}$ (samples with $z = 2$ are more likely to have 1 left-turn plus 1 through acceleration), then $z = 2$ represents the total duration starting with S_1 or S_2, and $T_{d_1} = T_{d_2} = \mu_{25}$, $T_{d_3} = \mu_{35}$. Otherwise z=2 represents the total duration starting with S_3 (2 simultaneous left-turn accelerations), and $T_{d_3} = \mu_{25}$, $T_{d_1} = T_{d_2} = \mu_{35}$. To this point, all phase durations are obtained from results of EM algorithm.

3.5 Traffic signal phase and timing prediction

If the traffic signals use fixed-time control system, then the future phase sequence and timing can be easily predicted given the phase durations. However, for intelligent control systems that add or skip phases according to real-time traffic demand, prediction is much more challenging without the knowledge of future traffic flow. In our case, prediction is all about whether or not to insert left-turn phases in the future phase sequence. We let the probability of inserting each left-turn phase equal to the occurrence probability of that phase within an hour of history. For example, during rush hour, there are proportionally more vehicles turning left, thus almost every stage contains a left-turn phase, so there should also be a left-turn phase in the prediction.

3.5.1 *Translate movements into phase sequence.*

All we have so far is movement data as in Table 1, and the time durations of 8 phases in the two stages (for a particular intersection). We want to find the probability of each phase from near history, therefore the first step is to translate the historical movements into phase sequences. Note that valid samples are already labeled with latent variable z by EM algorithm, so the phase sequences in those samples are known. However, due to missing reports, there may be time gaps between consecutive valid samples, so we need to translate the movements between them into phase sequences.

The approach is to go through all possible phase sequences and find the one that best matches. The process of enumerating all possible sequences follows 2 basic rules according to the phase transition diagram in Figure 3: (1) stage 1 and 2 are occurring alternately; (2) each stage starting with S_i has total duration T_{d_i} $(1 \le i \le 8)$. The degree of matching is reflected by a penalty score P_s, and lower P_s means better.

The calculation of P_s of a candidate phase sequence π is as follows. For each movement (*Mov*) occurred within the timespan of π, the server first finds out the phase that *Mov* belongs to, i.e., the nearest phase S_i that allows the movement. Then if *Mov* is an

acceleration (*Acc*), the penalty is the time difference between the *Acc* and the start of S_i. If *Mov* is a direct pass (*Pas*), there is penalty only if *Pas* did not happen within the timespan of S_i, in which case the penalty is the time difference between *Pas* and the nearest time point in S_i. The penalty score P_s is the sum of all penalties of movements in π. After P_s is calculated for all candidate phase sequences, the one with minimum P_s is chosen as the translated historical phase sequence. The historical phase sequence in the past hour is used to calculate the probability of each left-turn phase, which equals to the number of its occurrences divided by the number of traffic signal cycles. It also equals to the probability of inserting that phase in the prediction.

3.5.2 Predicting future SPAT.

Predicting future phase sequence is a dead reckoning process based on the probability of left-turn phases. The server needs to predict the phase sequence starting from the timestamp of the latest acceleration movement to the future time of vehicle's estimated arrival. According to the phase transition diagram in Figure 3, the server concatenates phases stage by stage, with the probability of inserting a left-turn phase equal to its recent occurrence probability. The concatenated phases form the future phase sequence, which is returned to the smartphone upon request.

3.6 Optimal speed calculation

Based on the future phase sequence from the server, the smartphone chooses the most appropriate phase to cruise through according to the vehicle's route and location. The route is planned beforehand since the system needs to know the desired movement at every intersection. Route planning can be similar to [31], so it is not a focus of this paper.

Having chosen the desired future phase, it is suboptimal, however, to arrive at the exact start of that phase. Since the predicted phase start time can be earlier than real start time, if the vehicle arrives just a few seconds before the traffic light turns green, it will have to stop briefly, defeating the purpose of our system. Instead, the vehicle should arrive after the delay of a time buffer.

3.6.1 Introducing the time buffer t_{buf}.

Without foreknowledge of future traffic flow, it's impossible to predict SPAT with high accuracy. Thus, to reduce the impact of inaccurate time prediction, given the desired phase to cruise through, the smartphone adds some time buffer t_{buf} (padding) to the start and end of the green phase time window. If the phase duration is shorter than $2t_{buf}$, then t_{buf} is set to be half the phase duration. With t_{buf}, the probability that the vehicle can actually cruise through the intersection in green phase will increase. However, this brings us a tradeoff. On the one hand the time buffer can save some fuel by increasing the chance of cruising through in green phase, on the other hand it might cause prolonged travel time, because the vehicle will approach slower due to the delay of t_{buf}. Excessively large t_{buf} could even backfire because longer travel time also causes higher fuel consumption. Therefore, Section 4.3.3 derives the optimal value of t_{buf} based on realistic simulation.

