National University of Singapore

Doctoral Thesis

Smartphone-based Decentralized Public-transport Applications

Submitted by:
Kartik Sankaran

Supervisors:
Associate Professor Chan Mun Choon
Professor Akkihebbal L. Ananda

Department of Computer Science
School of Computing
National University of Singapore

February 2016

National University of Singapore

Doctoral Thesis

Smartphone-based Decentralized Public-transport Applications

Submitted by:
Kartik Sankaran
(B.Eng., PES Institute of Technology)

Supervisors:
Associate Professor Chan Mun Choon
Professor Akkihebbal L. Ananda

A Thesis submitted for the degree of
Doctor of Philosophy

Department of Computer Science
School of Computing
National University of Singapore

February 2016

DECLARATION

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all sources of information which have been used in the thesis.

This thesis has not been submitted for any degree in any university previously.

Kartik Sankaran
February 22nd, 2016

Acknowledgements

First and foremost, I would like to thank my supervisor, Prof. Chan Mun Choon, for his support and guidance throughout my graduate study at NUS. I respect him for his energy and commitment to all his students. During paper deadlines, he often works together with us, many times late into the night. During the difficult times of my PhD, he has always encouraged me and guided me. My success is all due to his support. I am happy to get an opportunity to continue my research work under him after PhD as well.

I would like to thank Prof. Ananda A. L., who was my supervisor during the first four years of my PhD. He has always supported and encouraged me. He is more than a supervisor to his students, and has often spent time with us outside university. I look forward to meeting him once he is back in Singapore this year.

I thank Prof. Li-Shiuan Peh, who was my unofficial supervisor during the final three years of my PhD. Her suggestions, comments, and reviews during our weekly meetings and during paper deadlines were invaluable. She is always prompt in giving me feedback on my writing. I thank her for her guidance and support during my graduate study, and am happy to continue working under her after PhD.

I would like to thank the internal reviewers, Prof. Ben Leong and Prof. Bhojan Anand, and the external reviewer, Prof. Jakob Eriksson, for their invaluable comments and feedback on my thesis. Special thanks to Prof. Jakob Eriksson, who took time out from his New Zealand trip to attend my defense over Skype.

My parents and sister have been pillars of support throughout my PhD, and have always encouraged and helped me. They made sure that my graduate study went smoothly, and have frequently come all the way to Singapore to spend time with me during the final two years of my PhD when I was unable to travel out of Singapore. I thank them for their support.

I would like to thank all my paper co-authors: Pravein GK, Minhui Zhu, Xiang Fa Guo, and Wang Hui, for all of their valuable contributions to the papers. I would also like to thank Fang Zhao, Ajinkya Ghorpade, and Zuo Bingran from SMART FM for helping us during our research evaluation. Special thanks to Jason Gao and his colleagues in MIT who helped me collect data traces in Boston.

I thank all my colleagues in the CIR lab, including Padmanabha Venkatagiri Seshadri, Xiang Fa Guo, Pravein GK, Girisha Durrel De Silva, Mobashir Mohammad, Nimantha Baranasuriya, Luo Chengwen,

Wang Hui, Hong Hande, Paramasiven Appavoo, Chaodong Zheng, Sudipta Saha, as well as past-CIR lab members Mostafa Rezazad, Prashanth Raghu, Shao Tao, Manjunath Doddavenkatappa, Fai Cheong, and Hwee Xian Tan. Although we work independently on different problems, we frequently have productive discussions, both technical as well as philosophical. Everyone in the CIR lab is helpful to one another, making our lab a nice place to do research work. Special thanks to Dr. Padmanabha, who reviewed my thesis and gave me helpful comments and suggestions. He also constantly encourages everyone in the lab.

I thank all my friends, in particular Anusha Kopparam, Madhuri M S, Sampreet Sharma, Roshni MD, and Priya D, for their support during my long PhD study. They find time to Skype and chat in spite of living in different cities and time zones in the world. I treasure the letters and hand-made cards they posted to me over the last few years.

I would like to thank the National University of Singapore and the Singapore government for giving me the opportunity to pursue a PhD degree, and for granting me a scholarship for study. Finally, I would like to thank all other people who helped me directly or indirectly during my PhD.

Contents

Summary

Transportation is the lifeline of a country. Population growth has put a strain on road networks, causing delays due to congestion, particularly in cities of developing countries. Use of public-transport such as buses and trains is critical to reduce road congestion and improve overall efficiency of the transport system. Smartphone-based applications that provide users with information about transit routes, transit stops, and estimates of travel time during public-transport journeys are the key to make users choose public-transport over the convenience of a car. Existing smartphone-based applications make strong assumptions about the availability of cellular data connectivity and route maps, and fail to work in developing countries where these are often unavailable. These applications also typically track user location, causing concerns about privacy and battery consumption, leading users even in developed countries to turn off these applications.

In this thesis, we propose novel techniques for building public-transport applications on smartphones that are decentralized, and do not require cellular data or information from route maps. We use a new sensor modality - the barometer - together with local collaboration between users in the same bus to tackle the key challenges of context detection, route and transit-stop detection, and on-the-go deployment of applications. Our system makes very few assumptions about infrastructure, enabling it to work even in developing countries where other approaches fail. Since our system uses the barometer instead of location sensors, it reduces concerns about location privacy and reduces power consumption, encouraging even users in developed countries to use such applications. The use of the barometer also makes the system largely independent of user hand movement and phone placement, removing the need for training typically required by other systems.

This thesis work is the first in the literature to propose and implement novel uses of the barometer sensor for public-transport applications, and to demonstrate that smartphone-based public-transport applications can work even with limited infrastructure. It paves the way for future research work in building decentralized and location-'less' smartphone applications.

List of Publications

1. Kartik Sankaran, Akkihebbal L. Ananda, Mun Choon Chan, and Li-Shiuan Peh. Dynamic Framework for Building Highly-Localized Mobile Web DTN Applications. In *Proceedings of the 9th ACM MobiCom workshop on Challenged networks*, CHANTS Sept 2014.
2. Kartik Sankaran, Minhui Zhu, Xiang Fa Guo, Akkihebbal L. Ananda, Mun Choon Chan, and Li-Shiuan Peh. Using Mobile Phone Barometer for Low-Power Transportation Context Detection. In *Proceedings of the 12th ACM Conference on Embedded Networked Sensor Systems*, SenSys Nov 2014.
3. Kartik Sankaran, Akkihebbal L. Ananda, Mun Choon Chan, and Li-Shiuan Peh. Dynamic Framework for building highly-localized mobile web DTN applications. *Computer Communications, Volume 73, Part A*, Jan 2016.

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Transportation: a country's lifeline

Transportation is regarded as the lifeline of a country. It allows goods and people of the working class to move across different parts of the city quickly and efficiently. This has great impact on the economic growth of the country as a whole, and any disruptions or delays in the transportation system are considered major issues.

The growing population and increasing demand for goods has put a strain on existing transportation systems, particularly on road networks. This is caused by an increased number of vehicles travelling on roads [1]. For example, in the USA, a population growth of 20% over 20 years caused traffic to jump by 236% [2]. There are over 1 billion cars in the world today, which is expected to double by 2020 [2].

The increasing number of vehicles on roads causes delays due to congestion, especially at peak hours in the morning and evening when travel is more common. For example, in certain areas of Hong Kong, the increasing traffic during morning peak hour has reduced overall journey speed to even lower than 10 kmph, which is not much faster than the average walking speed of 5 kmph [3]. Congestion is even more common in developing countries where the road infrastructure has not kept up with the growing population.

1.2 Tackling congestion using public-transport

Tackling congestion on roads is challenging. Detecting and identifying congestion hotspots itself is difficult, since it requires significant economic expenditure in traffic monitoring systems to measure vehicle flow and speed.

Alleviating congestion hotspots is hard due to limited availability of land for expansion of road networks. Methods such as electronic road pricing in Singapore reduce the number of vehicles during peak hours, but are not sufficient.

A key approach to reducing congestion is providing reliable, efficient, and cheap public-transport as an alternative for travel. Public-transport vehicles include buses, trains, trams, and subways, which act as shared passenger transport services.

Public-transport increases the overall efficiency of the transportation system by moving a larger number of people across the city using a smaller number of vehicles. Using public-transport effectively decreases the number of cars on roads, reducing the strain on congestion hotspots.

The major problem faced when tackling congestion using public-transport is encouraging people to choose public-transport over the convenience of a car. Not only should the public-transport system be improved, but people need to be incentivized to shift from using cars to using public-transport. However, more is needed than just economic incentives.

A survey of commuters in Boston and San Francisco reveals that people are willing to ride a bus or train as long as they have the proper tools to plan and manage their commute [4]. This includes information about transit stops, delays, and time of travel along routes. By putting such information in the hands of the people, they are better informed and in control of their choices rather than left wondering whether they will reach their destination on time. This has led transport agencies to provide useful information about the public-transport system online, and increasingly, using smartphone apps.

1.3 Smartphones for public-transport apps

The sales of smartphones has surpassed the sales of feature phones [5]. With increasing penetration rate and computational power, smartphones have become a new means for providing useful information about public-transport to commuters.

Traditionally, transport agencies supplied information about the public-transport system after processing real-time traffic data from pre-installed infrastructure, such as road induction loops, speed sensors, and GPS devices installed on buses. However many cities, especially in developing countries, lack widespread monitoring infrastructure, and it is prohibitively expensive to install these systems.

Smartphones have opened up a new and cost-effective alternative to these systems by crowd-sourcing vehicle location data from users' smartphones. Utilizing the location sensor available on smartphones, these applications track users travelling on buses and trains, crowd-sourcing location data to a central server, which then processes this data to track the real-time location of public-transport vehicles, and estimate travel time.

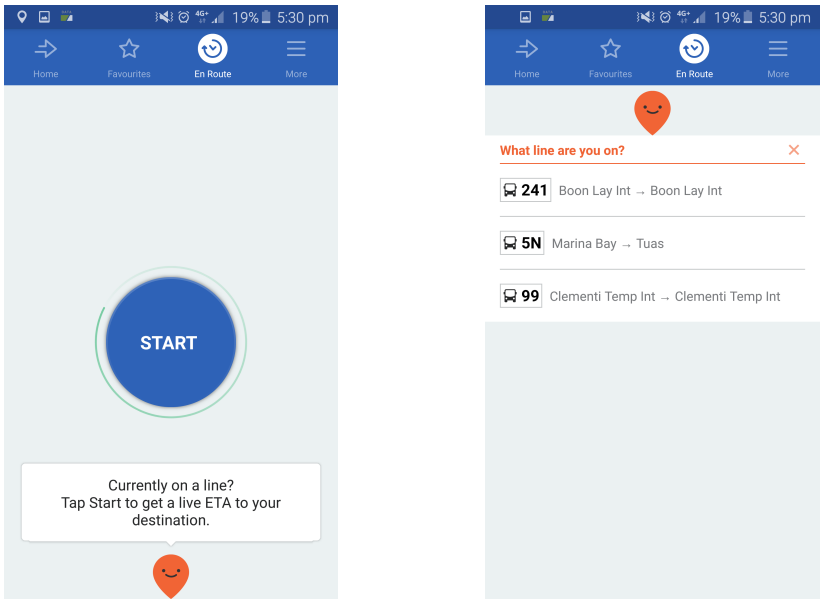
1.4 Challenges and limitations of existing apps

Public-transport applications based on smartphones are a cost-effective alternative to pre-installed infrastructure. However, in order to provide useful functionality, they need to tackle the following major challenges: detecting user context, detecting the bus route, detecting bus-stops, and deployment of specialized application services. We discuss existing solutions to these challenges below, as well as limitations of existing solutions.

Note that in this thesis, although we use the term 'bus', our work is applicable to other public-transport vehicles such as trains, and our evaluation involves traces from both buses and trains. For ease of exposition, we use the term 'bus' instead of the more general term 'public-transport vehicle', unless specified otherwise.

1.4.1 Detecting the travel context

Before displaying the Estimated Time of Arrival (ETA) to the user, public-transport applications first need to automatically detect when the user is travelling. Applications from LTA [6] and Moovit [7] place the burden on users by requiring them to manually press a button once they have boarded the bus, as shown in Figure 1.1a. Other applications utilize the low-power sensors on the phone to automatically detect the user's context. For example, Google Now [8] depends on Google's Activity Recognition [9] that uses the accelerometer to detect if the user is idle, walking, or in a vehicle, in order to trigger the location sensor.



(a) Starting a bus trip **(b)** Specifying the bus service

Figure 1.1: Public-transport application by Moovit that requires users to manually indicate when they are on the bus, and which bus service they are travelling on.

The accelerometer is the predominant sensor used for context detection due to its low power consumption, and because it does not have coverage issues faced by GPS and WiFi. However, accelerometer-based approaches suffer from problems caused by arbitrary hand movements while the user is waiting at a bus-stop, resulting in false vehicle detections.

They are also often unable to detect vehicles with smooth movement, such as in the SMRT trains in Singapore, where there are very few vibrations, and the accelerometer incorrectly detects user context as idle. For example, in our experiments, we found that Google's Activity Recognition algorithm has a low idle detection accuracy of 25% while the user is waiting for a bus, and only 15% vehicle detection accuracy on the circle line metro in Singapore. While the accelerometer typically has high accuracy for walking detection, good idle and vehicle detection still remains challenging.

1.4.2 Bus route and bus-stop detection

After the application has detected that the user is in a bus, it then needs to automatically identify the bus route and subsequently detect when the bus has reached a bus-stop in that route. For bus route identification, applications from Moovit and LTA ask the user to manually input the bus service number they are travelling in, as shown in Figure 1.1b. Google Now, on the other hand, tracks the user via the location sensor to automatically guess the route, based on past recurring journeys recorded in the user's travel history stored at the Google server.

For bus-stop detection, applications use publicly available bus route maps, which include bus-stop locations. By tracking the real-time location of the travelling user, applications can detect when the bus has reached a bus-stop in its route.

Throughout the bus journey, travel times between bus-stops are uploaded via cellular network to a central server, where they are stored as part of a historical travel time database of all users. The server uses this database to give estimates of arrival time to the destination stop.

The above techniques based on user location tracking have several drawbacks. First, the location sensor, especially GPS, is power-hungry and drains smartphone battery quickly. Second, recording user location history on a server, or even locally on the phone, causes location privacy concerns, as this data carries risk of unauthorized access. Users

often turn off such tracking applications to save battery and have better privacy. Third, locations sensors have coverage issues, often inaccurate or unavailable in subways and in urban canyons. Fourth, existing solutions fail to work in developing countries, where cellular data and even route maps may be unavailable (e.g. Dhaka in Bangladesh, Manila in Philippines, and Mexico City[1]). This provides motivation for research work in public-transport applications that can address these limitations and that can be applicable even to cities in developing countries.

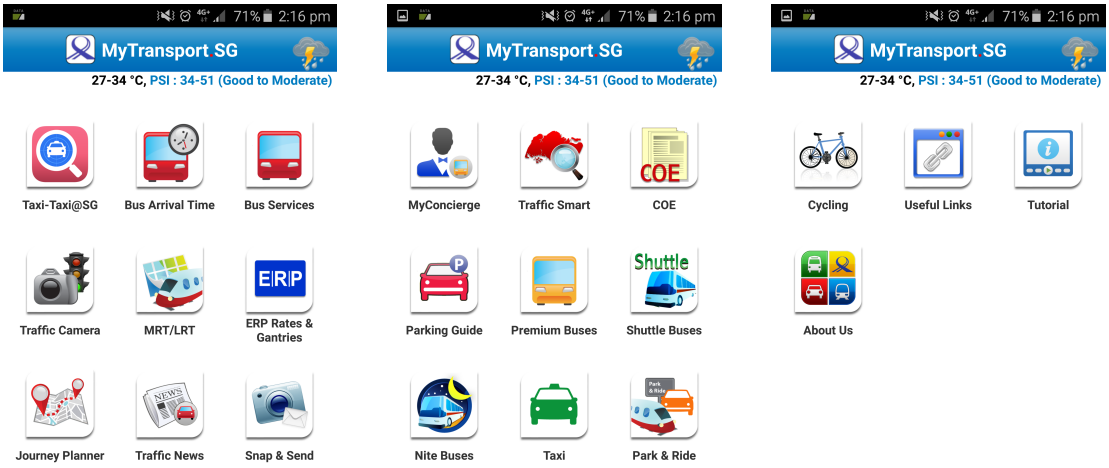
1.4.3 Deploying specialized applications

Another challenge faced by public-transport applications is deployment of specialized applications, such as providing information about bus wheelchair access, availability of nearby cycle rental pitstops, and on-the-spot traveller satisfaction surveys. Often, these applications are developed separately, and deployed as multiple apps on the app store. This becomes problematic for users, who need to install and update multiple (but related) applications on their smartphone.

One way to tackle this problem is to create a unified application. For example, the Land and Transport Authority of Singapore released a mobile application that contains all these services [6], handling everything from cycling to bus arrival times, as shown in Figure 1.2.

However, as applications become more automatic and intelligent, running all these services in the background is unnecessary, considering that users are often only interested in a few of these services, and use these on-the-go for short periods of time. Updating, adding, and removing new services is difficult as well. For example, transport authorities who conduct on-the-spot satisfaction surveys would need to modify the already bulky app, make users update the app, and ask users to rate their happiness during the few minutes they are at the bus-stop or bus. After the survey is finished, uninstalling the

[1] http://www.wired.com/2015/08/nairobi-got-ad-hoc-bus-system-google-maps/



(a) (b) (c)

Figure 1.2: LTA Application with all services bundled into one application.

service is equally cumbersome. These problems highlight the need for an easier and better way to deploy these specialized applications.

Table 1.1 summarizes the three challenges faced by smartphone-based public-transport applications, existing solutions to these challenges, and their limitations. In the next section, we discuss how these limitations are addressed.

1.5 Thesis contributions

In this thesis, we propose novel techniques to address the challenges and limitations highlighted in Table 1.1. To do this, we use a new modality - the barometer sensor[2] - that provides a tradeoff between pure phone motion sensors and absolute location sensors. The barometer measures surrounding air pressure, which can be translated into height above sea level (altitude) in metres. It can be used to measure characteristic

[2]available on Nexus 3/4/5/6, Galaxy S3/4/5/6, iPhone 6, and many more

Table 1.1: Challenges faced by smartphone-based public-transport applications, existing solutions, and their limitations.

Existing Solutions	**Limitations**	**Advantage of our work**
Challenge 1: Travel context detection		
Manual user input [6, 7]	Burden on user	Automatic context detection
GPS/WiFi-based [10, 11, 12, 13]	Coverage issues	100% coverage
Accelerometer-based [9, 14, 15]	Confusion cases caused by hand movement	Unaffected
Challenge 2: Bus route and bus-stop detection		
Based on location tracking [16, 17, 18]	High-power consumption	Low-power
	No location privacy	Privacy respected
	Coverage issues	100% coverage
	Reliance on route maps	Route maps not required
	Requires cellular data connectivity	Uses phone-to-phone comms
Challenge 3: Deploying on-the-go apps		
Using multiple/unified app [6]	Cumbersome to install/uninstall/update apps	Easy to add/remove/update apps

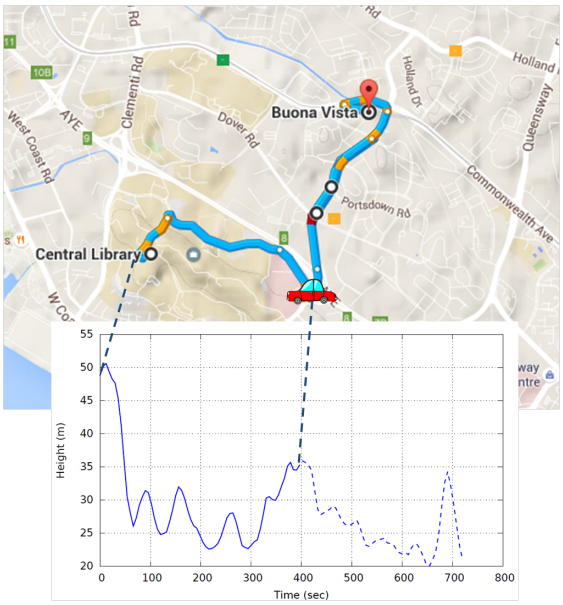


Figure 1.3: Barometer, which provides height variations, can act as a power-efficient alternative to absolute location.

terrain variations along roads when the user travels to the destination, and act as a power-efficient alternative to absolute location, while respecting location privacy. This is illustrated in Figure 1.3, which shows how height variation measured by the barometer can be used to get an idea how 'far' the user is from the destination, even without actual absolute location.

