IMPROVING SMART TRANSPORTATION APPLICATIONS WITH VEHICULAR EDGE COMPUTING

by

Bozhao Qi

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Electrical and Computer Engineering)

at the

UNIVERSITY OF WISCONSIN–MADISON

2020

Date of final oral examination: 08/13/20

The dissertation is approved by the following members of the Final Oral Committee:

Suman Banerjee, Professor, Computer Sciences, UW-Madison
Soyoung (Sue) Ahn, Professor, Civil and Environmental Engineering, UW-Madison
Yu Hen Hu, Professor, Electrical and Computer Engineering, UW-Madison
Younghyun Kim, Assistant Professor, Electrical and Computer Engineering, UW-Madison

To my families for their love, encouragement and support.

Acknowledgments

It is a wonderful journey at UW-Madison, which left me many cherished memories and valuable experiences about life and research. I feel so lucky to have the chance to study and conduct research at a renowned university with outstanding people.

First of all, I would like to thank my advisor, Suman Banerjee. He not only has done an amazing job in guiding my research works but also helped me with many personal matters. He helps me transform from a college graduate student to a Ph.D. with solid skills and creative ideas. He has always been there, with innovative and exciting research directions, allowing me to explore different ideas and learning from mistakes. I deeply appreciate his support, patience, tolerance, and trust.

I would also thank Professor Soyoung (Sue) Ahn, Professor Yu Hen Hu, and Professor Younghyun Kim for their willingness to be on my committee. Professor Soyoung (Sue) Ahn and Professor Xinyu Zhang served on my preliminary examination committee, I appreciate their time and efforts. Their valuable suggestions and comments help me shape this dissertation.

I am fortunate to work with many talented colleagues during my Ph.D. Lei Kang guided me in my early stage of Ph.D., he helped me to build the basic skills to do research and write papers. I also thank Peng Liu for his valuable advice in my research, career development and life. I am grateful for the help and support from my fellow graduate students, including Parikshit Sharma, Joshua Tabor, Wei Zhao, Chuhan Gao, Yilong Li, Yijing Zeng, Lance Hartung, Derek Meyer, Jayaram Raghuram, Shenghong Dai, Tao Ji, Haoran Qiu, Yanfang Le, Ming Gao, Keqiang He, Xuan Zhang and many others.

Apart from my colleagues, I have spent most of my spare time with my friends on the volleyball court. I especially thank Qi Ding, who brought me to the volleyball court and introduced me to my wife. Without your help, I would not have such a wonderful journey at Madison. Yue Qiu, Xing Wang, Huilong Zhang, Kaiyue Zheng, Bin Guo, Yixuan Feng, Weiwei Hu, Dongrui Zhao, Haiyun Jin, Xiaoping Bao, Jianqiao Zhu, Tianyi Jin, Yuzhou Zhao, Tingting Weng, Xiao Dong, Bocheng Lin, Shuo Li, Yuzhou Zhao, Muchuang Wang, Minghui Lou, Yixin Chen, Fu Tan, Peiru Yu, Qilin Hong, Jenny Huang, Zhenwei Ye, Baiyan Gong, Qiyu Zhou, Keyu Zeng, Ruisu Zhang, Ruihan Tong, Elaine Zheng, Minju Park, Jasmine Wang, Ping-Ni Wang, Cheng-Hsien Lee, Yi-Jiun Liao, Chi-Shian Dai, Shenghong Dai, Tao Ji, it is so nice to have you guys and thanks for all the joy and sorrow we experienced together.

Lastly but most importantly, I would like to express my sincere gratitude to my family - my parents Liping Jin and Yi Qi, my parents in law Hong Sun and Qian Cai, my grandparents Yulian Liu and Shuxiang Qi and my wife Bingqing Cai. Without their continuous support, constant love and encouragement, none of this would have been possible without the love of my family. This dissertation is dedicated to every one of them!

Contents

Contents iv

List of Tables vii

List of Figures viii

Abstract xi

1 Introduction 1
- *1.1 Overview* 1
- *1.2 Edge Enhanced Vehicular Sensing Framework and Its Application to Vehicle Tracking* 5
- *1.3 Augmented Driving Behavior Analytics* 6
- *1.4 Transit and Human Mobility Analytics* 8
- *1.5 Contributions* 9
- *1.6 Outline* 11

2 A Crowdsourced Sensing and Computing Framework Leveraging the Roaming Edge 12
- *2.1 Introduction* 12
- *2.2 Motivation* 17
- *2.3 System Design* 19
- *2.4 Vehicle Tracking Application* 27

2.5 Evaluation 36
2.6 Discussion 46
2.7 Conclusion 47

3 DrivAid: Augmenting Driving Analytics with Multi-Modal Information 48
3.1 Introduction 48
3.2 System Overview 51
3.3 Event Detection and Context Analysis 52
3.4 Driving Activity Evaluation 57
3.5 System Implementation 59
3.6 Evaluation 60
3.7 Discussion 68
3.8 Conclusion 69

4 A Vehicle-based Edge Computing Platform for Transit and Human Mobility Analytics 70
4.1 Introduction 70
4.2 Trellis System Design and Implementation 75
4.3 Our Approach to Track Individual 79
4.4 Passenger Activity Trends 87
4.5 Pedestrian Activity Trends 92
4.6 Impacts of External Factors 95
4.7 Discussion 99
4.8 Conclusion 102

5 Related Work 104
5.1 Edge Enhanced Crowdsensing Framework 104
5.2 Vehicle Tracking 105
5.3 Driving Behavior Analytics 106
5.4 Transit and Human Mobility Analytics 108

6 Summary and Future Work 110
6.1 Summary and Discussion 110
6.2 Future Work 111

Bibliography 117

List of Tables

2.1 Object Detection Results. 34
2.2 System Processing Delay Breakdown 45

3.1 Object Detection 56
3.2 The Accuracy of Driving Event Detection. 63
3.3 The Accuracy of Object Detection. 64
3.4 The Memory Usages of Pipelines. 67
3.5 The Accuracy of Lane Change Event. 68

4.1 Route Statistics 78
4.2 Collected Data Statistics 79

List of Figures

1.1 An edge-enhanced vehicular sensing framework and its applications . 3

2.1 An example application build upon Cascade. 15
2.2 Service management module overview. 21
2.3 Frame quality and size under different settings. 24
2.4 Vehicle tracking application overview. 28
2.5 Vehicle re-identification. 30
2.6 Vehicle re-identification and License Plate Recognition. 31
2.7 Vehicle information extraction. 33
2.8 Performance under different settings. 36
2.9 Performance under different frame qualities. 38
2.10 Vehicle tracking application overview. 39
2.11 Performance under different similarities. 40
2.12 Network parameter estimation accuracy. 41
2.13 Streaming performance evaluation. 43
2.14 Context-aware streaming evaluation. 44

3.1 Possible causes of a hard brake. 49
3.2 A high level overview of DrivAid. 50
3.3 The detection of driving activities. 51
3.4 Extracting context information using vision-based techniques. 54
3.5 The structure of decision tree evaluation model. 59

3.6 The hardware components of DrivAid. 62
3.7 System Usages. 65
3.8 Detailed analysis of a hard brake. 69

4.1 The on-board edge computing platform. 71
4.2 Different RSSI patterns between passenger and pedestrian. 74
4.3 Bus routes with labeled bus stops. 77
4.4 Distribution of devices by vendors in log scale. 78
4.5 Trellis architecture. 80
4.6 Illustration of two schemes and how to keep track of each passenger. . 81
4.7 An example of how to determine Type 1 pattern from RSSI slopes and vehicle stop information. 83
4.8 Four possible types of inference from RSSI and speed data patterns for detected passenger. 84
4.9 The CDF of station to station travel time (left) and distance (right). . . 85
4.10 The CDF of mobile device Wi-Fi signals' RSSI readings (left) and transmission rate (right). 86
4.11 Onboard passenger number ground truth and automatic passenger counting results. 88
4.12 The CDFs of passenger number estimation error with different schemes. 89
4.13 Riding patterns of different bus stops in the residential area (top) and the main campus (bottom). 91
4.14 Original-Destination matrices during morning hours (left), and evening hours(right). 92
4.15 The CDF of pedestrian number estimation error. 94
4.16 A comparison of the daily average pedestrian number on the street during daytime (10am-3pm) and night (9pm-11pm) hours. 95
4.17 The impacts of temperature on daily average pedestrian (left) and passenger (right) number. 96
4.18 Quantify temperature and weather impacts on human activities in region 7. 97

4.19 The comparison of on-time performance between different hours(left) and weather conditions(right). 98

Abstract

With the recent development of pervasive sensing, lightweight, connected devices are revolutionizing our lives and bring us to the era of the Internet-of-Things (IoT). Specifically, connected sensors, e.g., cameras, LiDARs, motion sensors, installed in a moving platform (e.g., mobile vehicle) can provide broad views of wide-area environments quickly and efficiently. If many vehicles incorporate such sensing systems, they together can be composed into a unique crowd-sourced platform, they can gather fine-grained, diverse, and noisy information at city-scales. However, multiple sensors can generate large amounts of data and it is hard to aggregate in a centralized cloud-hosted location. We need some form of interim processing platform to digest such data and avoid letting these devices constantly ping the cloud. Compared to cloud computing platforms, edge computing platforms can provide unique edge services with lower latency, greater responsiveness and more efficient use of network bandwidth. To push computational and storage capabilities even closer to end sensing devices, we consider a new edge computing paradigm called the roaming edge, in which the edge nodes themselves are in motion, e.g., mounted in vehicles. The roaming edge extends the notion of edge computing to provide local computing capabilities, without requiring offload computing tasks to the cloud. These characteristics make roaming edge computing platforms suitable for providing the first line of analytics support for vehicular sensing applications.

By loading computation tasks from cloud computing platforms to roaming edge computing platforms and even connected devices, we can achieve greater efficiency while accomplishing our desired goals. In order to understand how to leverage

the roaming edge in developing vehicular sensing applications, the following questions need to be answered: (i) How should we leverage the advantages of edge computing to accelerate and improve vehicular sensing applications? (ii) What are good ways to manage data access and to control what data each stakeholder in this ecosystem can access? (iii) What are good resource allocation strategies to achieve a predefined goal, and what are the trade-offs between resource allocation and system performance? Motivated by these questions, our research focuses on exploring how to design and implement vehicular sensing applications that can benefit from the roaming edge. Application in many domains can benefit from edge computing paradigms, we focus on applications of intelligent transportation systems in this dissertation. We start by designing and implementing a vehicle tracking application to explore challenges of building applications leveraging the roaming edge. Moreover, we explore the limitations of existing driving behavior evaluation solutions and proposed a multi-modal and lightweight real-time sensing system that leverages audio-visual cues to augment driving behavior analysis. Finally, we develop a low cost Wi-Fi-based in-vehicle monitoring and tracking system for transit and human mobility analytics. In each of these applications, the complex task or problem can be break into simpler sub-problems such that static edge compute nodes, roaming edge nodes and even connected in-range sensors are able to contribute to the end goal.

Our work on smart transportation applications demonstrates the advantages of roaming edge computing platforms for vehicular sensing applications requiring intensive computation, low latency and high privacy. Furthermore, we discuss how such an innovative structure can be applied to other applications that can improve the quality of life. Ultimately, we summarize what are the common characteristics for applications running at the roaming edge and how roaming edge computing can help improve vehicular sensing applications.

1 Introduction

1.1 Overview

Connected devices and sensors are becoming increasingly popular: an estimated 50 billion connected devices will be sold by 2020 [1]. With the advancement of pervasive sensing technologies, fruitful on-device sensors enable a wide range of applications in transportation systems to improve transportation safety, mobility and provide useful transportation analytics. Most city planners try to gather traffic flow information using a set of static connected sensors (e.g., roadside cameras, and road sensors) at the city-scale. For example, many intelligent parking services have been built using static sensors that are installed under or above individual parking spots to provide parking spot availability information. Public safety agencies leverage static security cameras across a city to search for and track people or objects of interest. However, installing these static sensors across a large area incurs massive deployment and maintenance efforts, and the coverage of these sensors is necessarily limited. We envision that these connected sensors can be installed on a moving platform to address the drawbacks of static sensors. More concretely, we focus on vehicles — autonomous or otherwise — that can be equipped with a plethora of sensors such as camera, LiDAR (light detection and ranging), RADAR and more. These in-vehicle sensors can provide context information of its surrounding and a

large set of distributed sensors mounted on vehicles can provide new and unique forms of information not available before.

Some of the sensors (e.g., camera, LiDAR) generate large amounts of data which need to be processed in a timely manner to support applications with strict latency requirements. Unfortunately, such data cannot be easily offloaded from vehicles to the cloud with current network architectures. To overcome the shortage of current technologies, a growing number of research studies have been done in the area of edge computing [2, 3, 4, 5]. Edge computing services distribute computing and storage resources closer to the connected devices. To bring computational and storage capabilities even closer to where the data is being generated, we consider a new edge computing paradigm called the roaming edge, in which the edge nodes themselves are in motion, e.g., mounted in vehicles. With the help of roaming edge nodes, various forms of data can be processed locally, without requiring offload. These characteristics make roaming edge computing platforms suitable for performing aggregate analysis without incurring transfer overheads. Further, once a reasonable fraction of vehicles are equipped with these sensing platforms, they can naturally become a crowd-sourced platform to gather, analyze, and organize various forms of data at the city-scale and beyond.

In this dissertation, we try to understand the advantages of roaming edge computing and how to leverage these advantages for a vehicular sensing framework that could bring improvements to various applications in transportation systems. We aim to answer the following questions:

(i) *How should we leverage the advantages of edge computing to accelerate and improve vehicular sensing applications?*

(ii) *What are good ways to manage data access and to control what data each stakeholder in this ecosystem can access?*

(iii) *What are good resource allocation strategies to achieve a predefined goal, and what are the trade-offs between resource allocation and system performance?*

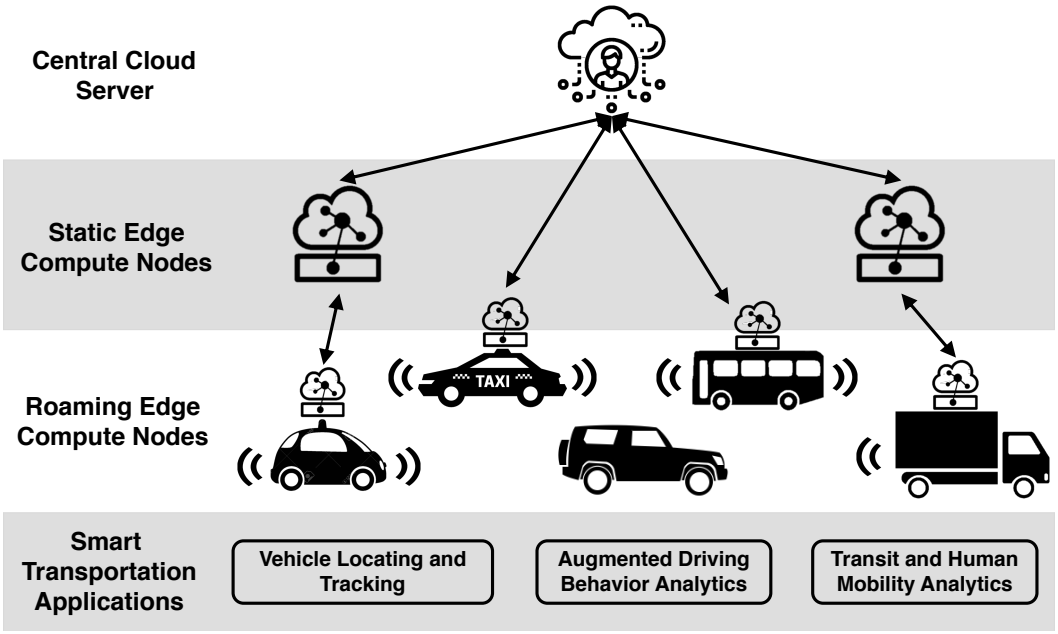


Figure 1.1: Our work in this thesis. An edge-enhanced vehicular sensing framework and its applications. We completed three applications leveraging the roaming edge compute nodes. A vehicle tracking application, a driving behavior analytics application and a transit and human mobility analytics application. Each application can choose to leverage resources on roaming edge nodes, static edge nodes and cloud-hosted services.

We have conducted research by developing three smart transportation applications to explore the answers to these questions. We envision that connected sensors can be installed on a moving platform and leverage roaming edge computing platforms to bring improvements to various applications in transportation systems. An overview of our methodology is summarized in Figure 1.1. We designed an edge computing based vehicular sensing and computing framework designed for smart transportation applications. Each application can choose to leverage resources on roaming edge nodes, which are general-purpose compute platforms mounted in participating vehicles that interact with in-range sensors to perform local processing, static edge nodes, that locally coordinate across a set of roaming edge nodes, and cloud-hosted services for global coordination. For this thesis, we put a previously

developed edge computing platform called ParaDrop [5] in the vehicle and build applications upon it. ParaDrop is an edge computing platform that provides computing and storage resources allowing developers to flexibly create various kinds of services. The ParaDrop platform supports multi-tenancy and a cloud-based backend through which computations can be orchestrated across many ParaDrop access points (AP). ParaDrop also provides APIs through which developers can manage their services across different ParaDrop APs.

We first built a vehicle locating and tracking application that can find a target vehicle over a wide area based on provided descriptions (e.g. an AMBER alert). Then, we have been building a multi-modal and lightweight real-time sensing system that leverages audio-visual cues to augment driving behavior analysis. The application overcomes the limitations of existing Inertial Measurement Unit (IMU)-based solutions and can provide fruitful contextual information for driver behavior profiling. In addition, we implemented a low-cost WiFi-based in-vehicle monitoring and tracking system that can passively observe mobile devices and provide various analytics about people both within and outside these vehicles, leading to interesting population insights at a city scale. City operators, law enforcement and other related organizations can benefit from such applications to provide better services. Building such applications is a great opportunity to understand how can applications benefit from the proposed edge computing enhanced vehicular sensing framework and explore how to distribute tasks such that static and roaming edge compute nodes can collaborate more efficiently for superior performance. We also have the chance to learn how to extract useful information from multiple resources and the complexity to manage the distributed services. We now describe each of these applications in more details in the following sections.

1.2 Edge Enhanced Vehicular Sensing Framework and Its Application to Vehicle Tracking

City-scale monitoring requires large amount of high quality sensing data with sufficient sensing coverage. It is a challenging problem as it uses static sensors to meet sensing and monitoring goals. A static infrastructure is more challenging to scale given such sensors need to be deployed at every location of interest in the city. If many vehicles incorporate such connected sensors, they together can be composed into a unique crowd-sourced platform, they can gather fine-grained, diverse, and noisy information at city-scales. We believe there exist significant opportunities to leverage many vehicles that ply daily city routes, e.g., transit buses, garbage trucks, taxis, and more — and equip them with sensing infrastructure to gather information at city-scale. However, multiple sensors can generate large amounts of data and it is hard to aggregate in a centralized cloud-hosted location. We start by exploring the design of a roaming edge that extends the notion of edge computing to provide some general-purpose computing capabilities in such vehicles to support diverse applications. A roaming edge node allows different sensors and data sources in a vehicle to connect to it, and supports data processing for necessary local analytics, for efficiency and privacy. By leveraging this roaming edge, we present Cascade, a crowdsourcing based sensing and computing framework designed for smart transportation applications. Cascade uses a two-stage edge structure — a roaming edge in vehicles and the traditional static edge both of which support different computing functions. Together, they allow various broad queries to be formulated as subproblems for compute nodes with different capabilities thus every compute node (including the ones in vehicles), can efficiently contribute to the end goal. We illustrate an example approach to design a vehicle tracking application with the Cascade two-stage structure. The vehicle tracking application consists of three major components. A query source (such as a public safety agency) issues the query, indicating the need to locate and track a vehicle — and provides important features such as vehicle type, color, make, model, and license plate

information. Workers (roaming edge nodes) are participating vehicles that are equipped with one or more cameras, optional motion sensors, and roaming edge compute nodes installed in their vehicles. Managers (static edge nodes installed at base stations, traffic lights, etc.) are responsible for coordinating the actions of all the workers in a predefined service area. We evaluate system and application performances over a large number of queries covering various scenarios (e.g., number and model of vehicles, the number of compute nodes, etc.) obtained through real-world experiences. Experimental results, under our assumptions, show that we can achieve an average identification accuracy of over 89% within reasonable latency. Further, we also explore some good resource allocation and task assignment strategies under different application scenes.

1.3 Augmented Driving Behavior Analytics

We now focus on augmenting driving behavior evaluation with multimodal sensor fusion on roaming edge computing platforms. We explore the solution of better evaluating a driver's behavior with surrounding context information. The way people drive vehicles has a great impact on traffic safety, fuel consumption, and passenger experience. Many research and commercial efforts today have primarily leveraged the IMU to characterize, profile, and understand how well people drive their vehicles. However, we observe that such IMU data alone cannot always reveal a driver's context and therefore does not provide a comprehensive understanding of a driver's actions. For example, a hard brake could be due to an inattentive driver suddenly realizing a vehicle ahead has already stopped (a bad driving action), or it could be to avoid a pedestrian who suddenly came in front of the vehicle (a good driving action). To find the missing context information, we believe that to better understand driving behaviors one can effectively leverage audio- visual cues, using a few vehicle-mounted cameras and microphones, to complement the existing use of IMU sensors. For instance, such an audio-visual system can easily discern whether a hard braking incident, as detected by an accelerometer, is the result of inattentive driving (e.g., a distracted driver) or evidence of alertness (e.g.,

a driver avoids a deer).

The focus of this work has been to design a relatively low-cost audio-visual infrastructure through which it is practical to gather such context information from various sensors and to develop a comprehensive understanding of why a particular driver may have taken different actions. Analyzing high resolution audio-visual data is often delegated to very high-end GPU-enhanced compute clusters located in different data centers. However, in our scenario, it is likely that the vehicles equipped with audio-visual sensors can easily generate a high volume of data rather quickly, which implies that the audio visual analytics should need to be performed in real-time. Therefore, the challenge of this work is how to leverage the roaming compute node to process audio and video in real-time inside the vehicle. In particular, we build a system called DrivAid, that collects and analyzes visual and audio signals in real time with computer vision techniques on a vehicle-based edge computing platform, to complement the signals from traditional motion sensors. To support efficient audio visual analytics in DrivAid, we put a GPU-enhanced embedded computing platform with optimized deep learning inference engines in the vehicle and deployed the analytics module on it. We also use smartphone motion sensors to detect different driving events and only conduct further analysis once an event is detected. Besides, with the in-vehicle setup, driver privacy is preserved since the audio-visual data is mainly processed locally. We implement DrivAid on a low-cost embedded computer with GPU and high-performance deep learning inference support. In total, we have collected more than 1550 miles of driving data from multiple vehicles to build and test our system. The evaluation results show that DrivAid is able to process video streams from 4 cameras at a rate of 10 frames per second. DrivAid can achieve an average of 90% event detection accuracy and provide reasonable evaluation feedbacks to users in real time. With the efficient design, for a single trip, only around 36% of audio-visual data needs to be analyzed on average.

1.4 Transit and Human Mobility Analytics

In this work, we focus on how to enhance public transportation and observe human populations using vehicle mounted sensors and roaming edge computing platforms. Public transit systems serve millions of users every year. An efficient and high quality public transportation system can not only benefit passengers, but also have a serious impact on city development. Hence, public transit has always looked for mechanisms that allow them to improve their services for people in terms of, say, what new routes or stops should be introduced, how do peak and off-peak behaviors be handled, and much more. However, efficient ways of gathering usage information such as popular origin-destination pairs and occupancy of the vehicle, is currently lacking. As mobile devices have transformed crowd-sourced data collection in a whole range of domains, we believe that transit systems and city operators can also benefit significantly from it. To solve this problem, we propose Trellis — a low-cost Wi-Fi-based in-vehicle monitoring and tracking system that can passively observe mobile devices and provide various analytics about people both within and outside a vehicle which can lead to interesting population insights at a city scale. Our system runs on a vehicle-based roaming edge computing platform and is a complementary mechanism which allows operators to collect various information, such as original-destination stations popular among passengers, occupancy of vehicles, pedestrian activity trends, and more. A key challenge is that how to distinguish passengers from pedestrians and determine when a certain passenger gets on and off the vehicle. To solve this problem, Trellis, takes advantage of these widely available mobile devices among passengers and pedestrians to quickly gather various forms of usage information at a significantly large (city) scale. Trellis makes this distinction by simply observing signal strength trends of Wi-Fi devices at instants when a vehicle is in motion, this localization problem becomes quite simple and can be solved fairly accurately. To conduct most of our analytics, we develop simple but effective algorithms that determine which device is actually inside (or outside) of a vehicle by leveraging some contextual information.

We demonstrate how such a system may be used from three major perspectives. First, we focus on passenger riding habits, i.e. what are the popular origin-destination pairs and how do these origin-destination pairs vary for different stations, at different locations, and at different times of the day. Next we study patterns of people on city streets. For example, urban planners often want to know how busy their city streets are, and where hotspots are during different times of days and periods of the year. Finally, we study the impact of weather on human mobility outdoors. More specifically, we observe how inclement weather (snow and rain) and outside temperature affects the number of people in the transit vehicles or out on city streets. We have deployed Trellis on a vehicle-based edge computing platform over a period of ten months, and have collected more than 30,000 miles of travel data spanning multiple bus routes. By combining our techniques, with bus schedule and weather information, we present a varied human mobility analysis across multiple aspects — activity trends of passengers in transit systems; trends of pedestrians on city streets; and how external factors, e.g., temperature and weather, impact human outdoor activities. These observations demonstrate the usefulness of Trellis in proposed settings.

1.5 Contributions

We built smart transportation applications to understand the advantages of using edge computing paradigms for vehicular sensing applications. This dissertation describes the design of edge computing enhanced vehicular sensing framework and implementations of applications based on the proposed framework.

In this dissertation, we focus on three vehicular related applications, target vehicle tracking, driving behavior analytics and transit and human mobility analytics. These applications demonstrate how to can connected devices, static and roam edge compute nodes cooperate with each other more efficiently, and how to achieve a pre-defined goal with minimum resources. We also explore the advantages of edge computing platforms on reducing network bandwidth usage, better responsiveness

and privacy protection. Specifically, our contributions of this dissertation are the following:

1. We proposed a two-stage crowdsensing framework enhanced by edge computing platforms - Cascade, which improves the efficiency of the system by fully leverage computing capabilities of both static and roaming compute nodes. Cascade achieves a better efficiency by formulating subproblems for compute nodes with different capabilities thus every compute node (including the ones in vehicles), can efficiently contribute to the end goal. We designed and implemented a vehicle locating and tracking application using Cascade, to demonstrate how to break a complex problem into simpler sub-problems such that both workers and managers are able to contribute to the end goal. To demonstrate the system flexibility and improve application performance, we developed a context-aware data encoding and streaming protocol, which can improve streaming efficiency and reduce latency. We presented a feature fusion model to demonstrate how to integrate extracted information from different workers and achieve improved information accuracy. We collect real-world driving data as well as publicly available data to evaluate application performance under different settings. Experimental results verify the effectiveness of the Cascade and give the idea of how to allocate computing resources in this specific application.

2. We designed and implemented DrivAid, a real-time low-cost sensing and analyzing system for driving behavior evaluation. We illustrated how such a system could be built that leverages audio-visual cues to augment driving behavior analysis based on IMU sensors in a holistic manner. DrivAidis a lightweight, powerful system that can be easily deployed in regular vehicles by adapting and integrating existing algorithms to let them run efficiently in vehicular edge computing nodes. We evaluated our system with more than 1500 miles' drive data. A prototype is deployed on a regular vehicle and evaluated through test drives of around 50 miles in real world environments. The evaluation results show that our system can process data in real time and

provide a good understanding of each driving behavior. We believe such a real-time sensing and analysis system can enable a wide range of applications.

3. We designed and implemented Trellis, a low-cost in-vehicle wireless monitoring system that can track passenger movements and study pedestrian behaviors to assist transit operators, and potentially city planners, with various forms of human mobility analytics. We developed several simple heuristic algorithms that can effectively separate passengers from pedestrians and identify where passengers get on or off a vehicle. To test the efficacy of our system, we deployed Trellis on vehicle-based edge computing platforms over a period of ten months and collected data from 3 bus routes. We evaluated how it can be used to infer origin-destination pairs that are popular among passengers over time and space. We demonstrate and quantify different impacts on human activities caused by different factors (e.g., weather and temperature). As we continue to work with our local transit partners, we continue to evaluate how such a system can be used to identify where to add new bus routes, or when to add non-stop services between various stations throughout the city at different times of the day and under different weather conditions.

1.6 Outline

The remaining of the thesis is organized as follows. In Chapter 2, we present the roaming edge-assisted crowdsensing framework and its application to vehicle locating and tracking. In Chapter 3, we propose DrivAid, a comprehensive driving behavior evaluation system using audio and visual cues. In Chapter 4, we present Trellis, a low-cost in-vehicle wireless monitoring system, explain how it can track passenger and pedestrian movements to derive various forms of transit and human analytics. We compare our work with existing solutions and discuss the related work in Chapter 5. We conclude this dissertation and discuss further research directions in Chapter 6.

2 A Crowdsourced Sensing and Computing Framework Leveraging the Roaming Edge

2.1 Introduction

Intelligent Transportation Systems (ITS) are receiving increasing attention recently as connected sensors enable innovative services to improve transportation safety, mobility and provide useful transportation analytics. Most planners try to gather city-scale traffic flow information using a set of static connected sensors (e.g., road-side cameras, and road sensors). For example, parking is always a challenge in many urban areas, which further leads to increased congestion, greater carbon emissions, and driver frustration. To provide parking spot availability information, many intelligent parking services have been built using static sensors that are installed under or above individual parking spots. However, installing these static sensors across a large area incurs massive deployment cost and effort, and the maintenance costs are also significant. Moreover, those static sensors are mostly deployed at fixed places and thus the coverage of these sensors is necessarily limited.

We envision that these connected sensors can be installed on a moving platform to complement the drawbacks of static sensors. In particular, we focus on vehicles — autonomous or otherwise — that can be equipped with a plethora of sensors such as camera, LiDAR (light detection and ranging), RADAR and more. These in-vehicle sensors can provide context information of its surrounding and once a

reasonable fraction of vehicles are equipped with these sensing platforms, they can naturally become a crowd-sourced platform to gather, analyze, and organize various forms of data at the city scale and beyond. However, in-vehicle sensors can easily generate a high volume of data rather quickly. For instance, a few cameras or a LiDAR system can create hundreds of megabits of data every second. Such data cannot be easily offloaded from vehicles to the cloud for further analysis, due to lack of available network capacity from the vehicles. Thus this requires finding an appropriate processing platform and designing efficient operation mechanisms that can improve data management and analysis efficiencies.