3.6.2 Remaining travel time estimation.

Being aware of user-specified travel time requirements, the smartphone needs to estimate the remaining travel time to choose the most appropriate speed adaptation mode. If there is not much time left, the vehicle should hurry up instead of running in most fuel-saving mode. To estimate remaining travel time, we use historical average travel time of vehicles in fuel-saving mode on each road segment. This means the smartphone will also request and upload average travel time of each road segment.

Our system aims to arrive at the destination at the user-specified time on average. If the users demand more assurance to arrive on time, they can specify earlier arrival time requirements. The amount of time that needs to be set earlier depends on the uncertainty of the historical travel time statistics, which is elaborated in [6] and can be crowdsourced [29]. Let function *getRemTime()* return the estimated remaining travel time.

3.6.3 Speed adaptation modes.

While there can be complicated speed control algorithm, we propose a simple solution with binary speed modes. Suppose the estimated remaining travel time is t_{rem}, and the time budget is t_{bud}. Then we define the late factor as $r = t_{rem}/t_{bud}$. If r exceeds a threshold r_1, the system assumes the fastest mode (MODE_{fast}). If r is less than a threshold r_2, the system assumes fuel-saving mode (MODE_{save}). To prevent oscillation between the two modes, we empirically set $r_1 = 1.1$ and $r_2 = 1.0$. Let function *getMode*(t_{rem}, t_{bud}) return the desired mode.

3.6.4 Speed calculation algorithm.

The smartphone requests the server for future phase sequence π of the 1st intersection ahead every 10 seconds. Upon receiving π, the smartphone scans π to find the desired phase (and stage). If the first desired stage in the sequential scan will occur too soon for the vehicle to arrive (exceeding speed limit is not allowed), the smartphone finds the next appropriate one. Note that if the vehicle plans a left-turn but the appropriate stage does not contain a left-turn phase, it still has to choose that stage. Suppose the desired phase in the appropriate stage is S, we define the cruise-through time window $W = [t_s + t_{buf}, t_e - t_{buf}]$, where t_s and t_e are the start and end timestamps of S, and t_{buf} is time buffer.

The cruise-through time window W allows the vehicle to pass in a wide time interval. Thus we add another optimization to consider the 2nd intersection ahead. The logic is, for example, if the 2nd intersection will be in red phase for a long time, the vehicle should pass the 1st intersection at the end of W to avoid too much slow down when approaching the 2nd intersection. In other words, considering two intersections ahead enables a longer "vision". Denote the predicted phase sequence of the 2nd intersection as π^n, the desired phase as S^n, and the time window as W^n.

When the system is in MODE_{fast}, the optimal speed v_{opt} is set to be the speed limit v_{limit}. In MODE_{save}, the system uses W and W^n to calculate speed. Let functions $V_{min}(W)$ and $V_{max}(W)$ return the minimum and maximum speed to arrive within the time window of W. If $V_{min}(W) > v_{limit}$, then the smartphone should consider the next appropriate stage. Denote v_{now} as the current driving speed. The optimal speed is calculated according to Algorithm 1.

4 SIMULATION

4.1 Simulation of Urban MObility (SUMO)

We use SUMO for the major part of system evaluation. SUMO is an open source simulation package designed to simulate large-scale, realistic traffic flows in urban settings [3, 17]. It is capable

Algorithm 1: Optimal speed calculation

Input: π, π^n, t_{bud}, v_{now}
Output: v_{opt}

```
t_rem = getRemTime()
mode = getMode(t_rem, t_bud)
if mode=MODE_fast then
    return v_limit
else
    for W of S in π do
        if V_min(W) > v_limit then
            continue
        for W^n of S^n in π^n do
            if V_min(W^n) > v_limit then
                continue
            else if V_max(W^n) > V_max(W) then
                return min{V_max(W), v_limit}
            else if V_max(W^n) > V_min(W) then
                if V_max(W^n) > v_now then
                    return min{V_max(W^n), v_limit}
            else
                return V_min(W)
```

of generating highly customized road networks, vehicle trips, and traffic signal schedules. SUMO is microscopic, meaning that each vehicle has its own editable physical property and accessible travel information such as route, speed, location, fuel consumption, etc.