The barometer is unaffected by hand movement, allowing the user to place and

use the phone in any way without affecting the sensor's output. The measured height variations during travel are stable since the terrain of the land does not change often. Due to these reasons, the barometer is a suitable modality to use for location-'less' operation of public-transport applications.

To overcome the absence of a central server or cellular data connectivity (such as in developing countries), we exploit local data available from users travelling in the same bus to provide key application functionality such as bus-stop detection and ETA, that works without prior-access to route maps.

Finally, we have written a framework that uses phone-to-phone communication for easy and flexible on-the-go localized deployment of applications written in Javascript, that can run in the browser with support of the framework for communication and context detection.

Use of the barometer sensor, use of local data collaboration between users in the same bus, and use of a framework for on-the-go deployment, form the basis of our thesis work. Table 1.1 lists the advantages of our system against the limitations of existing work.

Table 1.2 shows the placement of our thesis work w.r.t. related work and key public-transport application features. Compared to related work, our system makes only few assumptions on infrastructure and route map availability, and is hence applicable even to developing countries where other approaches fail to work. In this thesis, we focus on bus-stop countdown and ETA application features, while next bus feature is subject of future work.

To summarize, the contributions of this thesis are as follows:

1. **Barometer-based transportation context detection:** We propose and implement the first research work in literature that uses only the barometer for low-power transportation context detection of the states *idle*, *walking*, and *vehicle*. Unlike existing approaches that have high user dependence and require extensive

training, our barometer-based approach is inherently independent of the user. It has similar detection accuracy while consuming lower power. In the situations where the user is waiting for a bus, and while travelling on smooth vehicles, existing accelerometer-based techniques have less than 25% detection accuracy, while our approach has almost 100% accuracy.

2. **Barometer-based vehicle context detection:** We propose novel techniques for using only the barometer and local collaboration of users in the same bus for bus route and bus-stop detection. Our approach makes only few assumptions on infrastructure and availability of route maps, enabling it to work in even in developing countries where other approaches fail, since it is decentralized and does not require an Internet connection. In developed countries, it provides better location privacy and reduces the smartphone's power consumption.

3. **On-the-go deployment of applications:** We propose and implement a dynamic framework on Android for deploying on-the-go applications written in Javascript to users in proximity of places of interest, such as near public-transport stops. Users are notified of received apps, and can run them in the browser with support of the framework for communication without the Internet. After usage, the application is stopped by closing the browser tab, or automatically when users leave the place of interest. Our deployment framework removes the need for users to install multiple applications beforehand on their phone, and allows users to choose 'specialized' applications tailored to their needs. To illustrate this, we have implemented a simple on-the-go application to help the physically challenged (wheelchair) people board the bus.

We have evaluated our contributions using more than 60 hours of real-world barometer transportation traces from 3 countries and 15 volunteers during their daily commute, as well as using over 900 km of elevation data of 5 cities from Google Maps, and using

Table 1.2: Placement of this thesis work w.r.t. related work and key public-transport application features.

System type	**Bus-stop countdown and ETA feature**	**Next Bus feature**
Location and server-based using route maps	CoopTracking [17], Google Now [8], LTA App [6], Moovit App [7]	
Location-'less' and server-based using route maps	CellID [19, 20], CTrack [21]	Cell ID [20]
Location-'less' and server-'less' without route maps	**This Thesis Work**	Future Work

trace-based simulation of 4 bus routes involving more than 1500 users over 100 days.

1.6 System architecture

Figure 1.4 gives an overview of the architecture of our system. We envision the system as a decentralized set of mobile nodes, where each node is a smartphone carried by a user, capable of sensing, computation, storage, and communication with nearby phones.

Users can either install applications beforehand from the app store, or receive applications on-the-go and open them in the browser. Some applications may have special purposes and target only a subset of users, such as a wheelchair person who would like to check for wheelchair access and inform the bus driver for boarding.

Figure 1.4 shows the components running on each user's phone. At the highest layer are the end-user applications, either native apps from the app store, or Javascript apps deployed on-the-go and running in the browser. At the lowest layer are the components in the mobile operating system providing access to the barometer sensor and phone-to-phone communication.

Our thesis contributions are at the middle layer between the applications and the operating system. They consist of the following sub-systems:

1. **Transportation (travel) context detection:** This sub-system runs the context detection algorithm for the states *idle*, *walking*, and *vehicle* using the barometer, corresponding to our first thesis contribution.

2. **Vehicle context detection:** This sub-system runs the bus route and bus-stop detection using the barometer and phone-to-phone communication, corresponding to our second thesis contribution.

3. **Deployment framework:** This is the framework for on-the-go deployment of applications to users in the public-transport system, running Delay-Tolerant Network (DTN) protocols for communication, corresponding to our third thesis contribution.

The three sub-systems above are not independent, but are coupled with one another. The vehicle context detection (bus route and bus-stop detection) is turned on when the user is in a vehicle, to extract barometer data during the journey. The framework depends on the idle context detection to turn off power-consuming deployment protocols when the user is at home or work, and listen for on-the-go applications only when travelling.

Applications in turn use the APIs provided by the three sub-systems to perform end-user functionality, such as providing a bus-stop countdown, destination alarms, and estimating the arrival time to the destination.

We have implemented all components of our system on Android for power and execution time measurements. Additionally, we have provided the context detection and deployment framework APIs to students of the CS4222 Wireless and Sensor Network course in NUS for use in their course projects. A list of applications developed by students that uses these APIs is available online[3].

1.7 Thesis structure

The remainder of the thesis is organized as follows: Chapter 2 discusses related work, as well as their limitations, providing the motivation for our thesis. Chapter 3 describes

[3] `http://www.comp.nus.edu.sg/~kartiks/nusdtn/cs4222/project.html`

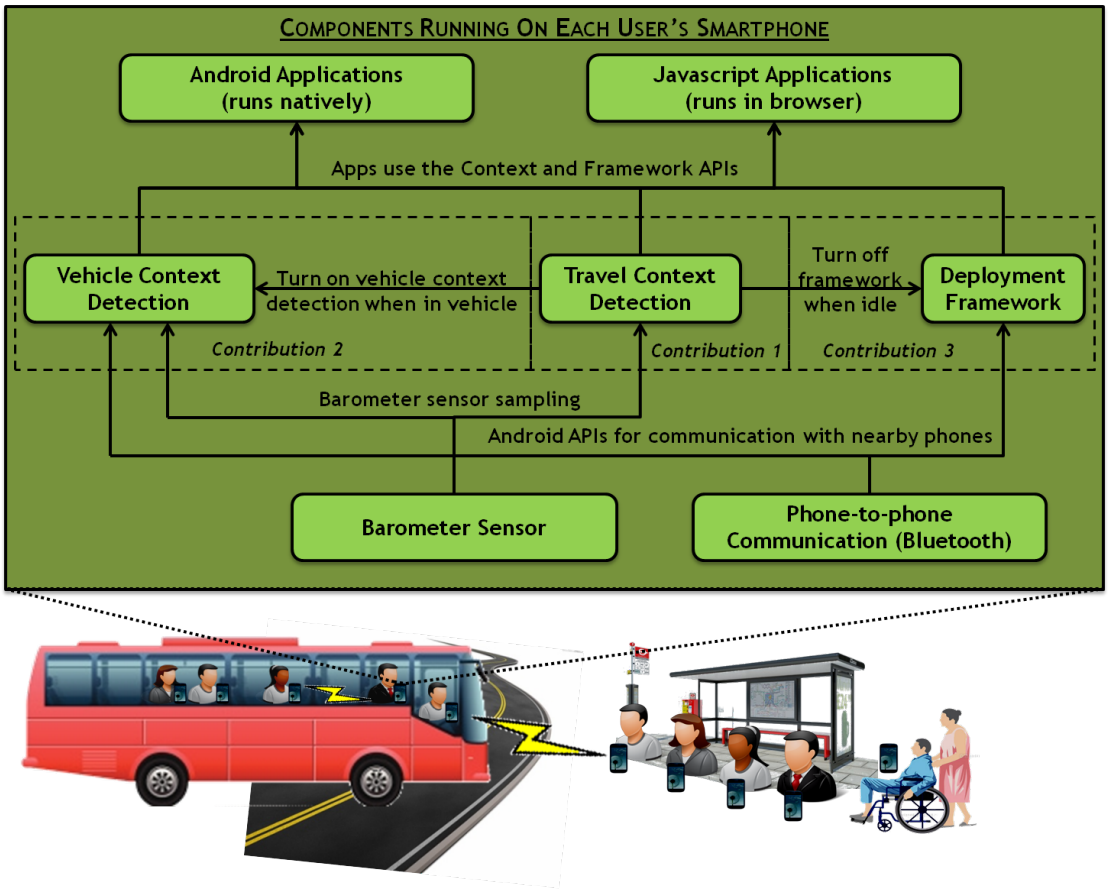


Figure 1.4: Overview of system architecture.

our barometer-based system for low-power transportation context detection. Chapter 4 describes our techniques for bus route and bus-stop detection using the barometer and phone-to-phone communication. Chapter 5 describes the design and evaluation of the dynamic framework for deploying localized transport apps. Chapter 6 concludes this thesis and discusses future work.

Chapter 2

Related work

In this chapter, we discuss existing work in the literature related to the three challenges tackled in this thesis: context detection, bus route and bus-stop detection, and deployment of on-the-go applications. At the same time, we also discuss their limitations.

First, we describe the features provided by typical public-transport applications in Section 2.1. Then, we discuss related works that use smartphone sensors for context detection in Section 2.2. Section 2.3 describes related work in bus route and bus-stop detection. Section 2.4 discusses previous work in web-based applications that use DTN.

2.1 Features of public-transport apps

Smartphone-based public-transport applications provide various features to assist users both before and during their commute. These features can be broadly divided into five categories described below.

2.1.1 Directions

This feature allows users to ask for directions to a particular destination. The suggested route may be static, i.e. based on static bus route maps and timetable schedules, or may

be dynamic, taking into consideration current traffic conditions. This feature is mainly only used before travelling to a new destination.

2.1.2 Estimated Time of Arrival (ETA)

Once the user has boarded the bus, one of the key features of public-transport applications is to provide an estimated time of arrival (ETA), together with a countdown of bus-stops remaining before reaching the destination. They additionally pop-up notifications (or vibrate the phone) to remind users when they have nearly reached their destination.

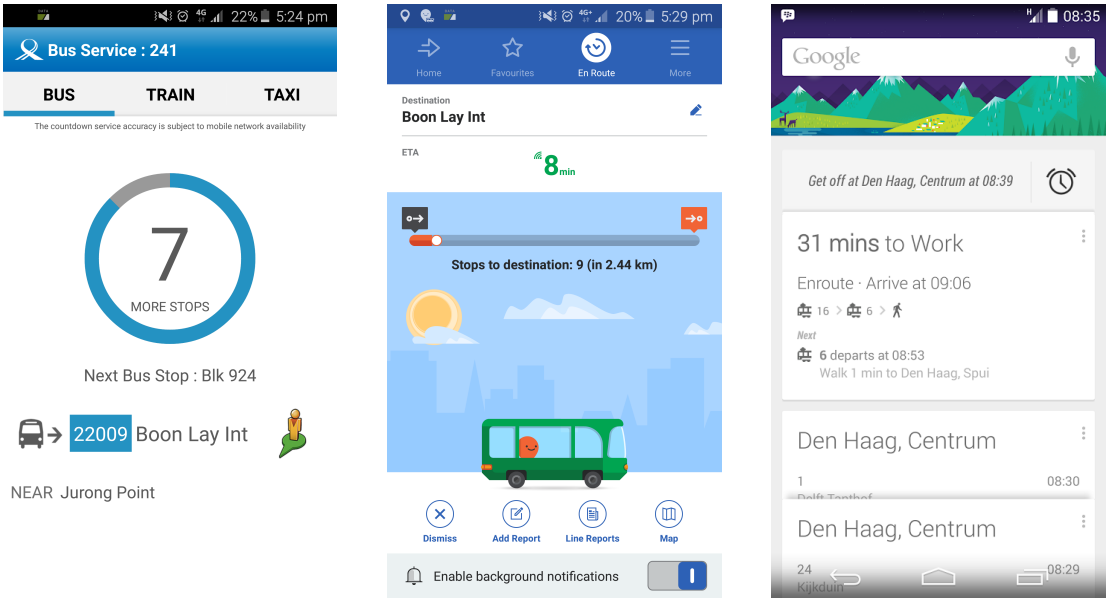
Figure 2.1 shows examples of three popular public-transport applications from Singapore's Land and Transport Authority (LTA) [6], Moovit [7], and Google Now [8], displaying a bus-stop countdown and ETA while users are on the bus. This feature is extremely useful to users travelling on a bus service for the first time, to users having long and tiring bus journeys during their daily commute, and to users who are visually challenged.

2.1.3 Next bus

Another important feature is checking the arrival time of buses while waiting at the bus-stop. This is an extension of the previous feature, in which ETA information is additionally provided to users at bus-stops.

2.1.4 Activity diaries

Another useful feature provided by public-transport applications is automatic activity diaries that record the travel history of the user. Not only are these diaries useful to users, but they are also useful to transport authorities to easily conduct travel surveys to understand user travel patterns and better improve the public-transport system. Moves



(a) LTA App **(b)** Moovit App **(c)** Google Now

Figure 2.1: Public-transport applications by LTA, Moovit, and Google Now, providing bus-stop countdown, estimated remaining travel time, and notifications to users before reaching the final stop.

[22] and Future Mobility Survey [23, 24] are examples of applications that provide activity diaries and conduct travel surveys.

2.1.5 Other services

Public-transport applications may also provide additional specialized applications to users, such as access to road traffic cameras, information about bus wheelchair access, wheelchair navigability maps, availability of nearby cycle rental pitstops, and on-the-spot traveller satisfaction surveys.

2.2 Context detection using smartphones

Research has been done in context detection using sensors for several years, differing in the type of user activities detected, sensors used, and classification techniques. An exten-

sive survey is presented in [25]. The accelerometer is the predominant sensor used, with 28 out of 36 papers listed using it. Other commonly used sensors include GPS, Cellular, WiFi, and other sensors. In the following, we focus on prior work using these sensors, describing their limitations, while Section 3.2 discusses how the barometer overcomes these limitations.

2.2.1 GPS

GPS is an extremely useful sensor for activity detection, since it provides the physical location of the user. A series of GPS readings can be analysed to calculate speed and bearing, typically used in combination with the accelerometer features to achieve higher classification accuracy. Reddy et al. [11] and Ryder et al. [12] use GPS in conjunction with the accelerometer to detect the modes idle/walking/running/bike/vehicle, and observe that GPS contributes to an increase in accuracy of about 10%. The Future Urban Mobility Survey (FMS) application [23, 24] uses GPS and the accelerometer to detect the modes walking/car/bus/MRT/bike, observing an increase of 27% in accuracy over just using the accelerometer sampled at 10 Hz. In contrast, Zheng et al. [10] uses solely GPS data to detect the modes walking/driving/bus/bike, by extracting additional features such as heading change rate, stop rate, and velocity change rate. The trade-off in GPS accuracy is its high power usage and low coverage indoors, underground, and in urban canyons [26]. To reduce power usage, GPS is turned on adaptively rather than periodically, using the accelerometer and/or cell tower change as a trigger. However, the power usage remains high outdoors, where charging sockets are typically unavailable.

2.2.2 Cellular and WiFi

Movement can also be detected from changes in the cellular and WiFi signals. Anderson and Muller [27] use the fluctuation in cellular signal strength to figure out if the user is stationary/walking/driving, as does Sohn et al. [28]. Using signal strength is chal-

lenging, since it can change unpredictably even when the user is stationary. Nawaz et al. [13] use a more robust WiFi beacon reception ratio as opposed to signal strength, to detect if a user has parked or is driving a car. An algorithm called BeaconPrint [29] uses the cellular/WiFi beacon IDs to detect movement, without requiring any signal strength information. Since WiFi has shorter range and the network deployment is denser, change in signal is observed faster than change in cellular signal. However, WiFi-based techniques work only in urban areas with dense WiFi access points. Cellular has better coverage, but the cell size can vary significantly, making cellular-based detection difficult to generalize.

2.2.3 Accelerometer

The lowest-power and most predominant sensor used for context detection is the accelerometer. Features extracted from the accelerometer data are used as input for a supervised machine learning algorithm, which classifies the user activities [30, 31, 32, 33]. Due to classifier complexity, majority of prior work perform the classification offline [14] instead of in real-time. Reddy et al. [11] implement their classifier on Nokia N95 phones, while performing the training offline, as does [34]. To avoid orientation problems, orientation-independent features (such as combined magnitude) can be used [14]. However, extensive training is still required to account for user and phone position dependence. Unlike Cellular and WiFi, the accelerometer is capable of fine-grained classification of vehicular modes. Hemminki et al. [15] implement a three-stage classifier on Android to detect travelling on bus/train/metro/tram/car, at a power consumption of 85 mW (excluding base-power consumption). Our barometer-based approach can be used as a low-power trigger for higher-power finer-grained vehicular classification.

2.2.4 Audio

The audio sensor (microphone) has been used for varied purposes in the literature, such as crowd counting [35] and analysing human conversation networks [36]. Pravein et al. [35] use the high frequency audio tones emitted by the smartphone's speakers to estimate the number of people in a particular area. Chengwen et al. [36] use the microphone to non-intrusively analyze human conversation networks. Audio, while extremely useful, is affected by ambient noise, such as in a bus or in crowded areas.

2.2.5 Light

The light sensor has been mainly used for indoor/outdoor detection. Zhou et al. [37] use the light sensor together with the magnetic and cell ID sensors to detect when the user is indoors or outdoors. However, the limitation of the light sensor is that it is affected by variable lighting conditions, such as cloud cover and presence of windows.

2.2.6 Magnetic

The magnetic sensor is often used in conjunction with other inertial sensors to re-orient the sensor readings in terms of the earth's magnetic axis. It has been used together with the accelerometer in dead reckoning [38, 39], and has also been used for indoor/outdoor detection together with the light and cell ID [37]. Like the audio and light sensors, the magnetic sensor is susceptible to interferences from the environment. Electrical devices in the surroundings have a sharp effect on the magnetic sensor's output.

2.2.7 Barometer

The barometer has been used for aiding GPS [40], which was the initial reason for its introduction into Android smartphones. Tanigawa et al. [41] uses the barometer as an aid in removing the accelerometer drift. Due to its excellent relative accuracy, the barometer has been used for floor-change detection [42, 43]. Stairs and elevator can be

Table 2.1: Smartphone sensors used for context detection and their limitations

Sensor	Limitation
GPS [11, 12, 23, 24, 10]	High Power and incomplete coverage
Cellular/WiFi [27, 28, 13, 29]	Variable cell sizes and low WiFi coverage
Accelerometer [14, 11, 34, 15, 30, 31, 32, 33]	Position dependence and extensive training
Light [37]	Affected by cloud cover and presence of windows
Audio [35, 36]	Affected by ambient noise
Magnetic [37, 38, 39]	Affected by magnetic interference from electrical devices

easily distinguished using vertical speed thresholds. [42] uses a sampling rate of 1 Hz, similar to our work. A higher sampling rate of 15 and 25 Hz can be used to reduce noise [43]. However, since newer barometer chips support internal hardware smoothing, such a high sampling rate is no longer required. So far, the barometer has been used only as an aid to other sensors. To the best of our knowledge, no prior work has used only the barometer for detection of the modes idle/walking/vehicle.

Table 2.1 summarizes the different sensors that have been used for context detection, as well as their limitations. These limitations are not faced by the barometer sensor, since it is unaffected by external factors except air pressure.

2.3 Bus route and bus-stop detection

We classify the related work for bus route and bus-stop detection into three categories: those that use GPS, those that use the cell ID, and those that use other low-power sensors. In this section, we give an overview of work done in each category, while highlighting their limitations.