To push computational and storage capabilities closer to end sensing devices, we consider an edge computing paradigm called the *roaming edge*, in which the edge nodes themselves are in motion, e.g., mounted in vehicles. Edge computing platforms provide computational support closer to where the data is created, which can be leveraged to provide greater data privacy, save network and computing resources and makes applications more efficient and responsive. For instance, a roaming edge node, installed in a vehicle may allow for local processing of dash camera videos, without requiring offload. These characteristics make roaming edge computing platforms suitable for providing the first line of analytics support. At the same time, we emphasize that in our description, roaming edge nodes are not being envisioned as the end device — they typically will allow end devices, such as cameras, LiDARs, and other sensors to connect (wirelessly) and provide a general-purpose computing support where new services can be flexibly installed and utilized. We believe that a distributed vehicular sensing approach could bring improvements to various applications in transportation systems.

In this paper, we present Cascade, a crowdsourcing based vehicular sensing and computing framework designed for smart transportation applications that leverage the roaming edge. Cascade has a two-stage structure and consists of the roaming edge nodes (in vehicles), referred to as workers, and static edge nodes, referred to as managers, which interact with a centralized cloud-hosted service (administrator) to manage broad queries being addressed to the system. Every manager is responsible for one specific service area.

Cascade can be used to pose and answer a large number of broad questions. For instance, the camera sensors in all vehicles could be used to track how frequently different traffic signals change from red to green in specific intersections and vice-versa. They can be used to determine the change and prevalence of honking in certain road segments. They can also be used to distributedly locate a vehicle sought for public safety, e.g., due to an AMBER alert that has been raised. In each of these examples, the task can be split among the various managers (static edge nodes), which can coordinate results across a number of workers (roaming edge nodes) that are collecting and continuously analyzing data from their local, in-range sensors.

With a reasonable number of distributed roaming edge nodes (workers), Cascade is able to provide new and unique forms of transportation related information that is not available before. Each roaming edge node in Cascade is a multi-tenant resource (such as those provided by ParaDrop [5]) that allows new functionality to be installed as a new third-party service. In the rest of this paper, we describe Cascade as a specific application — a vehicle tracking application using the Cascade framework to explore challenges of building applications leveraging the roaming edge. We consider the following questions when building this vehicle tracking application using Cascade. (i) What are good ways to manage data access and to control what data each stakeholder in this ecosystem can access? (ii) What are good resource allocation strategies to achieve a predefined goal, and what are the trade-offs between resource allocation and system performance? (iii) How should we fuse data from multiple sources and derive information with improved accuracy? As shown in Figure 2.1, the admin, could be the police department in this application, publishes a task asking the crowd tracking a specific vehicle. Although the license plate is the best identifier for vehicle tracking, it is very hard to correctly recognize it if the vehicle is not close enough. To improve tracking efficiencies, we prepare different tasks for workers and managers such that they can fully contribute to this task. Suppose we are tracking a white Honda CRV with license plate VFX-434, a worker is only responsible for tracking white SUVs using vehicle dash cameras and reports to its manager when necessary. The local static manager conducts further analysis based information received from one or multiple

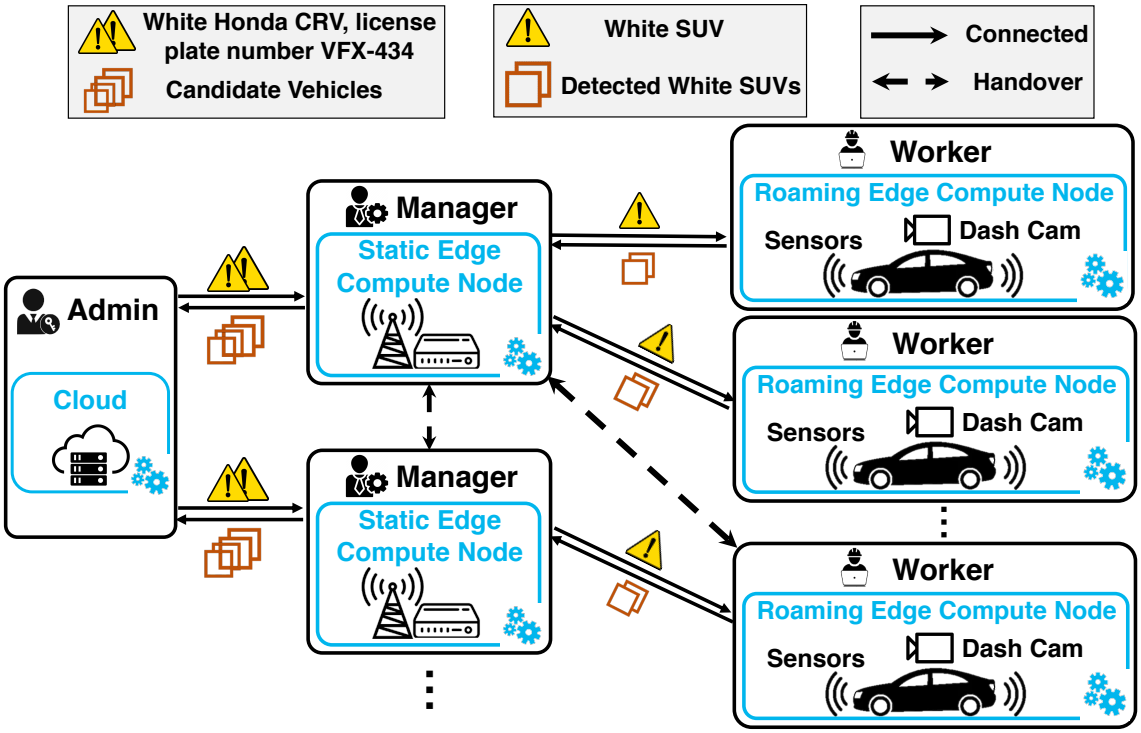


Figure 2.1: An example application build upon Cascade.

workers. If the target vehicle is identified, the manager reports its findings to the admin and asks workers to focus on tracking that vehicle. When a worker is about to leave a service area and the task is still running, the manager updates the current status with the manager in the adjacent service area (hand over in Figure 2.1). The worker collaborates with both managers until the hand over process finishes.

In summary, our Cascade system consists of roaming edge (worker) nodes, which are general-purpose compute platforms mounted in participating vehicles that interact with in-range sensors to perform local processing, static edge (manager) nodes, that locally coordinate across a set of workers, and a cloud-hosted administration service for global coordination. To minimize data being uploaded and maximize the usage of the roaming edge, Cascade breaks a complex problem into simpler sub-problems such that both workers and managers are able to contribute to the end goal. Given worker nodes have lesser computing capability, they

are assigned some simpler computational tasks, and manager nodes aggregate information from various local workers to obtain greater fidelity in results across a set of workers.

In this work, we collect real-world driving data as well as publicly available data to evaluate application performance under different settings. Experimental results verify the effectiveness of the Cascade and give the idea of how to allocate computing resources in this specific application. During the traffic peak hours (many vehicles are on the road), a fleet of five roaming compute nodes in one manager's service area can have a relatively high chance to successfully track the "suspect vehicle" with a reasonable latency. While during off-peak hours, three workers are enough to provide a precise "suspect vehicle" location (no matter how much the vehicles on the road look like the "suspect" one). On average, we can achieve an overall identification accuracy of over 89% considering traffic conditions change during the day.

Contributions: The contributions of this work can be summarized as follow:

- We propose Cascade, a two-stage vehicular crowdsensing framework enhanced by roaming edge computing, which can provide transportation related analytics at the city scale.
- To demonstrate a concrete example, we implemented a vehicle tracking application using Cascade, to explore how to overcome new challenges when building applications leveraging the roaming edge.
- We conduct a series of benchmark studies using real-world driving data and publicly available data to evaluate the application performance and demonstrate the usefulness and effectiveness of Cascade.
- We explore the best resource allocation and task assignment strategies under different application scenes based on experiment results.

2.2 Motivation

City-scale monitoring is a challenging problem as it uses static sensors to meet sensing and monitoring goals. A static infrastructure is more challenging to scale given such sensors need to be deployed at every location of interest in the city. In many applications, described below, we believe there exist significant interesting opportunities through a second complementary approach — leverage many vehicles that ply daily city routes, e.g., transit buses, garbage trucks, taxis, and more — and equip them with sensing infrastructure. Of course, if each such vehicle were to blindly upload all sensed data, it might be wasteful of available network capacity. Instead, we consider a multi-stage structure that can be used to support real-time (or offline) queries that can be posed of these diverse arrays of vehicles gathering and contributing data to address such queries. We explain our approach by using a few examples.

Tracking objects or people of interest: In many situations, public safety agencies need to search for and track people or objects of interest. It is possible to leverage static security cameras across a city for this purpose. But we note that increasingly vehicles are getting equipped with cameras for incident response and management, and if they can be re-purposed to create a city-scale sensing infrastructure, it can be a powerful tool for this highly time-critical query. Of course, if every vehicle attempted to simply stream its own video, from multiple cameras, it will overwhelm the available network capacity of any mobile infrastructure. Instead, the proposed roaming edge, with compute nodes installed in each vehicle, can provide the first stage of analytics, e.g., to match the vehicle color and type. Only if a match is detected in this stage, is the video or images uploaded to the second stage compute nodes, say at the static edge for license plate matches. This hierarchical approach naturally leverages the diverse computational capabilities that are likely available in different parts of the infrastructure.

Parking analytics: Parking is always a challenge in urban areas, and especially during peak hours in busy downtown hotspots. It is common that vehicles sometimes spend a significant amount of time circling city blocks in order to find parking,

which leads to increased congestion, greater carbon emissions, and driver frustration. Again, a similar infrastructure that leverages vehicle-mounted cameras to report available parking spots could be of great interest. Since parking information depends on recency, these queries may need to be addressed in near real-time. The detection of an available parking spot, if done using camera images, requires necessary processing, and again a multi-stage hierarchy might be useful to ascertain the availability of such spots. An approach based on such mobile, vehicle-mounted cameras (likely to be commonplace in vehicles) will significantly reduce the infrastructure costs needed for this application.

Traffic Analytics: City planners actively look for methods that can gather numerous forms of transportation analytics at the city scale, e.g., where are the hotspots right now that need congestion alleviation. While the city can be covered with various traffic cameras everywhere, extensive deployment of such a static infrastructure is expensive, and if the vehicle cameras can collaboratively share information, an intriguing solution to the problem is possible. Again the proposed approach of our work can be quite a handy solution to the problem.

Discussion on feasibility and other challenges: In the discussion of this proposed model of vehicle-mounted sensors to create city-scale monitoring infrastructures, we are definitely mindful of numerous challenges that will occur. First and foremost, perhaps, is privacy concerns, an issue that is likely important to any monitoring infrastructure. We believe that the use of a roaming edge, which processes raw data at the source, is particularly amenable to implement various privacy preservation mechanisms. It provides an opportunity to remove any private data at the source, thereby making the rest of the system to provide certain levels of guarantees. We do not focus on the privacy policies and mechanisms in this paper, but acknowledge that this would need to be a key part of individual problems being addressed.

Second, is the common concern around why vehicles will expend their own compute, power, and communication resources to serve an unrelated application. This is a concern that plagues every crowd-sourced application, and we believe an appropriate model can emerge if there is cost benefit to this approach. For instance, if a city realizes that it is going to save significant costs by not installing a

certain static infrastructure, they may be willing to provide some incentives to every vehicle that is willing to contribute data. Minimally, a city can deploy such sensors and roaming edge compute platforms on their own vehicles. Just installing such capabilities in city transit buses, garbage trucks, and other city-owned vehicular networks, might be sufficient to bootstrap a useful infrastructure with a greater reach and higher agility than a static infrastructure.

Third, could be a concern around malicious actors that may attempt to misguide the application in question. Of course, they can be addressed by the power of crowdsourcing where significant collusions are necessary to effect incorrect outcomes. Further, it may be possible to require compute nodes to attest their results, so that offenders in such settings may be easily identified. Finally, individual applications may choose to only trust compute results from certified edge nodes among the vehicles.

This paper is scoped to focus on the distributed compute infrastructure among the static and the roaming edges, and how examples applications can be designed to meet desired goals. Some of these above aspects are fairly significant and would need careful consideration in future efforts.

2.3 System Design

In this section, we present the overview of Cascade, and introduce design choices in detail.

Two-Stage Structure

As shown in Figure 2.1, Cascade proposes a two-stage structure that leverages the concept of crowdsourcing to solve a problem by combining the efforts from a large group of participants. As mentioned, a roaming compute edge node, in a vehicle, or a worker, with limited computing capabilities are often assigned simpler compute tasks. Similarly, a static edge compute node, or a manager, typically supports more complex computing functions, and aggregates information from various local

workers in its jurisdiction. Finally there exists a cloud-hosted administrator. Every manager is assigned to a specific service area. When a worker moves to a new area, it reports its current location to the administrator and the administrator assigns the worker to the local manager in that area. Depending on the application, a manager sets up a connection with one or more workers. The manager dynamically determines when to start a new task and assign the task to one or more workers. Connected workers follow instructions from the manager and upload collected data collected to the manager when necessary. With data collected from multiple workers, the manager conducts further data analysis and decide appropriate tasks for execution. When a worker is about to leave a service area and the task is still running, the manager updates the current status with the manager in the adjacent service area (hand over in Figure 2.1). The worker collaborates with both managers until the hand over process finishes.

As different types of end devices, e.g., cameras, LiDAR and connected sensors, can be utilized to address a large number of broad queries, Cascade provides a unified and easy to use programming interface to easily distribute and manage tasks across edge nodes. A virtualized environment is created on each edge node based on Docker [6] which allows application developers to select the programming languages, libraries, and platforms based on requirements of static and roaming edge nodes. To provide more flexibility, Cascade provides various APIs through which developers can manage the resources (e.g., sensors, RAM, CPU, etc.) of the edge node and monitor the running status of the application.

Edge Node Management: This type of API is responsible for edge node management tasks. For example, a worker reports to the admin for its availability and a nearby manager will be assigned to collaborate with that worker. When a worker is about to leave a service area and move to a new area that is managed by another manager. The worker works with both the old and the new managers during the handover process. All these communications and management processes can be done through API calls.

Data Transmission: Different types of sensors generate data at different rates and various applications have different latency requirements. Cascade implements

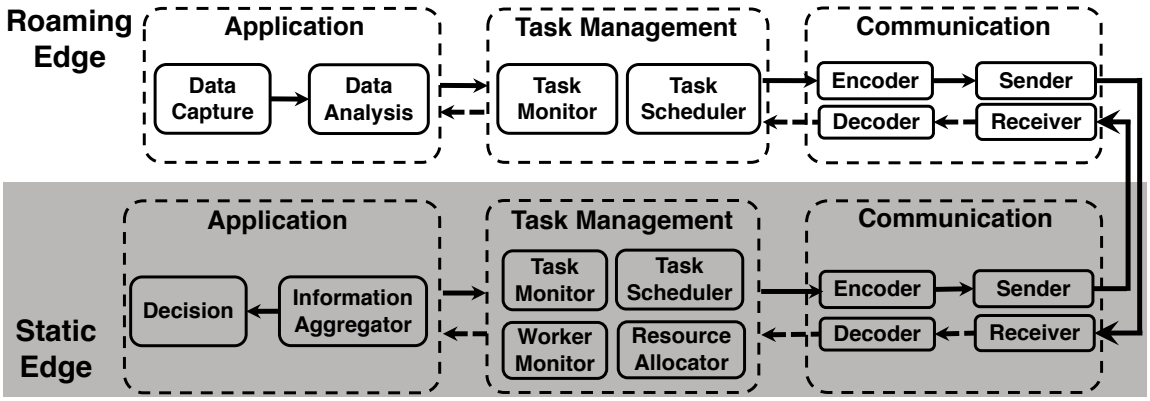


Figure 2.2: Service management module overview.

a pilot context-aware streaming protocols to accommodate various needs. Third party developers can choose appropriate protocols through APIs. Besides, Cascade allows end-users to easily add customized APIs to satisfy their extra needs.

Application Management: For each application, the admin needs to define specific tasks for managers and workers based on their capabilities before starting a new job. In Cascade, edge node executions and application level tasks are also controlled through APIs. Developers can develop application specific APIs to help them dynamically interact with their applications. Using vehicle tracking as an example, the manager sends a query (e.g., finding a red SUV) to a worker by calling an API. Workers send feedback to the manager through a call back function. The manager conducts further analysis to find a license plate and other information once receive the callback.

Service Management

Typically there are three modules of an edge node, data communication module, task management module, and application module. The responsibilities of these modules might vary for static and roaming edge nodes. Third party application developers can create more modules based on their needs. Figure 2.2 shows an overview of the service management modules at two stages. A worker's processing flow begins with the application pipeline defined at stage one and data flows over

the system to stage two until it reaches a decision at the stage two application layer. In the application module, users can choose the data source, develop their own algorithms to analyze data, and decide how to allocate tasks for stage one workers and stage two managers thus every compute node with different capabilities can efficiently contribute to the end goal. The task management module manages tasks running on a compute edge. In stage one, the task management module only needs to coordinate various tasks running on this single node. While in stage two, other than tasks running within the compute node, the task management module also needs to interact with other stage one compute nodes. The communication module manages connections and handles network communications between static and roaming compute nodes. To improve transmission efficiency, Cascade offers different data encoding and transmission techniques for different types of raw data. Application developers can choose appropriate techniques based on their requirements. Details are discussed in Section 2.3 and 2.3.

Data Encoding

Different types of sensors generate data at different rates. For example, proprioceptive sensors (motion sensors, GPS, etc.) generate few kilobytes raw data per second, while exteroceptive sensors (cameras, LiDAR, etc.) generate few gigabytes of data every second. We need a network with gigabit bandwidth to transmit such amount of data in real time, which is not easy to be handled by today's wireless networks. In our system, we implement a proof of concept data encoding and streaming protocol to reduce network bandwidth usages.

Proprioceptive Sensors

Proprioceptive sensors, like accelerometer, gyroscope, magnetic sensors, GPS and so on, generate time series data at a pre-set rate. It usually includes a time stamp and several bytes of sensory data. Hence, the size of the recorded data is relatively small compared to exteroceptive sensors. For example, motion sensors and GPS generate around 5 kilobytes of data every second if the sampling rates of motion

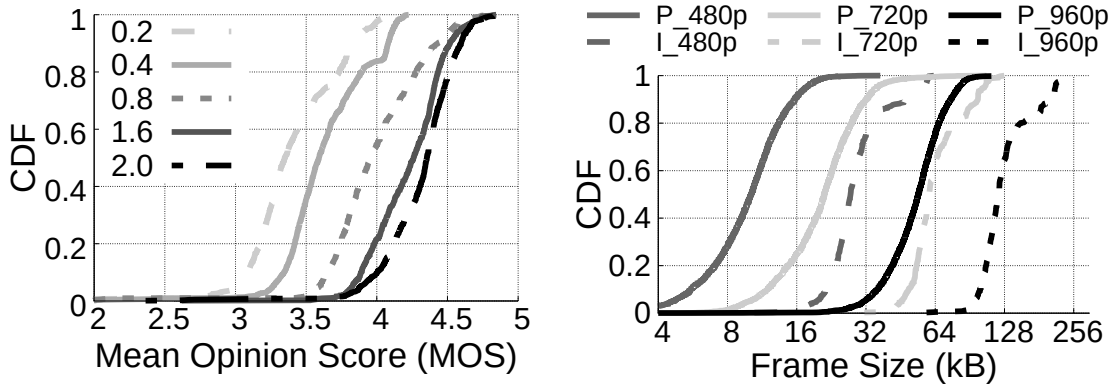
sensors are set to 100Hz and that of GPS is set to 1Hz. Generated raw data can be easily streamed through existing networks in real time. Hence, we stream such data directly when needed.

Exteroceptive Sensors

Since proprioceptive sensors generate a small amount of data, we mainly focus on exteroceptive sensors in this work. We use cameras as an example to demonstrate how to compress and stream large data sets. Suppose the resolution of a video frame is 960*720, the size of this frame is 3*960*720 bytes (around 2 megabytes). If 15 frames are streamed every second, we need a network with 200Mbps bandwidth. Although it is possible to set up a network with 200Mbps bandwidth, such resources would be precious and cannot be provided to the public. Various encoding techniques have been developed to solve this issue, and we implement a context-aware encoding technique to improve the streaming efficiency for our system.

Video Quality Typically there are three important factors that affect video quality, resolution, bitrate and encoding codec. To maintain the same video quality, a higher resolution needs a higher bitrate and results in a larger file size. To evaluate the frame quality, we use the Blind and Referenceless Image Spatial Quality Evaluator (BRISQUE) and calculate a Mean Opinion Score (MOS) [7] to evaluate the quality of the frame. The MOS score ranges from 1 to 5 with 1 represents the worst quality and 5 indicates the best quality. Figure 2.3a shows how different bitrates affect video quality. We encode a video using different bitrates (0.2mbps to 2mbps) and calculate corresponding MOSs. The video quality is better when the bitrate is higher under the same resolution. As the bitrate doubles, the video quality improves on a linear scale.

Frame Encoding We choose the H.264/MPEG-4 video encoding standard [8] to encode video frames. There are three types of frames in the H.264 standard, Intra-coded frame (I-frame), Predicted frame (P-frame) and Bidirectionally predicted picture (B-frame). The I-frame is a full frame that contains all the bits required to construct that frame. And the P-frame has to be constructed by the decoder



(a) Frame quality under different bitrates.

(b) CDFs of frame sizes under different resolutions.

Figure 2.3: Frame quality and size under different settings.

based on previous frames since it is derived by encoding a motion vector for each of the blocks from the previous frames. B-frame is generated using both previous and next frames to achieve a better compression rate, so it is not suitable for live streaming services. The sizes of I-frames and P-frames depends on the video encoding bitrate and resolution. Figure 2.3b shows the distribution of I-frame and P-frame sizes under various resolutions. We select three resolution and bitrate pairs, 640x480@1Mbps (480p), 960x720@2Mbps (720p) and 1280x960@4Mbps (960p) for comparison. On average, the size of I-frame is 2-4 times larger than that of P-frame under different resolutions.

Context-Aware Data Encoding

Encoding Parameter Other than frame resolution and bitrate, how frequently the I-frame will be used would also affect the video quality a lot. The frequency is defined as the I-frame interval and it is usually between 1-5s [9, 10]. A smaller interval is selected when the video contains a lot of motions, which reduces the chance of frame corruption. A larger interval is used for more static videos, where it can maintain the image quality while reducing the bandwidth requirement. It

is important to update the I-frame interval based on scene changes during the encoding process. Cascade improves the overall video encoding efficiency by understanding the surrounding context. The basic idea is to dynamically update the I-frame interval when the context changes.

Context Sensing We monitor context changes in each frame using the YOLOv3 object detection model [11]. If the difference of detected object numbers between two frames exceeds a threshold, we treat it as a context change. Moreover, motion sensors are used to monitor context changes by sensing vehicle dynamics. To track vehicle dynamics, we leverage motion sensors and the GPS to detect various driving events. We use similar techniques mentioned in [12] for turns, lane changes, brake detections. The context-aware encoding algorithm dynamically updates the I-frame interval based on detected context changes. Besides, Cascade provides APIs for users to change parameter settings to satisfy their needs.

Data Streaming Protocol

Most live streaming applications use User Datagram Protocol (UDP) with loss rate estimation and forward error correction mechanisms by default and switches to Transmission Control Protocol (TCP) if UDP traffic is blocked [9, 10]. In our framework, we implement the streaming protocol using both TCP and UDP, and conduct experiments to understand their performances. Note that we focus more on system design and proof of concept instead of improving streaming protocols.

Streaming Using TCP

TCP is a connection-oriented, end-to-end reliable protocol that offers "guaranteed delivery". If TCP is used for live streaming, then the receiver needs to wait for dropped packets before processing newer data. A socket buffer needs to be maintained on the sender side to keep track of the unacknowledged segments. In our implementation, we use the socket buffer size to estimate the TCP transmission rate and adjust the video encoding rate at the application layer. Based on the packet

loss rate in the network, the protocol will reduce the transmission rate and the application layer will reduce the video encoding rate.

Streaming Using UDP

In our UDP implementation, we add a forward error correction (FEC) algorithm to compensate for errors caused by packet losses. The packet loss rate and bandwidth are estimated on the receiver side. Estimation results are sent back using an acknowledgment mechanism.

Forward Error Correction To detect and correct errors, the sender sends a redundant error-correcting code along with the data frame. Each data frame is encoded into n packets of length l. Among the n packets, the first k packets are the raw data of the data frame, and the rest n-k packets are encoded as the redundant error-correction part. Each byte of the encoded redundant packet is the linear combination of the corresponding byte in the raw data packets. Therefore, the whole data frame can be recovered if any k of n packets are received at the receiver side. It is possible that each data frame has a different size, hence, k and n will be different for each data frames. To encode and decode one data frame, the corresponding packets should have the same length. We use a reference packet length to estimate the k first, and then assign each packet with equal length $(\textit{data frame size}/k)$. With this configuration, we need at most k bytes for the padding of the last packet. It reduces the extra padding from $O(l)$ to $O(k)$ (k is usually much smaller than l).

Loss Rate and Bandwidth Estimation The number of n and k are determined based on the loss rate when transmit data using UDP protocol. We estimate the loss rate on the server side. We assign a unique identifier starting from 0 to keep track of each frame and packet. For each frame, there are n numbers packets in total, the raw data packets have indexes ranging from 1 to k-1. And the encoded packets have indexes ranging from k to n-1. There is a buffer on the server side to maintain the frame identifier as well as the packet identifier. The buffer only stores packets of the current frame and if there are enough packets received to recover the whole frame, any future encoded packets of the current frame will be dropped.

Based on the received packets in the buffer we can have a sense of the loss rate and further update n and k accordingly.

If the bitrate is too large, the bandwidth might not large enough to satisfy the needs and the frame cannot be delivered to the server on time. Therefore, the encoding algorithm needs to know the network bandwidth to decide the best bitrate. In our setup, the bandwidth is also estimated on the server and update with the sender when necessary. We estimate the bandwidth every t seconds since each frame is divided into n packets and they may arrive in batch, so the bandwidth might be underestimated or overestimated if it is calculated based on received packets of each frame.

2.4 Vehicle Tracking Application

To demonstrate how to use Cascade, we illustrate an example approach to design a vehicle tracking application with our two-stage structure. We assume that the query source releases a query to the system indicating the need to locate and track a vehicle — and provides important features such as vehicle type, color, make, model, and license plate information. Once issued, the application attempts to locate and track such a vehicle. In our example, this vehicle tracking application consists of three major components described in Figure 2.4. A query source (such as a public safety agency) issues the query. Workers (roaming edge nodes) are participating vehicles that are equipped with one or more cameras, optional motion sensors, and roaming edge compute nodes installed in their vehicles. Managers (static edge nodes installed at base stations, traffic signals, etc.) are responsible for coordinating the actions of all the workers in a predefined service area. Figure 2.4 indicates the data flow from workers to the query source and interactions among three components.

Workers in this system have two limitations compared to the managers. First, any individual worker may have a limited field of view, especially to a license plate of another vehicle. Second, the compute capabilities in the workers are also limited compared to the managers. The latter, on the other hand, might be able to leverage

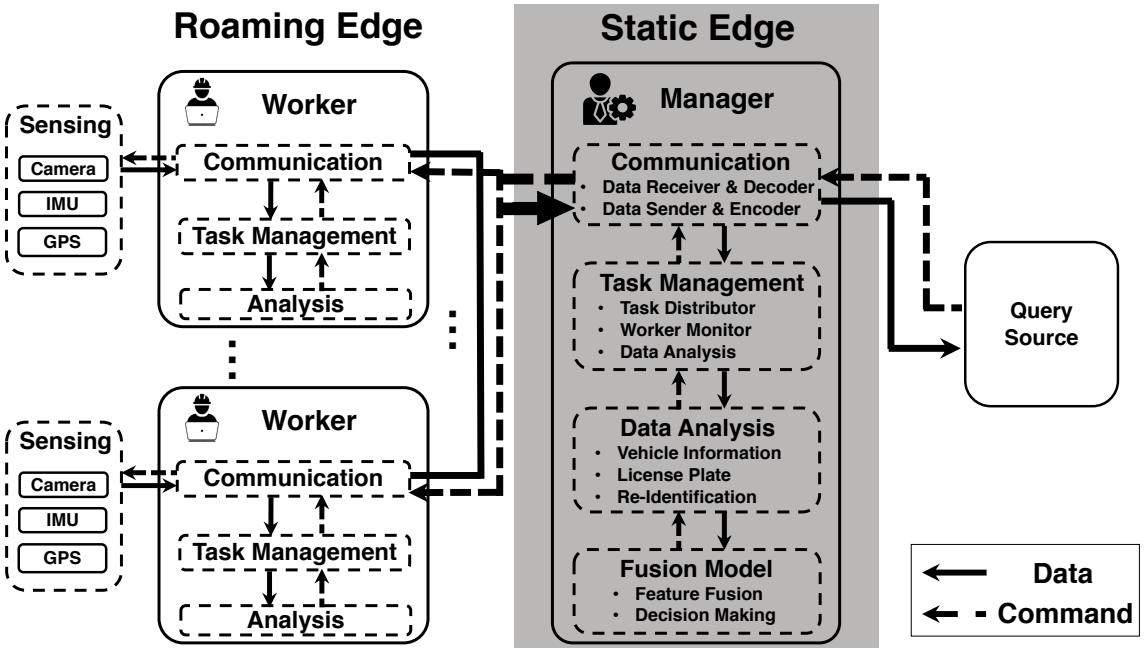


Figure 2.4: Vehicle tracking application overview.

information from multiple workers in its service area to construct improved views. With this in mind, we provide an example way to split computational tasks between the workers and the managers, noting that other alternatives to this design are definitely possible. Workers are tasked with identifying vehicles in camera range that match the vehicle type and color described in the query. If a worker finds a "reasonable" match[1], it will send the corresponding data (video clips, bounding boxes, etc.) to its manager for further analysis. The manager gathers information from multiple such workers in its service area to then identify the license plate and other information of the vehicle, from the query itself. The feature fusion model combines features of suspect vehicles extracted from one or multiple sources, and evaluates the similarity of suspect vehicles with the target vehicle. When the similarity level exceeds a certain threshold, the manager notifies the query source about its findings. The details of the process will be explained later in this section.

When a worker is available, it reports its availability and location information

[1]We refer to it as a suspect vehicle for the rest of the paper.

to the query source. The query source assigns a manager to that worker and notifies both the manager and the worker. The worker connects to the manager and waits for the manager's message to start working. Based on the current status, the manager schedules tasks for each connected worker. When a worker is about to leave the service area or it cannot keep tracking suspect vehicles, it notifies the current manager and waits to be assigned to a new manager. The worker works with both managers during the handover process.