We use OpenStreetMap [9] data to generate road network since it is realistic and large-scale. We deploy the intelligent traffic control system at 157 four-armed intersections. We set up 2 lanes per road segment, one is left-turn lane and the other for both through and right-turn movements. Every traffic signal follows the same phase transition diagram as shown in Figure 3. Note that the actual phase sequence depends on actual traffic demand, so different traffic signals will not be synchronized due to randomized traffic flows.

We use SUMO API *randomTrips()* to generate large scale randomized background traffic flows. Trip repetition rate controls how often a new vehicle starts a trip, thus controlling vehicle density. Adjusting the repetition rate ratio between test vehicles and other vehicles can be used to simulate different system penetration rates.

Figure 4: SUMO GUI.

4.2 Simulation parameters

To align our simulation parameters with real-world situations, we show the measurement results of preliminary experiments and explain basic parameters as following.

(1) Vehicle arrival rate r_v: r_v is the number of vehicles arriving at a junction per second per incoming arm. To get the range of r_v, we resort to the traffic cameras at District Department of Transportation [1], randomly choose 10 intersections and count the rate of vehicle arrivals. The r_v ranges from 0.039 to 0.15, with an average of 0.092. Thus in our simulation we choose r_v to be in this range. The basic three values of r_v are 0.04, 0.069 and 0.094.

(2) Movement false positives and false negatives: False positive rate r_{fp} is defined as the percentage of reported acceleration movement that did not actually happen. False negative rate r_{fn} is the percentage of missing acceleration reports. To choose realistic r_{fp} and r_{fn}, we first conducted an experiment to detect acceleration movement at an intersection. Out of 104 acceleration movements, there are 5 false negatives (4.8%) and 0 false positive. For simulation we investigate SPAT learning accuracy under $r_{fp}=r_{fn}=5\%, 10\%, 15\%$ by generating false or missing movement reports deliberately. In simulations apart from SPAT learning, we set $r_{fp}=r_{fn}=5\%$.

(3) Vehicle types: For simulation we have 3 vehicle types: random background vehicles (not using GreenDrive), test vehicles (using GreenDrive) and comparison vehicles (not using GreenDrive).

(4) System penetration rate r_p: This parameter is controlled by adjusting the ratio of the departure repetition rate between vehicles using GreenDrive and not using GreenDrive. The three r_p are 30%, 50%, and 70%.

(5) Traffic signal phase durations: We let all traffic signals have the same set of phase duration parameters: $T_{p_1} = T_{p_2} = 13s$, $T_{p_3} = 7s$, $T_{p_4} = 25s$, $T_{p_5} = T_{p_6} = 12s$, $T_{p_7} = 8s$, $T_{p_8} = 30s$. Note that although they share the same parameters, their phase transition will not be synchronized due to randomized traffic flows.

4.3 Simulation results

4.3.1 *Phase duration learning accuracy.*

The simulation generates random traffic flows and collects vehicle movement data over the period of 4 hours, Apart from the three basic r_v configurations, we add another higher rate r_v=0.14 with which congestion occurred at around 1/5 of all intersections. We evaluate the learned $T_{d_1}, T_{d_3}, T_{d_4}$ for stage 1 and $T_{d_5}, T_{d_7}, T_{d_8}$ for stage 2. The average (or maximum) phase duration error is the mean (or maximum) error of the 3 total durations of each stage.

In Figure 5 we plot the statistical distribution of phase duration error of 157 intersections. The median of maximum and average phase duration error are 4.3s and 1.8s, respectively, in worst case. CityDrive [33] reports 1.24s mean phase length error, that is because it targets fixed-control signal system. For the first three r_v, higher r_v and r_p lead to smaller error. But for the highest r_v (0.14), the errors become larger because traffic congestion is observed via SUMO GUI (right side of Figure 4), and some vehicles are unable to move at intersections. In the rest of our simulations, we no longer use r_v=0.14.

To investigate the impact of lower sensor quality on learning accuracy, we run simulation with $r_{fp} = r_{fn} = 0.05, 0.1, 0.15$ and $r_p = 30\%$. The phase duration error distribution is shown in Figure 6. The median error only increases slightly under larger r_{fp} and r_{fn}.