2.3.1 Traditional GPS-based approaches

Majority of papers in the literature use GPS data [16, 17, 18, 44, 45, 46]. While the papers discussed here have differing (but traffic-related) goals, their methodologies involve techniques for detecting routes and transit-stops.

Co-operative transit tracking [17] uses crowd-sourced GPS data from smartphone users to predict the arrival time of buses. They use prior-available route maps to identify which bus route the user is travelling in. They also use the accelerometer to detect train stops (and hence stations) in the underground subway where GPS is unavailable.

However, GPS is power-hungry, and uploading location data causes concerns about location privacy. Without route maps, identifying bus-stops is challenging. While the accelerometer can be used in vehicles where vehicle stops usually map to stations (such as in subways), this is challenging in buses, where stops may be due to traffic lights or congestion. GPS additionally has low coverage in urban canyons, a problem not faced by the barometer, which can be used even underground.

EasyTracker [16], unlike [17], does not require route maps, but instead uses unlabelled GPS data from buses to construct the map automatically, and identifies bus-stops by spotting 'stop-locations' in the data. The generated map and bus-stops are then used for arrival-time prediction. Similar to our work, the prediction does not require the real-world bus route IDs and bus-stop IDs.

Since [16] identifies 'stop-locations', traffic signals and stop signs may be falsely indicated as bus-stops. In contrast, our approach detects bus-stops by matching the barometer signal data between users in the same bus, and identifies only those points where users get on/off the bus.

VTrack [18] tackles the problems of high-power consumption and low-coverage in GPS by combining it with the alternate lower-power but noisier WiFi to estimate a user's trajectory. They use the detected trajectory and fused locations to discover road segments with congestion, and suggest alternative routes with shorter travel times. While their approach works well in identifying routes, transit-stop detection is not tackled.

2.3.2 Approaches using cell ID

Cell ID, which consumes much lower power than GPS, has also been used for bus route and bus-stop detection [19, 20], and for travel time estimation [21]. Zhou et al. use the cell ID to detect the bus route taken by users and predict bus arrival times [19], as well as to provide the next bus feature [20]. They observe that each bus route can be distinguished by the characteristic sequence of Cell-IDs during travel, and that bus-stops can be distinguished using the set of visible cell IDs.

While their system works well, it has three limitations: First, the system detects the user getting into the bus and subsequent bus-stops using the microphone to detect the 'tap' sound of the transport cards, which are typically unavailable in developing countries. Second, they use a server-based system, which assumes availability of cellular data connection. Third, from our experience of using the cell ID, we have observed that the actual set of cell IDs observed depends on the phone, whether the SIM is inserted or not, on the cellular network type (2G/3G), and on the service provider. These can lead to differing sets of cell IDs depending on the user's phone and SIM.

2.3.3 Approaches using low-power sensors

Recent papers have used sensors other than GPS for route detection (but not for transit-stop detection), such as the gyroscope and battery current.

The gyroscope has been used to identify a user's significant routes [47]. Using the idea that each journey can be identified by a characteristic sequence of vehicle turns detected by the gyroscope's z-axis, the authors are able to distinguish the user's journey in real-time. Similar to our work, this work allows for the gyroscope signal data to be stretched on the time-axis to reduce the effect of congestion. While the gyroscope may work well in a car, it is problematic in public-transport where the user would normally use and move the phone, triggering false vehicle turns. The amplitude of the gyroscope can also vary depending on the sharpness of turns. The gyroscope also would not work

in those parts of the journey with no turns (such as on the highway). The barometer, in contrast, is not affected by hand movement or position (hand, bag, pocket), and can provide distinguishing signal features throughout the journey.

PowerSpy [48] uses the battery current drawn in the smartphone as a distinguishing feature for each route. Observing that the current drawn by the cellular radio depends on the distance to the cell tower, the authors show that the current signal on a route remains similar even across days, and is possibly characteristic to the route. They then use this to estimate the user's location during travel. Unfortunately, the current drawn by the cellular radio alone is difficult to estimate when there are multiple applications running on the phone. Furthermore, the signal would change if the environment near the cell tower changes, for example during building construction or trimming of tree branches. The signal also suffers from hysteresis during cellular handoff, i.e. the signal measured on a road segment depends on the previous road segment (and hence cell) travelled on.

While the use of the cell ID, gyroscope, and battery current consume much lower power than GPS, they suffer from several problems described above, none of which are faced by the barometer sensor.

2.3.4 Subway-specific approaches

Several works are targeted specifically to subways where GPS is unavailable [49, 50, 17]. These works use the accelerometer to detect train stations, which is possible since trains do not stop in between stations. M-Loc [50] is closest to our work since it does not require route maps beforehand, and uses the barometer and magnetic sensors to distinguish the route. Their observation that the barometer signals are unique for different routes matches our observations.

However, these approaches are targeted for trains and subways, and are not applicable to buses, since they rely on detecting the train station using the accelerometer. Transit-

Table 2.2: Smartphone sensors used for route and transit-stop detection

Sensor	Limitation
GPS [16, 17, 18, 44, 45, 46]	High Power and incomplete coverage
Cell ID [19, 20, 21]	Cell IDs depend on phone, SIM, network (2G/3G), service provider
Accelerometer [49, 50, 17]	Not applicable to buses
Gyroscope [47]	Affected by hand movement
Radio Battery Current [48]	Affected by environment and hysteresis

stop detection using the accelerometer does not work for buses which stop for various other reasons (traffic lights, congestion, etc). Subways are less challenging than buses due to their fast speed, fixed stops, and much smaller number of route possibilities. In contrast, our thesis work is applicable to both buses and subways.

The barometer sensor has been used in prior works for aiding the GPS lock [40], and for floor-change detection [42, 43]. To the best of our knowledge, no prior work has used the barometer for bus route and bus-stop detection.

Table 2.2 gives a list of sensors that have been used for route and transit-stop detection, as well as their limitations. The barometer sensor used in this thesis does not face these limitations, since it is unaffected by hand movement and external factors other than air pressure.

2.4 Web-based applications using DTN

Our framework communicates using the Delay-Tolerant Network (DTN) [51] of smartphones carried by users. In this section, we discuss work related to web-based applications that communicate over the DTN, grouped under different categories. By describing limitations of existing work, we also provide a motivation for development of our dynamic deployment framework.

2.4.1 HTTP-over-DTN browsing

Efforts have been made to use DTN for web browsing [52, 53, 54, 55]. These papers concentrate on techniques for better serving browsing requests over DTN, such as bundling of related HTTP requests, pre-fetching, and caching. The underlying DTN is hidden from the webpages.

Our framework focuses on deploying DTN web *apps*, as opposed to web pages. Web apps are similar to mobile apps: they are self-contained, i.e. they contain all the scripts and web pages required for the app to work. Also, DTN web apps are fully aware of the underlying DTN, using the DTN API exposed by our framework.

2.4.2 Web-based DTN apps

Web apps such as Facebook and blogging have been written to use DTN [56, 57]. While these apps are 'DTN-aware', the work concentrates on how the apps work using DTN, and does not support localized dynamic deployment of web apps and protocols on-the-go.

2.4.3 PhoneGap

PhoneGap is a framework for creating cross-platform mobile apps using web technologies. Each web app runs in the PhoneGap container, which is essentially a 'super-browser': apps can access phone details (such as user contacts) via PhoneGap, normally not accessible to regular web apps. PhoneGap apps, while written in Javascript, are installed just like native apps. The advantage is that several app code versions are not required for different mobile platforms.

However, since the app must be installed like a native application, installation of PhoneGap apps do not meet the lightweight and convenience requirements of locally deployed applications. In addition, they lack access to DTN APIs.

2.4.4 QR codes

QR codes are useful to direct mobile users to web pages online by scanning codes using their camera. While these codes are convenient to post near places of interest, they require users to look for and manually scan the codes. Discovering web apps is not 'automatic' like in our framework.

2.4.5 Upcoming HTML5 APIs

Several APIs are being developed in order for web-based apps to access functionality on smartphones that was not possible before, such as access to sensors [58]. New APIs under the umbrella term 'WebRTC' are being developed for real-time communication between web browsers [58], including audio, video, and data. However, WebRTC is meant for real-time communication, and is not suitable for phone-to-phone communication between phones without Internet in proximity applications.

Our framework provides DTN APIs to applications for communication. Since it is dynamic, it enables applications to modify and use their own protocol stacks suitable for their needs (written in Java), bundled together with the application.

2.4.6 DTN middleware for smartphones

Several middleware have been written on mobile for development of DTN applications. Table 2.3 provides a list of existing middleware, along with the type of API exposed, and a brief description of each. To the best of our knowledge, these middleware do not expose their API to web applications (with an exception of Bytewalla, discussed below), limiting their use to native mobile applications only.

In addition, unlike our framework, these middleware are static, i.e. the underlying protocols are fixed at compile-time and shared by multiple applications. It is not possible to load and unload protocols on-the-fly, a feature required by our 'use-and-discard' web applications.

Table 2.3: Existing DTN Frameworks

Framework	API exposed to developers	Brief Description
Haggle [59]	Publish-Subscribe API (attribute-based)	Uses a search-based data-centric protocol
Mist [60]	Publish-Subscribe API (topic-based)	Uses a reliable broadcast with fragmentation
MaDMAN [61]	Sockets API	Switches between TCP/IP and DTN protocol stack
ubiSOAP [62]	Service-Oriented API	Floods WSDL files and SOAP messages
MobiClique [63]	Social-Networking API	Built on top of Haggle
DoDWAN [64]	Publish-Subscribe API (attribute-based)	Floods WSDL files and SOAP messages (with attributes)
Bytewalla [65]	Bundle Protocol API	First implementation of the Bundle Protocol on Android

2.4.7 Service-adaptation middleware

The work in [66] proposes a middleware that acts as a bridge between DTN apps written in different languages and DTN bundle service daemons running on different platforms. Bytewalla is the daemon running on Android, while PCs run the DTN2 service daemon. This middleware enables web applications to access DTN. However, like the web-based apps discussed earlier, it does not support localized deployment of web apps and protocols on-the-fly.

2.4.8 Dynamix

Dynamic frameworks are quite popular in the context-aware computing domain [67, 68]. In particular, a framework called Dynamix [68] provides context-awareness to web applications, by means of context-awareness components loaded at run-time. Architecturally, this framework is closest to our framework.

Although architecturally similar, Dynamix focuses on context-awareness, and its APIs are oriented around receiving 'context events'. In contrast, our framework's (DTN) APIs are communication-oriented. Dynamix's context-aware components are self-contained, while our protocol components are linked in the form of protocol stacks for each application.

Writing code to handle dynamic loading and unloading of components is complicated and error-prone when components are inter-dependent. Our framework handles this dynamism automatically: for example, the app is transparent to a change in the routing

protocol, and the routing protocol is transparent to a change in the link layer protocol.

To summarize this section, existing work have limitations with respect to the requirements of lightweight and convenient locally deployed DTN web applications. Our framework has been designed to address these limitations and make such dynamic DTN applications possible.

Chapter 3

Barometer-based transportation context detection

3.1 Introduction

With smartphones now reaching the computational power of personal computers, they are expected to behave intelligently: they should silently understand what the user is doing, help in ongoing or future tasks, and adapt accordingly. Google Now (July 2012) and Siri (Oct 2011), for example, behave as intelligent personal assistants. Cover [69], an Android application released in Oct 2013, automatically adapts the applications displayed on the lockscreen based on whether the user is at home/work/travelling. Google utilizes the user's activity and movement to improve the quality of location readings [70].

A key pre-requisite of such intelligent smartphone behaviour is context-awareness. The phone needs to continuously understand what the user is doing. Context is typically derived from the multitude of sensors on the phone. Since the phone's battery life-time is critical, context-detection algorithms must run at extremely low-power. In this regard, Apple and Google have taken the first steps forward to reducing power consumed for walking and step detection, by introducing the M7 co-processor [71] (Sept 2013) and

step counter [72] (Oct 2013) to offload sensor processing from the main CPU when the phone is asleep.

Transportation (or travel) context detection is a special case of context-awareness where the phone automatically understands the user's daily commute. Such awareness is immensely useful for maintaining activity diaries, conducting surveys, and urban planning. For example, Moves [22], the first to make use of Google's Activity Recognition API [9], is a popular application that maintains an activity diary for the user, and was featured in the Google I/O session in May 2013 when the API was introduced.

Being one of the lowest power sensors available on the phone, the accelerometer is the predominantly used sensor in transportation context detection [25]. It can detect acceleration in the phone's three axial directions. Using supervised machine learning and features extracted from the accelerometer readings, user activity can be classified as *IDLE*, *WALKING* or *VEHICLE*.

Although the accelerometer is low-power, its readings are phone orientation and position dependent, as well as user and vehicle dependent. The machine learning algorithm needs to be trained for all these different possibilities. To detect walking and vehicle activities accurately, sampling rate is typically 10 Hz and above. The high sampling rate, 3 axial directions, and position dependence make the classification complicated and increases power consumption (Section 3.5.4).

We present an alternative approach to context detection using only the barometer, a sensor now found in an increasing number of devices today[1], which measures the air pressure. Pressure can in turn be translated to altitude (height above sea level). Barometers were initially introduced on Android phones to reduce the delay of the GPS fix by providing the z co-ordinate (altitude). Since the MEMS sensor is sensitive enough to measure even a 1 metre change in height, they are also used for floor-change detection, and can differentiate between travelling on stairs/elevator. Some applications

[1]Nexus 3/4/5/6, Galaxy S3/4/5/6, Galaxy Note 1/2/3, iPhone 6, and many more

are attempting to crowd-source pressure data for weather forecasting [73, 74]. Fitness applications use change in height to better estimate calorie consumption.

We present the first work that uses only the barometer for transportation context detection. The barometer is inherently orientation and position independent. Using a low sampling rate of 1 Hz, coupled with the new sensor batching hardware, and using relatively simple processing based on intuitive logic, we demonstrate that the barometer can be used for basic context detection of the user activities *IDLE*, *WALKING*, and *VEHICLE* at extremely low power. These states are sufficient to characterise typical transportation context, and can act as a trigger for other higher-power finer-granularity classification of vehicular modes.

We evaluate our approach using 47 hours of transportation traces from 3 countries and 13 individuals. We compare the accuracy and power consumption to two other approaches: Google's accelerometer-based Activity Recognition algorithm [9], which runs at low power, and Future Urban Mobility Survey's (FMS) GPS and accelerometer server-based approach [23, 24], an app developed by the Singapore-MIT Alliance for Research and Technology (SMART), in conjunction with the Singapore government's Household Interview Travel Survey (HITS). FMS is the first smartphone survey app to have been field tested on a large deployment of over 1000+ users in Boston and Singapore, designed as an alternative to traditional surveys done via in-person interviews in HITS.

We chose FMS and Google as baselines due to their wide deployment and usage. In our evaluation, we find that our barometer-based approach consumes 26% lower power in comparison to Google, and has comparable accuracy to both Google as well as FMS.

In addition to real-world trace data, we have also evaluated our algorithm using over 900 km (30,000 data points) of elevation data available from Google Maps from 5 cities. The accuracy results from the map data are similar to the real-world trace data, and provide convincing evidence that sufficient elevation changes do occur in practice for the barometer to detect the user's context.

Table 3.1: Limitations of existing sensors for low-power activity detection

Sensor	Power	Limitations	Barometer advantage
GPS	Very high	Lack of indoor/underground coverage High power usage	Usable everywhere Ultra-low-power
WiFi/Cellular	High/Moderate	Requires dense access points/cellular towers	No external infrastructure
Accelerometer	Low	Extensive training required Classification complexity Position dependence	Simple calibration based on terrain Simple processing Inherently position independent

We have implemented our approach on Android. It runs in real-time on the phone locally, without requiring an Internet connection.

This chapter is organised as follows: Section 3.2 describes the motivation of using the barometer for context detection. A brief background is given in Section 3.3, and the context detection methodology is described in Section 3.4. Section 3.5 evaluates our approach. This is followed by a discussion in Section 3.6, and Section 3.7 concludes the chapter.

3.2 Motivation

Prior work have used multiple sensors including GPS, Cellular, WiFi, and the accelerometer for context detection. Then why it is advantageous to use the barometer for the same purpose?

Although several sensors are available, each has its own set of limitations in low-power activity detection, listed in Table 3.1. GPS consumes high power, and has poor coverage indoors and underground. In contrast, the barometer sensor is one of the lowest powered sensors on the phone, and can be used everywhere. WiFi and cellular-based approaches are better than GPS power-wise, but do not work without sufficient density of access points and cell towers. The barometer, on the other hand, is not dependent on any external infrastructure.

The accelerometer, like the barometer, is low power, and not dependent on external infrastructure. Consequently, it has become popular for context detection. However,

even the accelerometer has drawbacks. By nature, the accelerometer data is dependent on the phone's position (is the phone in a bag, pocket, or hand), and its orientation. Additionally, the readings vary from user to user, and vehicle to vehicle. Every user handles a phone differently, and different vehicles may produce different vibrations in the phone while moving. These dependencies can be offset by using orientation-independent features and training the machine learning algorithm with each dependency case. However, addressing these dependencies adds to the cost and complexity of the system, increasing the power consumption. Majority of prior accelerometer-based work implement the classification offline instead of on the phone due to this reason.

As we demonstrate later, the barometer is inherently position independent, requires simple processing and only minor calibration based on the terrain, overcoming the drawbacks of the accelerometer.

To summarize, the barometer has the following advantages:

- **Position independence:** The barometer measures air pressure, and is inherently position independent, as long as the phone is not kept in an air-tight environment.
- **Simpler calibration:** The accelerometer depends on the position, orientation, user, and vehicle, and requires sufficient training to work well in all cases. The barometer reduces dependencies drastically: it only requires calibration of a few parameters using the overall characteristics of the terrain of the land, which remains relatively unchanged over time.
- **Better *WAIT* detection:** In transportation applications, one of the important aspects of the journey is the waiting time. While the accelerometer is excellent for detecting when the phone is stationary, it faces problem when the user fiddles or makes minor movements with the phone, triggering false positives. This is especially problematic when the accelerometer is used to trigger higher power sensors like GPS. The barometer, unaffected by phone movements, yields fewer false pos-

itives compared to the accelerometer for user movement (Section 3.5.1 compares the accuracy of the accelerometer and the barometer for the *WAIT* state).

- **Lower-power:** The barometer's lower sampling rate and simpler processing reduces the power consumption. Accelerometer-based approaches typically require a sampling rate of 10 Hz or higher, consuming higher power.

The accelerometer data in 3 axial directions is both a boon and a bane. It provides more information, but complicates processing. This work ultimately tries to answer the question: Is one-dimensional height data more useful than three-dimensional accelerometer data? In other words, can more be done with less data?

3.3 Background

In order to better understand the barometer sensor's characteristics, strengths, and sources of error, it is constructive to look at the MEMS barometer at a deeper level.

Popular MEMS barometers are of the piezoresistive type. They consist of a thin diaphragm over a small air cavity of near vacuum pressure [75]. Piezoresistors are arranged on the diaphragm in the form of a wheatstone bridge circuit (Figure 3.1). The external atmosphere exerts a pressure over the diaphragm, causing it to get depressed into the air cavity. This deflection causes a change in resistance of the piezoresistors, which in turn changes the voltage output of the wheatstone bridge. Using two or more calibration points, the change in voltage can be translated into corresponding air pressure values in millibar. The entire MEMS package is extremely small (for example, Bosch's BMP280 barometer on Nexus 5 measures 2 x 2.5 x 0.95 mm).