Vehicle Detection

Object detection is a key objective of Cascade for both stages. The choice of a proper detection model should balance the run-time efficiency and accuracy. Object detection models are running on roaming edge compute nodes in stage one, while more powerful computing devices can be used for object detection in stage two. Therefore, we choose different models for different stages. In our example, for workers, we choose TensorFlow Lite [13] that can be easily run on lightweight compute nodes. TensorFlow Lite provides a pre-trained COCO SSD MobileNet model. To improve the accuracy of the detection model, we make use of transfer learning – take the weights of the pre-trained MobileNet model and train it with our own data, fine-tuning the layers from the MobileNet model. We choose DetectNet [14] as the object detection model in stage two since it can handle different input image sizes while providing reliable accuracies. To support real-time detection, we optimize the inference engine by using Nvidia TensorRT [15]. We prepare a large set of labeled images using our own data and open datasets [16, 17] to train detection models. Our models are able to detect five types of vehicles — sedan, SUV, minivan, truck and bus.

Vehicle Re-Identification

Vehicle re-identification (ReID) aims at identifying the same vehicle across multiple cameras. Most of the existing ReID solutions focus on finding the same vehicle across multiple stationary cameras while our goal is to find the same vehicle across

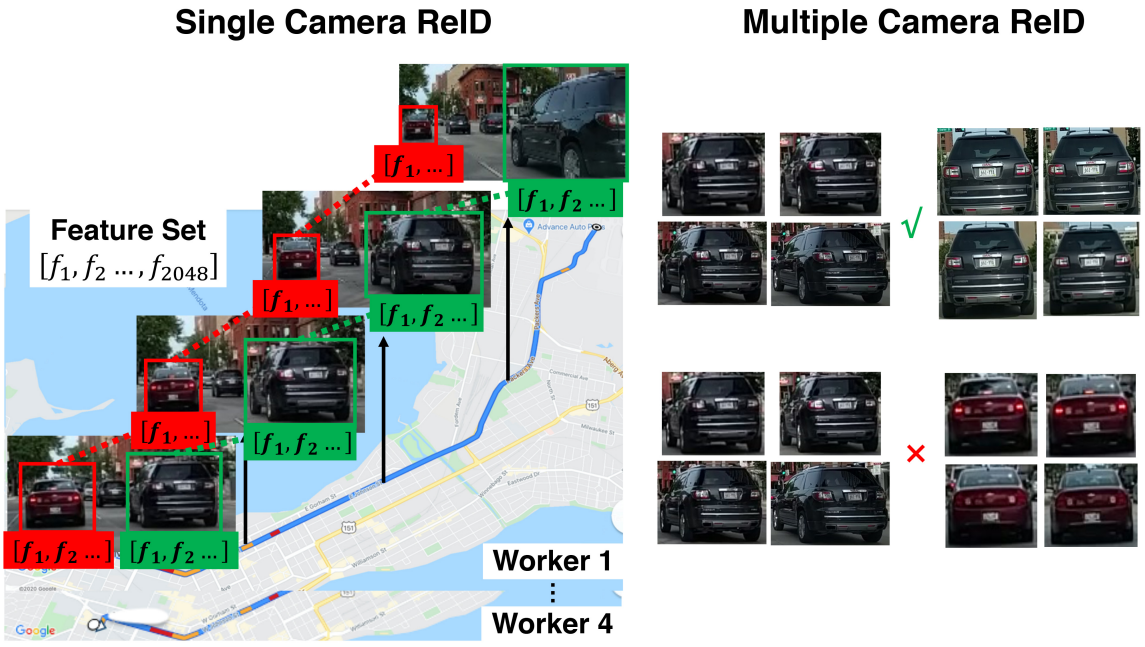


Figure 2.5: Vehicle re-identification.

moving cameras. We tackle this challenge in two steps. First, we locate as many vehicles as possible in every frame. We perform single-camera tracking to match vehicles across different frames. Next, we search for the same vehicles that appeared in multiple cameras by comparing features extracted from various tracklets. Tracklets with similar features are treated as the same vehicle. Figure 2.5 demonstrates a vehicle found in videos recorded by multiple workers.

In the first step, we train a Deep Convolutional Neural Networks (GoogLeNet [18]) to extract features from the detected vehicles. Next, we use the trained Deep Convolutional Neural Networks to generate a 2048-dimensional vector for each detected vehicle. To match vehicles in a single camera, we maintain a small buffer that can store a short period (e.g. 5 seconds) of the most recent video. Hungarian Algorithm [19] is implemented to link detected vehicles in different frames to form tracklets. The Hungarian Algorithm connects and relates detected vehicles based on the differences between 2048-dimensional features. Given the fact that the same

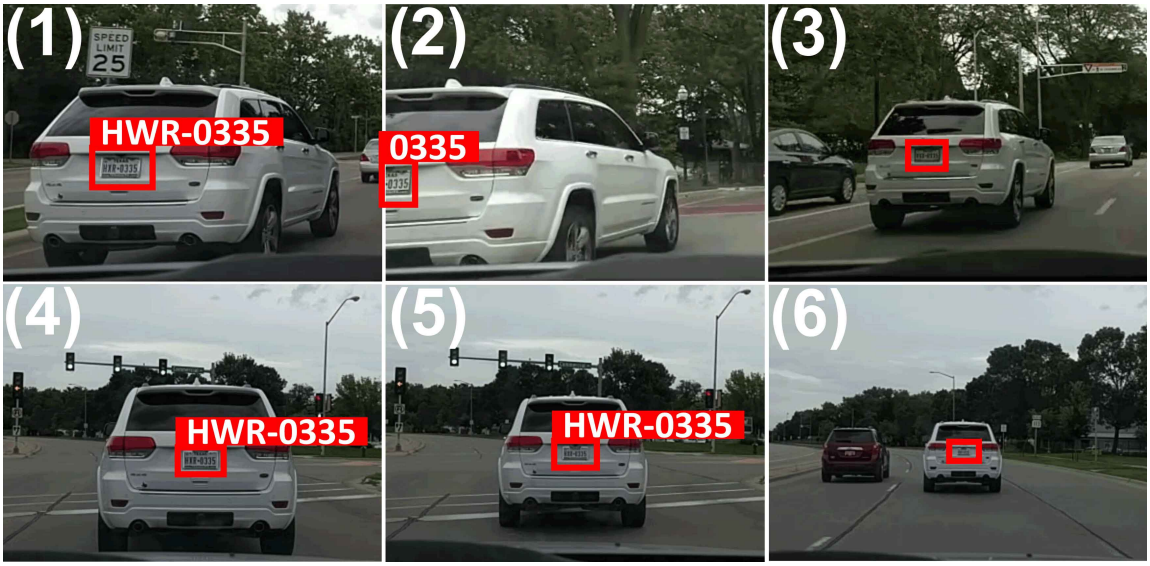


Figure 2.6: Vehicle re-identification and License Plate Recognition.

vehicle would appear in nearby locations in two consecutive frames, we also treat the location of the detected vehicle in each frame as a feature to improve matching accuracy. In the second step, we use pairwise comparisons to match tracklets detected from multiple cameras. In each pairwise comparison, we compute the difference between two tracklets' feature vectors (detected by different workers). If the difference between two tracklets is less than a threshold, these two tracklets will be grouped as a single tracklet.

License Plate Recognition and Matching

License plate is the most important information in the vehicle tracking application since it can uniquely identify the target vehicle. We implement our license plate recognition model based on an existing automatic license plate recognition system [20]. Most existing recognition models are trained using pictures taken at a fixed location, so the license plate numbers are large and clear enough to see. However, it is very hard to correctly recognize the license plate if the vehicle is far away. As shown in Figure 2.6, license plate recognition accuracies are highly impacted

by the distance (Figure 2.6 (3), (6)) and angle (Figure 2.6 (2)). To improve the accuracy, we trained the model using our own license plate dataset and existing open license plate datasets [21]. Our dataset includes license plated captured from different distances and angles (as shown in Figure 2.6). The license plate recognition module includes two neural networks, a license plate detection neural network and an optical character recognition (OCR) neural network for character recognition. During the training process, the license plate neural network learns how to "unwarp" the distorted license plate into a rectangular shape resembling a frontal view during the training process. The OCR neural network is designed based on the YOLO network to recognize each character from the unwrapped license plate.

Figure 2.6 shows license plates detected by different workers at different locations. Due to light conditions, camera angles and so on, the license plate recognition model produces different results on the same vehicle across different frames. If the result matches the provided query, the worker directly reports to the query source about this finding (no need to perform feature fusion on the manager). If there are several incorrect or missing characters, the worker keeps track of that vehicle until it can be ruled out as a suspect. Therefore, the problem becomes the longest common substring with k mismatch problem. We solve the substring matching problem using approaches presented in [22, 23]. If a detected license plate has less than k mismatches, we treat it as a target vehicle and send it to the query source.

Vehicle Feature Extraction

Vehicle Model and Make We implement a vehicle model and make classifier based on Spectrico's recognition model [24]. Spectrico's classifier is based on the MobileNet neural network architecture [25], which is known for smaller model size and good detection accuracy. It is able to process input images in real time. For instance, it takes 25 ms on Intel Core i5-7600 CPU and can be even faster with a more powerful processing unit. Detected vehicles are cropped from the original frame and resized to match the input requirements (224*224 pixels images) of the recognition model. Padded square images will be sent to the classifier for inference.

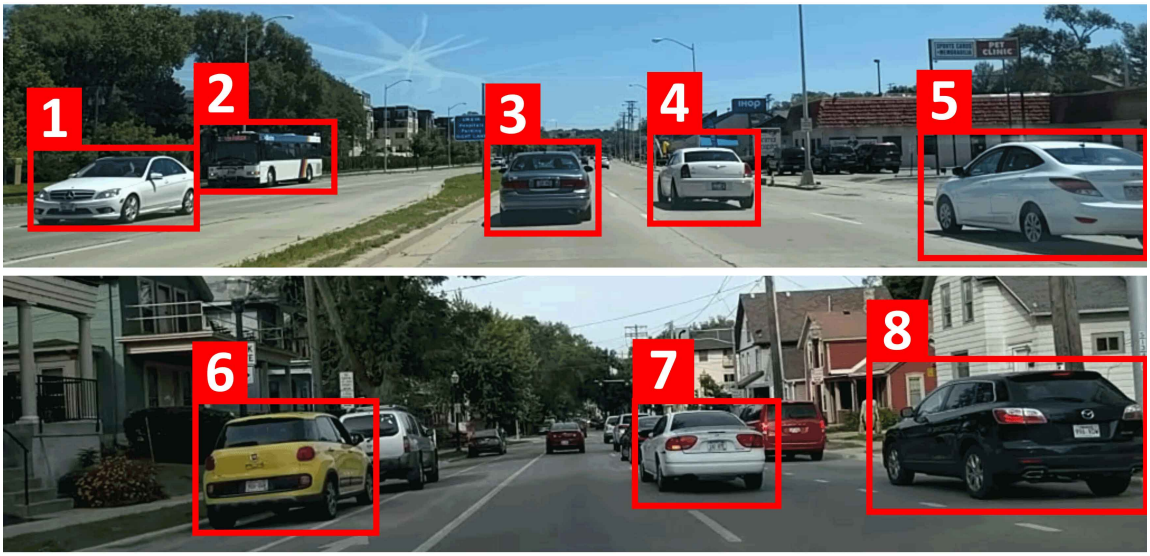


Figure 2.7: Vehicle information extraction.

Example inference results can be found in Figure 2.7 and Table 2.1.

Vehicle color We pass each detected vehicle to a classifier for color identification. The classifier is designed to classify 8 colors, including black, blue, gray, red, orange, green, white, and yellow. Each vehicle is represented using the Red, Green, and Blue (RGB) color histogram. We use the K nearest neighbors (KNN) algorithm to classify colors based on the extracted RGB color histogram features. By applying a KNN classifier on RGB color features, we would be able to preserve majority RGB values and identify the dominant color of an object. We label the color of each vehicle in our training data set. The training data set contains images of vehicles taken under different light conditions. Figure 2.7 and Table 2.1 show examples of detected vehicles with different colors.

Feature Fusion Model

When multiple workers capture the same vehicle, feature vectors generated from the workers can be fused to create a new feature vector to represent that vehicle with an improved confidence level. We implement the feature fusion model based on

Table 2.1: Object Detection Results

No.	Color	Make	Model
1	White	Mercedes-Benz	C-Class
2	White	Bus	-
3	Grey	Buick	LeSabre
4	White	Chrysler	300c
5	White	Hyundai	Solaris
6	Yellow	FIAT	500
7	White	Oldmobile	Alero
8	Black	Mazda	CX-9

the Dempster-Shafer theory (DST) [26] [27]. We use a probability vector with the size equals to the number of possible values of a feature, and each bit of the vector representing the confidence level (or probability)[2] of each value. The value with the highest confidence level is selected as the predicted value of a feature. Therefore, $p_c(A_i^j)$ denotes the probability of value j for feature i predicted by worker c. For instance, there are eight possible values of the color feature, sequenced by black, blue, gray, red, orange, green, white and yellow. $p_c(A_1)$ is an 8-dimensional color feature vector derived from worker c, consisting of $\{p_c(A_1^1), p_c(A_1^2), \ldots, p_c(A_1^8)\}$. Thus $p_1(A_1^1)$ represents the probability of a vehicle captured by worker 1 is classified as black. In total, there are three features considered by the manager for feature fusion, including vehicle color, make and model. According to the DST, the fused probability of ith feature in value r is:

$$p(A_i^{j=r}) = \frac{p_1(A_i^{j=r}) \cdot p_2(A_i^{j=r}) \cdot p_3(A_i^{j=r}) \ldots p_n(A_i^{j=r})}{\sum_{j=1,\ldots,k} p_1(A_i^j) \cdot p_2(A_i^j) \cdot p_3(A_i^j) \ldots p_n(A_i^j)} \tag{2.1}$$

If one of the values is not discovered by workers, a small value (< 0.1) will be assigned to the corresponding bit in the feature vector. Suppose two workers provide two color feature vectors on the same vehicle $\{p_1(A_1), p_2(A_1)\}$. If we want

[2]Confidence level is equivalent to probability and we use them interchangeably in this paper.

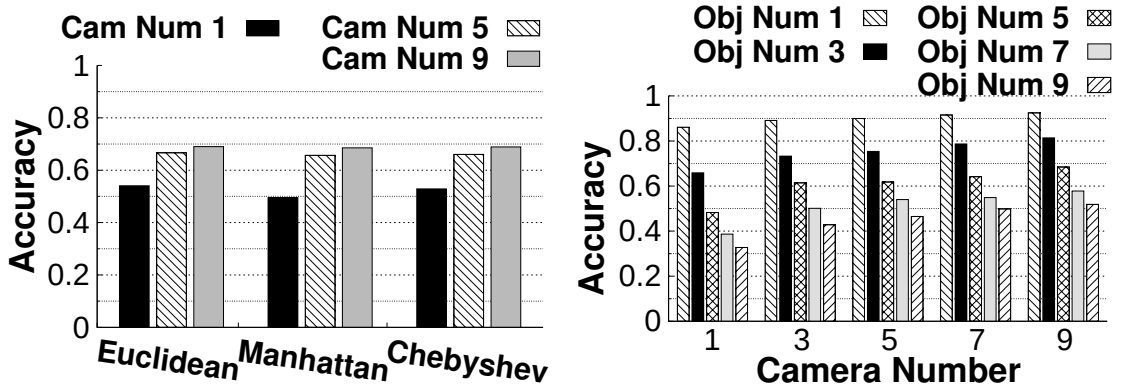
to know the fused probability of black, then $p(A_1^{j=1})$ can be calculated as follows:

$$p(A_1^{j=1}) = \frac{p_1(A_1^1) \cdot p_2(A_1^1)}{\sum_{j=1,\ldots,8} p_1(A_1^j) \cdot p_2(A_1^j)}$$
$$= \frac{p_1(A_1^1) \cdot p_2(A_1^1)}{p_1(A_1^1) \cdot p_2(A_1^1) + \ldots + p_1(A_1^8) \cdot p_2(A_1^8)}$$

Generally, the probability of feature value A_i^j is determined by inputs from multiple workers. The numerator of $p(A_i^j)$ can be treated as the shared belief across multiple workers in A_i^j. And the denominator of $p(A_i^j)$ is a measure of the amount of conflict between the workers. When the conflict is extremely large, the fused probability would be close to zero. While if there is no conflict, the denominator of $p(A_i^j)$ is 1, thus the fused probability only depends on the numerator. The denominator is between 1 to infinite and it can be used as the normalization factor.

Implementation

We implement a prototype of the vehicle tracking application to evaluate the feasibility of our design. We use the camera and motion sensors as the sensing devices. We develop an application for IMU sensors and GPS data collection as well as video capture. The application detects different driving events and leverages the TensorFlow Lite for vehicle detection. Based on detection results, the application streams encoded data to the manager. The edge compute node at stage two is built upon a previously developed edge computing platform called ParaDrop [5] and its extension EdgeEye [28]. The ParaDrop platform is built based on Docker [6], which provides an isolated environment allowing developers to flexibly create various kinds of services. We use multiple GStreamer pipelines to run analysis tasks and handle interactions among workers, managers and the query source. Two pipelines are developed to receive and send data. We implement three GStreamer elements for data analysis (vehicle information, license plate, and re-identification) and create GStreamer pipelines to manage various tasks. The GStreamer elements



(a) Prediction accuracies using three distance metrics.

(b) Prediction accuracies with different worker and vehicles.

Figure 2.8: Performance under different settings.

and pipelines were developed using C and C++. Neutral networks models are trained using a desktop with Ubuntu 18.04 LTS. The desktop has an Intel i7-6700 CPU @ 3.40GHz, an Nvidia GeForce GTX 1060 6GB, and 24GB system RAM.

2.5 Evaluation

In this section, we evaluate our application and system. The first part evaluates application performance, and the second part evaluates system performance.

Application Performance Evaluation

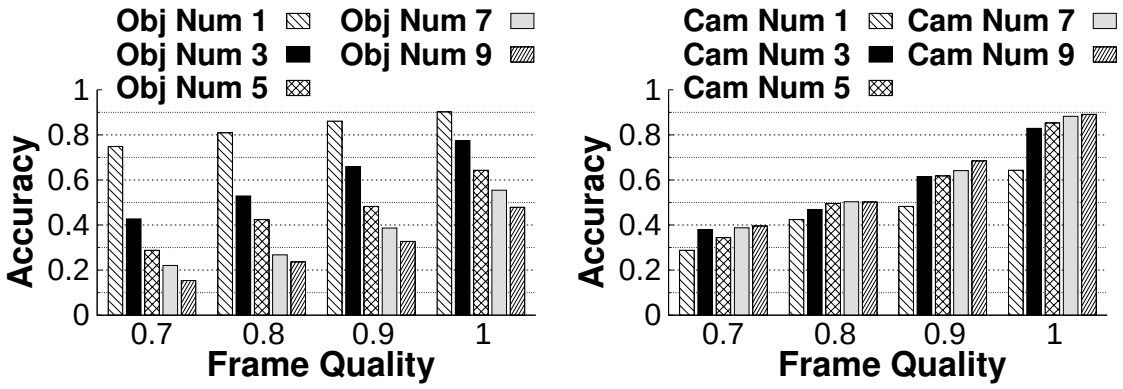
Methodology

We deploy our system in multiple vehicles and utilize video while driving simultaneously on the road in real-world conditions. We were only able to gather around 50 miles of driving data[3]. We conduct various experiments using observations derived from collected real-world driving data and a publicly available data set [29]. We

[3]Unfortunately, our data collection plan is interrupted due to the on-going global pandemic.

explore how different factors affect the target identification performance. We control four different parameters during the experiment process, they are numbers of available workers and vehicles in each frame, frame quality and the similarity level between the target vehicle and non-target vehicles. For each experiment run, we first generate a query, such as white Mazda CX-9. Next, we generate a feature vector for the target vehicle, which is the vehicle described in the query. A 28-dimensional combined feature vector, which consists of an 8-dimensional color feature vector, 10-dimensional vehicle make and 10-dimensional vehicle model vectors, is used to represent the target vehicle. Each dimension represents the confidence level of the corresponding feature value. If a value is selected as the predicted value of a feature, a confidence level greater than 0.5 will be assigned to the corresponding dimension. Other values' confidence levels will be assigned randomly (the summation of all the confidence levels of a feature is 1). We also consider the effects of frame qualities when generating vehicle feature vectors. Therefore, the generated feature vector's representative value could be partially inconsistent with the query, or the selected value has a low confidence level (less than 0.5). Next, we generate feature vectors for other vehicles on the road. We generate a random number to represent the similarity level between non-target vehicles and the target vehicle. Based on the similarity level, we change one or multiple feature values for non-target vehicles. Lastly, we randomly generate two numbers (between 1 and 10), representing how many non-target vehicles and available workers in this experiment run. Feature vectors of the same vehicle provided by different workers will be fused by the DST model to produce one feature vector representing the prediction of that vehicle.

We compute the distance between the fused feature vectors and the vector query issued using three common distance metric — Euclidean Distance (L2 norm), Manhattan Distance (L1 Norm) and Chebyshev Distance (L-infinity Norm). The vehicle that has the smallest distance would be considered as the vehicle we want to find. In total, we generate 20,000 experiment runs. If the target is successfully found, we treat this run as a successful run. The prediction accuracy is counted using the number of successful runs divided by the total number of runs. We evaluate the application performances under different settings. Details are discussed in the



(a) One worker. (b) Five vehicles.

Figure 2.9: Performance under different frame qualities.

following sections.

Overall Performance

We first study prediction accuracies when using different distance metrics and results are shown in Figure 2.10. The number of vehicles is randomly selected between 1 and 10 for each experiment scenario. The frame quality is set to good and the similarity level is randomly selected (between 0 and 0.8). According to the results, three distance metrics show similar performances, thus proving that our experimental results are robust enough. We can achieve an average accuracy of over 89%, which is reasonable given the fact that we randomly choose vehicle numbers and similarity levels. The Euclidean distance metric shows a slightly better overall performance, so we use this distance metric afterward. We next study how different numbers of vehicles and workers affect prediction accuracies. As shown in Figure 2.8b, when the number of workers is fixed, the prediction accuracy decreases as the number of vehicles increases. It is hard to find out the target vehicle when more and more non-target vehicles on the road. While the number of vehicles is very larger (always happen in the traffic peak hours), five workers can have a relatively high accuracy. Moreover, the prediction accuracy increases with more available workers when the vehicle number is fixed.

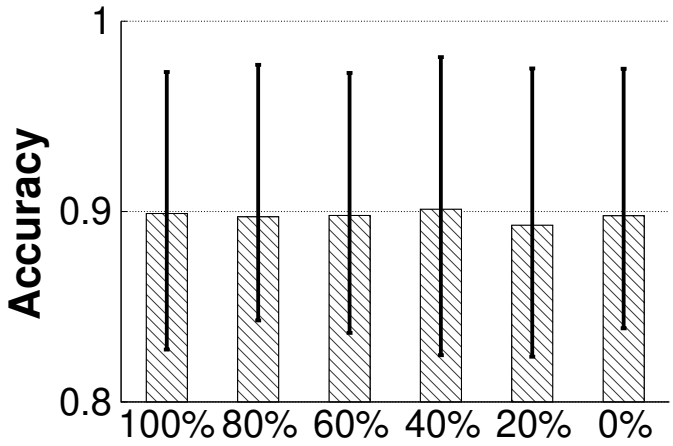
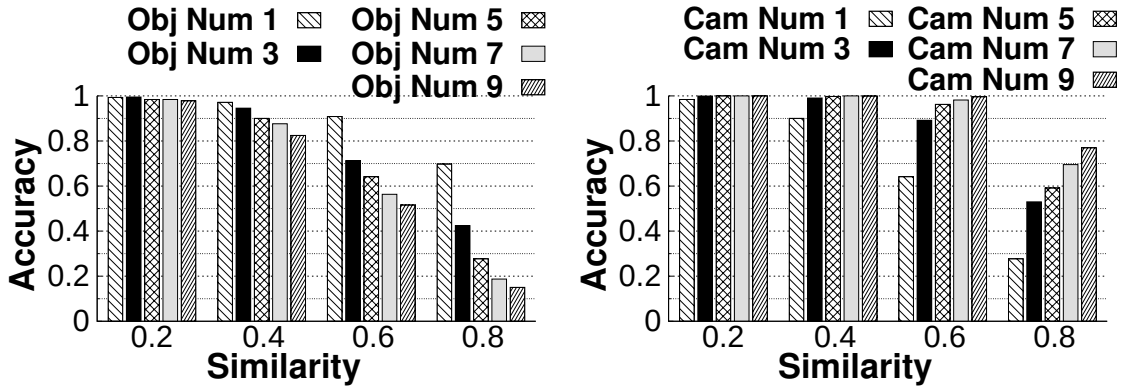


Figure 2.10: Vehicle tracking application overview.

In 20,000 experiment runs, we also consider the situation where the target has not been captured by any workers in one service area. Specifically, we consider an experiment run is a true positive if the target is captured and successfully found. Similarly, we consider the experiment run is a true negative if the target is not captured and no vehicle has been predicted as the target. In Figure 2.10, the x-axis represents the percentage that the target vehicle has been captured (e.g., 80% means 80% of 20,000 experiment runs have the target vehicle captured). Concluded from figure 2.10, the accuracy would not change much if we set a reasonable threshold as discussed before, which indicates our application is able to achieve a good performance under different situations.

Comparison between Different Frame Quality

Aqqa *et al* conducted a detailed study evaluating the impact of video quality changes on the detection performance of 4 state-of-the-art object detectors [30]. Evaluation results show that as the quality decreases, the detection accuracy decreases. Based on these findings, we conduct different frame qualities experiments by changing predicted feature values or decreasing the confidence level of predicted features. As shown in Figure 2.9a, when only one worker is recording, as the frame quality



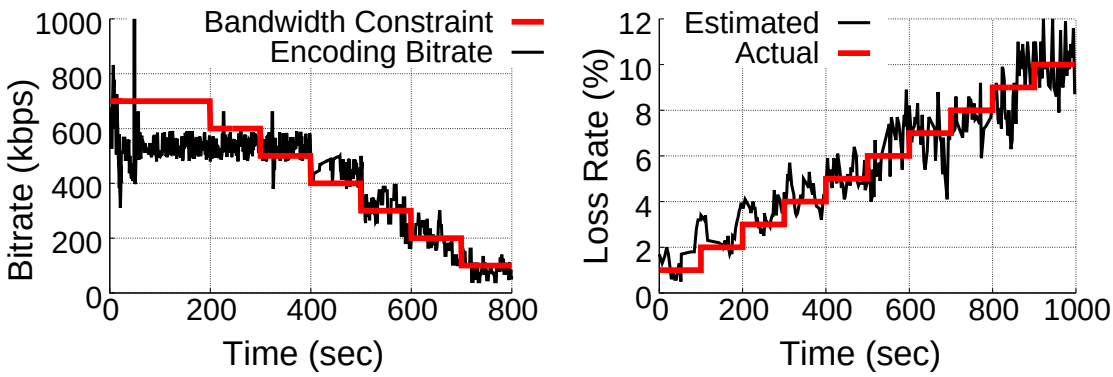
(a) One worker. (b) Five vehicles.

Figure 2.11: Performance under different similarities.

gets better, the prediction accuracy increases. The average accuracy is below 50% when the quality is poor and increases to more than 65% when the quality is good. Similarly, when there is a fixed number of vehicles on the road (we choose 5 in this case), as the frame quality gets better, the prediction accuracy increases (Figure 2.9b). The average accuracy increases from below 40% to above 80% as the frame quality gets better.

Comparison between Different Similarity

Besides frame quality, the similarity of non-target vehicles with the target vehicle can also affect the application performance. To understand its impact, we change similarity levels by controlling how much the non-target vehicle's feature vector is different from that of the target vehicle in different experiments. As shown in Figure 2.11a, when only one worker is working, as the similarity gets larger, the average prediction accuracy decreases from more than 95% to less than 40%. A higher similarity means one or more non-target vehicles share more similar features with the target vehicle, which makes the target vehicle harder to be found. We achieve an average accuracy of 90% when the similarity level is less than 0.5. While the accuracy drops sharply when similarity level changes from 0.6 to 0.8. Similarly, when there is a fixed number of vehicles on the road (we choose 5 in this case), as



(a) Bandwidth estimations.

(b) Loss rate estimation.

Figure 2.12: Network parameter estimation accuracy.

the similarity gets larger, the prediction accuracy decreases (Figure 2.11b).

Network Parameter Estimation

We first evaluate how our streaming protocol performs under different network conditions. We use the NetEm [31] to control loss rate, bandwidth and other parameters of an indoor Wi-Fi network. In this experiment, we change bandwidth over the Ethernet port of the server. A wireless router connects to the server through a cable. Our network parameter estimation algorithm is implemented on a customized Android app. The app streams a video via the wireless router to the server[4]. We decrease the bandwidth constraints from 700kbps to 100kbps and increase the loss rate from 1% to 10%. The Android app detects changes and adapts to the changes accordingly. As shown in Figure 2.12a, we can see that the app tries to find an optimal bitrate in the beginning and eventually decreases as the bandwidth decreases. In Figure 2.12b, as the loss rate increases 1% every 100s, our estimation protocol can keep track of the changes closely.

[4] In this experiment, we treat the Android phone as the roaming edge node and the server serves as the static edge node.

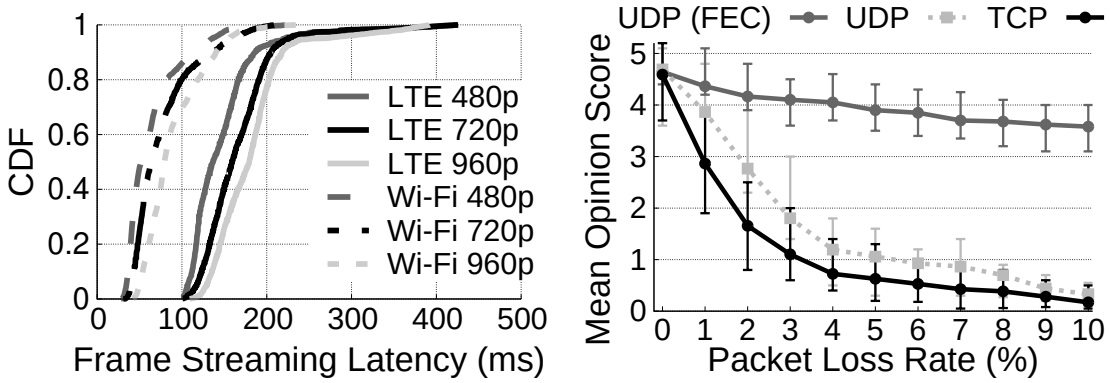
Data Streaming Services

Experiment Setup

We implement our context-aware streaming protocol on the Android platform. The app compresses raw video frames (in YUV420 format) using the H.264 standard, send them along with motion sensor data to a remote server in real-time[4].A GStreamer pipeline is running on the server to decompress each video frame and sending back the timestamp associated with the frame. We set up a UDP connection between the phone and the server to send encoded frames over both LTE and Wi-Fi networks, and measure the round trip latency of the encoded video frames transmission in both networks.