4.3.2 *Signal Phase and Timing (SPAT) prediction accuracy.*

Vehicles using GreenDrive request future SPAT every 10s regardless of how far away they are from the next intersection. When each vehicle chooses a target phase and decides the arrival time, the ground truth phase which covers the arrival time is logged. Ideally

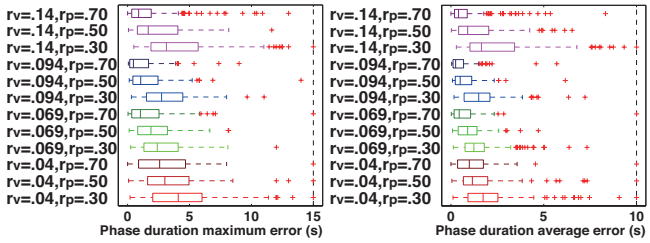

Figure 5: Phase duration error distribution in stage 1 (figure for stage 2 is similar). For each box, central mark is the median, edges are the 25th and 75th percentiles, whiskers covers 99.3% data points, and outliers are red crosses limited by the vertical dashed line.

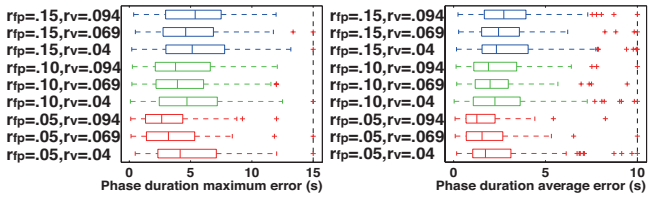

Figure 6: Phase duration error distribution under different r_{fp} (=r_{fn}).

the phase that occurs at the planned arrival time should be the same as the chosen predicted phase. But to judge the correctness of the predicted phase and timing, a simple right or wrong is not enough.

Here we distinguish four cases. Case 1: The predicted phase number is exactly the same as the ground truth. Case 2: The predicted phase number is wrong, but there happens to be a phase in that stage that allows the vehicle to cruise through. Case 3: The predicted phase number is wrong and the vehicle does not have an appropriate phase to pass in that stage, but the stage number is correct. Case 4: The predicted phase number is wrong and the predicted stage number is also wrong.

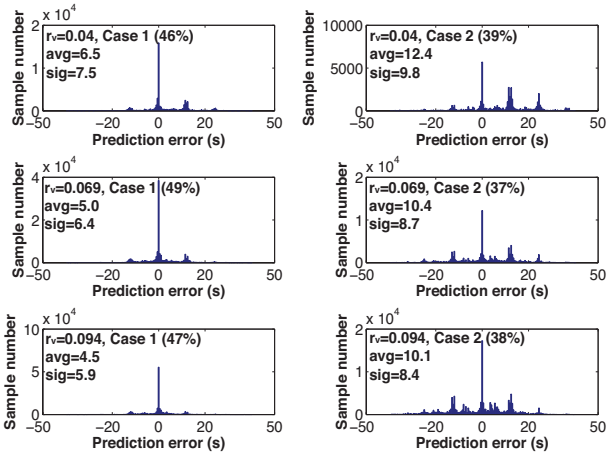

Figure 7: Prediction timing error distribution. avg and sig are average and standard deviation of prediction error.

Figure 7 shows the prediction timing error distribution histogram of the first two cases. The bucket size of the histogram is 0.5s. Case 1 means the phase number is correct, and the vehicle can cruise through in green phase. The prediction error here is the phase start time error compared to ground truth. It is likely to make wrong predictions about whether or not there is a left-turn phase before the target phase or in the previous stage, so the prediction error has small peaks around $\pm T_{p_1}$ or $\pm T_{p_5}$ ($\approx \pm 12s$). Case 2 means the predicted phase number is not correct, but the phase at the arrival time allows the vehicle's desired movement. In this case, the prediction error is the difference between the start time of predicted stage and the true start time of the stage. The stage start time is also likely to differ by one or two left-turn phase durations, so there are peaks around $\pm 12s$ or $\pm 24s$. Case 3 means that the predicted phase is not correct, and there is no phase in that stage that allows the desired movement. The stage is correct but whether the vehicle has to stop or not totally depends on the driver's behavior when approaching the intersection. For example, a vehicle plans a left-turn while the ground truth stage contains only a through phase. Thus, the chance of stop and wait is high and prediction time error has no meaning, so we don't show case 3 in Figure 7. Case 3 accounts for 13.2%, 12.1% and 13.1% under $r_v = 0.04, 0.069, 0.094$, respectively. Case 4 means the prediction of the phase and stage is wrong, and traffic flows of the conflicting stage are in motion. This means the vehicle has to stop and wait. Case 4 accounts for 1.72%, 1.93% and 1.86% under the three r_v values.