Possible sources of error in the barometer are as follows:

- **Vibration:** One would expect that a phone movement or vibration would cause a major deflection in the diaphragm, leading to unpredictable pressure values. How-

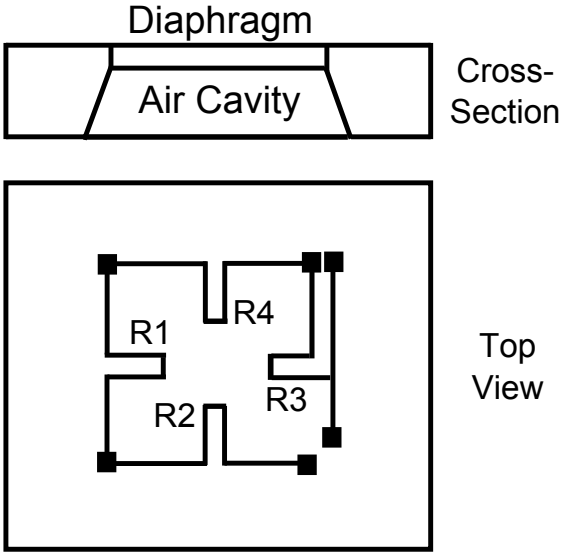


Figure 3.1: Simplified cross-section of MEMS piezoresistive barometer (adapted from [76])

ever, this is not the case. We conducted an experiment where we vibrated a Nexus 4 phone in the phone's 3 axial directions (one by one) using strong approximately 1 cm oscillations. However, this did not cause a change in the noise value (1 metre without smoothing), nor cause outliers. This is perhaps due to the small size of the diaphragm (in contrast, the MEMS accelerometer uses tiny rods in order to amplify vibrations).

- **Temperature:** A temperature above or below the room temperature (25 deg) causes a change in the resistance of the piezoresistors, leading to errors. Earlier barometer chips used a thermistor to compensate for this temperature error. Current chips contain a temperature sensor bundled into the package. The driver reads both the pressure as well as the temperature, and compensates for the error in software. Since the errors are usually second order or higher, a quadratic compensation is more effective than a linear compensation [77]. Since the barometer driver is part of the Android Open Source Project, we can check which phones perform the necessary temperature compensation (Table 3.2). Our experiments with

Galaxy S3 show us that the error caused by using a linear instead of a quadratic correction is small, and does not affect our context detection algorithm.

- **Installation bias and aging drift:** Installation bias (offset caused by soldering) is taken care of at the end of the phone's production line. Aging drift, which causes a drift in absolute pressure values as the barometer chip grows older, is in the order of months, and does not affect our algorithm which runs at a small time scale of a few minutes.

- **Weather drift:** Change in weather can cause a change in the air pressure, and consequently a change in the calculated height, even when the phone is still. Typical weather drift is a few metres in an hour, but intense storms can cause a drift of 3 to 4 metres even in 10 minutes. We discuss and evaluate weather effects in more detail in Section 3.5.1.

- **Sunlight and wind:** The diaphragm and resistors, if exposed, will be affected by sunlight and wind. However, the barometer is protected under the phone's outer case from direct light. The MEMS package contains only a tiny air hole to capture air pressure, protecting it from wind. This matches our observations in windy weather (section 3.5.1).

Barometer chips come with the capability to internally oversample and smooth pressure values to reduce noise. Our inspection of the driver code, and the chip specifications, tell us that the chips are already configured at optimized settings on Android. Table 3.2 summarises the chips found on popular phones.

Air pressure (in millibar) can be translated into height (in metres) above sea level. The absolute accuracy of the height depends on the sea level reference pressure used, and the time of the day. In other words, for the same reference level, the barometer can very well report significantly different altitudes at different times of the day. To get an accurate estimation of altitude, the mean sea level pressure for the phone's region

Table 3.2: Summary of barometer chips on popular phones

Baro Chip	Phones	Temp Sensor	Temp Compensation	Oversampling	Noise filter
LPS331AP (STM)	Galaxy S3	Yes	Linear (on chip)	Yes	No
BMP180 (Bosch)	Galaxy Nexus/S4, Nexus 4	Yes	Quadratic (in driver)	Yes	No
BMP280 (Bosch)	Nexus 5	Yes	Quadratic (in driver)	Yes	Yes

needs to be fetched from a local weather website at that time of the day, and used as a reference.

However, our context detection algorithm does not require absolute accuracy, but rather good relative accuracy. Barometer chips on newer phones are sensitive enough to measure a change in the height of even 1 metre, the reason why it is so useful for floor-change detection. The barometer's good relative accuracy is a strength exploited by our algorithm. Note that the height resolution is limited by the noise, which is approximately 1 metre without filtering. Nexus 5's chip, which performs internal filtering, has a lower noise value than other chips.

The study in [43] tests whether the change in height (i.e. relative accuracy) varies on different phones. Although different phones can report different absolute height values, the change in height values on different phones while moving are in sync.

In summary, by looking deeper into the MEMS barometer, we find that the main source of height error is the weather drift. We will evaluate the impact of weather drift in Section 3.5.1.

3.4 Methodology

In this section, we describe how exactly we use the barometer to detect the states of *IDLE*, *WALKING* and *VEHICLE*.

3.4.1 Activity definitions

Before describing our methodology, we need to first define the meaning of each state to avoid ambiguity. The state *VEHICLE* includes both motorised and non-motorised vehicles (including cycling). The state *IDLE* is not as strict as the typical definition in accelerometer-based works, since the barometer is unaffected by hand movement. If a user moves around in the same room or floor of a building, we still consider it as an *IDLE* state. We argue that in the context of transportation, such movements should be clearly differentiated from the *WALKING* and *VEHICLE* states since these movements include important transportation context such as waiting at bus stops, taxi stands and subway platforms. This definition also yields fewer false positives compared to the accelerometer when movement is used to trigger high power sensors such as GPS.

The states *IDLE*, *WALKING*, and *VEHICLE* are sufficient to characterise typical transportation context. Several popular applications such as Cover [69] and Moves [22] already make use of these fundamental three states. The *VEHICLE* state can also act as a trigger for other higher-power finer-granularity classification of vehicular modes. Similarly, additional sensors can be utilized to differentiate between stationary and waiting for transport.

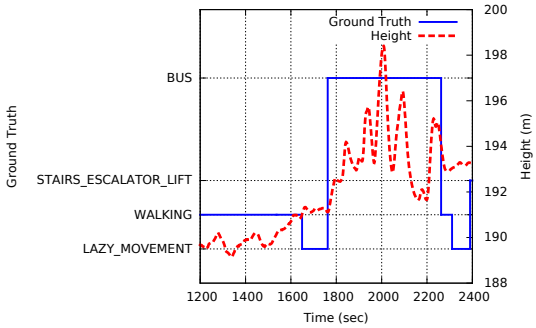
3.4.2 Intuition behind barometer context detection

This section describes the intuitive logic behind barometer-based context detection.

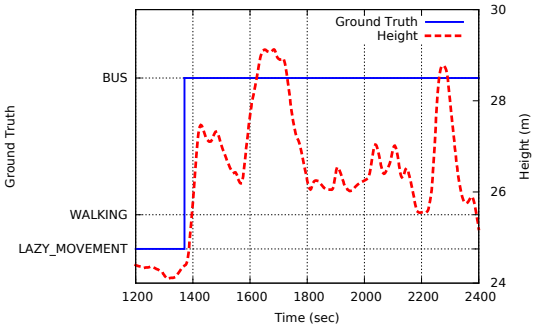
Roads are not perfectly flat. Their height changes slightly even when it is not visually obvious to the naked eye. The barometer is sensitive enough to measure this change in height when a vehicle moves along a road. Vehicle detection is based on the intuition that vehicles, because of their higher speed, tend to see more ups and downs and more rapid height changes, than walking in the same period of time (Note that vehicle bumps and jerks *do not* cause significant changes in height).

The graphs in Figure 3.2 explain this idea pictorially. The graphs are examples of

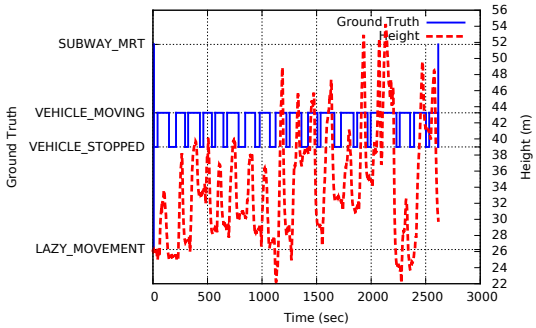
traces we collected, that plot the variation in the height value and the user's ground truth versus time (In the graphs, the state *LAZY* is the same as *IDLE*). As can be observed, the number of ups and downs and the rate of height change increase considerably when the user is in a vehicle (Figures 3.2a, 3.2b and 3.2c). On the other hand, a user walking about in the same floor of a mall (considered as *IDLE* by our definition) sees very little change in height, and no ups and downs at all (Figure 3.2d). A user who is walking on a road does see change in height, but however does not experience the large number of ups and downs as in *VEHICLE*. The height variation between *WALKING*, *IDLE*, and *VEHICLE* can be best observed in Figure 3.2a.



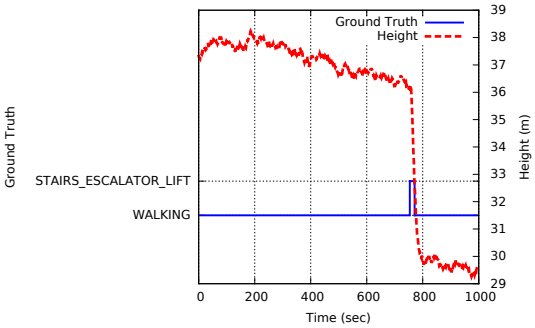
(a) Commute on a bus in Boston



(b) Commute on a bus in Singapore



(c) Subway ride in Singapore



(d) Walking between floors in a mall

Figure 3.2: Some examples of height variation against time for different user states

To summarise, context detection using the barometer is based on the intuitive logic that users in vehicles see more rapid changes in height, including larger number of ups

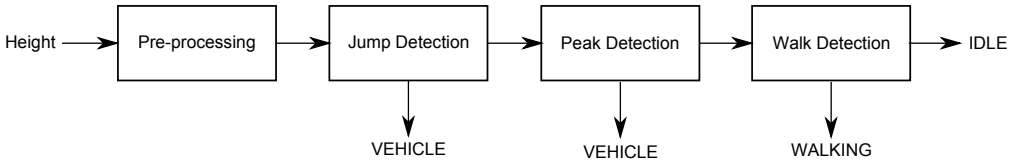


Figure 3.3: Overview of barometer-based transportation context detection

and downs. Users who are walking see a gradual change in height. Users who are idle do not see any change in height.

3.4.3 Overview of the context detection algorithm

Figure 3.3 gives a high-level overview of the barometer context detection algorithm. It consists of four stages: Pre-processing, jump detection (Jumpdet), peak detection (Peakdet), and walk detection.

Jumpdet and peakdet are jointly responsible for *VEHICLE* detection. Based on the order of the stages, it can be seen that a determination of the *VEHICLE* state overrides a determination of the *WALKING* state. This is due to the fact that the barometer is better at detecting vehicles than it is at *WALKING* (section 3.5.1).

The following sections discuss each stage of the algorithm in order.

3.4.4 Pre-processing

The barometer is sampled at a frequency of 1 Hz. Phones like the Nexus 4 return values at a higher sampling rate (typically 4 Hz), in which case the sampling rate is clamped to 1 Hz in code. In phones like the Nexus 5, due to internal smoothing in hardware, the values are returned at a lower rate, usually about every 2 seconds. In this case, linear interpolation is used to convert the actual rate into 1 Hz.

The pressure values returned by the barometer driver on Android is in millibar. This

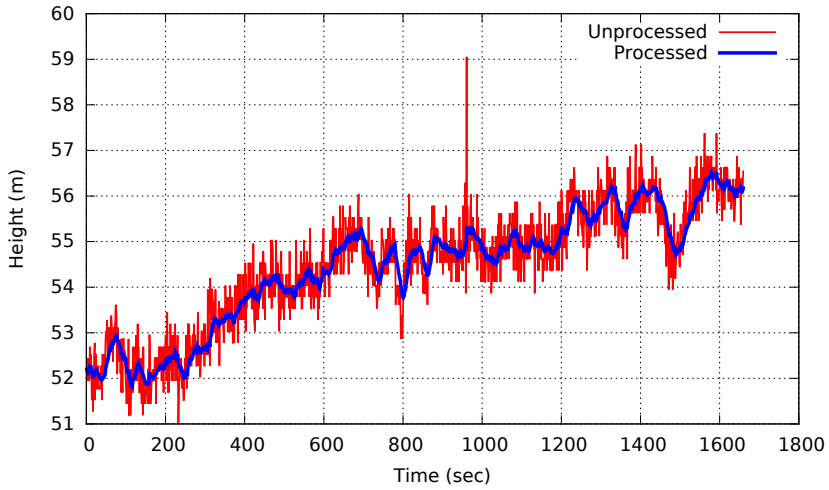


Figure 3.4: Barometer data after processing

is converted to height in metres using the standard pressure-to-height formula [78]:

$$h = 44330 * \left(1 - \left(\frac{p}{p_0}\right)^{\frac{1}{5.255}}\right)$$

where h is the altitude in metres, while p and p_0 are the measured air pressure and sea level reference pressure respectively in millibar. This conversion of pressure into height is available as a helper method in Android. The sea level reference does not matter in our algorithm, since we only use relative height change.

The height values are typically noisy (1 metre noise on average). They are smoothened using a simple filter:

$$currentHeight = \alpha * sensorHeight + (1 - \alpha) * prevHeight$$

where α is 0.1. On Nexus 5, the barometer chip performs smoothing internally, and is not required in code. Figure 3.4 shows an example of the barometer sensor data after it has been smoothened.

3.4.5 Vehicle detection

The following two sections describe the logic used to detect the *VEHICLE* state, namely Jumpdet and Peakdet.

Jump detection (Jumpdet)

Vehicles that move at a high speed, or vehicles that move on highly sloped roads tend to observe a large rate of height change. In other words, they undergo 'jumps' in height, both in the up and down direction. We can detect such vehicles using what we term as 'Jump Detection' (Jumpdet).

We define a 'jump' as a height change of more than 0.8 metres in 5 seconds. For every height reading obtained from the barometer (1 Hz), we check if there is jump in height, by calculating the difference with a height reading obtained 5 seconds earlier. We keep track of the number of jumps observed in a 200 second sliding window, along with their sign (up or down).

Our experiments have shown us that walking does not cause height jumps, even on sloped roads, since the speed of walking is limited. An exception to this is when a person takes the stairs/escalator or the elevator, all of which make it easier for a person to gain/lose vertical height with less effort. We can distinguish this from vehicle using the intuition that vehicles move both up and down, unlike a person that moves *only* up or *only* down the stairs/elevator.

When we observe that the ratio of the number of positive to negative jumps in the sliding window is in the range [30%, 70%], we classify the movement as vehicle. For the ratio to be meaningful, we calculate it only when the total number of jumps is more than 10. By capping the ratio to [30%, 70%], we ensure that we are seeing both positive as well as negative jumps in the sliding window of time, before deciding that the user's state is vehicle.

Peak detection (Peakdet)

Vehicles may move at a slower speed on roads that are not highly sloped, due to slow traffic or frequent stops. In such cases, Jumpdet fails to work since the rate of height change is not large enough. Instead, we detect such vehicles using what we term as 'Peak detection' (Peakdet).

Peakdet is based on the intuition that vehicles, due to their higher speed, observe a larger number of ups and downs in a given period of time compared to a person on foot. Every height reading from the barometer is fed through peak detection, a simple but online signal processing algorithm that can detect peaks and valleys of specified vertical size in a signal (in our case, we set the size to 1 meter).

While roads can be hilly, they are not undulating, i.e. the distance between ups and downs in a road are large and not small. Our experiments have shown us that a person on foot takes more than 200 seconds to cover the distance between undulations on highly sloped road, due to the large distance between them and slow walking speed.

Peakdet keeps track of the number of peaks and valleys in the height signal in a 200 second sliding window. If the number of peaks is more than 1, then the movement is classified as vehicle.

3.4.6 Walk and Idle detection

A user who is walking would see gradual rather than rapid changes in height (exceptions are while travelling on the stairs/escalator/elevator). The detection of *IDLE* and *WALKING* is based on the fact that the height varies while walking, but is stable while idle.

Our algorithm calculates the stddev value of the height in a 200 second sliding window. When the stddev rises above a threshold (0.3 metres), we classify the movement as walking, and idle otherwise.

Due to the weather drift, the height tends to change over time, even when the user

is idle. However, weather drift occurs over a larger time scale, and does not affect our algorithm which runs on a time scale of 200 seconds. Section 3.5.1 analyses the affect of extreme weather on our algorithm.

3.4.7 High-level stitching

The vehicle detection described in the previous sections leads to a fragmented output due to long vehicle stops (for example at traffic lights). These fragments need to be stitched together to capture the transportation context at the user's level. We employ a simple stitching algorithm: Two vehicle detections occurring less than 2 minutes apart are stitched together. This simple algorithm stitches the vehicle states together into a transportation journey more meaningful to the user and applications. To avoid false vehicle detections while idle or walking, vehicle state detections of less than 30 seconds duration are ignored.

3.4.8 Choice of thresholds and window sizes

In our algorithm, we use four configuration values, namely: the jump threshold (0.8 metres), the peakdet vertical height threshold (1 metre), the walking stddev threshold (0.3 metres), and the sliding window size (200 seconds). These values are characteristic of the overall terrain of the land.

For our algorithm, we have chosen the configuration values experimentally based on 10 hours of barometer traces collected before our evaluation's traces. In our evaluation using real-world traces and map elevation data, we find that the same configuration values however work well even in places with different terrains (Section 3.5), and find that the algorithm performs well even without re-calibration.

3.5 Evaluation

We evaluate our work based on the accuracy, power consumption, and latency, comparing it with two other approaches:

1. **Google Activity Recognition:** This is an accelerometer-based context detection algorithm implemented by Google [9]. Released in May 2013, it is part of the Google Play API, and is capable of detecting the modes *IDLE*, *WALKING*, *VEHICLE*, and *CYCLING*. Since it runs at very low power, we use this as a baseline for power consumption. The activity detection runs at an update interval specified by the application developer. Unless otherwise specified, in our evaluation the update interval is set to 10 seconds, since that is the highest frequency possible. We have used the Activity Recognition algorithm present in the Google Play Services version 4.3, which was the latest version available in March 2014 when the trace data was collected.

2. **Future Urban Mobility Survey (FMS) App:** This is a GPS and accelerometer-based context detection implemented by the Singapore-MIT Alliance for Research and Technology (SMART) group in Singapore, in conjunction with the Singapore government's Household Interview Travel Survey (HITS). They have developed a smartphone application for iOS and Android, called FMS, to conduct household travel surveys as an alternative to the traditional HITS approach of conducting in-person interviews. It is the first survey application to be deployed and tested on a large scale, on over 1000+ users in Boston and Singapore. Since volunteers took both the FMS and HITS survey, the accuracy of the app has been thoroughly validated using the ground truth. The FMS application works by uploading sensor data to the FMS server, which in turn runs the context-detection using machine learning. Users can access the FMS website to validate the travel information. To reduce power consumption and minimize data upload, the FMS application duty

Table 3.3: Summary of collected sensor trace data

Country	Volunteers	Total hours	Vehicle hours	Walking hours	Idle hours
Singapore	7	15	6.5	6.4	2.1
Boston (USA)	6	55.95	3.75	7.8	44.4
China	1	108.5	22	1.5	85

cycles even the accelerometer, and uses a very low sampling rate of 2 Hz when it is turned on. The accelerometer in turn is used as a trigger for the GPS, which is sampled at 1 Hz, and duty cycled even during the vehicle journey.

Our evaluation is based on 178 hours of barometer traces (of which 47 hours are transportation journey traces) collected from 3 countries with the help of 13 volunteers. 15 hours of data was collected from Singapore by 7 volunteers, while 10 hours of data was collected from Boston (USA) by 6 volunteers. In addition, 22 hours of data was collected from a cross-country train in China by a single individual. During the data collection, the barometer sensor data was logged to the `sdcard` of the phone at a frequency of 1 Hz. For Singapore, volunteers additionally collected context output from the Google algorithm and the FMS application.

Table 3.3 provides a summary of the traces collected from each country. There are more than 32 hours of vehicle activity and 15 hours of walking activity in total. As volunteers in Boston often leave the apps running on the phone for a long period of time while they are in office after they have completed their journeys, there are more than 40 hours of idle or stationary activity in this set of traces. This was also the case in China, with 85 hours of idle activity.