Real Time Streaming Latency

We conduct case studies with three resolutions of 640x480 (480p), 960x720 (720p), and 1280x960 (960p), with bitrate 1Mbps, 2Mbps, and 4Mbps, respectively. The I-frame interval is set to 1 second, and the frame rate is set to 10, so there will be one I-frame every 10 frames. We set up a UDP server with a global IP address in our lab served as the remote server when conducting measurements for both networks. For the measurements under LTE network, we put the Android phone on a vehicle dashboard and drive the vehicle for about 30 minutes to record measurement results. The recorded trip data cover an area of 1.5 square miles. For Wi-Fi connection measurements, we stream a pre-recorded video through an 802.11n Wi-Fi access point using the Android app. The streaming location is located approximately 5 miles away from where the UDP server is located. The cumulative distribution function (CDF) of round trip latencies under both networks are shown in Figure 2.13a. The median latency under Wi-Fi network is around 50 ms while that under LTE network is around 150ms. Since only one Wi-Fi access point is used in this experiment, the difference in latency is mainly caused by the protocol overhead, for instance, handoff, scheduling. Based on these results, we believe that our proposed streaming service is good for most applications. With the development of



(a) Frame streaming latency under different networks.

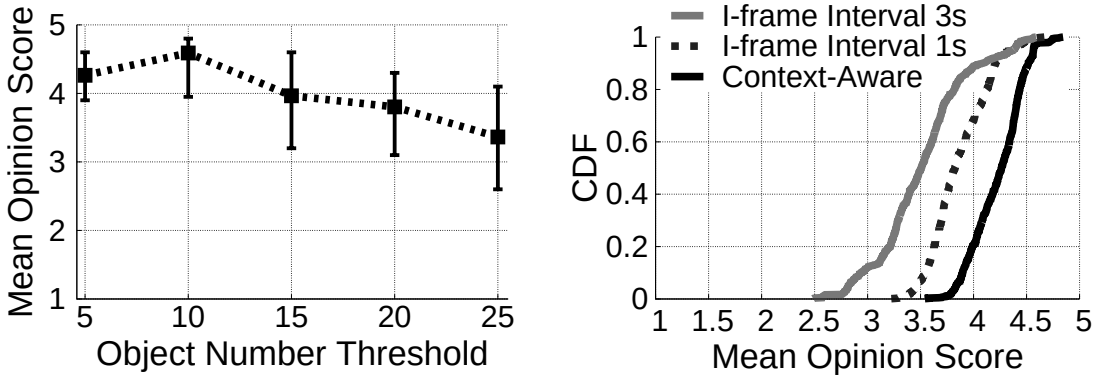
(b) Frame quality under different loss rates.

Figure 2.13: Streaming performance evaluation.

5G networks and carefully designed city-wide Wi-Fi networks [32], the streaming speed could enable more time-sensitive applications.

Real Time Streaming Quality

In this experiment, we evaluate the quality of received video transmitted under different loss rates using UDP and TCP, with or without FEC. Videos are streamed from an Android phone to a server. The server is set up on a laptop and connects to a wireless router through an Ethernet cable. The phone connects to the same router through Wi-Fi. Figure 2.13b shows that video encoding algorithms are very sensitive to packet loss. When the loss rate is greater than 2%, most frames are distorted if there are no error correction mechanisms involved. The reason is that if one packet of an I-frame gets lost, the current I-frame's quality and the qualities of the following P-frames will all be affected. The frame decoder cannot recover those frames correctly. Due to the missing packet of I-frame, errors will propagate over the following P-frames. Our UDP with FEC protocol works well even when the loss rate is above 5% (Figure 2.13b).



(a) Frame quality under different parameter settings.

(b) Frame quality using different encoding methods.

Figure 2.14: Context-aware streaming evaluation.

Context-Aware Data Encoding

We evaluate the performance of the context-aware encoding algorithm in a lab environment to avoid any other network interferences. An Android phone and a laptop running a server connect to the same Wi-Fi network. Recorded raw driving data includes different driving events and scenes are played on the Android phone. Once the server receives the data, it converts it back and evaluates the quality. The object number threshold controls how frequent the encoder produces an I-frame. The algorithm chooses different I-frame intervals based on the number of detected objects. A larger threshold produces fewer I-frames and a smaller threshold produces fewer P-frames. As shown in Figure 2.14a, quality decreases as the threshold increases. When the I-frame interval is relatively large, P-frames suffer bad qualities if the scene changes frequently, so the overall quality goes down. Next, we compare the context-aware encoding algorithm with the H.264 video encoding standard using fixed I-frame intervals. Figure 2.14b shows that a smaller I-frame interval leads to better quality, as more I-frames are sent to the server. However, more I-frames mean more data needs to be transmitted. The context-aware encoding method is able to decide when to send I-frame based on potential frame changes, which lead to better overall performance and less bandwidth usage

Table 2.2: System Processing Delay Breakdown

		480p	720p	960p
Stage One	Object Detector	35ms	38ms	42ms
	Color Classifier	12ms	13ms	15ms
Communication (Encoding, streaming, decoding)		65-240ms	75-260ms	90-285ms
Stage Two	ReID	35ms	45ms	50ms
	License Plate Recognition	11ms	15ms	18ms
	Vehicle Feature	21ms	25ms	28ms
	Feature Fusion	10ms	10ms	10ms
End to End		1.5-3.5s	2-4s	2.5-4.5s

than using other encoding methods.

End to End Latency

The overall end to end latency consists of latencies at different stages and the streaming latency. Latencies of object detector, color classifier and context-aware encoding are measured on mobile phones at stage one. At stage two, we measure re-identification, license plate recognition, vehicle feature extraction and feature fusion latencies. The data streaming latency is studied in Section 2.5. We record the timestamps when a module is called and an output is derived to calculate latency. The streaming latency is measured using timestamps embedded in packet headers. Table 2.2 shows the latency of each module and the end to end latency. On average it takes up to 4 seconds to identify the target vehicle. Vehicle reidentification is the most time-consuming task since it has to compare tracklets one by one.

We believe that it is reasonable to have a less than 4 seconds delay for vehicle tracking. Once a worker finds a suspect, it will keep tracking that suspect until

cleared. Hence, we can know the current location of that vehicle as soon as it is being identified as the target. The system processing delay can be improved from multiple aspects, e.g., using more powerful machines, better networks (5G). We leave these as future work.

2.6 Discussion

Hardware and Edge Computing Platform We used a previously developed edge computing platform - ParaDrop, which is built on WiFi access points. There are many other research efforts have been done on edge computing. For example, Mobile Edge Computing (MEC) [33] is a popular concept in industries. Jetson TX2 is a low-cost, low power consumption embedded system with GPU capabilities, which is suitable for complex computing at the edge. Integrating Cascade with other platforms would benefit both developers and crowdsensing applications.

Data Collection and Transmission Our vehicle tracking application mainly uses videos recorded using smartphones. Other cameras, like traffic cameras, can also be involved as a worker in this application. It can record videos from a different angle, which might be able to capture other important features of a vehicle. Cascade is able to accommodate various types of sensors and we will continue extending its capabilities. Further improvements can be made to improve data streaming efficiency. With the emerging 5G technologies, crowdsensing applications could have new paradigms for data transmission. Therefore, how crowdsensing can benefit from 5G networks can be an interesting research direction.

Vehicle Tracking In this work, we train existing models using our own dataset for vehicle information extraction. There are lots of improvements can be done on vehicle information extraction. For example, vehicle model, make and color can be respectively recognized using a single CNN model [34], and such information can be further used in vehicle re-identification [35]. Our color detector can only work with vehicles that have a single color. Therefore more work can be done to handle vehicles with multiple colors. We leave these as future work.

2.7 Conclusion

In this paper, we explore the feasibility of designing a two-stage crowdsensing framework with the help of roaming edge computing platforms. We aim to gather transportation related information at city-scale using crowd-sourced, distributed roaming edge compute nodes and their in-range sensors. To this end, we present Cascade, a roaming edge computing enhanced crowdsensing framework for smart transportation applications. We implemented a vehicle tracking application based on Cascade to explore and overcome challenges of building applications leveraging the roaming edge. During the development process, we demonstrate how to fully leverage resources at both static and roaming edge nodes to improve overall system performances. We evaluated this application using real-world driving data as well as publicly available data sets. The experiment results help us understand how to develop the best resource allocation strategy to achieve given goals.

3 DrivAid: Augmenting Driving Analytics with Multi-Modal Information

3.1 Introduction

Due to its many implications, monitoring and evaluating driver's behavior has always been a topic of great interest. Many commercial offerings (such as from Cambridge Mobile Telematics [36]) and research efforts have built systems that attempt to monitor and evaluate driving behaviors by analyzing information from On-Board Diagnostic (OBD) ports as well as motion sensor data (either from smartphones or other custom devices with Inertial Measurement Units (IMU) sensors). With the proper analysis, the data from these motion sensors can provide useful information about various activities within the drive itself, e.g., hard brakes, sudden lane changes, or fast acceleration [37, 38, 39, 40].

While IMU sensors can provide accurate analytics on *what* happened during a drive, e.g., hard brakes, sudden lane changes, etc., it should be apparent that such data alone does not answer *why* the driver acted in that manner. In particular, data from IMU sensors do not have sufficient contextual information to indicate whether such an action by the driver was good or bad. As shown in Figure 3.1, a hard brake could be due to driver distraction (a bad driving action), or it could be to avoid a deer that suddenly jumped to the front of the vehicle (a good driving action).

Introducing DrivAid: We believe that to better understand driving behaviors

Figure 3.1: Possible causes of a hard brake.

one can effectively leverage audio-visual cues, using a few vehicle-mounted cameras and microphones, to complement the existing use of IMU sensors. In particular, we build a low-cost and portable system called DrivAid that utilizes audio-visual sensors in the vehicle, processes such data feed locally and provides useful evaluation feedbacks to the driver both in real-time and offline.

Analyzing high resolution audio-visual data is often delegated to high-end GPU-enhanced compute clusters located in data centers. However, in our scenario, it is likely that the vehicles equipped with audio-visual sensors can easily generate a high volume of data rather quickly making the offload process either slow or expensive, or both. This implies that the audio-visual analytics should need to be performed locally and in real-time — if the analytics is any slower, then the data will continuously get backlogged and ultimately becoming stale at some point.

To address abovementioned challenges and support efficient audio-visual analytics, we experimented with in-vehicle GPU-enhanced computing platforms on which we deployed the analytics module. Compared to cloud computing, edge computing provides lower latency, greater responsiveness, and more efficient use of network bandwidth. *To avoid unnecessary resource-intensive image processing tasks and reduce computing workload of the whole system, we use smartphone motion sensors to detect different driving events and only conduct further analysis once an event is detected.* Figure 3.2 shows the workflow of DrivAid. DrivAid uses data from IMU and GPS sensors to detect driving events. When an event is detected, DrivAid fetches video and audio clips from the buffers to analyze. All the audio and video data are

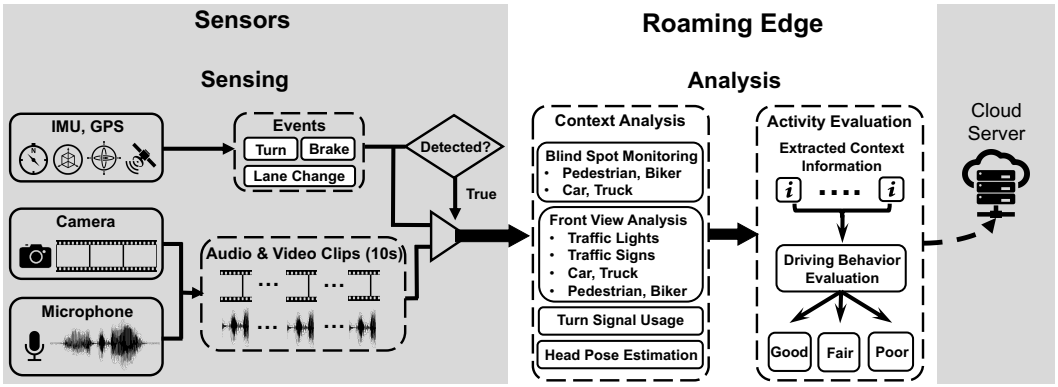


Figure 3.2: A high level overview of DrivAid.

processed locally in the vehicle to preserve privacy. By leveraging various sensor data and implementing an analysis pipeline that leverages deep neural networks, we are able to offer useful insights of driving events and generate analysis results in real time. We believe the proposed architecture of DrivAid can be particularly useful in analyzing driver behaviors, and could be an essential part of a driver analytics subsystem.

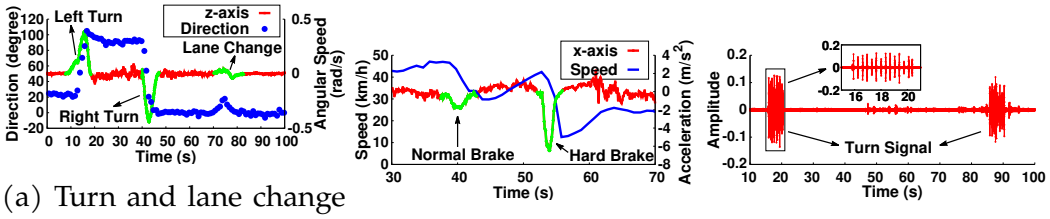
Usefulness of DrivAid: The concept of using audio-visual cues in improved situational awareness around vehicles is most famously used in autonomous vehicle industries (along with a host of other sensors). However, unlike the high resolution and very high accuracy needs of autonomous vehicles in making driving decisions, the DrivAid system requires far lower resolution and accuracy, resulting in a significantly lower cost. Today's autonomous vehicles install high-end sensors which aggregate a price of ~$100,000, while our system utilizes infrastructure with an aggregate cost of less than $500. DrivAid delivers significant benefit for driving analytics at a lower cost which could benefit a wide range of applications. First, with proper incentives, insurance companies may ask their customers to install extra hardware (e.g. camera) on their vehicle, and evaluate their driving skills with a system like DrivAid. Such a camera-based system could offer more information when an accident happens. Second, many fleet operators, e.g., public transit systems, school buses, freight trucks, are often interested in understanding the

behaviors of their drivers. DrivAid could identify unsafe habits and help fleet managers educate their drivers to take appropriate actions. Third, if certain drivers are known to be "experts," then the annotated actions of such drivers could also be used to provide useful inputs to the design of autonomous vehicles. Moreover, with adjustments, DrivAid could be used to verify whether the autonomous vehicle can handle unusual or rare traffic events.

3.2 System Overview

Design Consideration

In order to get a comprehensive understanding of a driver's performance, a major challenge is to acquire enough context information, e.g. driver's actions, and the surrounding traffic information. Current solutions, like XSense [41], V-Sense [37], can accurately detect various normal and abnormal events using data collected from motion sensors or OBD port, but fail to provide surrounding context information. To find the missing information, DrivAid buffers the most recent 10 seconds audio and video data from microphones and cameras for analysis. When an event of interest happens, DrivAid analyzes the data with computer vision techniques to get driving actions that cannot be detected by IMU and GPS sensors, including turn signal usage, mirror checking, blind spot checking, and so on. By combining information from various sources, DrivAid can provide more informative driver analytics and offer deeper insights into each detected events.



(a) Turn and lane change detection.

(b) Brake detection.

(c) Turn signal detection.

Figure 3.3: The detection of driving activities.

Our Solution

Figure 3.2 shows a high-level overview of the DrivAid design. DrivAid uses three types of sensors to collect data: i) IMU sensors and GPS of a phone, which can be used to detect driving events; ii) the rear camera of a phone to capture traffic ahead; iii) two cameras facing the blind spots on both sides, and one camera facing the driver. Timestamped sensor data, audio, and video are analyzed on an embedded computer deployed in the vehicle. We use an Android phone to record data from the accelerometer, gyroscope, magnetometer and stream them to the embedded computer for context analysis. Extracted context information is sent to an activity evaluation module, through which the detected event will be evaluated and categorized into different ratings.

IMU-based solutions can accurately detect various types of driving events with low power consumption. On the other hand, audio and video data analysis with deep neural networks are usually computationally expensive. Given the fact that a driver should have minimal operations when driving straight or there is not too much surrounding traffic. Hence, it is less important to monitor the driver during the whole trip, we only need to focus on how well the driver performs when a driving activity is detected. Following this principle, DrivAid uses motion sensors to continuously monitor various driving events, and analyze corresponding audio and video clips once an event is detected. Doing this can significantly reduce computing workload and power consumption of the whole system.

3.3 Event Detection and Context Analysis

The DrivAid system detects three types of driving activities (turn, hard brake, and lane change) with the sensors on the smartphone. Once a driving activity is detected, DrivAid starts extracting context information from the video and audio clips collected by the IP cameras. Data from different sensors are not synchronized due to buffering (video and audio) and different system clocks on those devices. We leverage the recorded turn signal sounds to synchronize the data from multiple

devices. A pattern matching algorithms is used to find the time difference between multiple signal traces.

Driving Event Detection

Coordinate Alignment

Smartphone-based mechanisms have been developed to detect various driving behaviors [42, 40, 37]. For instance, the vehicle's lateral dynamics can be obtained from the gyroscope sensor in the smartphone when a phone is aligned with the vehicle. DrivAid uses similar techniques as mentioned in [37, 41] for coordinate alignment.

Turn and Lane Change Detection

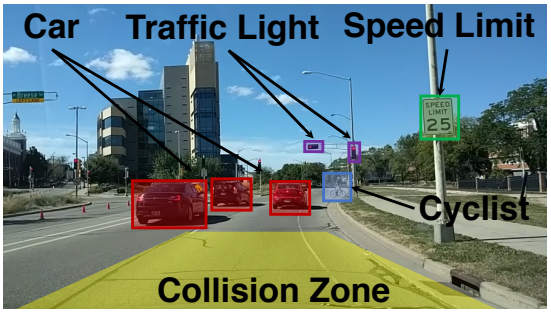
We detect turning and lane change events by using the gyroscope sensor and the GPS. Since the smartphone's coordinates are aligned with that of the car, the z-axis readings of the gyroscope sensor reflect lateral movements of the vehicle. Figure 3.3(a) illustrates the detection of left and right turns as well as a left lane change using the gyroscope sensor. Turning angles can be estimated by integrating the angular speed over time. A positive turning angle refers to a left turn while a negative turning angle represents a right turn. In addition to the gyroscope data, we monitor the vehicle heading changes with GPS coordinates. The vehicle turning events can be extracted by monitoring the changes in the traveling direction. This method could be a complementary method for turning detection. In Figure 3.3(a), the blue dots represent the traveling directions calculated from the GPS coordinates. Different angles stand for different directions. For instance, 0 corresponds to East, 90 indicates North, 180 denotes West, -90 represents South. From the figure, we can distinguish the turning events from the changes of direction angle. Using similar techniques, we can detect lane change events using both device orientation changes and direction angle changes. To differentiate between lane changes and driving on curvy roads, we implemented the same technique proposed in V-Sense [37].

Brake & Stop Detection

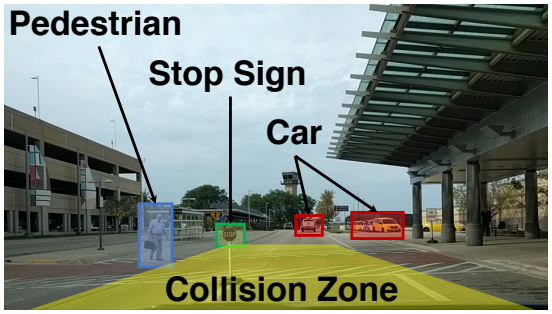
We use accelerometer readings as well as vehicle speed information to detect brakes and stops. Figure 3.3(b) illustrates the detection of normal and hard brakes. Brake events can be extracted from bumps in accelerometer readings and decreasing in vehicle speed. A hard brake is defined as any condition when the vehicle decelerates faster than 7 mph (Miles per hour) per second. Stop events can be inferred from the vehicle speed.

Driver Behavior and Context Analysis

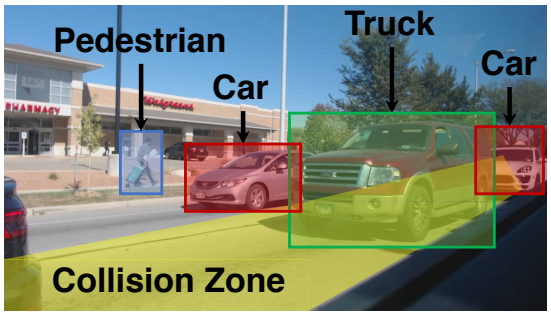
DrivAid analyzes the audio and video signal with signal processing and computer vision techniques to get driver's behaviors and also the context information.



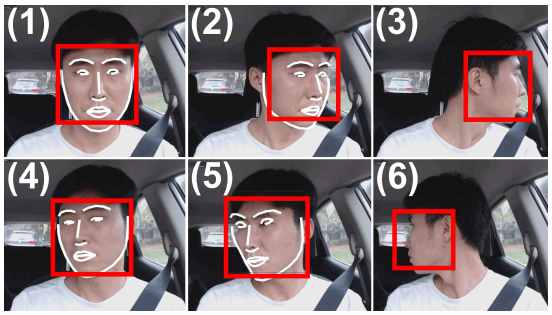
(a) Objects in front view.



(b) Objects in front view.



(c) Objects in blind spots.



(d) Examples of head poses.

Figure 3.4: Extracting context information using vision-based techniques.

Driver Behavior Analysis

The turn signal is a vital safety feature used to notify other drivers when making turns or switching lanes. DrivAid detects turn signal usages by analyzing turn signal sounds. We apply bandpass filters to filter out human speech as well as other background noises. Then a pattern matching technique is used to identify turn signals. Figure 3.3(c) shows examples of extracted turn signals from a camera's built-in microphone.

Drivers use mirrors to get a picture of their surrounding traffic, and check the blind spots when they need to make a turn or change the lane. DrivAid detects a driver's head pose to infer the driver's mirror and blind spot checking behaviors. We classify head poses into six categories, they are left and right wing mirror check, left and right blind spot check, rearview mirror check, and front view check. Figure 3.4d shows six different head poses of a driver. In Figure 3.4d (1), the driver is focusing on the front view. Figure 3.4d (2) and (5) show the driver is checking left and right wing mirrors correspondingly. The driver is checking the rearview mirror in Figure 3.4d (4). The driver is making left and right shoulder checks in Figure 3.4d (3) and (6). These six scenarios can be distinguished by checking head movements with respects to the yaw axis. The estimated rotation angles around the yaw axis of Figure 3.4d (1), (2), (4), (5) are -3.34, -40.20, 18.21, 49.70. As shown in Figure 3.4d (3) and (6), due to lack of facial features, it is hard to estimate head pose when the driver is checking right and left blind spots. To solve this issue, we track the rotation angle in a duration. If the angle keeps increasing and then disappears, then the driver is doing a right blind spot check, and vice versa. To calculate the head pose in 3D space, we need to know several locations of the face (eyes, nose, mouth, etc.) in 2D space, and also the 3D coordinates of those facial features in world coordinates. We assume the camera is calibrated so we are aware of the intrinsic parameters of the camera. DrivAid uses Dlib's facial landmark detector [43] to extract key points on a face from the captured image. And the solvePnP function of OpenCV [44] is used to get the head pose.

Table 3.1: Object Detection

Vehicle & Person	Traffic Sign	Speed Limit
Car	Stop Sign	25 mph
Truck	Red Traffic Light	30 mph
Pedestrian	Yellow Traffic Light	40 mph
Biker	Green Traffic Light	60 mph

Context Analysis

We analyze the audio and video signal to get the context of driving events. Object detection is the backbone of DrivAid and the choice of a right detection model should balance the run-time efficiency and accuracy. We choose DetectNet [14] as the object detection model since it can deal with input image of varying sizes and provide reliable accuracies. And it can be easily accelerated by the optimized inference engine—Nvidia TensorRT [15].

Most intersections are controlled by traffic signs, and it is important to follow the traffic rules at an intersection. DrivAid can detect a list of objects appeared in video frames captured in the front view camera, with a special focus on common objects at intersections. The front view object detection model could recognize vehicles, pedestrians, traffic lights, stop signs and speed limits. We train a DetectNet model using two public traffic image datasets [17, 16]. Figure 3.4a and 3.4b show the detected objects at intersections with traffic lights and stop signs. We define a collision zone (yellow shaded region shown in figures) in each front view frame, objects in the collision zone will be considered as "threats" to the driver. In general, the area of the collision zone is fixed when the vehicle travels under a speed threshold. The size will increase if the vehicle travels faster than the speed threshold, the higher vehicle speed, the larger the collision zone. The speed threshold and the size of the collision zone are defined based on empirical experiments. If there is an object exists in the collision zone, the driver might not have enough time to avoid hitting that object. Note that traffic signs, such as stop signs, speed limits, will always be considered by the evaluation module.

Any time before changing or merging lanes, a driver needs to check if there is

any object in blind spots. DrivAid monitors three most common types of objects that could appear in blind spot areas, they are vehicles (e.g. cars, trucks, etc.), pedestrians and bikers. As shown in Figure 3.4c, a pedestrian, a truck and two vehicles are detected in this frame. Similarly, we labeled a collision region to check whether there is anything in the blind spot area. The driver can make a safe lane change when the collision region is empty.

As mentioned in the previous section, DrivAid can detect four types of speed limits from the front view camera. DrivAid queries vehicle speeds from GPS every second. Once a speed limit sign is detected from the video frame, it will be compared with the speed from GPS. When risky events are detected, we check whether those events are caused by speeding. Besides, we can also infer if the driver has the speeding habit.

3.4 Driving Activity Evaluation

By combining the context information with the activity detection results with IMU and GPS, we can infer whether the driver is doing right when events occur.

Data Fusion

It is always hard to reveal the whole picture on the road using information acquired from a single sensor. Hence, DrivAid combines information acquired from multiple sensors to develop a deeper understanding. For instance, once a turning or lane change event is detected, the system will check if the driver follows correct rules before making the turn or lane change, e.g., using turn signals, checking wing mirror and blind spot, etc. If the detected turn or lane change is accompanied by a turn signal and a blind spot check, then the driver behaved properly. Otherwise, further analysis may be required to judge the rationality of the driver's behavior.

Comprehensive Driving Activity Evaluation

We implement a decision tree for driving behavior analysis. In this work, we mainly focus on evaluating three types of driving activities: turn, lane change, and hard brake. DrivAid caches the most recent 10 seconds of video and audio data. Once an activity is detected, DrivAid extracts context information from the 10-second data clips and use them to evaluate the driving activity. For each detected activity, we take different factors into consideration. Figure 3.5 shows the workflow of the analysis process. We use a decision tree to rate the driver's performance during an activity. Items in the red dashed boxes are the factors we take into consideration for each event. These factors serve as the inputs of the decision tree. The outputs of the decision tree are the classified categories of each event. For example, an aggressive turn is confirmed if the driver is speeding when making the turn. A conservative lane change is determined only when the maneuver duration is too long while the other operations are well done. If a hard brake is caused by the sudden appearance of a pedestrian, it will be classified as a reasonable one. To build the classification decision tree, we start with listing out all the possible conditions and cases for each event. Then, find out all the possible nodes, and choose the most important node as the starting point. The classification decision tree eventually grows upon the start node. Lastly, we make sure every possible condition can be covered by the tree.

Here are the details of key nodes for each event. For the turning event, F_0 represents the difference between vehicle speed and speed limit. F_1 checks the smoothness when making the turn. F_2 and F_3 measure the level of the turn. F_4 and F_5 are binary values showing whether the driver does mirror and blind spot check, as well as the turn signal usage. For the hard brake event, F_0 describes the smoothness of the brake. F_1 to F_4 are all binary values. F_1, F_3, and F_4 represent the appearances of traffic signals, the vehicle in front collision region and pedestrians correspondingly. F_2 describes whether the driver's attention is distracted or not. For lane change events, F_0 and F_1 represent the front and rear vehicle distances in the next lane. F_3 describes how long does it take to finish the lane change. We divide the distance into three categories, far, moderate, and close. F_2 and F_4 are

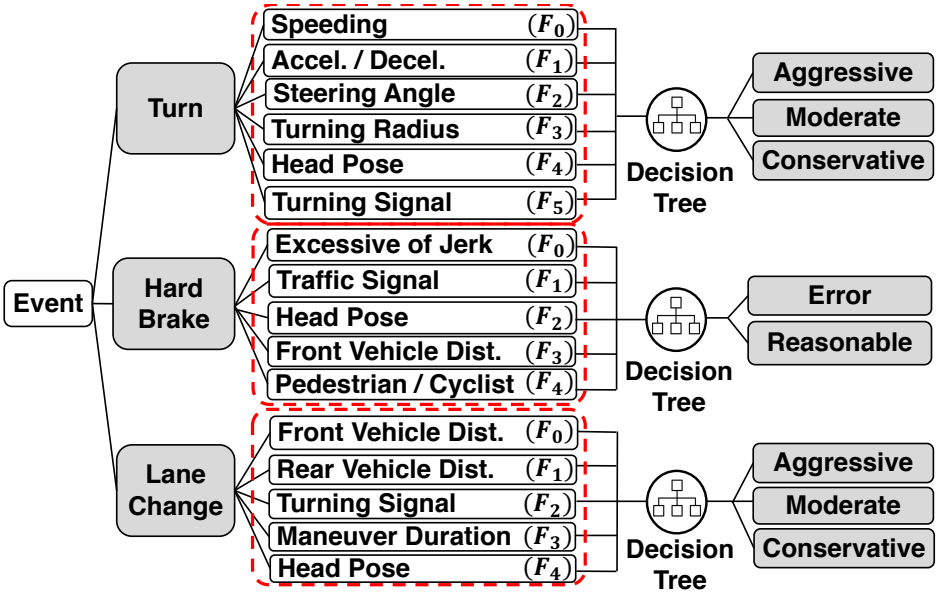


Figure 3.5: The structure of decision tree evaluation model.

binary values which describe the usage of the turn signal and whether the driver checks wing mirrors and blind spots.