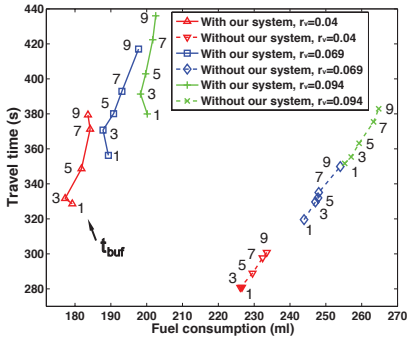

Figure 8: Fuel consumption and travel time for different t_{buf}.

4.3.3 *Fuel saving with optimal time buffer (t_{buf}).*

We show the fuel-saving results with the optimal value for t_{buf} when our system is in most fuel-saving mode (without travel time requirement). We set up 5 test cases with $t_{buf} = 1, 3, 5, 7, 9s$ and $r_p = 30\%$. In each case we create 100 vehicles using GreenDrive and another 100 without GreenDrive, and have them run in pairs on 10 routes with approximately the same distance. The departure time of consecutive pair is separated by 50s.

Figure 8 shows the results for each t_{buf} (t_{buf} is next to the markers). With respect to fuel saving, we observe that $t_{buf} = 3s$ is the best, achieving fuel saving of 21.6%, 23.9%, 22.8% under $r_v = 0.04, 0.069, 0.094$ respectively, while only increasing travel time by 18.1%, 12.4%, 10.1%, respectively. Our system is better than [13] which reports a 7% fuel saving under 50% system penetration using SUMO, and also better than [5] which achieves 13.7% saving using SUMO. The fact that travel time increases is due to our strategy to cope with the fundamental impossibility to accurately predict future signal schedule which depends on unknown future traffic. With the unpredictability of dynamic signal control, the time buffer increases the chance to pass in green phase but causes more delay compared to vehicles that always run the fastest, because vehicles

that always run the fastest would not miss even the last second of a green phase.

In Figure 8, we see that with the smallest $t_{buf} = 1s$, the vehicles have the least travel time, but the fuel consumption is higher since vehicles are more likely to arrive earlier than the phase transition and come to a brief stop. However, $t_{buf} \geq 5s$ can lead to overly extended travel time, thus more fuel consumption. Another reason why large t_{buf} does not help is that, our system needs some accelerations to infer future SPAT, so it is impossible that all vehicles don't encounter red lights. Because if so, no acceleration would be reported. Therefore, our system needs a dynamic equilibrium where some vehicles come to a stop due to prediction error, and then report accelerations to help the server improve prediction accuracy. Thus, no matter how large t_{buf} is, there will always be some random vehicles coming to a stop. One more point worth mentioning is that the travel time and fuel consumption of vehicles without our system also increases as t_{buf} goes up. This is because vehicles using GreenDrive could interfere with those without GreenDrive, such as blocking the lane of other vehicles.

These global effects and interactions are only observable in our large-scale realistic simulations based on SUMO, and are not reported by other related work because their simulations (e.g. [33] uses MatLab) are simplistic, small-scale, and do not model vehicle mobility well.

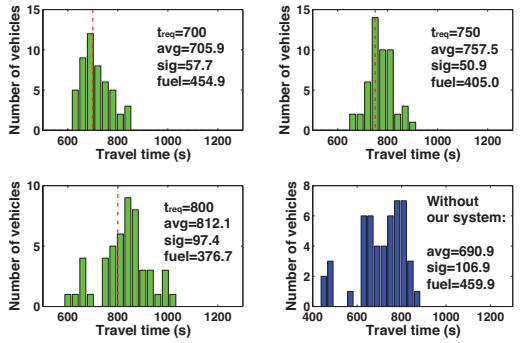

Figure 9: Travel time distribution under different travel time requirements. avg **and** sig **are average and standard deviation of recorded travel time.**

4.3.4 System performance under travel time requirements.

We investigate travel time distribution and fuel-saving under different travel time requirements t_{req}. The simulation parameters are $r_v = 0.04$ and $r_p = 30\%$. We choose a random route with estimated travel time 800s, and run 4 test cases, three with t_{req}= 700s, 750s, 800s, and another one without GreenDrive. For each test case, we create 50 target vehicles with departure time separated by 50s. Recall that, GreenDrive aims to arrive at required time on average; and different speed modes will be chosen based on time budget.