Volunteers were instructed to run the apps while carrying phones during their daily commute. No special instructions were given on how to carry the phones, and none of the journeys were decided beforehand. Volunteers recorded the ground truth by pressing buttons on the phone. Ground truth included activity at the user's level; low-level ground truth such as vehicle stops and minor walking stops were not logged. Use of

this high-level ground truth yields a more realistic analysis of performance of the activity detection algorithms at the user's level.

In addition to these journey traces, we have also collected additional barometer traces to analyse the effect of weather drift (Section 3.5.1), and to compare the accuracy of our barometer-based algorithm with Google's algorithm in a special case of the *IDLE* state (waiting) in Section 3.5.1, and in a special case of the *VEHICLE* state (subway) in Section 3.5.1.

The phones used by volunteers include Nexus 5, Nexus 4, Galaxy S3, and Galaxy S4, all updated to Jelly Bean. Our barometer-based detection algorithm processes the collected barometer data via a discrete-event trace-based simulator. The simulator is written such that the barometer algorithm code written in the simulator can be directly copied into the phone's application code. Since the simulator is deterministic and operates at a relatively large time scale (1 second) compared to processing, the output of the simulator matches that of running on the phone. Several traces have been checked to see that this is indeed the case, to ensure simulator correctness.

The Google and FMS algorithms are not simulated. The output of Google algorithm's detection is logged into a file in the phone, while the FMS detection results are available on the FMS website in the 'Activity Diary' of each user.

3.5.1 Accuracy

Accuracy is calculated by splitting each trace's timeline into intervals, and checking whether the context determined by the three algorithms matches the ground truth in each interval. The accuracy is calculated as the fraction of the total number of intervals that the user activity is correctly identified. An interval size of 200 seconds was chosen to effectively eliminate intermittent short-duration states (our accuracy calculation discards any interval where there is a transition in the ground truth).

For fairness of comparison, we have implemented a majority voting scheme for

Table 3.4: Barometer algorithm versus Google (accelerometer) and FMS (GPS+accelerometer) algorithms in Singapore

	Baro	**FMS**	**Google**	**GoogleSmooth**
Idle	76%	33%	76%	76%
Walking	54%	46%	79%	91%
Vehicle	81%	90%	31%	34%
Overall	69%	68%	56%	62%

Table 3.5: Barometer algorithm versus Google (accelerometer) algorithm in China

	Baro	**Google**	**GoogleSmooth**
Idle	99%	97%	98%
Walking	23%	40%	50%
Vehicle	78%	24%	25%
Overall	93%	82%	83%

smoothing of the Google algorithm's fragmented output using a window size of 200 sec (this gave better results compared to smaller window sizes). In this chapter, we refer to the smoothened Google algorithm as 'GoogleSmooth'.

Since all three algorithms are only available for Singapore, we will first present the accuracy comparison for this set of traces. Table 3.4 summarizes the results of the accuracy calculation for the traces from Singapore, for each algorithm and user state. Clearly, each algorithm has its own strengths and weaknesses. The Google activity recognition, which is accelerometer-based, performs well for *IDLE* (76%) and *WALKING* (79%) detection, but doesn't perform well for *VEHICLE* detection, achieving only 31% accuracy (the evaluation in Section 3.5.1 further illustrates this point). FMS, due to its use of GPS, has good *VEHICLE* accuracy (90%), but performs poorly in *IDLE* and *WALKING* detection. Both accuracies are below 50%. Our barometer-based approach does well for *VEHICLE* and *IDLE* detection, due to its independence of vehicular vibrations and user movements. The accuracy for *WALKING* state detection is however much lower.

Table 3.6 shows the confusion matrix for the barometer-based algorithm for Singa-

pore. There are two factors that contribute to the lower *WALKING* accuracy. Due to the long sliding window used in the barometer-based detection, the algorithm has a significant latency before it decides the user has gotten off the vehicle. *WALKING* states which usually happen right after *VEHICLE* states are classified incorrectly. The second factor is the terrain of the land. Sometimes, roads may not be sloped enough for the height to significantly change while walking. This is an inherent limitation of our approach. If the threshold for height change is set too low, then changes in the barometer readings caused by drift or environmental changes will be wrongly classified as walking. We fix the WALKING detection problem by fusing the barometer and the accelerometer in Section 3.5.5.

Tables 3.7 and 3.8 show the confusion matrices for Google activity recognition and FMS. The main source of error in Google's approach is in the *VEHICLE* detection. As the vehicle journey includes rides on subway which can be smooth between stations, and periods of traffic congestion on bus where movement is slow, the algorithm maps these vehicles states to the idle state in 38% of the cases. As for the FMS approach, it is unable to correctly determine the difference between idle and walking states in many instances.

The reason for the low detection accuracy of *IDLE* and *WALKING* by the FMS application is two-fold. First, it samples the accelerometer at a frequency of 2 Hz, which is much lower than related work in the literature, in order to save power and amount of data uploaded. Second, to reduce CPU awake power consumption, it duty cycles even the accelerometer, and hence there are periods of time where the accelerometer data may be unavailable. Note that unlike majority of related work, FMS is a real large-scale deployment, and needs to prioritize power over accuracy.

Table 3.5 compares the barometer approach with Google's algorithm in China. As in the case of Singapore, Google performs relatively better for *WALKING*, while our barometer algorithm has better accuracy for *VEHICLE*, in contrast to Google's poor

Table 3.6: Confusion matrix for the barometer-based algorithm

	Idle	**Walking**	**Vehicle**
Idle	76%	19%	5%
Walking	19%	54%	27%
Vehicle	6%	13%	81%

Table 3.7: Confusion matrix for Google's algorithm

	Idle	**Walking**	**Vehicle**	**Unknown**
Idle	76%	0%	0%	24%
Walking	10%	79%	0%	11%
Vehicle	38%	6%	31%	25%

VEHICLE accuracy. Majority of vehicle data from China was collected in a cross-country train, in order to cover larger area of terrain. These results convince us further that sufficient terrain variations indeed occur for the barometer to detect user activity.

Note that the *WALKING* detection accuracy of Google's algorithm is much lower in China (40%) compared to Singapore (79%) since the volunteer in China was with his kids, and may not have marked the ground truth for the short *WALKING* periods as accurately as for the long *VEHICLE* cross-country train. For Singapore and Boston, on the other hand, any trace where the ground truth was discovered to be marked incorrect during interview with the volunteer was discarded. For China, however, we did not discard the trace since only a single trace was available.

The effect of smoothing on Google's algorithm (GoogleSmooth) can be seen in Tables 3.4 and 3.5. While smoothing improves *WALKING* detection, the VEHICLE accuracy remains low since Google outputs *IDLE* majority of the time in subways/trains. Consequently, the overall accuracy did not improve much for Google's algorithm by smoothing.

Table 3.8: Confusion matrix for FMS algorithm

	Idle	**Walking**	**Vehicle**
Idle	33%	34%	33%
Walking	37%	46%	17%
Vehicle	6%	4%	90%

Table 3.9: Barometer algorithm accuracy for different locations

	Singapore	**Boston**	**China**
Idle	76%	85%	99%
Walking	54%	40%	23%
Vehicle	81%	72%	78%
Overall	69%	79%	93%

Location dependence

In this section, we evaluate the accuracy for Boston and China using the algorithm settings designed for Singapore. This allows us to test how sensitive the setting of our algorithm is, and also allows us to test the barometer value sensitivity at different locations. Note that besides having different terrain variation, Boston has a very different weather pattern, in particular temperature ranges, than Singapore (Boston traces were collected during the polar vortex in 2014). Table 3.9 shows the accuracy of our barometer-based algorithm for all 3 countries together. Note that the *IDLE* traces of Boston and China are included in the accuracy calculation, to check for any effect of weather patterns on *IDLE* detection.

As we can see from the table, the accuracy of the *VEHICLE* and *IDLE* detection still remained high, indicating that barometer-based detection can potentially work well in different locations even without re-calibration. However, the *WALKING* detection remains low and indicates a need for addition sensor fusion technique to complement the barometer-based algorithm to improve accuracy. This will be discussed in Section 3.5.5.

Weather dependence

Unlike the accelerometer and GPS, the barometer is affected by drift due to weather. To analyse the effect of weather on our barometer-based algorithm, we separately collected 29 hours of traces with raining and windy weather conditions from Singapore. The phone was kept idle and we check if changes in the barometer readings under such

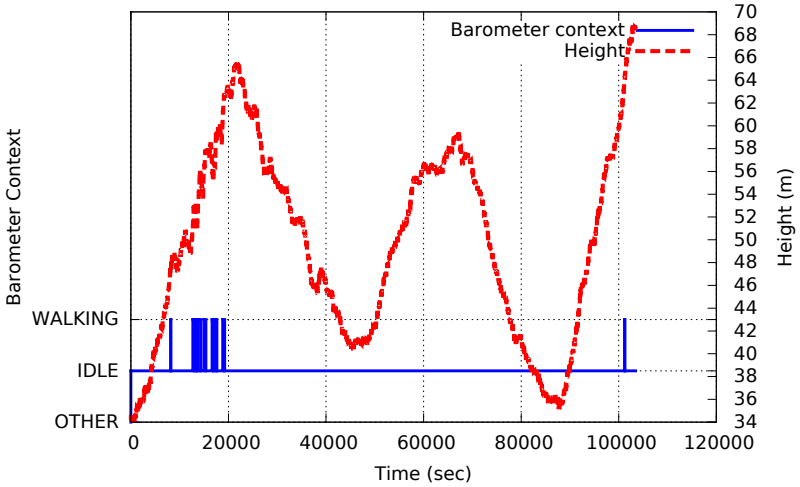


Figure 3.5: Barometer context detection during a rainy day (ground truth = IDLE). Shows diurnal pressure cycles.

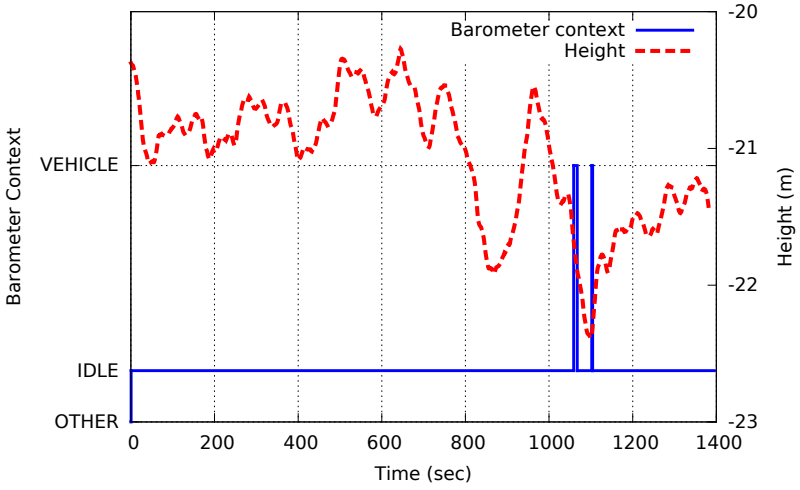


Figure 3.6: Barometer context detection during a windy day (ground truth = IDLE)

weather conditions can trigger false positives and detect *IDLE* as either *WALKING* or *VEHICLE*.

The accuracy of *IDLE* detection was found to be 96%, indicating that weather does not have a significant impact on the barometer-based algorithm. This is due to the fact that weather drift tends to be on a larger time scale than the sliding window of our algorithm (which is set to 200 sec). Rather than cause unpredictable height variations as one might expect, weather drift is usually in one direction, and gradual, typically following the diurnal pressure cycles. This is shown in Figure 3.5. Similarly, windy weather does not produce significant drift in the barometer. An example of how the readings change is shown Figure 3.6.

Finally, during the period of traces collection in Singapore and Boston, there are periods of dry weather (extended periods without rainfall), heavy thunderstorms, and snowfall. This further demonstrates that our approach of using the barometer sensors to determine relative height change is not significantly affected by weather patterns.

Accuracy of *WAIT* state

To illustrate how barometer can be advantageous for *IDLE* detection, we collected a separate 30 minute trace of waiting at a bus stop, running both the barometer and Google algorithms, while holding the phone in the hand.

We observed that even minor movements of the hand (while web browsing, for example), causes the accelerometer-based Google algorithm to get confused and output the state as *UNKNOWN* (see Figure 3.7). Consequently, it has a low accuracy for *IDLE* detection of just 25%. The barometer, in contrast, is unaffected by hand movements, and the accuracy is almost 100%.

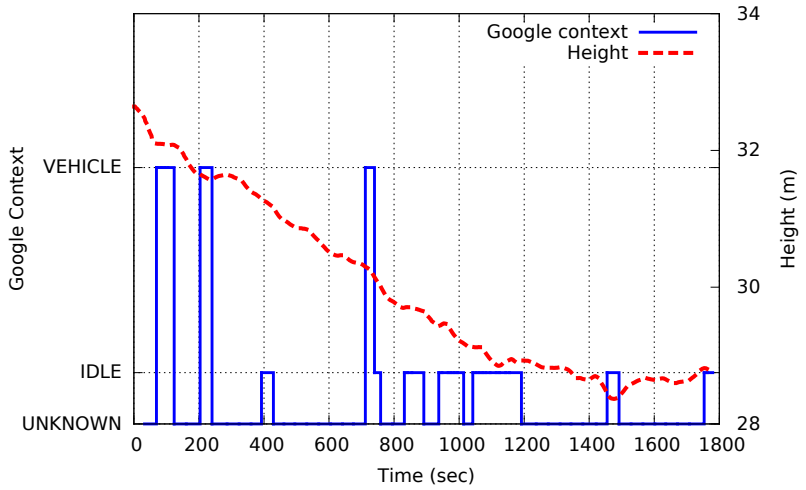


Figure 3.7: Google context detection while waiting at bus stop for 30 minutes (ground truth = IDLE)

Accuracy of *subway* state

Use of the accelerometer to determine vehicle movement depends significantly on the movement patterns of the vehicle in use. This is unlike the use of the barometer, where height change is relevant. To illustrate this issue, we collected a vehicle trace on the subway, running both the barometer and Google algorithms.

The subway ride between two stations is generally quite smooth with limited vibrations and jerks, except near stations. The accelerometer-based Google algorithm has difficulty classifying the user's state as *VEHICLE*, and has a poor accuracy of only 15%. A sample 1 hour subway ride segment is shown in Figure 3.8 where Google's approach often classifies the state as *IDLE*.

On the other hand, the barometer relies only on air pressure to detect height change. This approach has a high detection accuracy of almost 100%. While it may not be obvious to a human, there is substantial height variations in the paths travelled by the subway both in underground tunnels and on above the ground train tracks, making it easy for the barometer to detect a subway ride. The same activity segment but with

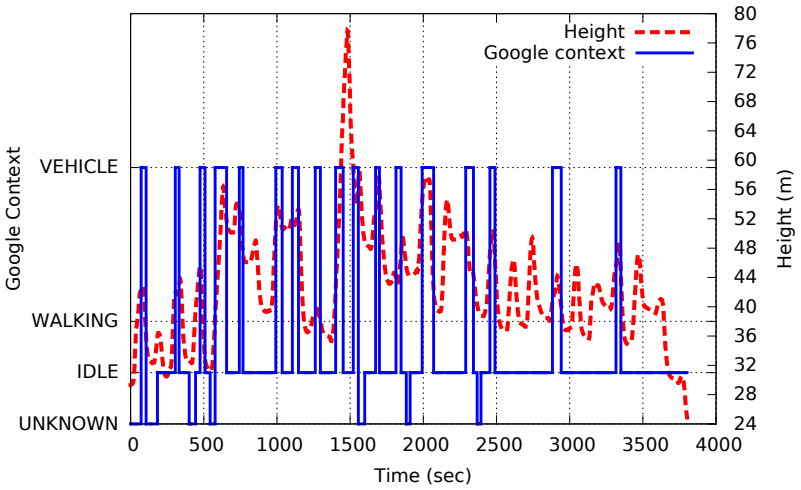


Figure 3.8: Google's context detection on Subway (ground truth = VEHICLE)

output states determined by our algorithm is shown in Figure 3.9.

The better accuracy of the barometer over the accelerometer in vehicle can be additionally observed from the results in the China cross-country train, where Google's algorithm had a poor accuracy of 24%, while our barometer-based algorithm accuracy was 78%. To summarize, although the accelerometer works well in vehicles with substantial vibrations (such as in cars and buses), it performs poorly in smooth vehicles like subways. The barometer works well irrespective of the type of vehicle.

Note on Google's algorithm

The version of Google's algorithm used in our evaluation was the Google Play Services 4.3, the latest version available at the time of collection of trace data in March 2014.

Google appears to have significantly modified their activity recognition algorithm in subsequent releases of Google Play Services, which can now distinguish between *WALKING* and *RUNNING*. The algorithm is also more sensitive to phone movement, enabling it to better detect subway rides. We compared the accuracy of Version 4.3 (March 2014) and Version 5.0 (August 2014)[2] in the subway. The new version has a significantly higher

[2]The apk of every Google Play Services release is available online and can be installed for accuracy

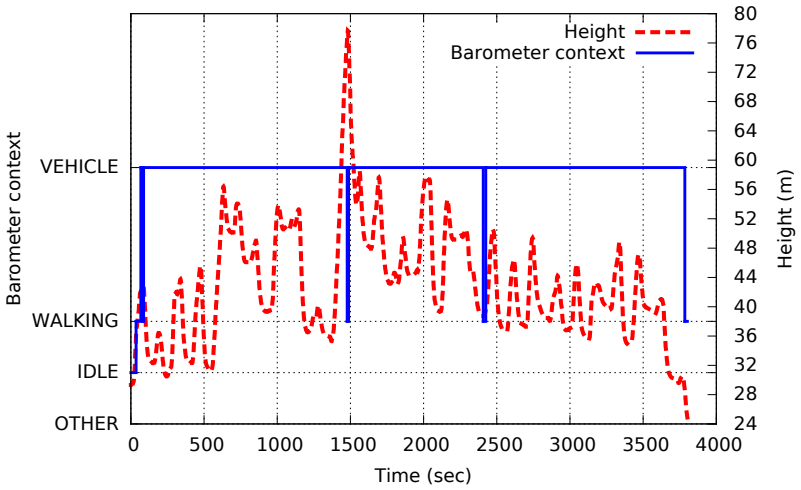


Figure 3.9: Barometer context detection on Subway (ground truth = VEHICLE)

Table 3.10: Accuracy of barometer-based context detection algorithm using map elevation data at different speeds

	Vehicle (50 kmph)	**Vehicle (35 kmph)**	**Vehicle (25 kmph)**	**Walk (5 kmph)**	**Walk (8 kmph)**
Kansas City	96%	93%	89%	73%	56%
San Francisco	92%	90%	76%	74%	66%
Lausanne	84%	83%	79%	58%	50%
Singapore	99%	99%	98%	63%	32%
Boston	99%	97%	91%	66%	58%

accuracy of 81%, compared to the older version with accuracy of 22%.

However, the extra sensitivity of the new version also leads to a larger number of *VEHICLE* false positives caused by minor hand movement. We compared the accuracy of the old and new version while waiting for a bus. The old version detected 3% *VEHICLE* false positives, while the new version performed poorly with 88% *VEHICLE* false positives (Note that both the old and new version have a low *WAITING* detection accuracy, since the old version reports *UNKNOWN* most of the time, as observed in Section 3.5.1). This comparison shows that while the extra sensitivity in the new version enables better subway detection, it causes a large number of false positives due to minor hand movement when the user is idle.

comparison

Table 3.11: Comparison of terrain characteristics (stddev in brackets)

	Avg Elevation Change (m)	**Avg Peak Distance (m)**
Kansas City	0.84 (0.99)	479 (494)
San Francisco	1.05 (1.17)	645 (709)
Lausanne	1.04 (1.19)	395 (536)
Singapore	0.69 (0.65)	332 (252)
Boston	0.56 (0.66)	476 (435)

3.5.2 Simulation using map elevation data

In addition to real-world trace data, we have manually pulled over 900 km (30,000 data points) of elevation data from non-overlapping roads of 5 cities from Google Maps. Using the map elevation data, we can evaluate the accuracy of our algorithm in a larger number of places over larger geographic areas having substantially different terrains, especially where we do not have real-world trace data available. An additional advantage is that we can vary the speed of travel and check the effect on the accuracy.