3.5 System Implementation

Supporting real-time data processing is a big challenge for DrivAid. To achieve this goal, we carefully choose the embedded computing platform and use the device-specific inference engine to execute the trained neural networks. The core component of DrivAid is the analysis module which runs in the Nvidia Jetson TX2 embedded computer. Jetson TX2 is an embedded system-on-module (SoM) with a hex-core ARMv8 64-bit CPU complex and a 256-core Pascal GPU [45]. We install JetPack 3.2 with L4T R28.2 on Jetson TX2. JetPack 3.2 bundles all the Jetson platform software, including TensorRT 3.0, CUDA 9.0, GStreamer, OpenCV and so on.

We implement DrivAid using GStreamer framework [46] in C and C++. GStreamer pipelines are used to manage each individual task, e.g., we create a pipeline for head pose estimation, and use another pipeline to monitor turn signal usage. A

GStreamer element is a procedure through which the audio-video stream is processed. A GStreamer pipeline consists of a chain of elements. Audio-video streams flow through the pipelines and are processed in a synchronized manner. To implement the context analysis module, we develop a GStreamer plugin including two GStreamer elements for head pose estimation and object detection. The head pose estimation element is implemented with Dlib. The object detection element loads a specific pre-trained DetectNet model and uses the model to recognize objects in input images. The object detection models are trained using the Nvidia Deep Learning GPU Training System (DIGITS) [47]. DIGITS is a wrapper for multiple deep learning frameworks including Caffe, Torch, and TensorFlow. We use Caffe to train our detection model, and copy the trained model snapshot to Jetson for inference applications. Our object detection models are trained on a Desktop with a Nvidia GeForce GTX 1060 GPU. Before using the trained model on Jetson TX2 for inference applications, we need to parse it and perform device-specific profiling and optimizations for the target deployment GPU. We use the TensorRT to perform such kind of optimization tasks.

In total, we have five pipelines to extract useful context information from audio and visual sensors. Here is a list of the pipelines we used:
Pipeline 1: Detecting vehicles, people, traffic signs and speed limits from the front view camera.
Pipeline 2 & 3: Detecting vehicles and people from left and right blind spot cameras.
Pipeline 4: Estimating head poses from the face camera.
Pipeline 5: Monitoring turn signal usage from audio streams. Pipeline 1 has two branches running for object detection in front view. These two branches load the Vehicle & Person and the Traffic Sign inference models correspondingly.

3.6 Evaluation

We deploy our hardware in vehicles and evaluate DrivAid under real-world conditions. In this section, we provide the details of our experiment settings, detection accuracies, and system resource consumption during the real-time process.

Experiment Setup

We deploy the hardware in two vehicles (Honda CRV and Nissan Rogue) and test DrivAid in real-world environments. We develop a custom Android application that runs on a Google Nexus 5X for IMU and GPS sensor data collection as well as video capturing in our prototype system. The data sampling rate is set to 10Hz for motion sensors and 1Hz for GPS. The phone captures video from the rear camera with a resolution of 640×480 @ 10 frames per second (FPS) 1 Mbps bitrate. The sensor and video data are streamed to the Jetson TX2 using TCP protocol through USB tethering service. We have three Logitech C920 video cameras capturing videos from the left and right blind spots and the driver. The video stream settings are the same as what we used for capturing video from the phone camera. Figure 3.6 shows the hardware components of DrivAid. Two cameras are mounted near wing mirrors facing the right and left blind spots. One camera facing the driver is placed on the dashboard behind the steering wheel. The phone is mounted on the center dashboard. Since there is only one USB 3.0 port on Jetson TX2, we use a USB hub to connect cameras and phone. The Jetson TX2 (maximum power consumption is about 7.5 Watt) is powered by the cigarette lighter. In total, we have collected around 1500 miles of data from multiple drivers[1]. Part of the data is collected by our partner [48]. We use the data to train and validate our models.

To evaluate DrivAid, we randomly select 10% of the original data. We mainly focus on analyzing the driving events and driver performance in urban areas since drivers tend to face more complex scenarios in urban environments than those on highways. To get the ground truth, we recruit three volunteers to find out all the events from the recorded data. For each detected event, we ask volunteers to identify driver's actions and surrounding traffic information from the corresponding video clips. Three volunteers should reach a consistent decision for each event, driver actions, and traffic information. For example, if volunteers recognize a lane change event from sensor data. Then, they will watch the corresponding video clip (captured from the front view camera) to double check this event. Further, they fig-

[1]We have received IRB approval for this research project.

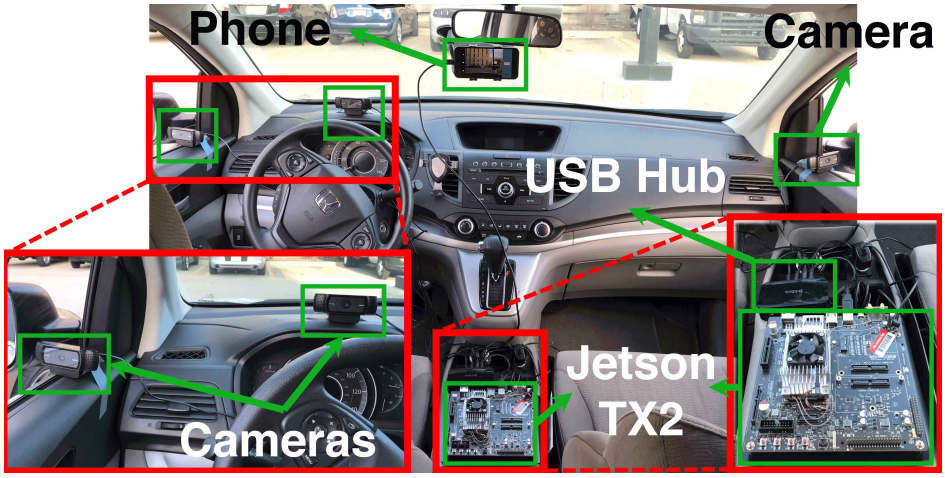


Figure 3.6: The hardware components of DrivAid.

ure out driver's actions (e.g. mirror check, turn signal usage, etc.) and surrounding traffic information (e.g. any vehicle in blind spot area, the distance of the vehicle in front, etc.) from the video clips taken by all cameras. We created a separate thread to record the performance data of the hardware, including CPU/GPU usage and frequencies, as well as memory usage.

Driving Event Detection

In order to avoid unnecessary computation tasks, our system is designed to detect various driving events mainly using smartphone motion sensors. DrivAid buffers audio and video streams in the most recent 10 seconds, and only conducts evaluation tasks once an event is detected.

First, we evaluate the detection accuracy of sensor-based algorithms. We use smartphone motion sensors to detect four different driving actions, namely turn, lane change, brake, and stop. Table 3.2 includes the detection accuracies of these four driving events. Brake and stop are relatively easy to detect and the precision and recall of these two events are 1 and around 0.99 correspondingly. The errors are mainly caused by vibrations of the vehicle (e.g. uneven road surface) and GPS signal lost (e.g. an underground parking lot). The precision and recall are more

Table 3.2: The Accuracy of Driving Event Detection.

Event	# of TPs[1]	# of FPs[1]	# of GTs[1]	PR[2]	RC[2]
Turn	113	7	125	0.94	0.90
Lane Change	67	27	82	0.71	0.80
Brake	306	0	311	1	0.98
Stop	89	0	90	1	0.99

[1] The number of ground truth (GT), true and false positives (TP, FP).
[2] Precision (PR) and Recall (RC).

than 0.9 for turn detection, the misjudgments are mainly caused by unexpected vibrations of the vehicle. We have a slightly lower precision and recall for lane change detection. Some drivers tend to make gradual lane changes and it is relatively hard to capture all of them using sensor data fusion techniques. Due to these factors, we achieve a precision of 0.71 for lane change events. We implement pattern recognition techniques for detecting these events. In order to make sure every event can be detected successfully, we loosen the constraints of matching algorithm when conducting the real-time evaluation. In other words, our system can detect almost every event, but there exist more false positives. To be more specific, the total number of detected events is 489, and around 3825 in seconds (a single event lasts for 8 seconds on average). The total driving time is around 13650 seconds, which means DrivAid only needs to analyze around 36% of the collected data (we extract and evaluate 10-second video-audio clips for each event).

Head Pose Estimation

In this experiment, we recruited 5 volunteers to evaluate the head pose estimation accuracy. As mentioned in Section 3.3, we define six states of head poses. Figure 3.7a (a) to (e) shows how estimated head poses (measured in degrees) change over time when the driver does right and left wing mirror check, right and left blind spot check and rearview mirror check. When the driver focuses on the front view, the estimated head pose is between ± 10. When the driver turns his head to the right, the estimated head pose is a positive number and vice versa. We ask each volunteer

Table 3.3: The Accuracy of Object Detection.

Event	# of TPs	# of FPs	# of GTs	PR	RC
Vehicle	5075	1904	7555	0.72	0.67
Pedestrian	1207	473	1942	0.71	0.62
Traffic Light	440	168	748	0.72	0.59
Speed Limit	427	138	721	0.75	0.59
Stop Sign	370	126	644	0.74	0.57
Turn Signal	162	17	173	0.91	0.93
Head Pose	116	23	150	0.83	0.77

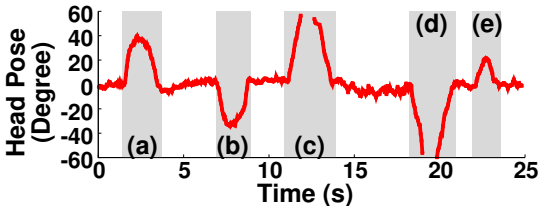
to sit in the vehicle, pretending he or she is driving it. Volunteers are asked to check mirrors and blind spots based on our instruction every 10 seconds. Each round lasts for five minutes. Table 3.3 shows the estimation accuracy. Our estimation module can achieve a precision of 0.83 and a recall of 0.77. Most misjudgments happen on rear mirror check since some people turn their heads very slightly when checking the rearview mirror. We use mean average precision (mAP) to measure the accuracy of object detectors. It is the average of the maximum precisions at different recall values. Figure 3.7b reports the mAPs of Pipeline 4 running at different frame rates. The higher the frame rate, the better the performance. With a higher frame rate, the pipeline can capture more head pose changes, thus, the classification result would be more accurate.

Front View & Blind Spot Monitoring

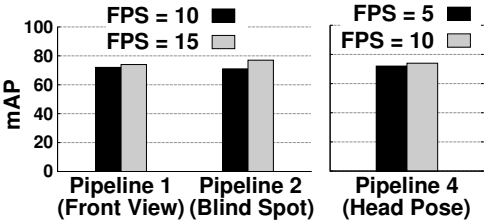
We prepared more than 1500 video frames from recorded videos in the real-world environment and labeled every object appears in each frame. First, we test our front view inference model by comparing the inference results with ground truth. Table 3.3 reports the precision and recall of each object. We find the precision values for the vehicle, pedestrian, traffic light and speed limit are more than 0.7. Based on our observation, the light condition affects detection accuracy most. Although we have used a relatively large data set for training, improvements can be made by expanding image data set as well as changing the design of CNN. We leave this for future work. Next, we replay the recorded videos on a desktop, the video is

displayed on a screen monitor. We start the front view pipeline and let the camera facing the screen. All the detected objects are plotted on the corresponding image frame and saved as a PNG file to a local disk. The total video length is around 20 minutes and we test the pipeline with frame rates of 10 and 15. The mAPs for front view pipeline (Pipeline 1) are shown in Figure 3.7b. We can see that the higher the frame rate, the higher the mAP value.

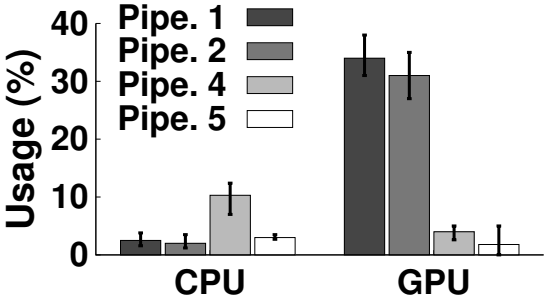
We use the same DNN model as front view pipeline for blind spot monitoring (Pipeline 2). Different from Pipeline 1, Pipeline 2 only uses the Vehicle & Person inference model since traffic signs and speed limit signs are less likely to appear in blind spot area. Therefore, we achieve a similar detection accuracy to Pipeline 1.



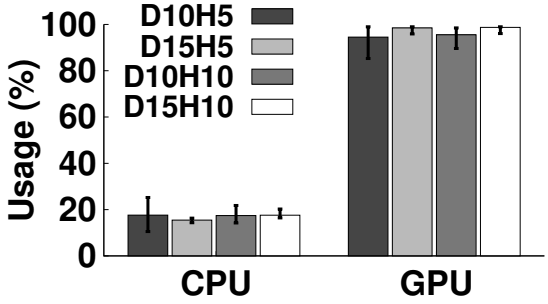
(a) The examples of head pose states.



(b) The mAPs of pipelines.



(c) The CPU and GPU usages of pipelines.



(d) The GPU and GPU usages of pipelines.

Figure 3.7: System Usages.

Audio Monitoring

In this experiment, we extract the audios from recorded video streams to test the turning signal detection pipeline. Table 3.3 summaries the detection accuracy. Both

the precision and recall are greater than 0.9. Most false positives and negatives happen on uneven roads since real turn signal pulses may be buried under vehicle vibration noises. Our detection algorithm works well in most cases and further improvements can be done to remove unexpected noises. Figure 3.7c and Table 3.4 report the system usage details of the turning signal detection pipeline (Pipeline 5). Pipeline 5 consumes the least system resources compared to other pipelines.

System Resource Usage

In this experiment, we evaluate the overall performance of DrivAid. As mentioned in previous sections, data from various sources, including phone, three webcams, a microphone, are pushed to five pipelines. We test DrivAid under different frame rate settings.

System Usage of a Single Pipeline: System usage of each pipeline is summarized in Figure 3.7c and Table 3.4. We measure the CPU, GPU, and memory usage when running each individual pipeline at 10 FPS. The main task of front view pipeline (Pipe. 1) is to detect various objects, it has a very low CPU usage (less than 5%) and a relatively high GPU usage (more than 30%). Table 3.4 shows the memory usage is around 970 Mb for Pipe. 1. The blind spot monitoring pipeline (Pipe. 2) consumes less resource than front view monitoring pipeline since it only has one branch. Our head pose estimation pipeline mainly uses CPU for computing and GStreamer uses a little GPU for video preparation. The memory usage is around 100 Mb, which is much lower than object detection pipelines. Based on our experiment result, a single pipeline can be run with the highest FPS of 30.

Overall System Usage: System usage details are summarized in Figure 3.7d and Table 3.4. "D10H5" means the FPSs of object detection pipelines are set to 10 and 5 for head pose pipeline. From the results, we can see that the memory usages improve a little bit compared to running a single object detection pipeline. GPU is fully occupied by DrivAid at 15 FPS and CPU usage is around 20% at 10 FPS. Based on evaluation results, DrivAid can support a max of 10 FPS for the head pose pipeline and 15 FPS for object detection pipelines simultaneously. Currently, the

Table 3.4: The Memory Usages of Pipelines.

	Pipe. 1	Pipe. 2	Pipe. 4	Pipe. 5
Usage (Mb)	972	964	138	9
	D10H5	D10H10	D15H5	D15H10
Usage (Mb)	1102	1103	1105	1108

head pose pipeline is running on a single core of the hex-core CPU, which is why the CPU is not fully occupied. Future improvements can be done to fully leverage the hex-core CPU.

Overall Performance in Real World Environments

We deploy DrivAid on a real vehicle and drive the vehicle on roads for around 50 miles. We perform normal actions during test drives, including turns, lane changes, brakes, and stops. There are two volunteers sitting in the vehicle and rate every event happens during the drive. The volunteers should come up with a consistent decision for each event. Their judgments serve as the ground truth. In general, DrivAid can achieve an accuracy higher than 90% for context extraction, and the outputs of decision tree analysis module are consistent with the ground truth. Table 3.5 summarizes the details of lane change event analysis results. Due to the space limit, we omit the analysis details of turn and hard brake. Figure 3.8(a) illustrates how DrivAid develops a comprehensive understanding of a hard brake. To evaluate this hard brake, DrivAid extracts useful context information from video clips collected from front view, face view and blind spot cameras. For instance, there is enough space between the two vehicles when the vehicle starts to decelerate (Figure 3.8(b)), driver's attention is distracted before the hard brake happens (Figure 3.8(c)), and so on. Using these factors as inputs, the decision tree classifies the hard brake into "Error" category, which is consistent with the ground truth.

Table 3.5: The Accuracy of Lane Change Event.

Event	TPs	FPs	GTs	PR	RC
Front Vehicle Dist.	25	9	31	0.74	0.71
Rear Vehicle Dist.	24	6	27	0.8	0.75
Turning Signal	31	2	34	0.94	0.91
Maneuver Duration	39	2	41	0.95	0.92
Head Pose	37	5	39	0.88	0.90

3.7 Discussion

Hardware Platform Selection: In order to support DNN-based video data analysis in real-time, we have to choose a hardware platform with hardware accelerators for deep learning inference. Low power consumption is another important factor because we want to deploy the system in regular cars. Nvidia Jetson TX2 is a good option for DrivAid because of its low power design and the integrated deep learning inference engine. However, there are other options available in the industry. For example, Intel's Movidius Myriad 2 Vision Processing Unit [49] is a good option we can try.

Deep Learning Models: Currently, DrivAid primary use object detection models based on DetectNet for context analysis. Many other deep learning models are available for object detection tasks, such as YOLO and SSD. We choose DetectNet because it is easy to be implemented and deployed in the current hardware platform. We may switch to other models after we finish evaluation on those deep learning models regarding both performance and accuracy. In addition, we may use other models like image segmentation for the context analysis in the future work. To get a more comprehensive understanding of the activity and context, we may also use RNNs (Recurrent Neural Networks) to models to analysis driving behaviors in the future work. DrivAid analyzes video frame one by one independently without leveraging the similarity of the neighbor frames. One possible improvement of the current implementation is combining the results of several neighbor frames.

Limitations: Current DrivAid prototype heavily depends on the audio and video signals from cameras, it may not work well at night or dark environment.

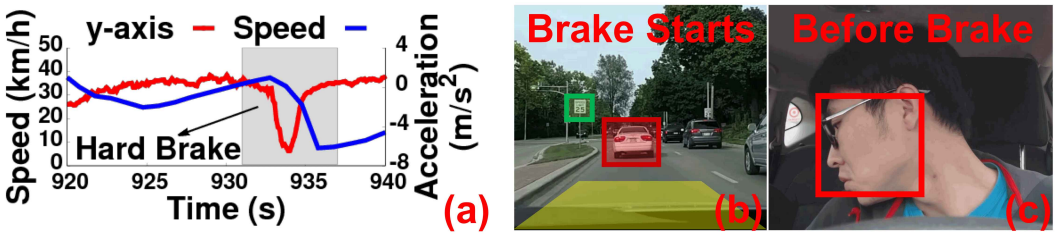


Figure 3.8: Detailed analysis of a hard brake.

3.8 Conclusion

This chapter introduces DrivAid, a multi-modal and light-weight real-time sensing system that leverages audio-visual cues to augment driving behavior analysis. Our system overcomes the limitations of existing IMU-based solutions and can provide fruitful contextual information for driver behavior profiling. DrivAid leverages mobile sensing and computer vision techniques to extract various context information of the driver and the surrounding environments. Further driving behavior evaluation is conducted using the extracted information. We build a prototype of DrivAid on a low-cost embedded computer with deep learning inference accelerator. The prototype is deployed to a regular vehicle and tested in real-world environments. The evaluation results show that our system can process data in real time and provide a good understanding of each driving behavior. We believe such a real-time sensing and analysis system can enable a wide range of applications.

4 A Vehicle-based Edge Computing Platform for Transit and Human Mobility Analytics

4.1 Introduction

A public transit system is an important part of public infrastructure provided by local governments. According to the American Public Transportation Association's report [50], 10.6 billion public transportation trips were taken by Americans in 2015. An efficient and high quality public transportation system both benefits passengers and also has a large impact on city development. Hence, public transit operators have always looked for mechanisms that allow them to improve their services regarding issues such as what new routes or stops should be introduced, how peak and off-peak behaviors are handled, and much more.

Traditionally, these decisions are often based on limited surveys — metro transit operators would recruit volunteers and ask them about their experiences and transit preferences. However, just as mobile devices have transformed crowd-sourced data collection in a whole range of domains, we believe that transit systems can also benefit significantly from them. In this chapter, we advocate a fairly low-cost and simple system through which a transit operator can gather significant user and usage analytics about its operations at a scale never possible before.

Transit systems need to learn much about transit usage to evaluate current transit routes/schedules, and to make decisions on adjustments [51]. Therefore,

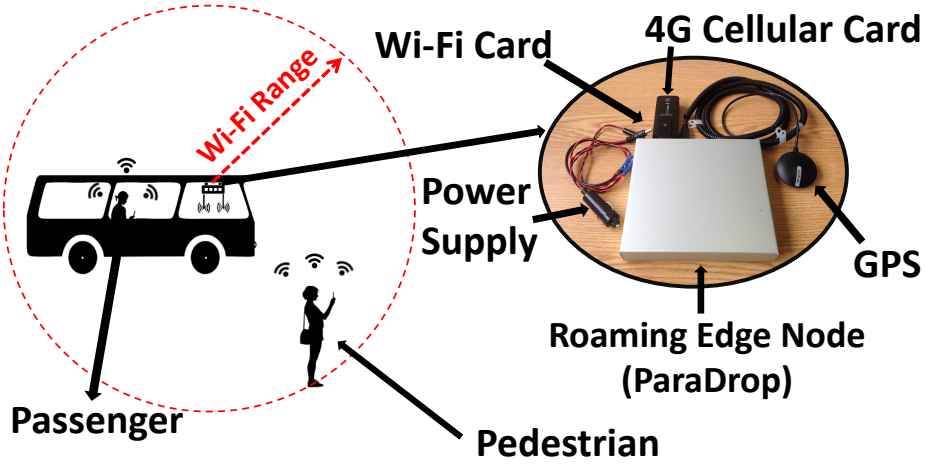


Figure 4.1: The on-board edge computing platform. The key challenge for Trellis is to determine whether an individual is located inside the vehicle (passenger) or outside of it (pedestrian). Based on the fact that pedestrians will eventually be out of monitoring range, Trellis solves this problem by observing device signal strength coupled with the vehicle's speed of movement.

transit operators are actively seeking approaches that can answer questions such as: What are the most popular stops at different times of the day? How long do people at bus stops wait for the next vehicle? How occupied are different vehicles at different times of the day? What do public mobility patterns look like throughout a year, especially during hot summers and cold winters? Some of these questions are significantly related to funding allocations— in particular, operators sometimes receive government funds based on how many passenger-miles they carry annually [52, 53, 54]. Transit operators use a number of low fidelity methods to collect such information. However, existing solutions either failed to answer this question or have been too expensive to be widely deployed. For instance, most ticketing systems on metro buses can infer where passengers get on a bus, but they do not record where/when passengers get off a bus. Some public transit operators rely on expensive sensor systems to count the number of passengers as they get on and off the bus. But these systems are not able to detect the specific origin and destination of individual passengers. Camera-based solutions

involve costly hardware and may generate privacy concerns when customers' facial identities are captured by the cameras. Even cameras are deployed, it is still very challenging to track individual passengers [55, 56]. What's more, pedestrian flows could eventually affect traffic conditions [57, 58, 59, 60]; however, there is not an effective method to estimate the number of pedestrians on the street. The approaches above tend to provide incomplete data or data with fairly low fidelity. In this chapter, we propose a low-cost, wireless-based mechanism to conduct spatial-temporal public transit analytics and answer these unresolved questions.

The usage of edge computing platform: In-vehicle computing platforms are becoming increasingly important as they enable advanced safety, efficiency, and diverse services such as entertainment, navigation, and much more. Compared to cloud computing platforms, such computing platforms in the vehicles provide unique edge services with a lower latency, greater responsiveness, and more efficient use of network bandwidth. These characteristics create such in-vehicle computing platforms as unique locations in which edge computing can be effectively implemented. For example, the massive amount of data generated by the sensors on a self-driving vehicle needs to be processed in a timely manner. A vehicle-based edge computing platform would be an ideal place to execute these kinds of computing tasks. In our efforts, we use a previously developed edge computing platform — ParaDrop [5] — as the platform of choice for deployment inside vehicles to flexibly provide computing and storage resources, allowing developers to create various kinds of services.

We use the ParaDrop platform as a computing platform in vehicles because of its flexibility and its benefits in this environment. The ParaDrop platform, implemented on low-end Wi-Fi Access Points (APs), supports multi-tenancy and a cloud-based back-end through which computations can be orchestrated across many such APs. ParaDrop also provides APIs through which developers can manage their services across diverse ParaDrop APs. In this project, we installed a ParaDrop AP into a public transit vehicle in order to focus on our desired problem — conducting transit analytics. Various kinds of analytics can be done on this vehicle-based edge computing platform, such as video analysis and obstacle

sensing. By loading computation tasks from the cloud to ParaDrop, our system achieves greater traffic efficiency while accomplishing our desired goals. Using ParaDrop, various relevant transit analytics can be quickly derived on-board and sent back to transit operators without incurring high data requirements from the vehicles. Additionally, it is easy to deploy and manage such applications in multiple vehicles across a whole city using ParaDrop.

A first look at Trellis: Wi-Fi-enabled mobile devices have skyrocketed in most parts of the world, and many reports point to their deep penetration among their populations [61]. Our proposed system, Trellis, takes advantage of these widely available mobile devices among passengers and pedestrians to quickly gather various forms of usage information at a significantly large (city) scale. A Wi-Fi-based monitoring system has been widely used in many related scenarios, such as tracking human queues [62], estimating vehicle trajectories [63], and understanding network performance [64]. Trellis uses this kind of mechanism in a similar but much simpler way. As shown in Figure 4.1, our system uses a low-end Wi-Fi monitoring unit mounted on the vehicle to distinguish passengers from pedestrians and determine when a certain passenger gets on and off the vehicle. The approach relies on the fact that many mobile devices typically have their Wi-Fi function turned on, which makes them trackable by another Wi-Fi observer. Most analytics in Trellis are based on the ability to distinguish between which individual is actually inside the vehicle and which is actually outside. While one may be tempted to utilize any one of a slew of Wi-Fi based localization techniques [65, 66, 67], the accuracy of these systems are often not sufficient to distinguish between a passenger seated inside the vehicle and a person who is just outside.

The approach in Trellis to make this distinction is fairly simple — when the vehicle is in motion, the signal strength of a passenger's Wi-Fi device as perceived by a vehicle-mounted Wi-Fi observer is likely to be fairly stable; while the signal strength of an outside pedestrian will vary in a predictable way before eventually disappearing (Figure 4.2). Thus, by simply observing the signal strength trends of Wi-Fi devices while a vehicle is in motion, this localization problem becomes quite simple and can be solved fairly accurately. This basic observation forms the core of

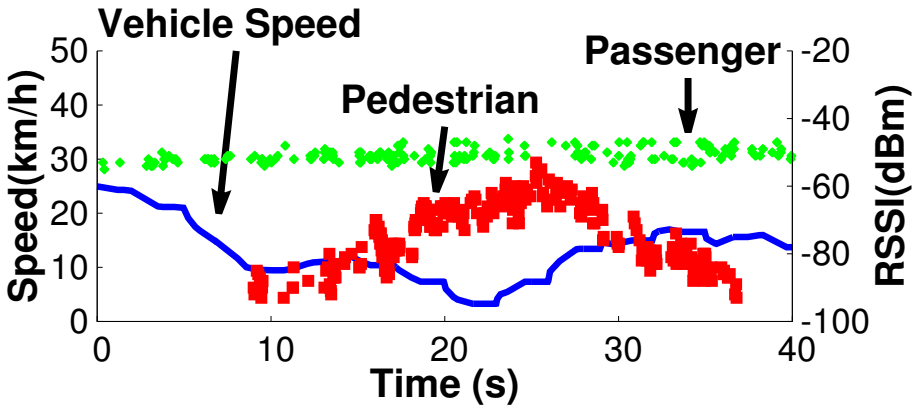


Figure 4.2: Different RSSI patterns between passenger and pedestrian.

many of our analytics presented in Trellis.

Obviously systems such as Trellis will not be able to count for passengers who travel without mobile devices or those with their Wi-Fi function turned off, but our observation shows that we can still track general trends in transit behavior quite effectively [1]. We recommend our current version of Trellis be used to track relative trends in transit systems, as opposed to using it for exact and absolute counts.

We believe that a simple and low-cost infrastructure such as Trellis mounted on public transit vehicles can be effectively used to perform transit analytics as well as answer questions regarding human mobility behavior studies. For the purpose of this work, we demonstrate how such a system may be used from three major perspectives. First, we focus on passenger riding habits, i.e. what are the origin-destination pairs of the user population and how does the popularity of these origin-destination pairs vary for different stations, at different locations, and at different times of the day. Next we study patterns of people on city streets: For example, how busy city streets are, and where hotspots are during different times of day and periods of the year. A system such as Trellis can provide such insights. Finally, we study the impact of weather on outdoor human mobility. Specifically, we

[1]Beginning with iOS 8, Apple introduced randomized MAC address techniques. This may lead to miscounting, but the trends still hold. We will discuss this effect more in section 4.7.

observe how inclement weather (snow and rain) and outside temperature affects the number of people in transit vehicles or out on city streets.

In the end, Trellis provides an unique approach to collect transit information (in addition to other kinds of information) in real-time and can potentially be combined with other existing or complementary approaches. If a reasonable fraction of buses are equipped with Trellis, they can naturally become a crowd-sourced platform to gather and analyze various forms of transit usage analytics at a scale never possible before. Overall, Trellis provides a new lens of human mobility at large scales. While we provide some initial aspects one can learn from this system, we believe many significant opportunities potentially exist.

4.2 Trellis System Design and Implementation

In this section, we discuss the overview, design, implementation and deployment of the system.

System Overview

Trellis tracks people by tracking their Wi-Fi-enabled devices. It achieves this goal in two steps. First, our system performs device detection tasks by capturing Wi-Fi transmissions from each device. As long as its Wi-Fi function is turned on, a device will send out probe request packages scanning for available access points. Our system takes advantage of this feature to capture Wi-Fi enabled devices. The system distinguishes devices by checking their MAC addresses. Once the system successfully detects a device, it will determine whether the device is inside or outside a vehicle. Our system determines the position of the device by observing RSSI coupled with the vehicle's speed of movement. After these two steps, the system records the device data into databases. All of these recorded information can be made available to traffic control centers or other related orgnizations in real time, and facilitate possible real-time responses and updates to various emerging situations. Furthermore, a distributed infrastructure for gathering and analyzing such

transit usage data can provide widespread understanding of public transportation services.