Figure 9 shows that stricter travel time requirements lead to less fuel-saving. Compared with vehicles without our system, we are able to save 15.9% and 9.6% fuel consumption under t_{req}= 800s, 750s, respectively. The reason that this test achieves less fuel saving than in Section 4.3.3 is that the system would occasionally switch to MODE$_{fast}$ to meet travel time requirements. When t_{req}=700s, our system is mostly in MODE$_{fast}$ mode, thus the fuel consumption is approximately the same as without our system.

5 REAL-WORLD EXPERIMENTS

Limited resources make it difficult to conduct large-scale experiments with many real vehicles. So we choose to report observed real vehicle movements using smartphones and have a real vehicle running under the speed advice of our system.

5.1 Experiment setup

We target four real major intersections and let one person stand at each intersection to monitor and record observed vehicle movements using a smartphone. Then the smartphone will manipulate recorded movements to simulate false positives/negatives, system penetration rate, etc. The detail of Android program workflow is the following. When a movement is observed, the button that represents its type is clicked, and it also records the timestamp. Then the incoming and outgoing arms represented by buttons are clicked sequentially. This operation is then followed by clicking the number of vehicles n_v observed in each movement. To simulate system penetration rate, the number of vehicles using our system is randomly generated in the integer interval $[0, 2n_v r_p]$. Then for each vehicle a random float number between $[0, 1.0]$ is generated and compared with r_{fp} and r_{fn} to determine if a false positive or false negative is to be generated. To generate a false negative, the program simply ignores the movement. To generate a false positive, the program randomly subtracts the timestamp of this movement to make it appear in the wrong stage.

In all our real-world experiments, the observed r_v is about 0.05. We choose $r_p = 30\%$, and $r_{fn} = r_{fp} = 5\%$ as parameters in the Android program. The test vehicle is Nissan Altima. We plug in the ELM 327 interface to read the vehicle's real-time on-board diagnostics (OBD) information and send it to the smartphone via Bluetooth. The OBD sample rate is 3Hz, and it gives accurate readings of fuel consumption, speed, engine RPM, etc.

The test trip is repeated traveling on a straight road consisting of the four intersections with intelligent traffic control system. Each direction of the road has two lanes. The driver makes round trips along the same route and tries to stay at suggested speed. The travel distance of each round trip is about 2.2 kilometers and the travel time is about 5 minutes.

5.2 Experiment results

5.2.1 Traffic signal schedule learning.

As the first stage of our system, we use smartphones to record vehicle movements of the four intersections, each for four hours. The collected data are fed to the EM algorithm to learn the signal schedule of each intersection. To get the ground truth of the phase durations, we use a stopwatch to measure the average duration multiple times. The results are in Table 2, where the values in parentheses are ground truth. It is shown that the average phase duration error is 1.78s, which is slightly better than the 1.85s mean phase length error claimed in SignalGuru [16] using direct image processing.

5.2.2 System performance under travel time requirements.

The server makes predictions using duration results in Table 2 and real-time vehicle movements sent from smartphones. The average driving speed used for estimating remaining time is 7.2 m/s. We record fuel consumption and travel time in the following 3 test cases. Test case 1 is the baseline, in which the vehicle makes

Table 2: Phase duration learning result of 4 intersections.

	Int. 1	Int. 2	Int. 3	Int. 4
T_{d_1}	55.8 (54.9)	45.9 (45.0)	40.7 (45.3)	33.1 (35.0)
T_{d_3}	58.5 (62.0)	39.6 (42.1)	37.3 (35.0)	32.8 (35.0)
T_{d_4}	40.4 (41.1)	28.6 (28.0)	33.5 (31.5)	30.5 (30.0)
T_{d_5}	49.8 (52.2)	69.4 (70.3)	70.5 (72.8)	70.1 (72.0)
T_{d_7}	49.7 (49.0)	74.8 (75.7)	61.3 (62.4)	69.9 (72.9)
T_{d_8}	35.7 (34.5)	59.4 (55.9)	60.6 (61.0)	64.4 (62.0)

10 round trips as fast as possible without our system. It's the same as driving constantly in MODE$_{fast}$. This mode saves most time but fuel consumption is high. The average travel time is 249.6s. Test case 2 is 10 round trips using our system with required travel time t_{req}=270s. During the travel the in-vehicle smartphone calculates the available time budget and adaptively switches between the two speed modes. Test case 3 is 10 round trips with t_{req}=400s, which allows the vehicle to be in MODE$_{save}$ most of the time. In this case our system saves the most fuel but causes 55s more travel time.