Map elevation data was collected from roads in Kansas City (USA), San Francisco (USA), Lausanne (Switzerland), as well as from Singapore and Boston. Data was collected at 30 meter points, which is the highest resolution possible. Elevation data in between points are interpolated. This data has been used to emulate the barometer sensor data, and fed to our algorithm to evaluate accuracy.

Table 3.10 shows the accuracy our barometer-based algorithm at different speeds. Two walking speeds (5 and 8 kmph) and three vehicle speeds (25, 35 and 50 kmph) are considered. 5 kmph is the average walking speed of a person, while 8 kmph is a fast pace. Note that the accuracy of vehicle for speeds higher than 50 kmph is expected to be higher, since number of ups and downs encountered would also be higher, and is hence not shown in Table 3.10.

Note that even in places like Kansas, sufficient terrain variations occur while travelling for the barometer to distinguish between user states. Even at very low vehicle speed

Table 3.12: Latency (sec) for each user state for the barometer and Google's algorithms (stddev in brackets)

	Baro	**Google**
Idle	176 (142)	78 (66)
Walking	158 (138)	26 (24)
Vehicle	211 (192)	122 (135)

of 25 kmph, the accuracy is high, which increases with increasing speed. The walking detection accuracy is lower, but can be fixed using fusion with the accelerometer (Section 3.5.5).

Table 3.11 compares the terrain characteristics of the 5 cities, calculated using over 30,000 elevation data points, each spaced at 30 meters intervals, collected manually over non-overlapping roads from Google Maps. Average Peak Distance is the average distance between ups and downs on the road. In other words, if you plot a graph of elevation versus distance travelled, peak distance is the distance between two peaks in the graph. Smaller the value, more undulated the terrain, i.e. more peaks and valleys are encountered over the same distance travelled. From Table 3.11, Lausanne and Singapore have the most undulated terrain.

Average Elevation Change is the average change in elevation for every 30 metres of distance travelled. A higher value of average elevation change indicates higher road steepness. From Table 3.11, San Francisco and Lausanne have the steepest roads.

The accuracy of our barometer-based algorithm using map elevation data over these different terrains gives us confidence that this approach can indeed be generalized to other cities as well.

3.5.3 Latency

The barometer and Google's algorithms run in real-time on the phone. In this section we compare the latency of these algorithms (FMS, in contrast, uploads data to the server and does post-processing). Latency is calculated as the average delay between transition

to a user state, and detection of that state by the algorithm in question (for example, the average delay between a person starting to walk and the walk activity being detected). Table 3.12 lists the latency for each user state for both Google's and the barometer algorithms.

The Google algorithm's *IDLE* latency is low since it has been designed to detect even short vehicle/walking stops. However, applications typically prefer to ignore short user stops (detected by Google's algorithm), and rather focus on longer stops (detected by our barometer-based algorithm) which indicate higher-level user activities like home/office/shopping.

For the *VEHICLE* state, both Google's and the barometer algorithms have higher latencies (2 to 3.5 min). This is due to the poor VEHICLE detection of Google's algorithm and the long sliding window of our barometer-based algorithm. The impact of this latency on applications depends on the journey duration. Analysis of real-world bus trip data from Singapore over 2,256,911 bus trips shows that the average duration of a bus ride is 14 minutes, while the maximum duration can be as large as 156 minutes. For long bus trips, we expect the vehicle latency to be acceptable, since it is a fraction of the total bus trip duration.

Applications interested in the *WALKING* state (Eg: Fitness apps) require low latency. We exploit the low latency of walking detection of Google's algorithm by fusing both the barometer and the accelerometer together in Section 3.5.5.

Note that although Google's activity detection has overall lower latency than our proposed barometer-based algorithm, the output of Google's algorithm is highly fragmented, a consequence of it being too reactive to state change.

3.5.4 Power usage

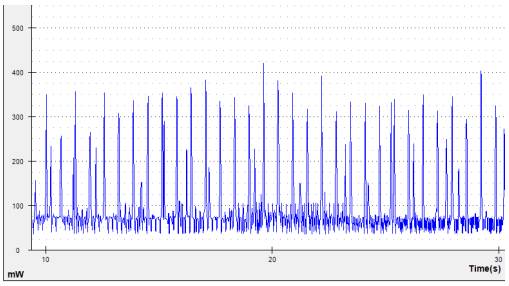
In this section, we compare the power consumption of the barometer-based context detection to Google's Activity Recognition. Measurements were performed on a Galaxy

S3 using the Monsoon Power Monitor. Our application is run in the background after acquiring a wake lock to keep CPU processing on. The screen and all wireless interfaces as well as data sync are kept switched off. The Android OS is an unmodified and unrooted version 4.3, which came bundled with the phone from the manufacturer.

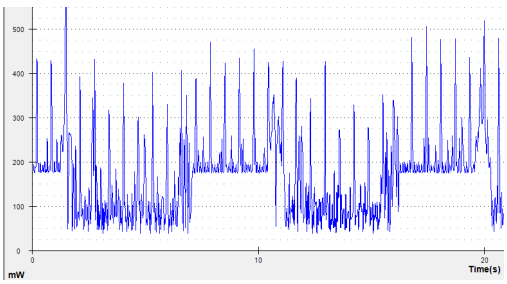
The barometer data is sampled using the Android sensor manager API. Note that although we specify a sampling rate of 1 Hz to the sensor manager, the Galaxy S3 driver returns data at a higher rate of 5 Hz. However, we clamp and process data at 1 Hz in our code.

Google's algorithm does not run continuously. It runs for 5 seconds each time it is triggered. For an update interval of 10 seconds, the program is triggered every 10 seconds, runs for 5 seconds, and sleeps for the remaining 5 seconds (Figure 3.10b). In contrast, the barometer-based algorithm runs continuously. The power consumption can be further reduced if sensor batching is utilized. This is dicussed in Section 3.6.

Since the accelerometer sampling rate and calculations involved are higher, the power usage is also correspondingly high. The barometer, on the other hand, consumes lower power due to its low sampling rate and simple calculations involved (refer to Figures 3.10a and 3.10b).



(a) Power profile of barometer-based context detection



(b) Power profile of Google's context detection

Figure 3.10: Power profile of Google and barometer algorithms

Table 3.13 shows the power consumption. The power values listed for the barometer and the Google approaches include the base CPU awake power, the sensing power, as

Table 3.13: Power usage

	Power (mW)
CPU Idle	25
CPU Awake	85
Google	120
Baro	88

well as the computation power.

In spite of running continuously, the barometer-based algorithm consumes 32 mW lower power than the accelerometer-based algorithm. Google consumes 35 mW over the base power, while the barometer uses only 3 mW over the base power, a significant improvement.

We had a detailed discussion with the authors of [43] on the measurement methodology. The power measurement methodology in their work differs significantly from ours in several aspects. First, they run the power measuring app in the foreground rather than background. Second, they do not process sensor data but log sensor readings to the `sdcard`. Finally, the accelerometer and the barometer were read and processed at different frequencies. Conversely, as we elaborated at the beginning of this section, we run our power measuring app in the background, process the sensor readings (either with Google's algorithm for the accelerometer or ours for the barometer), and do not save the readings to `sdcard`.

3.5.5 Fusion of barometer and accelerometer

We can fuse both the barometer and the accelerometer together to increase detection accuracy. Although the power consumption increases compared to using a single algorithm, with the advent of sensor hubs and offloading of activity detection into hardware, power consumption may reduce drastically, making it worthwhile to fuse multiple sensors for higher context detection accuracy.

In this research work, we found that the barometer and the accelerometer have com-

Table 3.14: Fusing barometer and Google algorithms

	Baro	**Google**	**Fusion**
Idle	76%	76%	76%
Walking	54%	79%	88%
Vehicle	81%	31%	77%
Overall	69%	56%	81%

plementary strengths and weaknesses: the barometer is good for *IDLE* and *VEHICLE* detection, but poor in *WALKING* detection, while the accelerometer is good for *WALKING* detection, but poor in *IDLE* and certain *VEHICLE* detections. A simple fusion technique can combine the strengths of both the sensors, by first giving precedence to the accelerometer for *WALKING*, and then to the barometer for other states.

Table 3.14 shows the accuracy using fusion of the barometer and Google algorithms. The overall accuracy improves drastically over using a single sensor. Fusion also fixes the *WALKING* detection problem faced by the barometer.

3.6 Discussion

In this section, we discuss two issues pertaining to the use of the barometer for context detection:

3.6.1 Sensor batching

The main source of power consumption in current activity detection algorithms is the need to keep a wake lock on the main processor to process sensor data continuously. Hardware implementations such as the M7 co-processor reduce this power, but are inflexible. A new hardware feature has been introduced in the Nexus 5, called sensor batching, where sensor data can be buffered while the processor sleeps, and processed in a batch later when the processor wakes. This provides a nice balance between power and software flexibility. The number of readings that can be buffered is limited by the sam-

pling rate and the size of the data. The low sampling rate of the barometer (1 Hz) makes it excellent for sensor batching, compared to the 3-axial accelerometer which requires higher sampling rates and larger data. The barometer data can be buffered for several minutes even when using small-size buffers. With sensor batching, the barometer would become truly ultra low power. Unfortunately, we were unable to tap into the battery of the Nexus 5 for power measurements, and so are unable to provide an evaluation of the effect of sensor batching on the power consumption.

3.6.2 Combining temperature with pressure

One potential way to improve the *WALKING* detection is the use of the temperature along with the pressure. Barometer chips contain embedded temperature sensors, so reading the temperature comes at no extra cost. Air pressure and temperature are co-related. When the phone is idle, a change in pressure would be associated with a corresponding change in temperature. While walking, this co-relation is disturbed since the change in pressure gets additionally affected by the change in altitude. This could act as an indicator for the *WALKING* state. In our measurements of the pressure and the temperature indoors on the Galaxy S4 (which has an ambient temperature sensor), this co-relation appeared to be correct. However, when we performed measurements outdoors while idle, this co-relation does not seem to hold, as seen in Figure 3.11. Ideally, even when outdoors, when the user is idle, pressure and temperature should show a co-relation. One reason this is not the case is perhaps the slower reaction of the temperature sensor to change in the temperature, or low resolution. Further exploration of this idea using higher quality and higher resolution sensors is left as future work.

3.6.3 Integration with the FMS app

We are looking into integrating our algorithm into the FMS App, to eliminate the 'movement' false positives often generated by the accelerometer which trigger the high-power

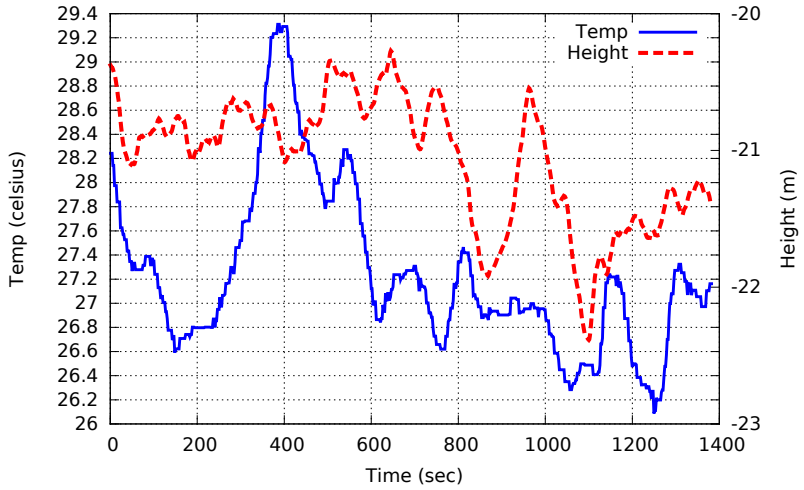


Figure 3.11: Variation of temperature and pressure outdoors when IDLE

location service. This can help reduce power consumption since the location service will not run unnecessarily.

3.7 Conclusion

In our work, we demonstrate an alternative approach for low-power transportation context detection using only the barometer sensor, a relatively new sensor now present in an increasing number of phones. Unlike the accelerometer, the predominant sensor in use today, the barometer is unaffected by phone position and orientation, but instead depends only on the overall terrain of the land. Using a low sampling rate of 1 Hz, and simple processing based on intuitive logic, we show that the barometer can be used for the detection of the states *IDLE*, *WALKING*, and *VEHICLE* while consuming 32 mW lower power than even the accelerometer, at the same time achieving comparable accuracy to Google's Activity Recognition algorithm and the FMS application. The barometer also solves the problems of the accelerometer in detecting the *WAITING* state, and certain vehicles like the subway. Finally, we found that fusion of the barometer and the accelerometer combines the strengths of each sensor.

Chapter 4

Barometer-based vehicle context detection

4.1 Introduction

The sales of smartphones has surpassed the sales of feature phones [5]. With increasing penetration rate and computational power, smartphones have opened up new possibilities for the way applications work.

Applications for aiding users travelling by public-transport have been and continue to be important in urban cities for reducing road congestion. Traditionally, such applications operate by collecting traffic data from costly pre-installed infrastructure (such as road induction loops and GPS devices installed in public-transport vehicles). Smartphones have opened up a new alternative for crowd-sourcing vehicle location data from the users' smartphones.

Two fundamental requirements of such smartphone-based public-transport applications is the detection of bus routes and bus-stops, i.e. identifying which bus route a user has taken, and when a bus has reached a bus-stop in its route. This is necessary in order to provide functionality such as bus-stop arrival time prediction, and reminding

users that they have almost reached their destination stop, features provided by apps like NextBus Muni[1], Moovit[2], and SBS iris[3].

Existing techniques for bus route and bus-stop detection rely on tracking the user's (and thereby vehicle's) location using GPS/WiFi/Cell-ID during the journey, and uploading this data to a central server. The server in turn uses this data to determine which bus route a user has taken, and when a bus has reached a bus-stop in the route. The location of bus-stops is known either from prior-available bus route maps, or is determined by processing unlabelled GPS traces from several users [16].

However, these techniques suffer from multiple drawbacks. First, the privacy of users is violated by tracking their location and uploading to a server. Second, using the location sensor, in particular the GPS module on the phone, consumes significant power. Third, majority of existing works assume prior-availability of route maps with locations of bus-stops for detection to work. Finally, users need to upload their data through the cellular network, which may hinder user deployment due to sharply reducing data limit for cellular data plans.

In this chapter, we propose a new smartphone-based technique for bus route and bus-stop detection using the barometer sensor. We show that by using just the readings from the barometer sensor, we can detect the bus route taken by the user. By matching barometer measurements between users in the same bus, we can also detect with reasonable accuracy when a bus has reached a bus-stop in its route. We then use the knowledge of bus route and bus-stops to perform bus journey time prediction.

To clarify what exactly our technique tries to accomplish, consider a user who takes bus numbers A and B everyday, with 5 stops and 10 stops respectively. When the user takes Bus A, our technique is able to distinguish the route (i.e. it is A and not B), detects when the bus reaches each of the 5 stops in route A, and uses this information

[1] `www.nextbus.com`
[2] `www.moovitapp.com`
[3] `www.sbstransit.com.sg/iris`

to predict when the bus will reach the destination stop. Note that the real-world bus route IDs and bus-stop IDs are not required for doing the prediction. Also, note that while we use the terms 'bus journeys' and 'bus-stops', our approach is applicable to any transit-stop and public-transport vehicle, such as subway stations in a subway journey (see Section 4.4 for our evaluation on buses and trains).

Unlike existing approaches, our work does not need the user's location, is completely de-centralized, requires no knowledge of bus routes beforehand and works without an Internet connection. Our system can thus be deployed with minimum support and infrastructure, including cities in developing countries where Internet and cellular data is not widely available, and where even route maps are not available (e.g. Dhaka in Bangladesh, Manila in Philippines, and Mexico City[4]).

The key contributions in this chapter are the following: (1) We describe a novel use of the barometer sensor for detecting bus routes and bus-stops providing location privacy and lower power consumption. (2) Using an approach based on Dynamic Time Warping (DTW), we show that it is possible to group a user's journeys across multiple days into correct clusters with high accuracy using only barometer readings. (3) We propose a variation of the DTW algorithm so that even when journeys overlap only partially, data from different users over different time periods can be combined to extract information that is not available on a single device: intermediate bus stops, and untraversed portions of the bus route, which are useful for journey time prediction.

We evaluate our approach using 61 real-world barometer traces of 7 different journeys using 3 volunteers and 5 different phones. We also evaluate using a simulation of real-world bus traces from 4 bus routes involving more than 1500 users over 100 days. The results show that we are able to detect bus routes and bus-stops with reasonable accuracy. Our route clustering technique has a precision of 87% and recall of 81%, and we can detect bus-stops with an average error of 2 minutes. We show that we can use local data

[4]http://www.wired.com/2015/08/nairobi-got-ad-hoc-bus-system-google-maps/

from users in the same bus predict bus-stop arrival time with an average error of 1.22 minutes. We have also implemented our system on Android, and demonstrated that the power consumption is much lower compared to a traditional GPS-based system.

The rest of this chapter is organized as follows: Section 4.2 gives an overview of our system, while Section 4.3 describes its implementation. We evaluate our work in Section 4.4. Section 4.5 describes our application implementation on Android. Section 4.6 discusses future work, while Section 4.7 concludes the chapter.

4.2 System overview and motivation

This section gives a brief overview of our system's operation and functionality. We then provide the motivation for the use of the barometer sensor for low-power low-latency operation.

4.2.1 System overview

We explain the operation and functionality of our system using an illustrative example, depicted in Figure 4.1, of 2 users and a tourist travelling on the same bus. User A gets on the bus at bus-stop B1, and gets off at bus-stop B3, while user B gets on at B2, and gets off at B4. The tourist gets on at bus-stop B3 and gets off at bus-stop B4.

Figure 4.1 illustrates the functionality provided by our system during the bus journey from the point of view of each user. It also shows the terrain variation in terms of altitude observed indirectly using the barometer sensor on each user's smartphone. In addition, the figure also shows a high-level view of the operations occurring behind the scenes in the system. As depicted in the figure, our system operates in three modes: single-user, multiple-user, and tourist.

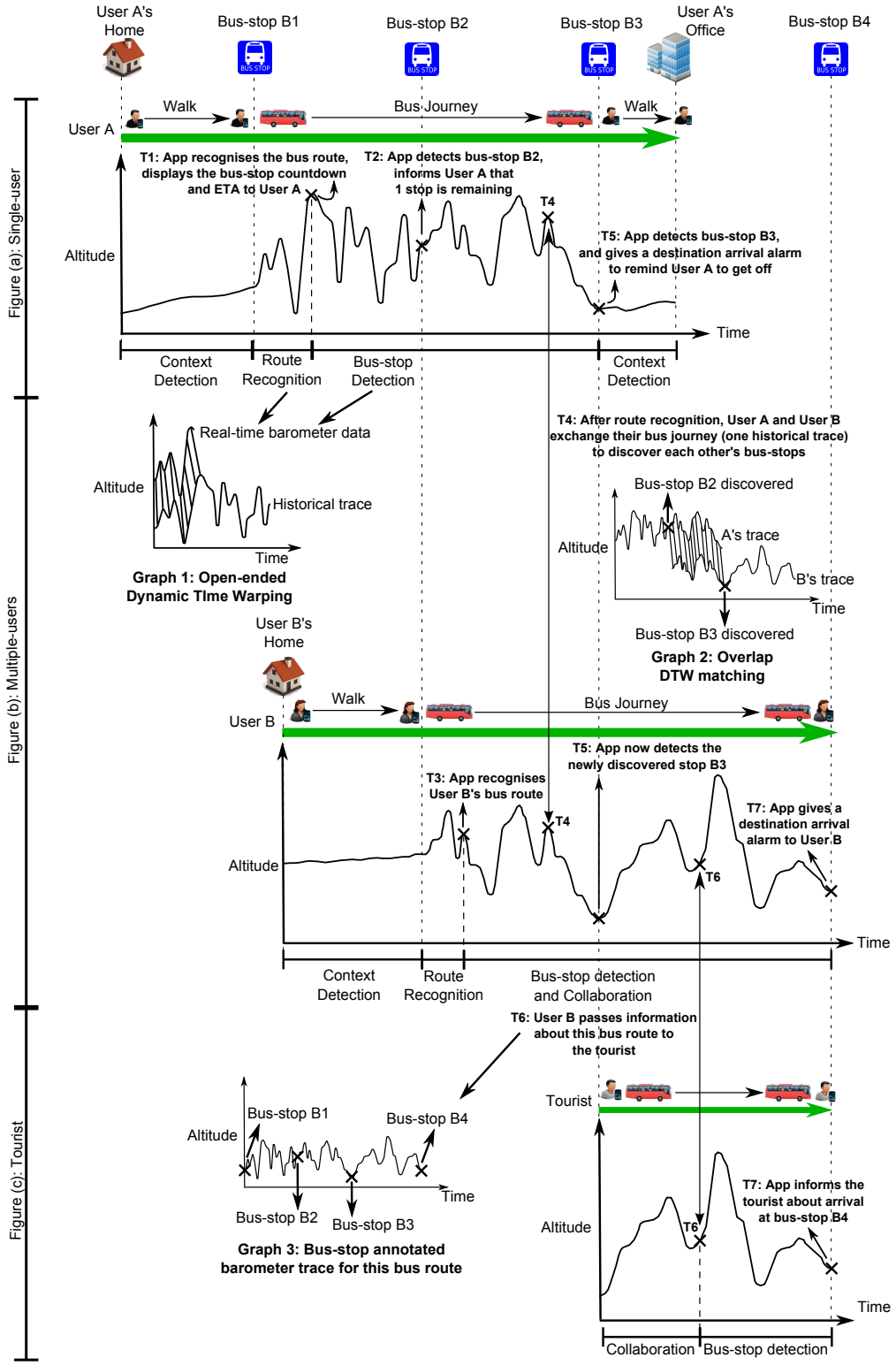


Figure 4.1: Overview of system operation using an illustrative example.