System Design

Our system uses a front-end monitoring module to collect Wi-Fi devices' signals and transit GPS information, and it uses a back-end processing module to reconstruct transit schedules and human mobility patterns. The monitoring module performs sniffing tasks and collects the data from mobile devices. The collected data will be saved in a local database along with corresponding GPS location information. Meanwhile, the sniffing module can send calculated passenger and pedestrian numbers to a remote server in real-time through a cellular link, i.e., for the purpose of real time monitoring. Although our system supports real-time communication, we use a separated program to send the data from the databases to a remote back-end server. The back-end server reconstructs public transit schedules and human mobility patterns from the collected data. It further combines the data from multiple transit sniffing system instances to provide a more complete view of the transit schedules and human mobility patterns. On top of the abstraction and aggregation modules, we construct an origin-destination matrix and pedestrian flow heat map to analyze transit efficiency in spatial and temporal domains.

System Implementation

The Wi-Fi monitoring system is operated on the Ubuntu 14.04.1 64 bit distribution (with Linux kernel version 3.19.0-28-generic), that runs on PC Engines APU platform [68]. The APU platform is a mobile embedded platform that is equipped with a 1GHz dual core CPU and 4G DDR3 DRAM. We conducted the sniffing tasks by using a multi-thread program written in C/C++. One thread runs the monitoring module to collect Wi-Fi packets from the specified wireless interfaces. Another module checks the correctness of received packets by validating the Cyclic Redundancy Check (CRC). The GPS module senses location changes and sends the GPS location information to a third thread. Both packets and GPS data are stored in

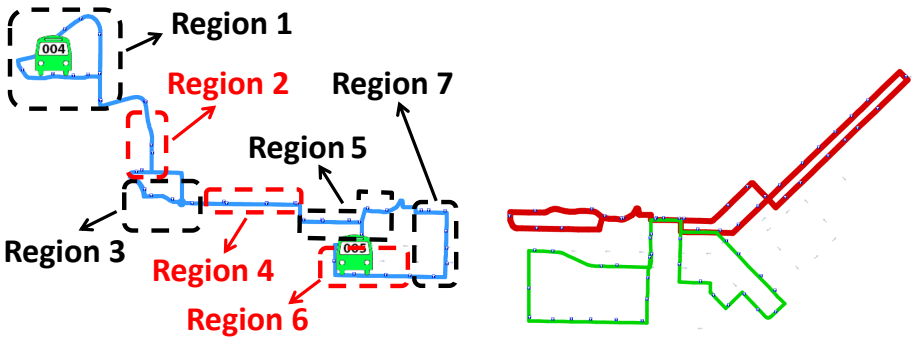


Figure 4.3: Bus routes with labeled bus stops. Route 80 (blue) map on the left, Route 81 (red) & 82 (green) on the right. The map size is roughly 1.5 mile $\times$ 2 mile. We segment the route for bus line 80 into seven disjoint regions for easy analysis.

SQLite database files. To protect public privacy, the private information included in each packet, e.g. MAC addresses, are hashed before saving into databases. And the real data is dropped immediately. The data analysis modules are written in Java.

System Deployment

We deploy our Wi-Fi monitoring system in two city buses. Those two buses have been assigned to three bus routes that are illustrated in Figure 4.3. The bus routes cover a large public university's main campus area as well as a residential area that accommodates graduate students and visiting scholars. The details of each route are shown in Table 4.1. The two city buses are operated by one local bus company.

These two buses are usually scheduled to be on the road from 6am to 6pm on route 80. Buses are also occasionally scheduled to operate during night hours on route 81 and 82. Detailed statistical information about recorded data is summarized in Table 4.2. We collected data from both buses for around 300 days for 12 hours per day. In total, during these 300 days, two buses travel more than 32,000 miles. Among the collected data traces, the two buses ran on route 80, 81 and 82 for 258, 23, and 24 days accordingly. More than 300,000 unique Wi-Fi devices were detected by our system. By looking at the Organizationally Unique Identifier (OUI) [69] of

Table 4.1: Route Statistics

	Route 80	Route 81	Route 82
Trip Distance (miles)	7.91	5.65	5
Trip Time (mins)	45[1]	30	30
Service Span	6am-3am	6:30pm-3am	6pm-3am
Total Station Number	47	31	34
Frequency (mins)	7-50[2]	30	30

[1] The actual trip time ranges from 40 to 50 minutes during different hours of a day.

[2] The frequency will change during different hours of a day, e.g. during rush hours, it will have a higher frequency to satisfy high volume riding demands.

the MAC address (the first three octets), we are able to compare the distribution of various vendors. As shown in Figure 4.4, Apple dominates all other vendors.

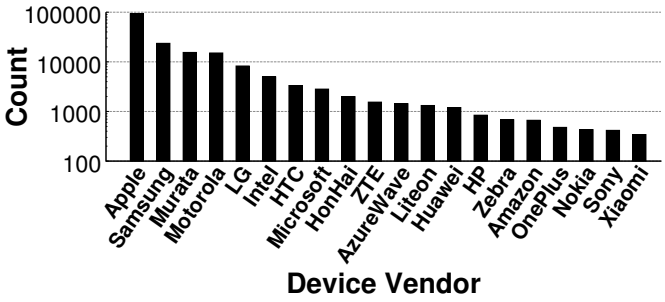


Figure 4.4: Distribution of devices by vendors in log scale.

Starting from iPhone 5s and iOS 8, Apple introduces randomized MAC address in probe requests under certain settings to protect user privacy. According to Zebra Technologies' white paper [70], the MAC randomization can only be triggered when both cellular data and location service are off, with Wi-Fi turned on but not

Table 4.2: Collected Data Statistics

	Route 80	Route 81	Route 82
Days	258	23	24
Hours	3,225	126	65
Distance Covered (miles)	31,510	1,425	510

connected. We also performed the same kind of experiments using iPhone 6 with iOS 8 and Wireshark toolkit. And we have similar observations. According to a recent study [71], iOS devices can be re-identified by checking sequence numbers and timing information contained in the probe request packets. MAC randomization certainly overestimates the number of iPhone users, but it exposes limited impacts on statistical transit analytics.

4.3 Our Approach to Track Individual

In this section, we describe how to reconstruct bus schedules and passenger riding patterns.

Passenger and Pedestrian Tracking

Trellis keeps track of each individual by tracking their Wi-Fi enabled devices, and it separates different devices based on the MAC addresses included in the Wi-Fi packets. Figure 4.5 illustrates the architecture of Trellis. Trellis first determines the type of the device based on received 802.11 packet type, that is, Trellis identifies that the device is a Wi-Fi access point or a mobile device. If the device is a mobile device, further analysis will be conducted. As discussed in previous sections, when the vehicle is in motion, the signal strength of a passenger's Wi-Fi device as observed by a vehicle-mounted Wi-Fi observer should be fairly stable; while the signal strength of an outside pedestrian will vary in a predictable way before eventually disappearing. We developed two schemes to discern which device is inside the bus and which is

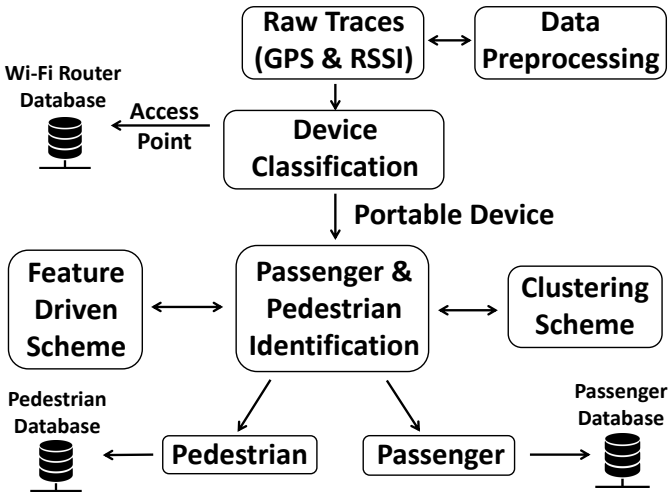


Figure 4.5: Trellis architecture. The raw GPS and RSSI is processed together to identify passenger and pedestrian. Passenger and pedestrian are sent to corresponding modules for further information extraction.

not. A feature driven scheme is a straight forward identification mechanism. We set different thresholds on RSSI, distance and duration to determine who is inside the bus. However, it is hard to select one set of thresholds that can work under different scenarios. Hence, we extracted features from GPS and RSSI data, then used a hierarchical clustering algorithm to distinguish passenger and pedestrian.

Some unpredictable factors certainly affect our system accuracy. For example, some people may still use feature phones or have more than one smartphone; some people will turn off their Wi-Fi function to save power. Under these circumstances, our system will either overestimate or underestimate the total device number. However, we are focusing on the statistical trends of human activities, not the exact number of passengers and pedestrians. Our observation shows that we can still quite effectively track general trends in transit behavior from a long-term view. Next, we explain how Trellis gathers enough information for transit usage analysis.

Passenger Detection

The most challenging task for passenger detection is to extract useful information from collected data. First, the RSSI readings are highly fluctuating. Therefore, we cannot use RSSI alone as the indicator to identify if one passenger is on the bus. Second, the Wi-Fi signals are opportunistically received. The Wi-Fi signals' transmitting frequencies are based on user activities, such as screen being on and off. We have developed two schemes, a feature driven scheme and a clustering

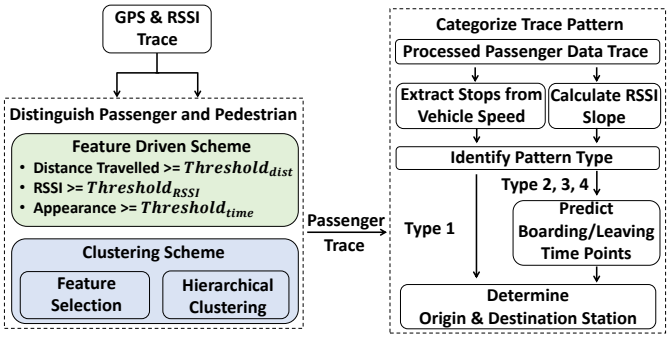


Figure 4.6: Illustration of two schemes and how to keep track of each passenger.

scheme to identify whether one subject is on the bus or not. Figure 4.6 provides an overview of the two schemes.

Feature Driven Scheme: We use multiple RSSI readings observed at different locations to determine the location of that subject. If there are consistent high RSSI readings from a specific device after the bus has been traveling a certain distance (or readings that appear for a certain period), this device is on the bus with high probability. We will discuss how to find the RSSI threshold δ_{on}, distance threshold β_{on} and duration threshold θ_{on} in section 4.3.

Clustering Scheme: As shown in Figure 4.10, emission power of on-bus devices varies greatly. Since the feature driven scheme uses a threshold-based algorithm, the classification results may be affected by some bias factors. To resolve this potential problem, we use a clustering algorithm to classify passenger and pedestrian. Here is a list of features we used for a hierarchical clustering algorithm.

Packet: Mean, Median, Standard deviation, Percentage of RSSI readings greater than -70, Packet receiving rate, $\frac{\sum \text{RSSI}}{\text{Duration}}$
GPS: $\frac{\text{Total Distance}}{\text{Duration}}$, $\frac{\text{Total Distance}}{\text{Packet Size}}$, Average speed, Speed standard deviation
To eliminate potential bias, all features are normalized to values between 0 and 1.

Passenger Tracking

We divided the entire bus route into continuous road segments, and each road segment is between two consecutive adjacent bus stations. Hence, each passenger travels with a bus for at least one road segment. Ideally, we can observe an increasing trend of RSSI values which would indicate that the passenger gets on the bus. Then RSSI readings stay stable for a period of time and then decrease when the passenger gets off the bus. Based on observation, RSSI & speed patterns can be categorized into four types. Figure 4.8 demonstrates four possible types of inference from RSSI & speed patterns.
Type 1: An ideal RSSI & speed pattern, the place and time that a passenger gets on and off the bus can be clearly identified. (Figure 4.8a)
Type 2: Only boarding information can be inferred. (Figure 4.8b)
Type 3: Only leaving information can be inferred. (Figure 4.8c)
Type 4: Neither boarding nor leaving information can be inferred. (Figure 4.8d)
Boarding and leaving information are hard to infer for patterns Type 2, 3, and 4. A prediction method has been developed to handle imperfect cases. (Figure 4.8d)
Determine Pattern Type: We determine the type of RSSI & Speed pattern by checking RSSI slopes with vehicle stop points (where vehicle speed is zero). First, we calculate the slopes of RSSI values using equation 4.1.

$$\text{Slope}_i = \frac{\triangle \text{RSSI}}{\triangle \text{Time}} = \frac{\text{RSSI}_{i+1} - \text{RSSI}_i}{t_{i+1} - t_i} \tag{4.1}$$

Ideally, the slope values should be positive at the beginning indicating the passenger gets on the bus and the slope values should be negative at the end. RSSI slope values are close to zero while the passenger is on the bus. Then, we extract vehicle stop points based on vehicle speed. Combining vehicle stop and RSSI slope information,

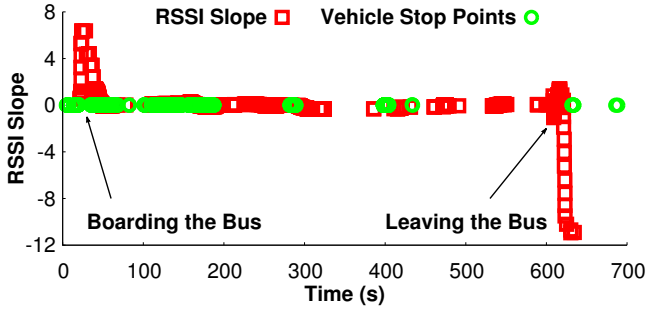
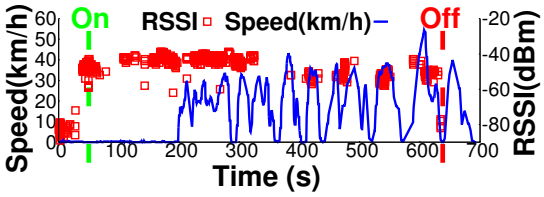


Figure 4.7: An example of how to determine Type 1 pattern from RSSI slopes and vehicle stop information.

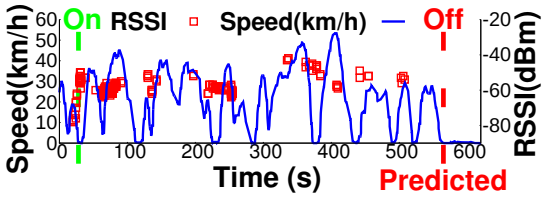
passenger boarding and leaving points can be inferred. Figure 4.7 shows an example of RSSI slopes with vehicle stop points of Type 1 pattern. Slope values are positive in the beginning and negative in the end. The vehicle stops when the peaks appear, indicating the stations where the passengers get on and off the bus. Couple RSSI patterns with vehicle speed of movement, passenger riding information can be clearly inferred for this example. For Type 2 and 3 patterns, only positive or negative RSSI slope values could be observed in the beginning period or at the end of the trip correspondingly. All RSSI slope values are close to zero for Type 4 pattern.

Handling Type 1: For this scenario, the starting bus stop of the first road segment is recognized as the location where the passenger gets on the bus. The ending bus stop of the last road segment is recognized as the location where the passenger gets off the bus.

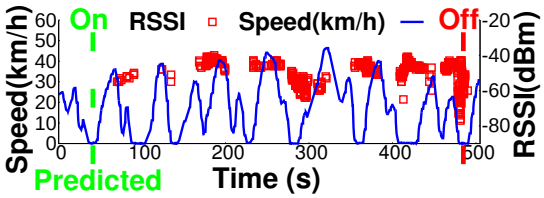
Handling Type 2, 3 and 4: When the phone screen is off, the system will reduce the frequency of sending out probe request packets to save power. Figure 4.10 illustrates the transmission rate for different brands of devices. Theoretically, the device should send out a probe request every τ seconds, which means we could detect that device at least once every τ seconds. It is possible that the passenger could get on and off the bus during this τ second period. This is why Trellis fails to identify the boarding or leaving or both locations for Type 2, 3, and 4 scenarios.



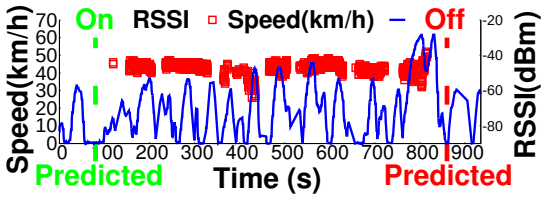
(a) An example data trace showing where a passenger gets on and off the bus.



(b) An example data trace showing only a passenger's boarding information can be inferred.



(c) An example data trace showing only a passenger's leaving information can be inferred.



(d) An example data trace showing that no passenger detailed information can be inferred.

Figure 4.8: Four possible types of inference from RSSI and speed data patterns for detected passenger.

To handle these scenarios, we developed a model to predict when the passenger gets on or off the bus. First, based on frequencies of the received packets from that specific device, we estimate τ using equation 4.2.

$$\tau = \Sigma_i^n f_i * Duration_i \tag{4.2}$$

where $Duration_i$ is the time difference between i-th and $(i+1)$-th packets, and f_i is the appearance frequency of $Duration_i$. To determine boarding time, we begin by using the first half of the received packets to derive τ, and vice versa. Second, we explore all the places that the bus traveled during τ seconds (looking ahead for boarding station predictions, and looking behind for leaving station predictions). The system chooses the bus stop where the bus stops before/after τ seconds as the boarding/leaving bus stop.

Pedestrian Detection

Pedestrians can be detected anywhere along the bus route. Our pedestrian identification algorithm first checks the total distance a bus traveled. If the travel distance is less than a distance threshold β_{off}, then it will check the RSSI readings. In contrast to the passenger RSSI readings, a portion of α readings should be less than the threshold δ_{on}. If an individual satisfies the above two conditions, it has a high probability of being on the road. As shown in Figure 4.10 (left), the RSSI readings from a device in a car tailgating the bus are much less than those from a passenger's device. Therefore, this device cannot be classified as a passenger device. What's more, the total distance a given bus traveled should be much longer than β_{off}. Thus, this device will be treated as a pedestrian device.

Parameter Selection

Figure 4.9 summarizes the cumulative distribution functions (CDF) of station to station travel time and the distance of the three routes. Detailed information of each

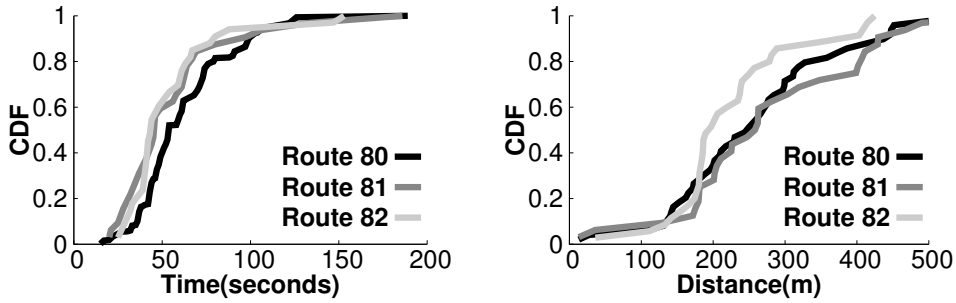


Figure 4.9: The CDF of station to station travel time (left) and distance (right).

route is acquired from the GTFS [72] data published by the local metro company. From the figure we can see that 80% of the travel time between stations is more than 50 seconds. More than 80% of travel distance between stations is longer than 150 meters. For a feature driven scheme, we set the distance threshold β_{on} as 200 meters, choose 100 meters for β_{off}, and assign 1 minute to θ_{on}.

After we recognize one detected device is a passenger device, we can study the Wi-Fi module emission power and compute packet transmission rates for different devices of various vendors. The CDF of emission power of on-bus devices are summarized in Figure 4.10 (left). To plot this CDF, we use RSSI readings observed

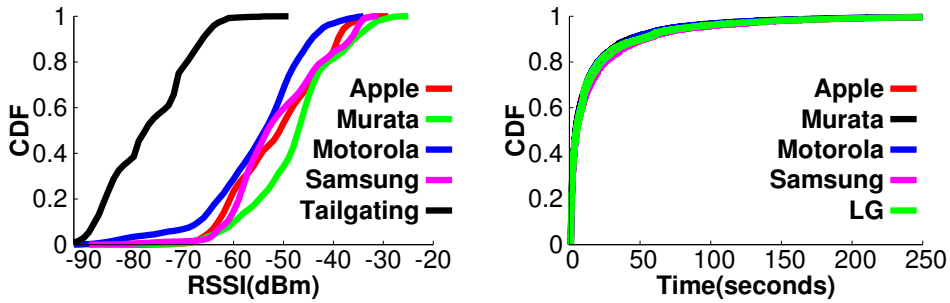


Figure 4.10: The CDF of mobile device Wi-Fi signals' RSSI readings (left) and transmission rate (right).

from on-bus devices. In this chapter, we show devices made by the four most popular vendors. In general, the signal strength for on-bus devices are greater than -65 for around 90% of time. In order to deal with the case of a car following the bus for multiple stops, we collected some data by driving a car tailgating the bus. We put an iPhone 6s Plus and a Nexus 5X in the car, and followed the bus at a close but safe distance. We did this experiment twice, each time for around a half an hour. The RSSI readings are shown in Figure 4.10 (left). The signal strengths from a device in the car following the bus are between -90 to -65. Hence, both the feature driven and the clustering scheme will not consider such a device to be a passenger device. What's more, with the speed and GPS information, our system could tell this is not a pedestrian device (For normal people, it is very hard to run as fast as a bus for a long distance). Therefore, we test the feature driven scheme with δ_{on} ranging from -70 to -60 db. The CDF of the Wi-Fi signal transmission rates is shown in Figure 4.10 (right). Since the transmission rate is within 50 seconds for 90% of the time, this transmission resolution suggests that our individual tracking algorithm has the ability to handle corner cases such as when a passenger is on the

bus for only one road segment.

Transit Schedule Reconstruction

In order to perform public transit analytics, e.g., route design, scheduling, etc., we track each bus on the route to record when it passes each stop. To reconstruct the transit schedule from collected data, we build stop tables for each route. Each bus stop in the table is labeled by an index, GPS location information and direction. All the information is gathered from the published GTFS feeds. Based on the vehicle driving direction as well as the location information, we can infer when the buses pass each station. This module essentially establishes when the bus arrives at each station and how long it stays there. This information is important for transit operators to evaluate the on-time performance of each bus.

Origin-Destination Matrix

For most kinds of analyses in the field of traffic planning and analysis, there is a need for origin-destination (OD) matrices, which specify the travel demands between the origin and destination nodes in the network. Hence, we built an origin-destination matrix, which essentially records how many passengers ride from one bus station to another. We divide the 47 bus stations in route 80 into seven geographically adjacent regions for easy analysis (as illustrated in Figure 4.3). Within each of the seven regions, there are 11, 4, 6, 7, 7, 5 and 7 bus stations, respectively. Based on this matrix, we can analyze passenger region-to-region movement patterns. Additionally, we may add other dimensions to understand passenger behaviors, i.e., time domain and weather conditions, to analyze passenger riding patterns during different periods of the day or under different weather conditions.

4.4 Passenger Activity Trends

In this section, we evaluate Trellis by demonstrating various transit usage analysis results.

Tracking Bus Occupancy

Bus occupancy is an important factor for transit operators to make transit plans, improve the transit efficiency and seek government funding. After reconstructing the transit schedules and identifing passengers, we gathered enough information to count passenger and record how many (essentially which) passengers get on and off at each bus station. Traditional methods such as questionnaires and bus driver counting could help transit operators understand riding patterns. However, these methods require a lot of human labor work, and are time consuming.

Our system provides a low-cost approach to assist or even replace existing counting methods. We evaluate the counting algorithms by comparing estimated passenger numbers and ground truth. To get ground truth data, we recruited several volunteers, and asked them to take the bus and count the number of passengers getting on/off the bus at each bus stop. Volunteers counted the numbers and recorded them using a customized Android application. We collect the ground truth data in 20 trips on 20 different days. For each trip, each volunteer stayed on the bus for around one hour. The aggregated ground truth data covers every hour from 9am to 9pm, including weekdays and weekends. The ground truth data was then synchronized with the data collected by the sniffing system based on time and GPS location.

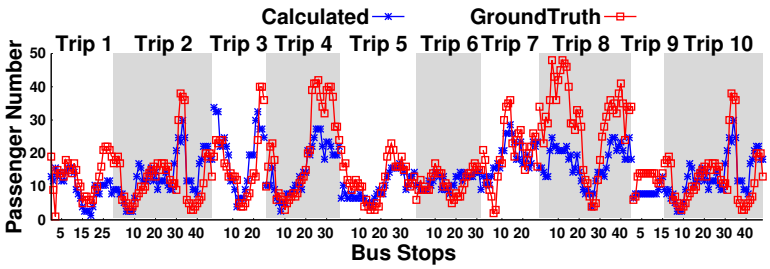


Figure 4.11: Onboard passenger number ground truth and automatic passenger counting results.

Based on our observation, Trellis can, on average, detect 65% of the total passengers. Wang *et al.* reports that their system can discover around 40% of customers

waiting in a queue [62]. Musa *et al.* claims that their system can detect a passing smartphone 69% of the time if the Wi-Fi is turned on [63]. Consider the increasing trend of smartphone penetration rates [73], we believe 65% is a fairly reasonable discovery rate. Hence, Trellis scales the estimated passenger number by 1.5. We summarize the calculated passenger numbers and the ground truth passenger numbers in Figure 4.11. The x axis represents the number of bus stops counted for that trip; the y axis shows the passenger number on the bus between two consecutive bus stops. Due to space limitations, we only show 10 trips in this figure. Each red point represents the actual number of passengers on the bus at that bus stop. The ground truth passenger numbers were counted at each station where the bus stopped, after existing old passengers left the bus and new passengers got on. The total passenger numbers stay the same between two consecutive stops. Each blue point refers to the total number of passengers calculated by Trellis. The estimated passenger number is the total number of passengers on the bus between two consecutive bus stops. Figure 4.11 shows that the major estimation error is

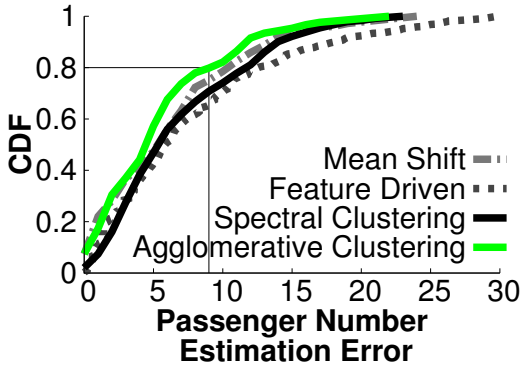


Figure 4.12: The CDFs of passenger number estimation error with different schemes.

caused by passenger bursts. The bursts usually occur when a large volume of students finishes one class together, gets on the bus at one stop, then gets off the bus at next stop together to take another class in a nearby building. Some students may not turn on phone screen during this short trip, so we may lose track of them. However, we still can see the rising and falling trends from estimated passenger

numbers when bursts happened. Hence, the loss of tracking has little effects on long term statistical analysis.

Figure 4.12 summaries the passenger number estimation errors of different schemes. For the feature driven scheme, when the RSSI threshold δ_{on} is set to -65db, Trellis has the best performance. We tested with five clustering algorithms, namely the Affinity Propagation, Mean Shift, Spectral Clustering, DBSCAN and Agglomerative Clustering. Due to space limitations, we show the results of only the best three algorithms in Figure 4.12. Agglomerative Clustering has the best estimation performance. The passenger number estimation error is within 7 for around 80% of the time.

Key Observations: Regarding the evaluation of passenger counting algorithms, Trellis can discover 65% of total passengers. Passenger counting error is within 7 for 80% of the time.

Transit Riding Patterns

The ability to track each individual allows us to conduct transit statistical analysis. For instance, we can study passenger riding behaviors and discover various types of passenger riding patterns at different stops. As previouly noted, the bus route covers a residential area and a main campus area, and our analysis shows that passenger riding habits are periodic during weekdays among bus stops in different regions. We summarize the average number of passengers getting on and off at two specific bus stops located in two regions during each hour of one week in Figure 4.13. The top graph shows the passenger riding patterns at a bus station located in the residential area (region 1 in Figure 4.3) and the bottom graph is for a bus station located in the main campus (region 5 in Figure 4.3). There are two main observations generated from these two figures. First, same riding patterns (including getting on and off) repeats from Monday to Friday, and changes during weekends. Second, bus stops located in different regions have different riding patterns. For instance, in the residential area, people are going out for work in the morning and going back home in late afternoon. Hence, there are obvious riding

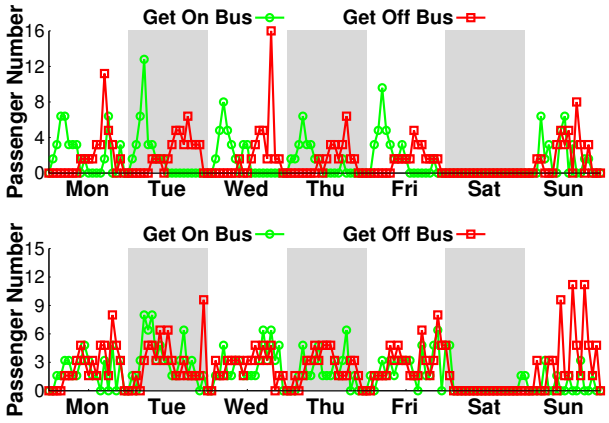


Figure 4.13: Riding patterns of different bus stops in the residential area (top) and the main campus (bottom).

peaks during those hours. Further, undergraduate students live on campus. They travel between dormitories and campus buildings for different classes throughout the day, so there are peaks in the number of passengers getting on and off the bus throughout the day.

Key Observations: Regarding the study of passengers' riding behaviors at stops in a residential area as well as in the main campus area, at a residential bus station, most people get on the bus in the morning and get off the bus in the evening whereas in the main campus area, people get on and off the bus throughout the day.

Transit Scheduling Analytics

We build OD-matrices using passenger region-to-region movements data. Figure 4.14 shows two OD-matrices during morning hours (7am to 9am) and evening hours (5pm to 7pm).