The tradeoff between saving fuel and travel time is shown in Table 3. The case without travel time requirement and the case with t_{req}=270s save 31.2% and 23.0% fuel consumption, respectively, which is better than the 20.3% fuel-saving in SignalGuru [16]. SignalGuru uses ad-hoc networks and requires relay nodes at intersections, thus vehicles far from the intersection may not receive speed advice. Our centralized approach enables speed advisory service all the way to the next intersection, which yields even more fuel saving. CityDrive [33] claims 58.8% energy saving, but it is calculated using GPS speed information instead of real fuel consumption readings.

Table 3: Tradeoff between saving fuel and travel time.

	Avg. time (s)	Req. time (s)	Avg. fuel (g)	Fuel saved
Case 1	249.6	/	158.2	Baseline
Case 2	261.2	270	121.6	23.0%
Case 3	304.4	400	108.9	31.2%

6 CONCLUSION

This paper presents GreenDrive, a smartphone-based intelligent speed adaptation system that helps reduce fuel consumption and meet travel time requirements. Both realistic and large-scale SUMO simulations and small-scale real-world experiments show that our system is able to effectively learn traffic signal schedule, and offer real-time optimal speed advice to drivers according to travel time requirements. In the future we plan to improve the SPAT learning algorithm to tolerate even lower system penetration rate.

ACKNOWLEDGMENTS

This work was supported in part by NSF grants CNS 16-18627, CNS 13-20209, CNS 13-29886 and CNS 13-45266.