Single-user (Figure 4.1 (a))

In single-user mode, each user runs the application locally on his/her phone with no real-time collaboration or data sharing with other users. This is illustrated using the example of user A in Figure 4.1.

User A walks from home to bus-stop B1, where he gets into the bus. The system runs a low-power context detection algorithm to detect when the user has boarded the bus. It then runs a route recognition algorithm to determine which of his historical routes user A is most likely travelling on. At time T1, after route recognition, the system displays details about the recognised bus route to the user, including the number of bus-stops remaining, the ETA, and the name of the next bus-stop. Following this, the system runs a bus-stop detection algorithm to track the bus's location in real-time in terms of stops along the route. For example, at time T2, the system detects arrival at bus-stop B2, and updates the bus-stop countdown accordingly. Finally, at time T5, when the bus reaches the destination at bus-stop B3, the system rings an alarm to remind user A to get off the bus. The system detects the user getting off the bus and starting to walk, and switches back to running only low-power context detection.

For route recognition and bus-stop detection, the system applies open-ended dynamic time warping (shown in Graph 1 in Figure 4.1, and described in Section 4.3) using real-time barometer sensor data from user A's smartphone, and a historical barometer trace for that bus route, annotated with points corresponding to the positions of bus-stops (shown in Graph 3 in Figure 4.1). The barometer traces for bus routes can be collected and annotated by either the transport authorities, or crowd-sourced by other users and downloaded to the phone beforehand. Unlike works involving low-power motion sensors, which usually require extensive data collection over several users and/or training, the barometer trace is independent of the user and phone placement, and one data trace is usually sufficient for any user. In Section 4.3, we explain in detail about the route recognition and bus-stop detection process.

Multiple-users (Figure 4.1 (b))

Many cities in developing countries do not have consolidated bus route maps (Section 4.1). Transport services are chaotically operated by multiple private operators, bus-stops are not designated or marked by any signs, and people are often not even aware of available bus routes and stops along the route. In such situations, co-operation from transport operators is limited, and manually collecting annotated barometer traces with bus-stop positions is time consuming and difficult to do without bus route maps. In such cases, we need a method to automatically 'discover' bus-stops along the route, based on the points where users get on/off. Our system provides bus-stop discovery using real-time sharing of barometer data between multiple users in the same bus. This is in addition to the functionality provided in the single-user mode.

Multiple-user operation is described using the example of users A and B in Figure 4.1, who travel along different parts of the same bus route. After user B has boarded the bus at bus-stop B2, the system recognises the most likely route at time T3 based on user B's historical traces. However, unlike in the single-user case, the locations of intermediate bus-stops is unknown, except for the source and destination stops.

Once both users' smartphones have completed route recognition, they exchange one historical trace of their respective bus journeys using direct phone-to-phone communication. This exchange can happen at anytime during the journey when both users are in the bus, enabling the system to perform neighbour discovery at a lower duty cycle to save power. In Figure 4.1, this exchange occurs at time T4. Processing the exchanged traces enables both users to discover each other's source and destination bus-stops. In this example, users A and B became aware of all 4 bus-stops in the route after their data exchange. After collaborating for bus-stop discovery, the system can now perform bus-stop detection of intermediate stops as in the single-user mode, shown at time T5 and T7 for user B, where she is notified of the newly discovered bus-stop B3. By exchanging data with more users with different but overlapping journeys, more bus-stops

are discovered over time, and in the steady state, each user ends up with an annotated barometer trace for the entire bus route (shown in Graph 3 in Figure 4.1), but obtained automatically rather than manually. While the data exchange could be performed via a server, using phone-to-phone communication on the bus removes the need for setting up a server and Internet connectivity, both of which may not be available in developing cities in spite of the high smartphone penetration.

Bus-stop discovery is performed using a variant of dynamic time warping called overlap matching. This is shown in Graph 2 in Figure 4.1, and is explained in Section 4.3.

Tourist (Figure 4.1 (c))

Over a longer time scale, once sufficient data is exchanged, the entire bus-route with the associated bus-stop information is derived and stored locally on each user's smartphone. When a tourist or a commuter gets onto the bus for the first time, he/she can obtain the bus-stop and journey time information for each stop on the route, on encountering one of the regular commuters who runs the application.

This is illustrated in Figure 4.1, where the tourist obtains bus route information from user B at time T6, enabling the system to inform the tourist on arrival at bus-stop B4 at time T7 where he gets offs the bus.

Applications

This thesis focuses on using the barometer sensor for bus route recognition and bus-stop discovery and real-time stop detection. We have built an application to provide bus-stop countdown, destination reminder alarms, and real-time tracking of the bus in terms of stops, to demonstrate how these techniques can be integrated into a single application.

While not explored in this thesis, we expect that our techniques can be easily applied to additionally act as a back-end to systems providing next bus arrival services, by

tracking bus location even in between stops using historical barometer traces annotated with high frequency GPS data. Our barometer-based system can act as a power-efficient alternative or complement systems using GPS devices. In this thesis, however, we focus on the underlying challenges of route recognition and bus-stop discovery that enable applications to provide these functionalities.

4.2.2 Motivation for using the Barometer

Prior works have used GPS, cell-ID, gyroscope, battery current, and magnetometer. In our system, we instead use the barometer sensor[5], a low-power sensor that measures air pressure. This can be converted into elevation (height/altitude) above sea level. This section provides the motivation for using the barometer over other sensors.

During a vehicle journey, changes in elevation as small as 0.5 metre can be sensed by the barometer. When plotted as a time-series graph, the barometer signal effectively shows the terrain variation in terms of relative altitude change as the user travels to the destination.

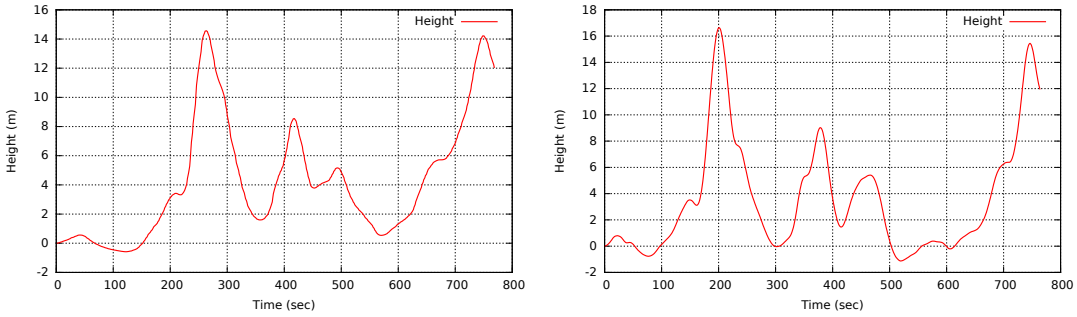
A key observation that motivates this work is that terrain, measured using the barometer sensor, for the same bus route, remains fairly stable across different days and different smartphones. As an example, Figures 4.2a and 4.2b show the changes in elevation for the same journey, but on 2 different days and 2 different phones (Galaxy S4 and LG Nexus 5). As can be observed, the signals have a stable and unique set of features. This is expected since the terrain rarely changes. Consequently, variation in height provides a sufficient and stable signature for route matching. Note that it is not necessary for the height/terrain measurements to be similar in absolute values across different days. Instead, similarity is required only for the relative variations.

Unlike other low-power sensors, the barometer output is unaffected by user hand movement and phone placement, removing the need for training or extensive data col-

[5]available on Nexus 3/4/5/6, Galaxy S3/4/5/6, iPhone 6, and many more

lection typically required by other systems. In contrast to GPS, the barometer operates at a low power and provides continuous data even when duty cycled.

The barometer can be used as a proxy for location to track the bus. This is illustrated in Figure 4.3, which shows how terrain variation measured by the barometer can be used to get an idea how 'far' the user is from the destination, even without actual absolute location.



(a) Journey on day 1 on Nexus 5 **(b)** Same journey on day 2 on Galaxy S4

Figure 4.2: Barometer time-series signal for the same journey on different days and different phones (note that journeys are typically of different durations, although they appear similar in this figure).

To summarize, the barometer has the following advantages over other sensors:

1. **Low-power:** Using the barometer saves power over using GPS, even while operating at low detection latency.

2. **Continuous sampling:** Using the sensor batching feature available on many smartphones, the barometer can be duty cycled without losing sensor data when the CPU is asleep. GPS, in contrast, misses location data when the GPS chip is turned off.

3. **Phone position and hand movement:** Unlike sensors like gyroscope, the barometer signal is insensitive to the phone position and user's hand movement.

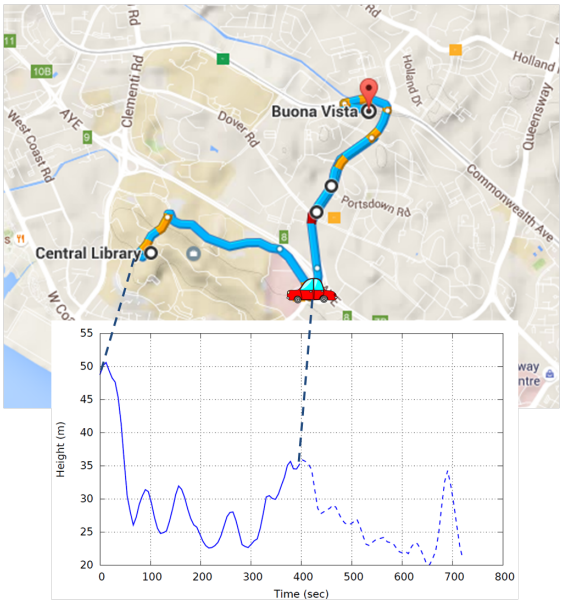


Figure 4.3: Barometer, which provides height variations, can act as a power-efficient alternative to absolute location.

4. **Full coverage:** Unlike GPS, barometer has 100% coverage, working even when the vehicle is underground.

5. **Signal stability:** GPS, Cell-ID, and battery current are sensitive to changes in the environment, and are thereby more noisy. On the other hand, barometer measures terrain indirectly and yields a relatively stable signal as long as the terrain remains unchanged.

Terrain variations in cities

Our use of the barometer is based on the assumption that there is terrain variation during travel. As observed in previous works [79, 80], there is indeed sufficient terrain variation in many cities for the barometer to be exploited. In our evaluation, we observe that not only is there sufficient terrain variation, but this variation it is sufficiently unique per user to recognise bus routes.

Weather effects

Change in air pressure due to weather affects the barometer output. However, as also observed by prior works [79, 80], weather drift in the barometer occurs over a long period of time, typically 3 to 4 metres over 1 to 2 hours. During travel, change in air pressure due to vehicle movement over undulating terrain dominates over change in air pressure due to weather, and weather has negligible effect on our system.

4.2.3 Motivation for user collaboration

Traditional approaches use a central database and crowdsourcing of data from all users travelling in public transport. However, in developing cities, setting up and maintaining a server can be challenging, and Internet connectivity may be limited in spite of high smartphone penetration.

We envision the set of smartphones as a 'mobile distributed database' consisting of mobile nodes travelling in the public transport network, each node containing useful data. When a user gets into a bus, he/she only has access to local data from other users travelling in the same bus. In this thesis, we show that it is possible, from this collection of local data only, to derive information about the bus journey and obtain a reasonable estimate of arrival time to various bus stops (Section 4.4.2).

To further motivate the feasibility and usefulness of local data sharing, we look at the statistics of a real public bus transportation system in a large urban city which consists of 235 bus routes and 2,256,911 trips over a single day. Figure 4.4 shows the CDF of the number of users on the same bus with different (source, destination) stops.

From the figure, it can be seen that when a user gets into the bus, 70% of the time there are more than 10 other users in the same bus with different (source, destination) stops, i.e. they have different but overlapping journeys. This implies that it is often possible for users to extract information about intermediate bus-stops.

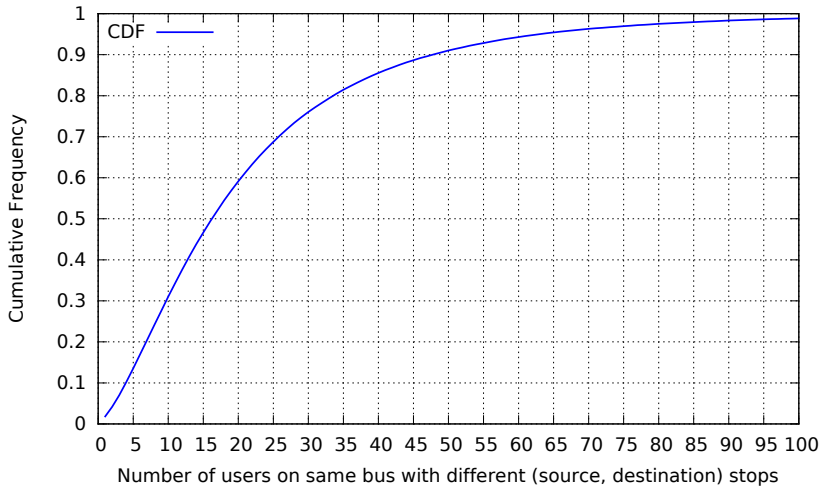


Figure 4.4: CDF of Number of users on the same bus with different (source, destination) bus-stops.

4.3 System implementation

In this section, we describe the processing done on the barometer data to provide our system's functionality, that was illustrated in Section 4.2. First, we give a background on Dynamic Time Warping, different variants of which are used in our system. Then, we describe how the route recognition and bus-stop detection is performed.

4.3.1 Background

In this work, we use different variants of Dynamic Time Warping (DTW) to find the dissimilarity between two barometer signals, as well as to match signals between different users. Here, we give a high-level description of the basic version of the DTW algorithm, while its variants are discussed later.

Dynamic Time Warping (DTW) is an algorithm for finding the dissimilarity (or distance) between two time series which have been warped (compressed or stretched) by different amounts at different parts of the time axis. While originally proposed for speech recognition [81], it is now widely used in several domains, including gesture recognition,

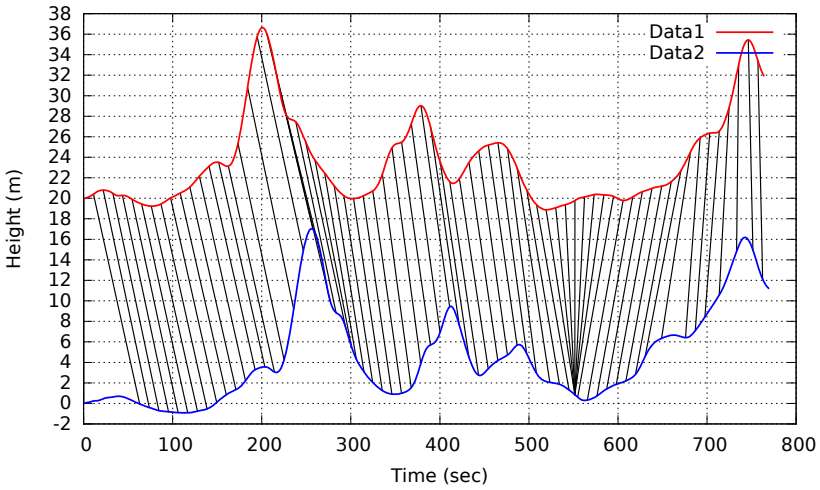


Figure 4.5: Example of DTW matching between two barometer signals (Data 1: offset by +20m).

bioinformatics, and data mining in general [82].

The barometer signal is essentially a time series, being an ordered sequence of timestamped height values. In this chapter, the term 'signal' refers to the time series of barometer height values.

Figure 4.5 shows an example of DTW matching between two barometer signals. The DTW algorithm not only outputs the dissimilarity value between the signals, but also maps the points in one signal to corresponding points in the other signal, as shown by the black lines in the figure. When a signal is stretched or compressed, it is possible that one point is mapped to multiple other points in the other signal.

For illustration purpose, in Figure 4.5 (and similar figures in the rest of this chapter), we purposely add an offset to one of the signals on the y-axis, to show a gap between the two signals even though there may be none. In figures appearing later, we also sometimes offset one of the signals along the time axis for the same reason. In the processing, all timestamps and height values are relative to the origin. Also, note that the signals are typically of unequal total duration, depending on vehicle speed during the journey.

While several distance measures exist for computing the dissimilarity between two

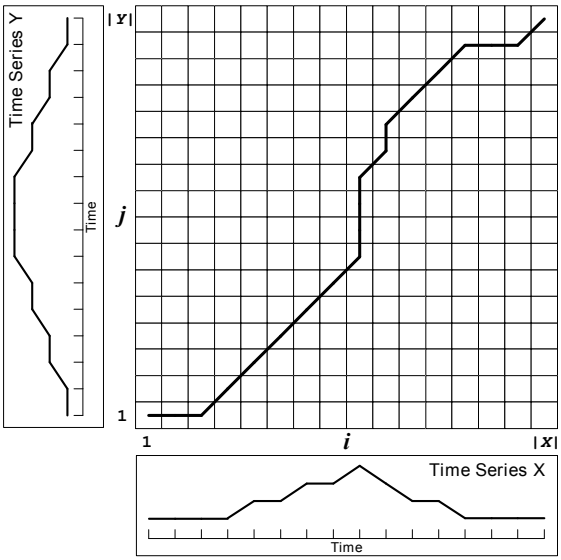


Figure 4.6: Warp path in DTW (image as appears in [85])

signals, including Euclidean distance and Edit distance [83], we chose DTW for three reasons: First, it operates on continuous signals, avoiding the need to discretize height values. Second, it allows for warping w.r.t. the time axis, making it possible to match barometer signals that have been stretched or compressed due to differing vehicle speeds during the journey. Third, unlike Edit distances, which are more suited for noisy signals, DTW particularly works well for clean signals like barometer [84]. To the best of our knowledge, this is the first work that applies DTW to a barometer signal.

If X is a time series consisting of N height values $\{x_1, x_2, ..., x_N\}$, and Y is a time series of M height values $\{y_1, y_2, ..., y_M\}$, then the mapping of points from one signal X to the other signal Y can be alternatively represented by a *warp path* going a matrix L of size $N \times M$, as shown in Figure 4.6. If the warp path passes through the matrix element $L(i, j)$, then the i^{th} point of X is mapped to the j^{th} point of Y. These mappings correspond to the black lines shown in the graph in Figure 4.5. As can be observed in Figure 4.6, the warp path is often restricted to start at element $(1, 1)$ and end at element (N, M).

If L is a local distance matrix (i.e. each element $L(i,j)$ stores the distance between x_i and y_j), then the DTW dissimilarity value between the two signals X and Y is the summation of all $L(i,j)$ elements the warp path passes through. In simple terms, the dissimilarity value is the sum of the distances between mapped points.