Suppose OD represents this OD-matrix and OD_{ij} denotes each element in the matrix (i represents the index of y axis and j represents the index of x axis). The value of OD_{ij} refers to the total number of passengers getting on the bus at bus stops located in region i and getting off at bus stops located at region j. Darkening colors

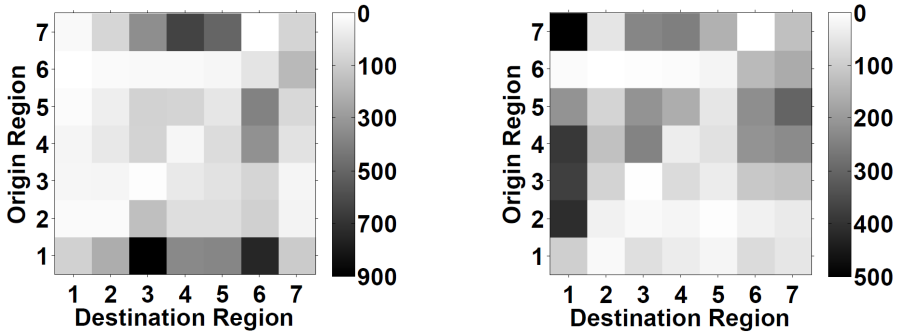


Figure 4.14: Original-Destination matrices during morning hours (left), and evening hours(right).

correlate to increases in passengers. As can be seen from Figure 4.14 (left), most of the passengers travelling from region 1 are going to region 3 or region 6 during morning hours. From this observation, we can provide suggestions to operators. For example, extra direct buses connecting region 1 and 3 (or 6) can be added to the route during morning hours, which could reduce the travel time for passengers who want to go to region 3 or region 6 since the bus stops less frequently, while the rest of passengers can have a better riding experience due to less passengers being on the bus. During the evening rush hours, most of the passengers get on the bus from different regions and are going back to region 1, which means passengers are going back home.

Key Observations: By building OD-matrices for transit scheduling analysis and evaluation it is dscernable that during morning hours, Regions 1 and 3 and Regions 1 and 6 are the most popular origin-destination pairs. Most traffic goes to Region 1 between 5pm and 7pm.

4.5 Pedestrian Activity Trends

Pedestrian activities are of great importance for both the design and evaluation of public transit. Traditional transit evaluation approaches lack ways to gather

pedestrian information. Our system brings the possibility of using human mobility patterns to evaluate transit systems.

Pedestrian Activity Analysis

Pedestrian flows have been shown to be in close relationship with traffic flows [74], i.e. pedestrian flow can reflect and affect traffic conditions. Detecting pedestrian activity could help transit operators have a better understanding of traffic conditions.

Overview

Trellis conducts pedestrian detection concurrently with passenger tracking. As opposed to passengers on the bus, pedestrians are far away from the Wi-Fi monitor. Thus, the RSSI readings from a pedestrian device would be smaller than those of a passenger. Based on this fact, Trellis identifies pedestrians by checking the RSSI readings. Details of the detection algorithm are discussed in section 4.3. While this simple mechanism can miscount some people in a nearby building, this is a very limited negative effect. Based on our experiment results, the chance of such a person being detected is limited, since the distance between the bus and the building is long and the bus passes that building within seconds. To evaluate the accuracy of the pedestrian detection algorithm, we conducted two different kinds of experiments. First, we evaluated the detection accuracy by comparing the estimated pedestrian number with ground truth data. Second, we conducted experiments in an open field area, to study how people inside nearby buildings influence the estimation accuracy. Along route 80, there is an open field area where there are no buildings on both sides of the street. This area runs along a large lake, and it also contains a marsh and a softball field. This area is shown in Figure 4.15 (right); the bus route in this area is labeled in red. For the rest of route 80, buildings are located on both sides of the street. To get ground truth data, we asked volunteers to take the bus, and count the pedestrians on the street as the bus passed by. Figure 4.7 (left) shows the CDFs of the pedestrian number estimation error. In general, Trellis

performs better in the open field area. Around 80% of times, the estimation error is within 7 people. The evaluation results show that Trellis performs well for most cases and will not be affected heavily by people inside a nearby building. Again, in this work, we focus on the general behavior of passengers and pedestrians, not exact numbers.

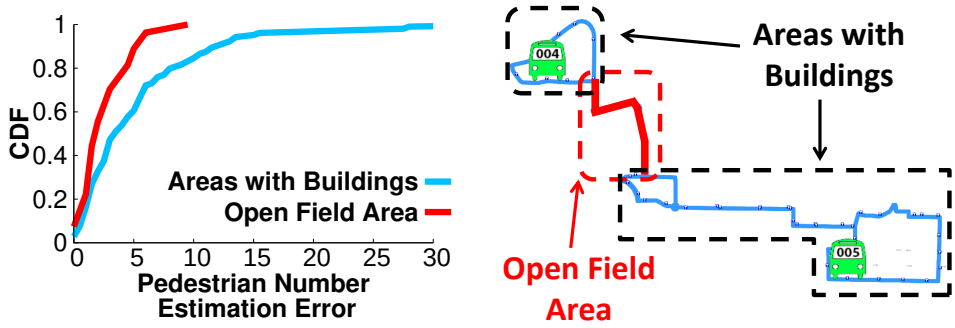


Figure 4.15: The CDF of pedestrian number estimation error.

Time Impacts

Figure 4.16 explains the daily average pedestrian number along route 80 during the day and night. We used data collected from route 80 to plot these heat maps [2]. The top left part in each map represents the residential area while the bottom right part is the main campus. University hospitals and medical schools are located in the middle left. We can see major population centers appear differently throughout a day.

During the daytime, high population density areas are mainly distributed in the residential area, the hospital and some parts of the main campus. This is because various kinds of activities happen in those areas, such as going to school for classes and traveling between different campus areas. During the night hours, the library area is the only active area. Students gather in the library to do homework or participate in group studies. From the GTFS feeds, service frequency varies

[2]Route 81 and 82 only operate during night hours (6pm to 2am).

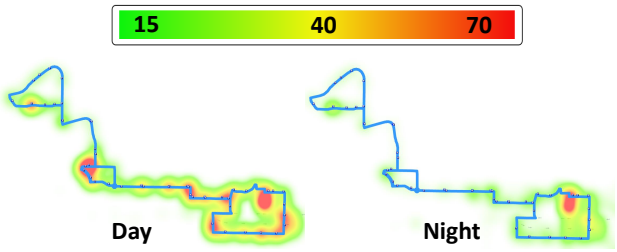


Figure 4.16: A comparison of the daily average pedestrian number on the street during daytime (10am-3pm) and night (9pm-11pm) hours.

from 7 minutes to 50 minutes during a day for route 80. During the morning and evening rush hours, the frequency is 7 minutes. And after rush hours, the frequency decreases from 7 to 14, and then to 25 minutes. Finally, the frequency shifts down to 50 minutes at night. Transit operators carefully design this schedule to serve public efficiently. Our pedestrian activity analysis observations follow and support these facts. We believe Trellis is reliable to perform these transit analytics effectively at a large scale.

Key Observations: By identifying pedestrian activity patterns during different times of day, we can locate several popular areas during the daytime and see that the library area is the only active area at night.

4.6 Impacts of External Factors

In this section, we discuss how external factors, such as weather and temperature, impact human outdoor activities.

The weather data was collected using the Dark Sky Forecast API [75]. We requested hourly weather data and stored those data in a database. The requested weather data contains the following properties: icon (rain, clear, snow, etc.), precipitation intensity (inch/hr), precipitation probability, precipitation accumulation (inches), temperature (Fahrenheit), wind speed (mph), humidity and visibility (miles). Each hour's weather information entry in the database is indexed by a

time integer key, e.g. 201512251300 represents Dec. 25, 2015 at 13:00.

Overview

Trellis was deployed in a northern city in the US. The data was collected through a mild summer and a severe winter. The temperature could be as high as 90 Fahrenheit in the summer but as low as −10 Fahrenheit during the winter. Therefore, outdoor temperature could be a key factor that affects human behaviors as well as traffic.

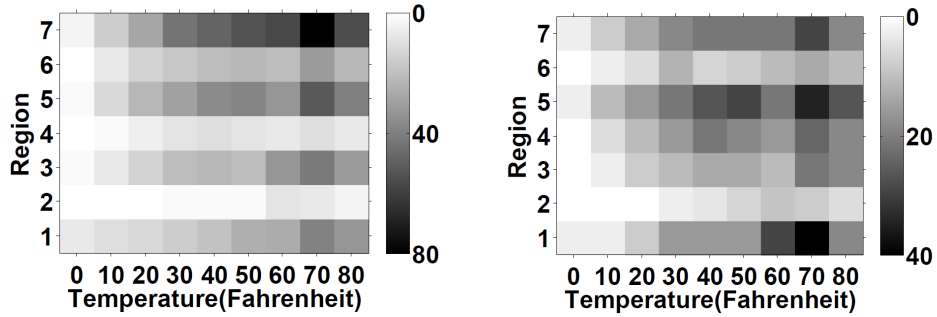


Figure 4.17: The impacts of temperature on daily average pedestrian (left) and passenger (right) number.

We built region-temperature matrices to show how would people react to different outdoor temperatures. We counted passenger and pedestrian numbers region by region, then, for each data entry, we queried the temperature at the time it was detected. Finally, we built the region-temperature matrices shown in Figure 4.17. The left one demonstrates pedestrian behaviors while the right one shows passenger behaviors. For the left figure, each box in the color map is the average daily number of pedestrians detected in that region at a certain temperature range, e.g. suppose the matrix is RT, then RT^{1}_{50} indicates the average daily pedestrian number in region 1 within the temperature range of 45 to 55 Fahrenheit. Similarly, for the right figure, each box in the color map is the average daily number of passengers that get on the bus from the station in that region at a certain temperature range. Darkening colors correlate to increases in the daily average number of people. As can be seen from

these two figures, temperature affects both passengers and pedestrians in a similar way. For each region, there is an increasing trend of numbers as the temperature increases. When the temperature is 0 Fahrenheit (around −17 Celsius), we can rarely see people outside. As the temperature increases, the number of people increases, and reaches a maximum when temperature is around 70 Fahrenheit (around 21 Celsius).

Key Observations: The results of this evaluation of temperature's impacts on human outdoor activities indicate a positive correlation between temperature and human outdoor activities.

Quantitative Analysis

To have a numerical intuition of the weather and temperature impacts, we conducted a quantitative analysis of these two factors based on the data we have. The figure on the left side of Figure 4.18 shows the relationship between the temperature and the number of people in region 7. We can see a positive correlation between the temperature and the number of people, i.e. as the temperature increases, more and more people are willing to go outside. To be more specific, when temperature increases ten degrees, around 15% more pedestrians show up on the street, and around 10% more passengers are present on the bus.

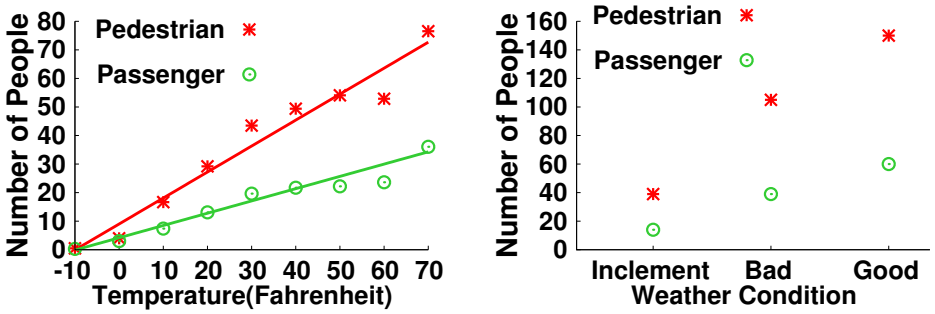


Figure 4.18: Quantify temperature and weather impacts on human activities in region 7.

According to the Dark Sky Forecast API [75], we define weather conditions based on precipitation intensity. If it is raining or snowing and the precipitation intensity is greater than 0.35, we treat it as an inclement weather condition. If the intensity is between 0.05 and 0.2, then we regard it as bad weather. Finally, we identify good weather only when it is clear or sunny. We only tend to use these three conditions as a showcase for quantifying weather impacts. The figure on the right side of Figure 4.18 indicates there is a positive correlation between weather conditions and the number of people. More people participate in more outdoor activities as the weather improves.

Key Observations: Temperature and weather changes affect human outdoor activities.

Impacts on On-Time Performance

Traffic conditions vary dramatically during different times of the day and are affected by various factors. Under inclement weather conditions, especially when weather is snowy or icy, drivers increase headway, decrease acceleration rates, and reduce speeds, which collectively results in traffic congestion and schedule delay. Providing an efficient public transportation system can essentially alleviate traffic congestion.

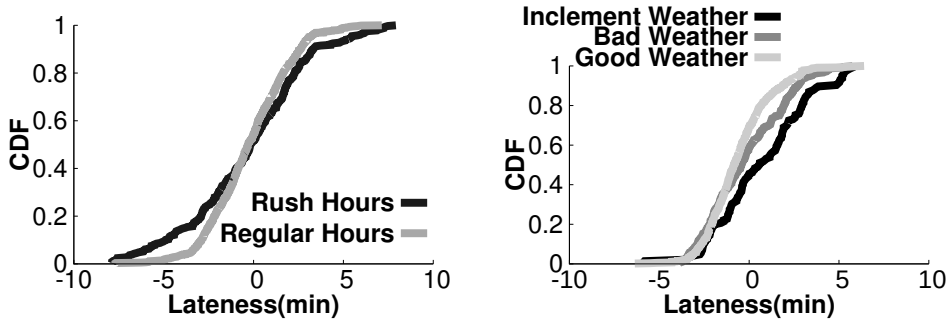


Figure 4.19: The comparison of on-time performance between different hours(left) and weather conditions(right).

In Figure 4.19, we summarize the on-time performance under different scenarios. For this comparison, we focus on the data collected from route 80. The figure on the left shows the difference between peak and off-peak hours. A negative value means that the bus arrived earlier than the scheduled time while a positive value represents how late the bus was. During rush hours, bus drivers need to reduce speeds and brake frequently due to the high volume of vehicles on the road. What's more, passenger demands also increase. Hence, schedule delay is likely to happen during rush hours. In the left figure, we can see the difference between two CDFs, which shows a longer delay during rush hours. Compared to regular hours, early or late arrivals happen more frequently during rush hours. During regular hours, the bus may arrive early or late within a 3-minute interval. For worst case scenarios during peak hours, passengers would experience an 8 minutes' wait, or the bus may arrive 7 minutes earlier than scheduled. One thing that needs to be mentioned is that the transit schedule already takes traffic conditions into consideration, i.e., transit operators schedule more buses on the road during rush hours. Hence, from the CDFs we can see the difference is not that significant, which reflects the effectiveness of current transit schedules. The right figure shows the CDFs of lateness under inclement, bad and good weather conditions. When bad weather happens, traffic slows down and drivers drive more carefully, which all may contribute to a schedule delay. It is shown that as the weather worsens, the delay increases. Under inclement weather conditions, the bus could have a 7 minutes delay for a worst case scenario.

Key Observations: Passengers experience a longer delay during rush hours or when bad weather happens.

4.7 Discussion

In this section, we discuss the limitations of our system and propose other potential applications.

Limitations

These limitations expose challenges for our tasks, but they do not affect the practicability of our system.

Tracking Accuracy

As has been previouly noted, passenger tracking and pedestrian detection accuracy are limited by some unpredictable factors. For example, some passengers are still using feature phones or their Wi-Fi function is turned off. For these cases, the Trellis system is not able to detect the presence of the individual. Some passengers may have multiple Wi-Fi enabled devices, e.g., a tablet and a smartphone. In this case, Trellis may overestimate the number of passengers. Although we use distance and RSSI thresholds to filter passengers and pedestrians, people driving a car following or parallel to the bus may be detected and cause overestimation. Either overestimated or underestimated device numbers will affect the system accuracy. However, since our goal focuses on the statistical trends of transit systems, such impacts can be ignored from a long-term view.

iOS MAC Randomization

Apple introduced MAC randomization in iOS 8; the system uses fake MAC addresses in their probe request packets. It is hard for our system to identify this fake MAC address. However, from our own observations, this feature requires several prerequisites to be triggered. Various discussions and reports [76, 70] also claim the similar findings. According to Freudiger's work [71], iOS devices can be re-identified by checking sequence numbers and timing information contained in the probe request packets. Again, our work focuses on providing statistical analysis on transit efficiency to assist public transit operators instead of tracking every single passenger.

Pedestrian Detection

Our system can detect pedestrians along the route, but it has no capability of tracking each detected pedestrians. However, if we could deploy Trellis on more buses in the city metro network, we might have the ability to track pedestrians by building a communication network among buses. Therefore, several buses can track a single pedestrian and rebuild the pedestrian movement pattern. Our pedestrian detection algorithm may miscount some people in a nearby building along the routes. According to our evaluation results, such counting errors have little affect from a long-term view.

Smartphone Penetration

Another issue to note is the penetration of smartphones with built-in Wi-Fi through the passenger community. While it is hard for us to measure the true distribution of Wi-Fi devices within the passenger community, we can expect that older people and babies are less likely to carry smartphones. However, smartphones with built-in Wi-Fi are increasingly popular and accessible to larger populations. From a long-term view, the riding demand and trend of the passenger community will hold.

Tap-based Fare System

Tap-based fare systems are available all over the world [77]. Transit operators employ two strategies to collect riders' payments. One way is to set a fixed price for the whole route; the other way is to set the price based on the distance traveled. For the first mechanism, passengers only need to "tap in" to the transit system at their origin station. A distance-based pricing strategy requires passengers "tap in" at their origin station and then "tap out" at their destination station. Tap-based fare systems are more efficient than traditional conductors but cause customer delay during peak hours [78]. More recently, Metro Transit riders are able to pay transit fares using their smartphones [79]. Mobile applications have been

developed for passengers to buy tickets online. Trellis could be part of such mobile applications. When a passenger enters the station, Trellis could determine his/her origin and destination, then calculate the price for each trip the passenger has taken. In the meantime, Trellis could evaluate transit usage and send real time statistics to transit operators. Since the whole process can be done automatically by the mobile application, passengers do not need to physically "tap in" and "tap out", thus minimizing queuing delays.

Other Applications

Currently, Trellis is deployed on two city buses, and it has been assigned to three routes. As we continue to cooperate with the local metro transit, Trellis will run on more buses and routes simultaneously, which will further extend its applications. For instance, with data collected from several buses on different routes, we could provide evaluation on transit interchange performance.

We mainly focus on providing transit analytics in this work, however, many more applications could be developed given this rich data set. For example, it is possible to predict the riding route of each individual passenger. Wi-Fi related applications can also benefit from an accurate predication of a passenger's presence [80, 81, 82].

4.8 Conclusion

This chapter presents Trellis, a low-cost, vehicle-mounted wireless monitoring system that can track passenger movements, detect pedestrian flows, and evaluate how external factors impact human mobility, thereby providing useful analytics to transit operators, and potentially urban planners. Trellis uses a passive approach to gathering individual mobility data. The system uses RSSI readings coupled with vehicle location information to distinguish passengers from pedestrians, and it can monitor each detected individual. This capability gives us the flexibility

for evaluating a public transportation system across a city in a very cost effective manner.

In this work, we attempted to provide analytics of specific aspects of passengers on transit vehicles and of human mobility as observed from these vehicles. With ParaDrop, Trellis can be easily deployed and managed in vehicles at a large scale. Various transit analytics can be derived on-board quickly. We believe this approach of observing human populations at city scales has many more interesting and useful applications. Such explorations will form a part of our future work.

5 Related Work

In this chapter, we discuss various prior efforts and existing projects that are related to the work presented in this thesis. We first discuss the related work on edge computing enhanced crowdsensing frameworks and vehicle searching and tracking applications. Then we discuss prior approaches for evaluating driving behaviors. Finally, we discuss existing work on transit and human mobility analytics.

5.1 Edge Enhanced Crowdsensing Framework

Edge Computing enabled Crowdsensing Frameworks

Edge computing is a very popular research area recently, many researchers have explored the combination of edge and cloud computing platforms [83, 84, 85]. ParaDrop [5] is an edge computing platform built with WiFi access points which allows developers deploy services at the network's extreme edge. Firework [86] is a noval edge-cloud computing framework which facilitates distributed data processing and sharing for Internet of Everything (IoE) applications. ENORM [87] proposes an edge node resource management framework to address the resource management challenge between edge nodes and the cloud server. Fernando *et al* [88] presents a work-sharing model which aims to load balance independent jobs among heterogeneous mobile nodes. Cascade borrows the concept of crowdsourcing and

applies it to the edge-cloud framework. It solves big problems using a divide-and-conquer strategy. Sub-problems are simple enough to solved directly at edge nodes. Cascade presents a novel way to solve complex problems using edge-cloud framework.

Streaming and network latency

Existing cloud meeting applications, such as Skype, Hangouts, aim to provide stable and reliable streaming solutions. Lots of research efforts have studied how these applications handle different network conditions. Xu *et al* [89] evaluated the performance of Google+, Skype, iChat under various simulated network conditions. Li *et al* [90] conducted a measurement study of Skype over LTE networks and demostrated the ineffeciencies of protocols. Another study, presented by Yu *et al* [10], studied how different networks (mobile, wireless) affect video qualities of various VoIP applications. Zhang *et al* [9] performed a detailed study to under Skype's protocol. In the study, various packet loss rates, network bandwidth, and propagation delay are simulated to learn how Skype adjusts its FEC redundancy, sending rate to accommendate changes in networks. Applications using Cascade have similar data streaming requirements as existing cloud meeting solutions. Different from existing solotions and studies, the proposed context aware streaming method leverages vehicle dynamics and context of surrounding environments to adjusts streaming parameters.

5.2 Vehicle Tracking

Smart Transportation Applications Using edge computing for intellegent transportation systems is a hot topic in both acadimia and industies. Li *et al* [91] extends the Firework [86] framework and develops a city-wide vehicle tracking application. Vigil [92] is a scalable, novel edge computing architecture for wireless video surveillance system. The system leverage computing resources at the edge and the cloud to achieve real-time tracking and suiveillance. DrivAid [12] puts edge computing nodes into a vehicle and use that to evaluate driver performance in real-time. ParkMaster [93] also leverages the edge computing instance inside a

vehicle to help drivers find parking spaces in urban environments. To achieve real-time processsing or other complexed tasks, most applications also require edge nodes have certain amount of computing power [82, 94, 95]. Since big problems are broke into simpler sub-problems, Cascade has less requirements (in terms of computing power) for its edge nodes. In other words, Cascade tackles down the hardware requirements for its edge nodes, which make it easy to be deployed.

Vehicle Dynamics

Sensing vehicle dynamics using various kinds of sensors have been a popular topic in both industries and research communities [12, 96, 97]. V-Sense [37] senses various vehicle maneuvers, such as lane-changes, turns, and driving on curvy roads by only using non-vision sensors on the smartphone. OmniView [98] provides a driver with a traffic map about the relative positions of surrounding vehicles. To achieve this goal, the proposed system uses cameras of multiple collaborating smartphones in the surrounding vehicles. CarSafe [42] leverages the front and back cameras of a smartphone to detect dangerous driving conditions and behaviors. The smartphone front-facing camera is used to detect the driver tiredness and distraction while the rear-facing camera monitors the road conditions. Cruz *et al.* [99] proposes an efficient algorithm that can localize vehicle positions in surrounding environments. Chen *et al.* [100] proposes multi-view 3D neworks and a sensory-fusion framework that processes data from Lidar and cameras for object detection on the road. Our work levearages existing context sensing solutions and develops a context aware encoding and streaming protocol.

5.3 Driving Behavior Analytics

Transportation-related analytics has been an active research area, such as transfer feasibility at train station [101, 102], human mobility analytics [103], driving behavior analysis [104, 98, 105], and so on. Bigroad [104] leverages motion sensors and phone cameras to collect road data at a large scale. Wang *et al.* [106] utilized embedded sensors in smartphones, for sensing vehicle dynamics to determine driver phone use. The motion sensors capture differences in centripetal acceleration

due to vehicle dynamics. Yang *et al.* [107] addressed the same issue by leveraging high frequency beeps from car speakers. Vision-based driver assistant have been a popular topic in both industries and research communities. The on-board cameras enable a wide range of safety applications that protect drivers and reduce accidents in commercial vehicles [108]. With high quality camera built in, smartphones become a sensor platform for researchers to explore next generation advanced driver assistance systems. SideEye [109] is a smartphone based system to monitor the blind spot on the driver side and alert the driver about the presence of a vehicle. Dooley *et al.* [110] uses a single fisheye camera for detecting and tracking vehicles in blind zones of a vehicle. Leveraging these technologies, insurance companies have programs that offer discounts for drivers with good driving habits [111, 112]. Auditeur [113] is an energy efficient, mobile phone based acoustic event detection system. It can notify users when a registered acoustic event happens.

PrivateHunt [114] presents a public transportation demand-supply model based on large-scale urban transportation data. Basu*et al.* [115] explored the possibility of teaching autonomous vehicles driving in different styles. Liu*et al.* [116] presents a novel clustering approach that identifies hot spots of moving vehicles in urban areas. DrivAid could be deployed on regular vehicles as well as public transportation tools and help to collect fruitful context information for above-mentioned studies.

U-Net [117] is an efficient convolutional network designed for biomedical image segmentation. Vivek Yadav from Udacity borrow the idea of U-Net and develop a small U-Net for vehicle detection [118]. Chen*et al.* [100] propose multi-view 3D neworks and a sensory-fusion framework that processes data from Lidar and cameras for object detection on the road. We plan to compare existing successful deep learning network designs and make appropriate improvements to our current model. We will also explore other methods to acquire meaningful context information from surrounding environments.

5.4 Transit and Human Mobility Analytics

The concept of using a wireless-based approach for transit analytics was first considered in a recent position paper [119]. However, the prior work did not consider the broader opportunity of using this platform for analyzing pedestrians in city streets, nor did it consider the ability to analyze human mobility across different external factors, such as weather and changing temperatures, etc. More importantly, it did not conduct a significant evaluation of these opportunities in practical settings.

Passenger Counting

Transit operators need to collect transit usage statistics either by manually counting or using expensive sensor systems. They are required to submit usage information to national transit database [120]. APC presents a passive, non-radiating infra-red technology to detect and count people moving through a door or gate [52]. The system has the ability to detect the number of passengers, but it needs expensive hardware and is not able to track each individual that is riding between each pair of stations. Chen *et al.* [56] use a video based algorithm to count passenger numbers. Meanwhile, some Asian cities use a system that requires each passenger to tapping IC card when gets on and gets off the bus [121, 122]. These systems cannot count those passengers who are paying cash. More importantly, tapping card may cause extra delays and queues at each bus station. Trellis does not require any passenger operations.

Human Mobility Study

Many applications such as traffic engineering and urban planning need to understand human mobility [123, 124]. Gonzalez *et al.* [125] demonstrated the regularity of human trajectories by tracking user smartphone. Zhang *et al.* [126] studied human mobility based on multiple data resources, e.g., cellphone and transit data, to avoid biased judgment by single data resources. [127] infers human mobility by using taxicab location traces. Our work did similar things and proposes new applications by performing individual monitoring. However, we propose a novel way to conduct public transit analytics by using Wi-Fi monitors on city buses, which separate our work from existing ones.

Human Tracking by Wi-Fi

Wang *et al.* proposed a system that can track human queue length based on received Wi-Fi signal features and analyze the waiting time in the queue [62]. Depatla *et al.* [128] estimate the total number of people in an area based on Wi-Fi device power measurements. However, this technique requires customers' smartphones to become connected with APs and to generate network traffic. VTrack [129] uses smartphone inertial sensors to estimate people's trajectory, which is fundamentally different from our approach. Musa *et al.* [63] deploys multiple monitors on the road to estimating the trajectory of smartphone holders.

Traffic Monitoring Research work has been done on traffic monitoring using instrumented probe vehicles to get sparse probe data [130, 131]. Herring *et al.* [132] use an HMM based probabilistic modeling framework to estimate arterial travel time distributions using collected sparse probe data. Thiagarajan *et al.* [133] proposed a smartphone based system that can shorten the expected waiting times. The study uses smartphones' accelerometer sensors and GPS to determine user position, and estimate traffic conditions. VTrack [129] also uses smartphones as sensor platforms and presents an energy efficient algorithm to predict traffic delay. It proposes an HMM based map matching scheme and travel time estimation model. In contrast to Thiagarajan's work, the VTrack method uses Wi-Fi data instead of inertial data. Nericell [134] takes advantage of mobile phones' multiple sensor readings and proposes a system that can monitoring road and traffic conditions. It requires the smartphone to send various sensor data, which may not work on all phones, and it is energy inefficient. In our study, we use a low-cost passive Wi-Fi monitoring system to get sparse probe data, and we are focusing on the different factors that can help improve transit efficiency.

6 Summary and Future Work

6.1 Summary and Discussion

With the goals of understanding the benefits of connected devices and edge computing platforms for vehicular sensing applications, we have successfully developed three smart transportation applications mentioned in Chapters 1, namely vehicle searching, driving behavior analysis and transit and human analytics. We have shown how to properly formulate tasks for different edge computing nodes and aggregate information extracted from multiple sources to derive information with improved accuracy. These applications demonstrate the usefulness of edge computing platforms as well as some challenges we need to face. As smart transportation applications require a large amount of distributed sensors and devices, the complexities of managing such distributed and heterogeneous resources and services are the biggest challenges. Although edge computing helps preserve privacy, better privacy protection is still an area that requires us to pay more attention to as the system scales up.

Based on the experiences with these applications, the usage of edge computing does not mean cloud computing will be eventually obsolete. Instead, we truly believe edge and cloud computing will be complementary to each other. With the development of edge computing infrastructure, the combination of edge and

cloud computing can enable more powerful applications in the future. Developers can leverage the strengths of a hybrid edge-cloud architecture to build useful applications.

6.2 Future Work

We believe that this dissertation is successful in developing smart applications for transportation systems. We now discuss potential future research directions. We envision the future work can be done from three directions, exploring new opportunities for edge computing, developing applications in related fields, and improving the existing framework. Many research efforts have been done on edge computing leveraging emerging technologies and hardwares. We have only focused on vehicle-based edge computing for smart transportation applications in our work, as technology evolves and more and more end device going mobile and getting connected, many other services and applications can be benefited from edge enhanced frameworks. Besides, when solving the problems mentioned in previous chapters, we realized that most applications require similar modules to function. We would like to build easy-to-use tools and APIs for developers and researchers in this field such that applications using our framework can be easily designed and implemented.

Exploring New Opportunities for Edge Computing

Roaming Edge on UAVs We explain the usage of vehicle-mounted roaming edge in this dissertation. Besides vehicles, the unmanned aerial vehicle (UAV) technology has continuously been evolving recently with exceptional growth. Many industries are looking for opportunities to cut costs by implementing drone technology in their applications or services. Similar to autonomous vehicles, UAVs are also equipped with a set of sensors, such as cameras, LiDAR, GPS and so on. Hence, a crowd-sourced sensing platform could be built with a reasonable amount of UAVs to gather and analyze various forms of data at city scale. As drones could fly in the

sky, they could provide information from a completely new dimension, which could potentially bring new opportunities to many applications. Different from vehicles, UAVs have even limited resources in terms of power, space, and so on. So there are a lot of challenges we need to solve to build a sensing system using UAVs. For example, data generated by cameras and LiDAR need to be processed in a timely manner at the roaming edge on the UAV. Otherwise, the data will continuously get backlogged when the drone is in the sky. Given the power and space limitations of the drone, it is challenging to process such a large amount of data on the UAV. So a good research direction is to explore what tasks are suitable to run on the UAV. In order to gather and derive useful information over a large area or at city scale, it usually requires a group of UAVs to collect and share data. Here are some issues that need to be addressed to implement such a system, how to achieve a given goal with limited resources, what are the best strategies to share data and other information with others. We believe there are many opportunities that exist in this field and it is worth exploring these possibilities.