REFERENCES

[1] 2017. District Dept. of Transportation. (2017). http://app.ddot.dc.gov.
[2] Behrang Asadi and Ardalan Vahidi. 2011. Predictive Cruise Control: Utilizing Upcoming Traffic Signal Information for Improving Fuel Economy and Reducing Trip Time. In *IEEE CST*.
[3] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz. 2011. Sumo-simulation of urban mobility-an overview. In *IEEE SIMUL*.
[4] Lu Shen Cai and Guo Bao Ning. 2013. Adaptive Driving Speed Guiding to Avoid Red Traffic Lights. In *IEEE ICCSEE*.
[5] David Eckhoff, Bastian Halmos, and Reinhard German. 2013. Potentials and limitations of green light optimal speed advisory systems. In *2013 IEEE Vehicular Networking Conference*. IEEE, 103–110.
[6] Raman Ganti, Mudhakar Srivatsa, and Tarek Abdelzaher. 2014. On limits of travel time predictions: Insights from a new york city case study. In *IEEE ICDCS*.
[7] Jianwei Gong, Yanhua Jiang, Guangming Xiong, Chaohua Guan, Gang Tao, and Huiyan Chen. 2010. The recognition and tracking of traffic lights based on color segmentation and camshift for intelligent vehicles. In *IEEE IV*.
[8] Victor Gradinescu, Cristian Gorgorin, Raluca Diaconescu, Valentin Cristea, and Liviu Iftode. 2007. Adaptive Traffic Lights Using Car-to-Car Communication. In *IEEE VTC*.
[9] Mordechai Haklay and Patrick Weber. 2008. Openstreetmap: User-generated street maps. *Pervasive Computing, IEEE* 7, 4 (2008), 12–18.
[10] K Larry Head, Pitu B Mirchandani, and Dennis Sheppard. 1992. Hierarchical framework for real-time traffic control. In *Transportation Research Board (TRB)*.
[11] Tsin Hing Heung, Tin Kin Ho, and Yu Fai Fung. 2005. Coordinated road-junction traffic control by dynamic programming. In *IEEE ITSC*.
[12] N Hounsell, J Landles, R Bretherton, and K Gardner. 1998. Intelligent systems for priority at traffic signals in London: The INCOME project. In *IEEE RTIC*.
[13] Konstantinos Katsaros, Ralf Kernchen, Mehrdad Dianati, and David Rieck. 2011. Performance study of a Green Light Optimized Speed Advisory (GLOSA) application using an integrated cooperative ITS simulation platform. In *2011 7th International Wireless Communications and Mobile Computing Conference*. IEEE, 918–923.
[14] Markus Kerper, Christian Wewetzer, Andreas Sasse, and Martin Mauve. 2012. Learning Traffic Light Phase Schedules from Velocity Profiles in the Cloud. In *IEEE NTMS*.
[15] Lawrence A Klein, Milton K Mills, and David RP Gibson. 2006. Traffic Detector Handbook: -Volume II. In *No. FHWA-HRT-06-139*.
[16] E. Koukoumidis, L.-S. Peh, and M. R. Martonosi. 2011. SignalGuru: leveraging mobile phones for collaborative traffic signal schedule advisory. In *ACM MobiSys*.
[17] Aron Laszka, Bradley Potteiger, Yevgeniy Vorobeychik, Saurabh Amin, and Xenofon Koutsoukos. 2016. Vulnerability of Transportation Networks to Traffic-Signal Tampering. In *2016 ACM/IEEE 7th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 1–10.
[18] G. Mahler and A. Vahidi. 2012. Reducing idling at red lights based on probabilistic prediction of traffic signal timings. In *IEEE ACC*.
[19] Masako Omachi and Shinichiro Omachi. 2009. Traffic light detection with color and edge information. In *IEEE ICCSIT*.
[20] Kartik Pandit, Dipak Ghosal, H Michael Zhang, and Chen-Nee Chuah. 2013. Adaptive Traffic Signal Control With Vehicular Ad hoc Networks. In *IEEE TVT*.
[21] V. Paruchuri, S. Chellappan, and R. B. Lenin. 2013. Arrival time based traffic signal optimization for intelligent transportation systems. In *IEEE AINA*.
[22] Christian Priemer and Bernhard Friedrich. 2009. A Decentralized Adaptive Traffic Signal Control Using V2I Communication Data. In *IEEE ITSC*.
[23] Akcelik R., Besley M., and E. Chung. 1998. An evaluation of SCATS master isolated control. In *ARRB Transport Research Conference*.
[24] P. Schuricht, O. Michler, and B. Bäker. 2011. Efficiency-Increasing Driver Assistance at Signalized Intersections using Predictive Traffic State Estimation. In *IEEE ITSC*.
[25] Nirav Shah, Farokh B Bastani, I Yen, and others. 2006. A Real-Time Scheduling Based Framework for Traffic Coordination Systems. In *IEEE SUTC*.
[26] T. A. Suramardhana and H. Y. Jeong. 2014. A driver-centric green light optimal speed advisory (DC-GLOSA) for improving road traffic congestion at urban intersections. In *IEEE APWiMob*.
[27] Robin M Waterman, Mark F Makuch, PE LeVance H James, and Michael J Cloutier. 2014. Traffic Control Signal Design Manual.
[28] Haitao Xia, Kanok Boriboonsomsin, and Matthew Barth. 2013. Dynamic eco-driving for signalized arterial corridors and its indirect network-wide energy/emissions benefits. 17 (2013), 31–41.
[29] Shuochao Yao, Md Tanvir Amin, Lu Su, Shaohan Hu, Shen Li, Shiguang Wang, Yiran Zhao, Tarek Abdelzaher, Lance Kaplan, Charu Aggarwal, and others. 2016. Recursive ground truth estimator for social data streams. In *Information Processing in Sensor Networks (IPSN), 2016 15th ACM/IEEE International Conference on*. IEEE, 1–12.
[30] Shuochao Yao, Shaohan Hu, Yiran Zhao, Aston Zhang, and Tarek Abdelzaher. 2017. DeepSense: A Unified Deep Learning Framework for Time-Series Mobile Sensing Data Processing. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee.
[31] Dejing Zhang, Tian He, Shunjiang Lin, Sirajum Munir, and John Stankovic. 2014. pCruise: Online Cruising Mile Reduction for Large-Scale Taxicab Networks. *IEEE TPDS* (2014).
[32] Yiran Zhao, Shen Li, Shaohan Hu, Hongwei Wang, Shuochao Yao, Huajie Shao, and Tarek Abdelzaher. 2016. An experimental evaluation of datacenter workloads on low-power embedded micro servers. *Proceedings of the VLDB Endowment* 9, 9 (2016), 696–707.
[33] Yiran Zhao, Yang Zhang, Tuo Yu, Tianyuan Liu, Xinbing Wang, Xiaohua Tian, and Xue Liu. 2014. CityDrive: A Map-Generating and Speed-Optimizing Driving System. In *Proc. INFOCOM*.