Other Edge Computing Platforms In this dissertation, we choose a previously developed edge computing platform - ParaDrop, which is built at the extreme network edge, WiFi access points. There are many other research efforts that have been done on edge computing. For example, lots of industry companies are very interested in Mobile Edge Computing (MEC) [33]. Verizon and Amazon Web Services are working together on the 5G Edge pilot. MEC provides computational and storage resources in the access network, which is also being treated as a key emerging technology for 5G networks [135]. It is definitely worth paying more attention to it as 5G networks are just around the corner. Different from LTE networks, 5G technology achieves significant improvements on network bandwidth and latency, and also supports a higher number of connected devices than current LTE networks. Therefore, there will be strong economic incentives for service providers to host resources in the access network and provide edge computing Infrastructure as a Service (IaaS). With the emerging 5G networks, it is an interesting direction for us to explore how would edge computing applications benefit from these technologies. For instance, how could latency sensitive applications, such

as AR and VR, benefit from 5G and edge computing? It is likely that we will switch to 5G eventually, so how can we develop edge computing applications or services that could work with both 4G and 5G seamlessly? What is the role of cloud computing when 5G-based edge computing services are available? Do we still need a central cloud for such services? Above are some next steps that naturally need to be addressed for 5G and edge computing.

More Applications

We will continue to explore vehicle-based edge sensing applications for improving transportation systems. What's more, we also would like to leverage emerging technologies such as 5G, UAVs and so on. Below are two directions we plan to work on in the future.

Building a Pedestrian Collision Avoidance System for Bus Drivers According to NHTSA's report [136], on average a pedestrian was killed every 2 hours and injured every 8 minutes in traffic crashes in 2014. Road traffic safety, especially pedestrian safety, is still a challenge that requires more government attention. Traditionally, a driver would sound the horn to alert pedestrians when the bus would leave a bus stop or make a turn. However, this generates a lot of noise and can be annoying at night. A pedestrian alert system can be developed for bus drivers to help them locate pedestrians in blind spots. The idea is to use cameras installed on the bus and bus stations to monitor pedestrians in blind spots and send out alerts when necessary. Videos captured from different cameras can be analyzed on either the edge compute node at bus stations (the static edge) or the edge compute node on the bus (the roaming edge) Although camera-based methods are relatively accurate in detecting objects, they might be restricted by the possible obstacles that might arise in a pedestrian line-of-sight (LOS). To overcome these limitations, we will place a Bluetooth beacon in each corner of the bus. These beacons will broadcast messages periodically. To be protected, pedestrians should agree to install an application on their mobile devices that will send location information to a central server. Once there is a bus nearby, the application will start scanning the

Bluetooth beacons. Based on the signal strength received from different beacons, our system could estimate the relative location of the pedestrian. The combination of cameras and proximity sensors overcomes the limitations of existing solutions and improves the system performances.

Smart Transportation Applications using UAVs As mentioned in the previous section, UAVs could provide insights from a new dimension and enable improvements for various applications in transportation systems. They can continuously overlook the city as they fly along the city streets, numerous forms of transportation analytics questions can be easily answered. For example, to monitor parking spot availability, many intelligent parking services have been built using static sensors that are installed under or above individual parking spots. With the help of UAVs, we can take a bird view picture of the parking spot and quickly locate empty spots in the picture. Understanding how traffic flows at a large scale is an important problem with which city planners and its traffic engineering team continuously grapple. Most planners try to gather this information through a set of static sensors (e.g., roadside cameras, and road sensors), which gives them a limited view. Similarly, we can take pictures of roads from the top view using UAVs, which will present a clearly view of the current situation without any blockage. There are many further example applications that can benefit from the UAV-based edge sensing framework. This great diversity will also bring some challenges for the roaming edge on UAVs, such as it needs to be built as a multi-tenant resource, property data access should be assigned to each involved stakeholder involved (e.g., traffic engineers, city planners, car owners, law enforcement, etc.). Future work can be done to explore the design of UAV-based edge sensing framework and understand the unique advantages and problems of such a framework.

Improving the Edge-based Sensing Framework

In this dissertation, we proposed an edge enhanced vehicular sensing framework, and we focused on exploring the use of edge computing and IoT devices for smart transportation applications. We plan to polish it to further improve its capabilities,

e.g., building dedicated portable but powerful hardwares that can be easily deployed in vehicular settings, developing easy-to-use tools and modules that allow developers to use them by calling APIs. To be more specific, here are three aspects that can be improved for future work.

Security The security of edge computing infrastructure is an important future work. Unlike cloud computing, the resources of an edge computing platform might not be controlled by the platform owner, and developers might have physical access to the resources. What's worse, edge computing devices can be stolen or physically manipulated. More research is needed to figure out how to restrict access to sensitive information and guarantee the correctness and reliability of edge computing platforms.

Context Aware Communication Module The communication module is responsible for communication among the roaming edge, static edge and the cloud. Our current design does not require communication between roaming edges, however, they might need to talk to each other in some application scenarios. How to allow communication with devices at various locations could be a challenge to latency-sensitive applications. An efficient context-aware communication protocols with good encryption can be implemented to preserve privacy for applications. For instance, the motion sensing video streaming protocol illustrated in Chapter 2.3.

Energy Consumption Monitor Energy consumption is an important factor for edge computing applications, especially for IoT devices as they are usually small in size and has limited space for the battery. Therefore, an energy consumption monitor would help developers with battery management of their applications. In general, there are two ways to monitor energy consumption, software-based and hardware-based approaches. Software-based approach is easy to be implemented, as it just constantly pull battery levels from the device. We would like to start with this approach and prepare APIs for developers to check battery levels from multiple devices at once. Next, we would like to design specific hardware for battery level monitoring. The hardware is not designed for all types of connected devices that can be used in every scenario, it works for some types of applications. For example, it would be useful to design a hardware that can be hooked on smartphones for energy

monitoring since a wide range of applications uses smartphones as a connected sensor. We believe a battery usage monitor could help developers with application development and users can also benefit from it as the application performance improves.

Bibliography

[1] Fog Computing. the internet of things: Extend the cloud to where the things are, 2016.

[2] Rajesh Balan, Jason Flinn, Mahadev Satyanarayanan, Shafeeq Sinnamohideen, and Hen-I Yang. The case for cyber foraging. In *Proceedings of the 10th workshop on ACM SIGOPS European workshop*, pages 87–92. ACM, 2002.

[3] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, 8(4):14–23, 2009.

[4] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[5] Peng Liu, Dale Willis, and Suman Banerjee. Paradrop: Enabling lightweight multi-tenancy at the network's extreme edge. In *Edge Computing (SEC), IEEE/ACM Symposium on*, pages 1–13. IEEE, 2016.

[6] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment.

[7] Brian Bauman and Patrick Seeling. Towards predictions of the image quality of experience for augmented reality scenarios, 2017.

[8] Thomas Wiegand, Gary J Sullivan, Gisle Bjontegaard, and Ajay Luthra. Overview of the h. 264/avc video coding standard. *IEEE Transactions on circuits and systems for video technology*, 13(7):560–576, 2003.

[9] Xinggong Zhang, Yang Xu, Hao Hu, Yong Liu, Zongming Guo, and Yao Wang. Profiling skype video calls: Rate control and video quality. In *2012 Proceedings IEEE INFOCOM*, pages 621–629. IEEE, 2012.

[10] Chenguang Yu, Yang Xu, Bo Liu, and Yong Liu. "can you see me now?" a measurement study of mobile video calls. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pages 1456–1464. IEEE, 2014.

[11] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[12] Bozhao Qi, Peng Liu, Tao Ji, Wei Zhao, and Suman Banerjee. Drivaid: Augmenting driving analytics with multi-modal information. In *2018 IEEE Vehicular Networking Conference (VNC)*, pages 1–8. IEEE, 2018.

[13] TensorFlow. `https://www.tensorflow.org/lite/models/object_detection/overview`, 2020.

[14] A Tao, J Barker, and S Sarathy. Detectnet: Deep neural network for object detection in digits. *Parallel Forall*, 2016.

[15] NVIDIA. `https://developer.nvidia.com/tensorrt`, 2020.

[16] Udacity. Annotated driving dataset, 2017.

[17] LISA. `http://cvrr.ucsd.edu/LISA/datasets.html`, 2020.

[18] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[19] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[20] S. M. Silva and C. R. Jung. License plate detection and recognition in unconstrained scenarios. In *2018 European Conference on Computer Vision (ECCV)*, pages 580–596, Sep 2018.

[21] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[22] Tomás Flouri, Emanuele Giaquinta, Kassian Kobert, and Esko Ukkonen. Longest common substrings with k mismatches. *Information Processing Letters*, 115(6-8):643–647, 2015.

[23] Chris-Andre Leimeister and Burkhard Morgenstern. Kmacs: the k-mismatch average common substring approach to alignment-free sequence comparison. *Bioinformatics*, 30(14):2000–2008, 2014.

[24] spectrico. `http://spectrico.com/car-make-model-recognition.html`, 2020.

[25] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[26] Arthur P Dempster. A generalization of bayesian inference. *Journal of the Royal Statistical Society: Series B (Methodological)*, 30(2):205–232, 1968.

[27] Glenn Shafer. *A mathematical theory of evidence*, volume 42. Princeton university press, 1976.

[28] Peng Liu, Bozhao Qi, and Suman Banerjee. Edgeeye: An edge service framework for real-time intelligent video analytics. In *Proceedings of the 1st International Workshop on Edge Systems, Analytics and Networking*, pages 1–6, 2018.

[29] Zheng Tang, Milind Naphade, Ming-Yu Liu, Xiaodong Yang, Stan Birchfield, Shuo Wang, Ratnesh Kumar, David Anastasiu, and Jenq-Neng Hwang. Cityflow: A city-scale benchmark for multi-target multi-camera vehicle tracking and re-identification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[30] Miloud Aqqa, Pranav Mantini Mantini, and Shishir Shah. Understanding how video quality affects object detection algorithms. In *2019 VISIGRAPP 14th International Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications. INSTICC*, 2019.

[31] Michael Kerrisk. `http://man7.org/linux/man-pages/man8/tc-netem.8.html`, 2020.

[32] Zhenyu Song, Longfei Shangguan, and Kyle Jamieson. Wi-fi goes to town: Rapid picocell switching for wireless transit networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 322–334, 2017.

[33] Tarik Taleb, Konstantinos Samdanis, Badr Mada, Hannu Flinck, Sunny Dutta, and Dario Sabella. On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration. *IEEE Communications Surveys & Tutorials*, 19(3):1657–1681, 2017.

[34] Afshin Dehghan, Syed Zain Masood, Guang Shu, Enrique Ortiz, et al. View independent vehicle make, model and color recognition using convolutional neural network. *arXiv preprint arXiv:1702.01721*, 2017.

[35] Haiyun Guo, Chaoyang Zhao, Zhiwei Liu, Jinqiao Wang, and Hanqing Lu. Learning coarse-to-fine structured feature embedding for vehicle re-identification. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[36] DriveWell. `https://www.cmtelematics.com/`, 2017.

[37] Dongyao Chen, Kyong-Tak Cho, Sihui Han, Zhizhuo Jin, and Kang G Shin. Invisible sensing of vehicle steering with smartphones. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 1–13. ACM, 2015.

[38] German Castignani, Raphaël Frank, and Thomas Engel. Driver behavior profiling using smartphones. In *Intelligent Transportation Systems-(ITSC), 2013 16th International IEEE Conference on*, pages 552–557. IEEE, 2013.

[39] Bozhao Qi and Suman Banerjee. Goniosense: a wearable-based range of motion sensing and measurement system for body joints: poster. In *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*, pages 441–442, 2016.

[40] Cagdas Karatas, Luyang Liu, Hongyu Li, Jian Liu, Yan Wang, Sheng Tan, Jie Yang, Yingying Chen, Marco Gruteser, and Richard Martin. Leveraging wearables for steering and driver tracking. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*, pages 1–9. IEEE, 2016.

[41] Lei Kang and Suman Banerjee. Practical driving analytics with smartphone sensors. In *VNC, 2017 IEEE*, pages 303–310. IEEE, 2017.

[42] Chuang-Wen You, Nicholas D Lane, Fanglin Chen, Rui Wang, Zhenyu Chen, Thomas J Bao, Martha Montes-de Oca, et al. Carsafe app: Alerting drowsy and distracted drivers using dual cameras on smartphones. In *ACM MobiSys*, 2013.

[43] Davis E. King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10:1755–1758, 2009.

[44] OpenCV. `https://opencv.org/`, 2018.

[45] Nvidia. `https://developer.nvidia.com/embedded/buy/jetson-tx2`, 2018.

[46] GStreamer. `https://gstreamer.freedesktop.org/`, 2020.

[47] Nvidia. `https://developer.nvidia.com/digits`, 2018.

[48] China Automotive Engineering Research Institute. `http://www.caeri.com.cn/`, 2018.

[49] Intel. Intel movidius myriad vpu 2: A class-defining processor, 2018.

[50] Virginia Miller. Americans Took 10.6 Billion Trips on Public Transportation in 2015. `http://www.apta.com/mediacenter/pressreleases/2016/Pages/160331_Ridership.aspx`, 2016.

[51] Avishai Ceder. *Public transit planning and operation: Modeling, practice and behavior*. CRC press, 2016.

[52] Horst E Gerland and Kurt Sutter. Automatic passenger counting (apc): Infra-red motion analyzer for accurate counts in stations and rail, light-rail and bus operations. *INIT GmbH Innovations in Transportation*, 1999.

[53] InfoDev Automatic Passenger Counting. `http://www.infodev.ca/vehicles/counting-passengers.html`, 2017.

[54] Thomas Kimpel, James Strathman, David Griffin, Steve Callas, and Richard Gerhart. Automatic passenger counter evaluation: Implications for national transit database reporting. *Transportation Research Record: Journal of the Transportation Research Board*, 2003.

[55] Chao-Ho Chen, Yin-Chan Chang, Tsong-Yi Chen, and Da-Jinn Wang. People counting system for getting in/out of a bus based on video processing. In *Intelligent Systems Design and Applications, 2008. ISDA'08. Eighth International Conference on*. IEEE, 2008.

[56] Chao-Ho Chen, Tsong-Yi Chen, Da-Jinn Wang, and Tsang-Jie Chen. A cost-effective people-counter for a crowd of moving people based on two-stage segmentation. *Journal of Information Hiding and Multimedia Signal Processing*, 2012.

[57] Sheng Jin, Xiaobo Qu, Cheng Xu, and Dian-Hai Wang. Dynamic characteristics of traffic flow with consideration of pedestrians' road-crossing behavior. *Physica A: Statistical Mechanics and its Applications*, 392(18):3881–3890, 2013.

[58] Rui Jiang, Qingsong Wu, and Xiaobai Li. Capacity drop due to the traverse of pedestrians. *Physical Review E*, 65(3):036120, 2002.

[59] Dirk Helbing, Rui Jiang, and Martin Treiber. Analytical investigation of oscillations in intersecting flows of pedestrian and vehicle traffic. *Physical Review E*, 72(4):046130, 2005.

[60] Rui Jiang and Qing-Song Wu. Interaction between vehicle and pedestrians in a narrow channel. *Physica A: Statistical Mechanics and its Applications*, 368(1):239–246, 2006.

[61] Jacob Poushter. Smartphone ownership and internet usage continues to climb in emerging economies. *Pew Research Center*, 2016.

[62] Yan Wang, Jie Yang, Yingying Chen, Hongbo Liu, Marco Gruteser, and Richard P Martin. Tracking human queues using single-point signal monitoring. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*. ACM, 2014.

[63] ABM Musa and Jakob Eriksson. Tracking unmodified smartphones using wi-fi monitors. In *Proceedings of the 10th ACM conference on embedded network sensor systems*. ACM, 2012.

[64] Yu-chung Cheng, Mikhail Afanasyev, Patrick Verkaik, Péter Benkö, Jennifer Chiang, Alex C Snoeren, Stefan Savage, and Geoffrey M Voelker. Automating cross-layer diagnosis of enterprise wireless networks. In *In Proc. of ACM SIGCOMM*, 2007.

[65] Paramvir Bahl and Venkata N Padmanabhan. Radar: An in-building rf-based user location and tracking system. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 775–784. Ieee, 2000.

[66] S. He and S. H. G. Chan. Sectjunction: Wi-fi indoor localization based on junction of signal sectors. In *Communications (ICC), 2014 IEEE International Conference on*, pages 2605–2610, June 2014.

[67] Suining He, S-H Gary Chan, Lei Yu, and Ning Liu. Fusing noisy fingerprints with distance bounds for indoor localization. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 2506–2514. IEEE, 2015.

[68] Apu platform, 2017.

[69] IEEE OUI Registry. `http://standards-oui.ieee.org/oui.txt`, 2017.

[70] Zebra Technologies. Analysis of ios 8 mac randomization on locationing. *http://mpact.zebra.com/documents/iOS8-White-Paper.pdf*, 2015.

[71] Julien Freudiger. How talkative is your mobile device?: an experimental study of wi-fi probe requests. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, page 8. ACM, 2015.

[72] General Transit Feed Specification (GTFS). `https://developers.google.com/transit/gtfs/`, 2017.

[73] Aaron Smith. Record shares of americans now own smartphones, have home broadband. *Pew Research Center*, 2017.

[74] Takashi Nagatani. The physics of traffic jams. *Reports on progress in physics*, 65(9):1331, 2002.

[75] The Dark Sky Forecast API. `https://developer.forecast.io`, 2017.

[76] Bhupinder Misra. iOS8 MAC Randomization - Analyzed! `http://blog.mojonetworks.com/ios8-mac-randomization-analyzed/`, 2014.

[77] Wikipedia. List of smart cards – Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/List_of_smart_cards`, 2017. [Online; accessed 2017].

[78] Justin Antos and Michael D Eichler. Tapping into delay: Assessing rail transit passenger delay with data from a tap-in, tap-out fare system. *Transportation Research Record: Journal of the Transportation Research Board*, (2540):76–83, 2016.

[79] Jake Sion, Transit App, Candace Brakewood, and Omar Alvarado. Planning for new fare payment systems: An equity analysis of smartphone, credit card, and potential mobile ticketing adoption by bus riders in nassau county. In *Transportation Research Board 95th Annual Meeting*, number 16-0179, 2016.

[80] Justin Manweiler, Naveen Santhapuri, Romit Roy Choudhury, and Srihari Nelakuditi. Predicting length of stay at wifi hotspots. In *INFOCOM, 2013 Proceedings IEEE*. IEEE, 2013.

[81] Wenmin Wang, Wei Zhao, Xiaohan Wang, Zhihong Jin, Yuanchen Li, and Troy Runge. A low-cost simultaneous localization and mapping algorithm for last-mile indoor delivery. In *2019 5th International Conference on Transportation Information and Safety (ICTIS)*, pages 329–336. IEEE, 2019.

[82] Lei Kang, Wei Zhao, Bozhao Qi, et al. Augmenting self-driving with remote control: Challenges and directions. In *ACM HotMobile*, 2018.

[83] Zhenyu Zhou, Haijun Liao, Bo Gu, Kazi Mohammed Saidul Huq, Shahid Mumtaz, and Jonathan Rodriguez. Robust mobile crowd sensing: When deep learning meets edge computing. *IEEE Network*, 32(4):54–60, 2018.

[84] Martina Marjanović, Aleksandar Antonić, and Ivana Podnar Žarko. Edge computing architecture for mobile crowdsensing. *IEEE Access*, 6:10662–10674, 2018.

[85] Wei Zhao, Liangjie Xu, Bozhao Qi, Jia Hu, Teng Wang, and Troy Runge. Vivid: Augmenting vision-based indoor navigation system with edge computing. *IEEE Access*, 8:42909–42923, 2020.

[86] Quan Zhang, Qingyang Zhang, Weisong Shi, and Hong Zhong. Firework: Data processing and sharing for hybrid cloud-edge analytics. *IEEE Transactions on Parallel and Distributed Systems*, 29(9):2004–2017, 2018.

[87] Nan Wang, Blesson Varghese, Michail Matthaiou, and Dimitrios S Nikolopoulos. Enorm: A framework for edge node resource management. *IEEE transactions on services computing*, 2017.

[88] Niroshinie Fernando, Seng W Loke, and Wenny Rahayu. Computing with nearby mobile devices: a work sharing algorithm for mobile edge-clouds. *IEEE Transactions on Cloud Computing*, 2016.

[89] Yang Xu, Chenguang Yu, Jingjiang Li, and Yong Liu. Video telephony for end-consumers: measurement study of google+, ichat, and skype. In *Proceedings of the 2012 Internet Measurement Conference*, pages 371–384, 2012.

[90] Li Li, Ke Xu, Dan Wang, Chunyi Peng, Kai Zheng, Haiyang Wang, Rashid Mijumbi, and Xiangxiang Wang. A measurement study on skype voice and video calls in lte networks on high speed rails. In *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*, pages 1–10. IEEE, 2017.

[91] Qingyang Zhang, Quan Zhang, Weisong Shi, and Hong Zhong. Distributed collaborative execution on the edges and its application to amber alerts. *IEEE Internet of Things Journal*, 5(5):3580–3593, 2018.

[92] Tan Zhang, Aakanksha Chowdhery, Paramvir Bahl, Kyle Jamieson, and Suman Banerjee. The design and implementation of a wireless video surveillance system. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pages 426–438, 2015.

[93] Giulio Grassi, Kyle Jamieson, Paramvir Bahl, and Giovanni Pau. Parkmaster: An in-vehicle, edge-based video analytics service for detecting open parking spaces in urban environments. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, pages 1–14, 2017.

[94] Wei Zhao, Jiateng Yin, Xiaohan Wang, Jia Hu, Bozhao Qi, and Troy Runge. Real-time vehicle motion detection and motion altering for connected vehicle: Algorithm design and practical applications. *Sensors*, 19(19):4108, 2019.

[95] Bozhao Qi, Wei Zhao, Xiaohan Wang, Shen Li, and Troy Runge. A low-cost driver and passenger activity detection system based on deep learning and multiple sensor fusion. In *2019 5th International Conference on Transportation Information and Safety (ICTIS)*, pages 170–176. IEEE, 2019.

[96] Tao Wang, Liangjie Xu, Guojun Chen, and Wei Zhao. A guidence method for lane change detection at signalized intersections in connected vehicle environment. In *2019 5th International Conference on Transportation Information and Safety (ICTIS)*, pages 32–38. IEEE, 2019.

[97] Bozhao Qi, Wei Zhao, Haiping Zhang, Zhihong Jin, Xiaohan Wang, and Troy Runge. Automated traffic volume analytics at road intersections using computer vision techniques. In *2019 5th International Conference on Transportation Information and Safety (ICTIS)*, pages 161–169. IEEE, 2019.

[98] Rufeng Meng, Srihari Nelakuditi, Song Wang, and Romit Roy Choudhury. Omniview: A mobile collaborative system for assisting drivers with a map of surrounding traffic. In *ICNC*, pages 760–765. IEEE, 2015.

[99] Susana B Cruz, Traian E Abrudan, Zhuoling Xiao, Niki Trigoni, and João Barros. Neighbor-aided localization in vehicular networks. *IEEE ITS*, 18(10):2693–2702, 2017.

[100] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In *IEEE CVPR*, volume 1, page 3, 2017.

[101] Wei Zhao, Liangjie Xu, Zhijie Sasha Dong, Bozhao Qi, Lingqiao Qin, et al. Improving transfer feasibility for older travelers inside high-speed train station. *Transportation Research Part A*, 113:302–317, 2018.

[102] Wei Zhao, Liangjie Xu, Jing Bai, Menglu Ji, and Troy Runge. Sensor-based risk perception ability network design for drivers in snow and ice environmental freeway: a deep learning and rough sets approach. *Soft computing*, 22(5):1457–1466, 2018.

[103] Bozhao Qi, Lei Kang, et al. A vehicle-based edge computing platform for transit and human mobility analytics. In *ACM/IEEE SEC*, 2017.

[104] Luyang Liu, Hongyu Li, Jian Liu, Cagdas Karatas, et al. Bigroad: Scaling road data acquisition for dependable self-driving. In *ACM MobiSys*, 2017.

[105] Wei Zhao, Liangjie Xu, Shaoxin Xi, Jizhou Wang, and Troy Runge. A sensor-based visual effect evaluation of chevron alignment signs' colors on drivers through the curves in snow and ice environment. *Journal of Sensors*, 2017, 2017.

[106] Yan Wang, Jie Yang, Hongbo Liu, Yingying Chen, Marco Gruteser, and Richard P Martin. Sensing vehicle dynamics for determining driver phone use. In *MobiSys*, pages 41–54. ACM, 2013.

[107] Jie Yang, Simon Sidhom, Gayathri Chandrasekaran, Tam Vu, Hongbo Liu, Nicolae Cecan, Yingying Chen, Marco Gruteser, and Richard P Martin. Detecting driver phone use leveraging car speakers. In *MobiCom*, pages 97–108. ACM, 2011.

[108] Mike Monticello. Guide to lane-departure warning & lane-keeping assist, 2017.

[109] Sanjeev Singh, Rufeng Meng, Srihari Nelakuditi, Yan Tong, and Song Wang. Sideeye: Mobile assistant for blind spot monitoring. In *ICNC*, pages 408–412. IEEE, 2014.

[110] Damien Dooley, Brian McGinley, Ciarán Hughes, Liam Kilmartin, Edward Jones, and Martin Glavin. A blind-zone detection method using a rear-mounted fisheye camera with combination of vehicle detection methods. *Transactions on Intelligent Transportation Systems*, 17(1):264–278, 2016.

[111] Progressive snapshot. `https://www.progressive.com/auto/snapshot/`.

[112] Drive safe & save. `https://www.statefarm.com/insurance/auto/discounts/drive-safe-save/mobile-app`.

[113] Shahriar Nirjon, Robert F Dickerson, Philip Asare, Qiang Li, Dezhi Hong, John A Stankovic, Pan Hu, Guobin Shen, and Xiaofan Jiang. Auditeur: A mobile-cloud service platform for acoustic event detection on smartphones. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 403–416. ACM, 2013.

[114] Xiaoyang Xie, Fan Zhang, and Desheng Zhang. Privatehunt: Multi-source data-driven dispatching in for-hire vehicle systems. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 2(1):45:1–45:26, March 2018.

[115] Chandrayee Basu, Qian Yang, David Hungerman, Mukesh Singhal, and Anca D Dragan. Do you want your autonomous car to drive like you? In *International Conference on Human-Robot Interaction*, pages 417–425. ACM/IEEE, 2017.

[116] Siyuan Liu, Yunhuai Liu, Lionel M Ni, Jianping Fan, and Minglu Li. Towards mobility-based clustering. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 919–928. ACM, 2010.

[117] O. Ronneberger, P.Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention* (*MICCAI*), volume 9351 of *LNCS*, pages 234–241. Springer, 2015. (available on arXiv:1505.04597 [cs.CV]).

[118] Vivek Yadav. Small u-net for vehicle detection, 2018.

[119] Lei Kang, Bozhao Qi, and Suman Banerjee. A wireless-based approach for transit analytics. In *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications*, pages 75–80. ACM, 2016.

[120] National transit database. *http://www.ntdprogram.gov/*, 2017.

[121] IC Card in Hong Kong Transit. `http://www.octopus.com.hk/home/en/`, 2017.

[122] Pengfei Zhou, Yuanqing Zheng, and Mo Li. How long to wait?: predicting bus arrival time with mobile phone based participatory sensing. In *Mobisys*. ACM, 2012.

[123] Fereshteh Asgari, Vincent Gauthier, and Monique Becker. A survey on human mobility and its applications. *arXiv preprint arXiv:1307.0814*, 2013.

[124] Yanping Hao, Liangjie Xu, Bozhao Qi, Teng Wang, and Wei Zhao. A machine learning approach for highway intersection risk caused by harmful lane-changing behaviors. In *CICTP 2019*, pages 5623–5635. 2019.

[125] Marta C Gonzalez, Cesar A Hidalgo, and Albert-Laszlo Barabasi. Understanding individual human mobility patterns. *Nature*, 2008.

[126] Desheng Zhang, Jun Huang, Ye Li, Fan Zhang, Chengzhong Xu, and Tian He. Exploring human mobility with multi-source data at extremely large metropolitan scales. In *Proceedings of the 20th annual international conference on Mobile computing and networking*. ACM, 2014.

[127] Raghu Ganti, Mudhakar Srivatsa, Anand Ranganathan, and Jiawei Han. Inferring human mobility patterns from taxicab location traces. In *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*. ACM, 2013.

[128] S. Depatla, A. Muralidharan, and Y. Mostofi. Occupancy estimation using only wifi power measurements. *IEEE Journal on Selected Areas in Communications*, July 2015.

[129] Arvind Thiagarajan, Lenin Ravindranath, Katrina LaCurts, Samuel Madden, Hari Balakrishnan, Sivan Toledo, and Jakob Eriksson. Vtrack: accurate, energy-aware road traffic delay estimation using mobile phones. In *Sensys*. ACM, 2009.

[130] Wei Zhao, Liangjie Xu, and Bin Sun. Relationship between vehicle emissions and air quality within a transfer center area of downtown: a case in wuhan, china. In *CICTP 2015*, pages 3408–3418. 2015.

[131] Lei Kang, Bozhao Qi, Dan Janecek, and Suman Banerjee. Ecodrive: A mobile sensing and control system for fuel efficient driving. In *MobiCom*, pages 358–371. ACM, 2015.

[132] Ryan Herring, Aude Hofleitner, Pieter Abbeel, and Alexandre Bayen. Estimating arterial traffic conditions using sparse probe data. In *Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on*, pages 929–936. IEEE, 2010.

[133] Arvind Thiagarajan, James Biagioni, Tomas Gerlich, and Jakob Eriksson. Cooperative transit tracking using smart-phones. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, pages 85–98. ACM, 2010.

[134] Prashanth Mohan, Venkata N Padmanabhan, and Ramachandran Ramjee. Nericell: rich monitoring of road and traffic conditions using mobile smartphones. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 323–336. ACM, 2008.

[135] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. Mobile edge computing—a key technology towards 5g. *ETSI white paper*, 11(11):1–16, 2015.

[136] National Highway Traffic Safety Administration et al. Traffic safety facts: 2013 data: pedestrians. 2015.