The DTW algorithm finds the best warp path through the matrix L that results in the smallest dissimilarity value. In other words, it finds the best mapping between points of the two signals such that the sum of distances between mapped points is minimum. In simple terms, it finds the best warp path from $(1,1)$ to (N,M) in the local distance matrix L.

DTW, as the name suggests, uses dynamic programming to find the best warp path. It constructs a *cumulative* distance matrix C (instead of a local distance matrix), and calculates each element $C(i,j)$ as follows:

$$C(i,j) \;=\; distance(x_i, y_j) \;+\; min\{C(i,j\,-\,1), C(i\,-\,1, j\,-\,1), C(i\,-\,1, j)\} \quad (4.1)$$

with the base conditions $C(0,j)$ and $C(i,0)$ as infinity, with the exception of $C(0,0)$ which is set to 0.

After this calculation, each element (i,j) in the cumulative distance matrix C stores the value of the best warp path from element $(0,0)$ to the element (i,j). The DTW dissimilarity value is thus stored in $C(N,M)$. The base conditions ensure that the warp path starts at (x_1, y_1), while choosing the dissimilarity value from element $C(N,M)$ ensures that the warp path ends at (x_N, y_M). The warp path can be obtained by backtracking from $C(N,M)$.

DTW has a space and time complexity of $O(N \times M)$. If only the dissimilarity value is required, and not the warp path (as is the case in this work), then the space complexity can be reduced to $O(N)$.

In our algorithm implementation, we use the *symmetric2* step pattern for the warp

path [86], which ensures that all points of both signals are mapped, and no points are skipped (skipping points is more suited for noisy signals). It also makes it simpler to normalize the DTW dissimilarity value by dividing by the total length of both sequences:

$$Dissimilarity_{normalized} = C(N, M)/(N + M) \tag{4.2}$$

While the un-normalized DTW dissimilarity value is the summation of distances between mapped points (see Figure 4.5), the normalized DTW dissimilarity value can be interpreted as the average distance between two mapped points in the two signals.

4.3.2 Overview of steps involved

Figure 4.7 shows an overview of the steps involved in our system. It consists of four major steps: Journey clustering, Route discovery and bus-stop detection, Collaboration, and Bus-stop discovery. Two of the steps (route and bus-stop detection and collaboration) are performed on the bus itself and have real-time constraints, while the other two (journey clustering and bus-stop discovery) are performed when the user is at home, and hence do not have real-time constraints.

The user acts as a mobile node in a distributed data store, where the node is the user's smartphone, storing historical journey data of the user. This data builds up on the phone each day.

Note that while we use the term 'bus' and 'bus-stop', our system works for other modes of public transport, such as trains and subways (see Section 4.4 for our evaluation on buses and trains).

4.3.3 Assumptions

Our system makes the same assumption as prior works [19, 47, 48, 16] that there is a low-power context detection system running on the phone to detect when the user

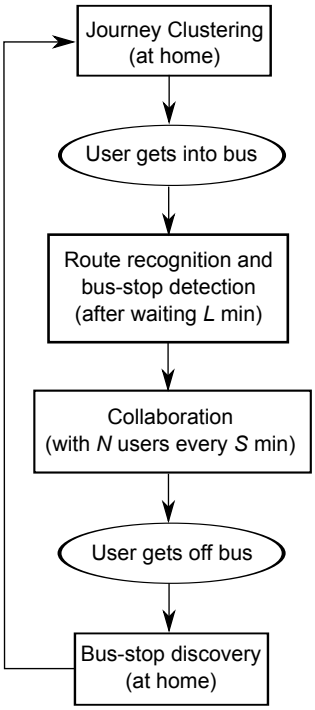


Figure 4.7: Overview of our Methodology

gets on/off a bus. This can be accomplished using low-power sensors like accelerometer [87], or using the microphone [19]. In our system, we use the barometer-based context detection algorithm described in Chapter 3.

4.3.4 Data smoothing

When the user gets into the bus, the phone starts logging barometer sensor data. The barometer is sampled at 1 Hz, and is smoothed to eliminate noise. Height values are stored relative to the height at the start of the journey, as explained in Section 4.3.1. When the user gets off the bus, logging is stopped. The logged barometer height values constitute the journey data, which is later annotated with additional information.

4.3.5 Journey clustering (at home)

The first step in our system is journey clustering, performed when user is at home and the phone is being charged. In the typical case where a user travels on two public-transport vehicles per day, after 30 days of travel, the user would have roughly 60 journey data points. To make the system scalable, the data points for the same journey are clustered into different clusters. In the example above, we would expect to obtain two clusters of 30 data points each, assuming to-and-fro trips from home to work everyday involving a single vehicle. Note that if both home-to-work and work-to-home trips consisted of 2 vehicle journeys each (for example in a bus and train), then the user would have 4 vehicle journeys per day, 120 data points in a month, and they would be clustered into 4 clusters.

We use the basic version of DTW (Section 4.3.1) to calculate a dissimilarity matrix for all journeys. If there are J number of journeys, then the dissimilarity matrix is of dimensions $J \times J$, where $J(i, j)$ stores the dissimilarity value between the i^{th} and j^{th} journey. Note that the dissimilarity matrix is not built from scratch, rather it is built incrementally as journey data is added everyday.

The journeys are then clustered by using the calculated dissimilarity matrix in k-medoids [88], a clustering technique suitable when only the dissimilarity matrix is available (as opposed to co-ordinates in space), and is more resilient to outliers than k-means.

The system is designed to run with minimum user involvement and the number of clusters is not assumed to be known. To find the value of 'k', the number of clusters, we use the Silhouette method [89], which like k-medoids, can work using only the dissimilarity matrix. It also does not assume any distribution of cluster points. The silhouette method outputs a value between [-1, 1] for each data point, where 1 indicates good tight clustering, 0 indicates that the point is an outlier, and -1 indicates that the point should be put into the neighbouring cluster. The overall silhouette value of a clustering result is the median of the silhouette values of all data points (again, using the median is more

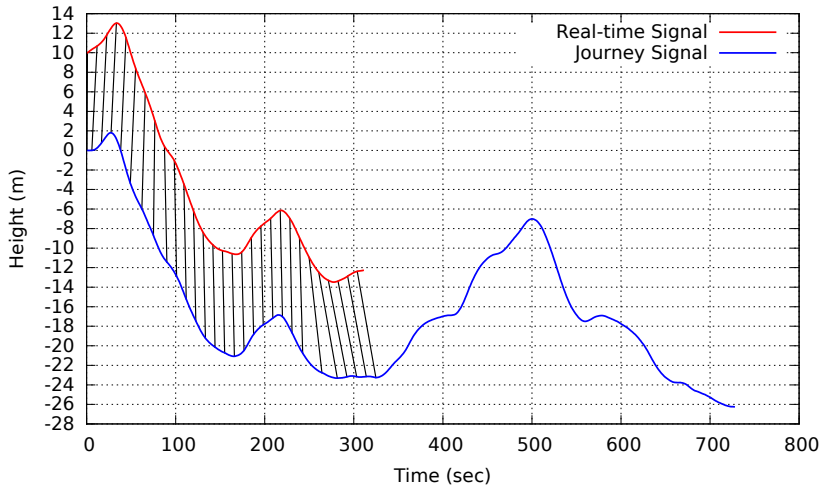


Figure 4.8: Open-ended DTW for real-time journey detection (Real-time signal offset by +10m).

resilient to outliers). If 'k' is too small or too large, then the silhouette value drops. We run k-medoids for k = 2 to J, and choose the value of 'k' which gives the maximum overall silhoutte value.

After clustering, representative journeys are chosen (one per cluster), by choosing the data point in a cluster with the highest silhouette value. These representative journeys are used for route recognition. Without clustering and choosing such representative journeys, route recognition would need to process all the J journey data points, which is not scalable.

Note that the time consuming computation of the clustering step is not the clustering itself, but the calculation of dissimilarities. This is characterized in Section 4.5.

4.3.6 Route recognition and bus-stop detection (real-time)

As a result of the clustering step, we obtain a number of journey clusters. We choose a representative data point for each cluster (the point in a cluster with the highest silhouette value), and use these points for route recognition.

Once the user gets into the bus, the phone has to first automatically detect which of

his/her prior representative bus journeys the user is currently on (in other words, match which journey cluster the current journey belongs to), but in real-time and with small delay.

This can be accomplished using Open-ended DTW [90] (OE-DTW), a version of DTW where a real-time incoming signal can be matched to the beginning portion (prefix) of a full journey signal, as shown in Figure 4.8. Recall in Section 4.3.1 that the DTW dissimilarity value is obtained from the matrix element $C(N, M)$ in Figure 4.6, ensuring that the warp path ends there. If X is instead a real-time signal (current journey data), and Y is the full journey signal (cluster representative), the best OE-DTW match of X into a prefix of Y can be found by searching for the minimum dissimilarity value from the rightmost matrix column ($C(N, j)$, for $1 \leq j \leq M$). The warp path then starts at $C(0, 0)$, but now ends at $C(N, j_{min})$, mapping all points in signal X but only to a prefix of Y (j_{min} is the j value that yields the minimum dissimilarity value in the rightmost column). j_{min}, an index into the full journey signal Y, gives us an idea how far into the barometer signal we have travelled. In Figure 4.8, the blue (longer) signal is cluster representative Y, while the red (shorter) signal is the real-time journey X.

To detect which journey the user is currently on, the dissimilarity value is computed between the current journey and every cluster representative. The representative with the smallest dissimilarity is considered as the detected journey.

This step involves a trade-off between latency and accuracy. The longer we wait before doing the detection, the more real-time data we have for the current journey, making detecion more accurate. In our system, we wait for a duration of L minutes before performing detection. The impact of L on accuracy is evaluated in Section 4.4.

Once the route has been recognised, the OE-DTW algorithm is continued in order to detect bus-stops. The OE-DTW algorithm, as shown in Figures 4.8 and 1.3, gives us an idea how 'far' the user has travelled towards the destination, and when the user reaches a bus-stop. These bus-stop points are annotated in the historical traces, using

the bus-stop extraction step described later.

4.3.7 Collaboration (real-time)

After a user performs route recognition, the user then communicates using phone-to-phone communication (Bluetooth/WiFi-Direct) with other users in the same bus, and exchanges the best cluster representative, that is output from the journey detection step, along with any previously detected bus-stop data. Note that the cluster representative exchanged is of a prior user journey on the bus. The ongoing (and hence incomplete) journey data on the bus is not exchanged.

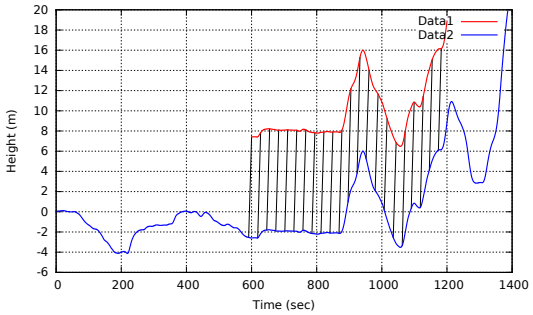
The bulk of the data exchanged is the barometer signal of the cluster representative, which due to the low sampling rate of 1 Hz, is only a few kB in size. For example, considering a a 30 minute journey, the barometer signal would have about 1800 timestamped height values. Each value is 8 bytes (4 bytes for timestamp, 4 bytes for height value), so the signal size is only 14 kB without compression.

As explained in the next step, we would like to exchange data with a variety of users getting on/off different bus stops. To achieve this, we communicate with N users every S min, where S is ideally the average time to travel between bus-stops in the city. A high value of N and low value of S involves more collaboration but at the expense of power. Suitable values for N and S is evaluated in Sections 4.4 and 4.5.

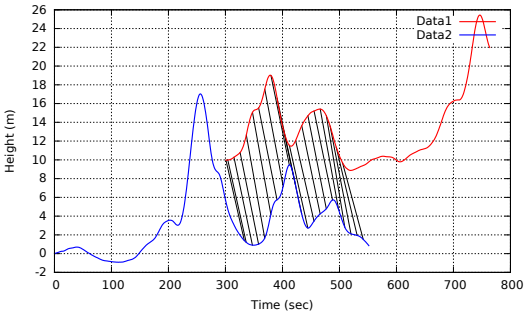
Without collaboration, users only have barometer data for their own journeys, and can identify only their own source and destination bus-stops. Collaboration not only enables bus-stop discovery for intermediate stops (explained next), but also enables the user to obtain annotated barometer data for parts of the bus route *before* the source stop and *after* the destination stop, by appending the barometer data from other users. This is useful for tourists when they receive route information from neighbouring bus users.

4.3.8 Bus-stop discovery (at home)

At the end of the bus journey, the phone obtains the barometer signals from several users on the same bus. In this step, we match these signals to identify those points in the barometer signal corresponding to intermediate bus-stops. This step can be performed at the same time as clustering, when the user is at home and the phone is being charged.



(a) Subset matching (Data 1: offset by +7.5m,+600sec)



(b) Overlap matching (Data 1: offset by +10m,+300sec)

Figure 4.9: Different cases for matching signals between two users: Subset and Overlap matching.

Different users in the bus most likely have different (source, destination) bus-stops. Considering a pair of users in the bus, two cases arise: either one user's journey is a subset of the other's journey, or the two users' journeys only overlap, as illustrated in Figure 4.9a and Figure 4.9b respectively.

In bus-stop discovery, we identify those points in the barometer signal corresponding to bus-stops. For example, in Figure 4.9a, if the blue (longer) signal is User A, and the red (smaller) signal is User B, then we would like to find the 2 points in User A's signal where User B got on and got off. These 2 points would correspond to bus-stops. Similarly, we would like to do the same in the case of overlapped journeys in Figure 4.9b.

This matching is possible using two versions of DTW. For subset matching, we use a version of DTW called open-ended open-beginning DTW (OE-OB-DTW) [91], as shown in Figure 4.9a, which is an extension of OE-DTW. By modifying the base conditions in

Section 4.3.1 to set the leftmost column to zero ($C(0, j) = 0$ for $0 \leq j \leq M$), we allow the warp path to start at any position in the signal Y. Similar to OE-DTW, we check the rightmost column for the minimum dissimilarity value. So, the warp path starts anywhere in the leftmost column, and ends anywhere in the rightmost column, covering only a subset of Y, the larger signal. In the graph in Figure 4.9a, the blue (larger) signal is Y, while the red (smaller) signal is X.

For overlap matching, as shown in Figure 4.9b, we modify the base conditions to set the leftmost column to zero ($C(0, j) = 0$ for $0 \leq j \leq M$), like in OE-OB-DTW. Unlike OE-OB-DTW, we check the topmost row for the minimum dissimilarity value ($C(i, M)$, for $1 \leq i \leq N$). This ensures that the warp path starts in the leftmost column and ends in the topmost row, mapping a prefix of signal X to a postfix of signal Y. In the graph in Figure 4.9b, the blue (lower) signal is Y, while the red (higher) signal is X.

To our knowledge, we are not aware of another use-case of the overlap version of DTW, and so far have not found a name for it in the literature. In this chapter, we refer to the 2 variants as subset and overlap matching.

After a initial build-up period involving iterations of the above steps, each user obtains a barometer signal for each journey annotated with bus-stop points. This is useful to provide a 'countdown' of bus-stops as the user travels to his/her destination, and upload the location of the bus to a server, if Internet connectivity is available.

4.4 Evaluation

We evaluate our system using real-world barometer data as well as in a trace-based simulation. We present 4 sets of results for real-world data, and 2 sets of results for simulation, described in the following sections.

Table 4.1: Summary of collected journey traces

Sl No.	Journey Type	Data points	Journey duration	Num of stops	Distance (km)
1	Bus	9	5-10 min	4	1.8
2	Train	9	15-20 min	5	8.9
3	Bus	9	15-24 min	8	3.1
4	Bus	10	15-20 min	9	4.3
5	Train	10	19-24 min	6	11.9
6	Bus	10	10-15 min	8	2.7
7	Car	4	12-17 min	N/A	12.6

4.4.1 Real-world data

We have collected 61 real-world barometer traces from 7 different journeys using 3 volunteers and 5 Android phones (Galaxy S3/4, Nexus 4/5/6) in different positions (hand, bag, pocket). The traces are from the volunteers' daily commute, and are not controlled journeys. Out of the 61 traces, 26 traces were collected during peak hours with slow traffic and congestion. Table 4.1 summarizes the traces collected.

The following sections evaluate the 4 different steps in our system using the real-world traces.

Journey clustering

To evaluate clustering, we assume that all 61 traces have been collected by a single individual, and need to be clustered into different journey clusters using k-medoids and the silhouette method discussed in Section 4.3.

Table 4.2 shows the confusion matrix after clustering. The ground truth clusters are shown as rows, and the detected clusters are shown as columns. As can be seen, the silhouette method detected 8 instead of 7 clusters.

The clustering results are however quite good. 4 clusters (Journey 2, 4, 5, 7) are perfectly identified, and 1 cluster (Journey 6) is almost correctly identified. The 2 clusters (Journey 1 and 3) are partially correct, with 6 points each correctly identified.

3 points of Journey 3 are placed in a fictitious detected cluster 8. This is not very

Table 4.2: Confusion matrix for Clustering. Rows are ground truth clusters and columns are detected clusters.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Remark**
1	6	0	0	0	0	3	0	0	Harmful
2	0	9	0	0	0	0	0	0	Correct
3	0	0	6	0	0	0	0	3	Not harmful
4	0	0	0	10	0	0	0	0	Correct
5	0	0	0	0	10	0	0	0	Correct
6	0	0	0	0	0	9	0	1	Almost correct
7	0	0	0	0	0	0	4	0	Correct

problematic since splitting of clusters simply yields multiple cluster representatives for the same journey during journey detection. However, 3 points of Journey 1 are grouped with detected cluster 6, which already has 9 points from Journey 6, which can possibly lead to a wrong choice of cluster representative for Journey 6. The reason for the bad clustering of Journey 1 is because it is an extremely short journey with little terrain variations, and hence lacks prominant signal features.

In spite of 1 harmful cluster, the overall clustering result is good, with a precision of 87.1% and recall of 81.9%, a RAND index of 95.9%, and an adjusted RAND index of 82.1%.

The good clustering result demonstrates two points: First, that there is sufficient terrain data variation between journeys to distinguish them, and second, that the barometer yields a stable and clean signal irrespective of hand movement, phone position, and orientation.

Note that our traces are of relatively short journeys, often in slow-moving vehicles. We expect clustering to be even better for longer journeys and faster vehicles, which would generate a larger number of features in the barometer signal. Also, note that clustering is performed only for that particular user's journeys, and requires that the signals are unique only for that user. The barometer signal does not need to unique w.r.t. the global set of barometer signals of all journeys in a city for all users.

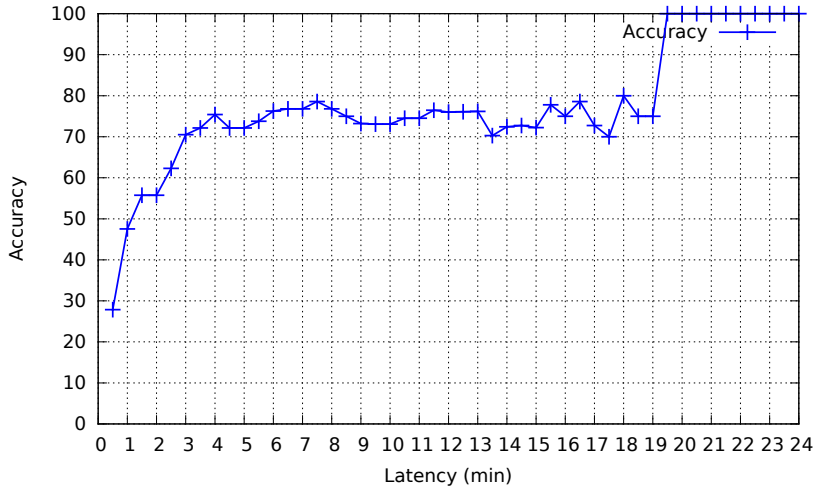


Figure 4.10: Trade-off between latency and detection accuracy.

Journey Detection

We use the real-world traces to evaluate the trade-off between latency and accuracy during journey detection, in order to choose a suitable value for L in Section 4.3.

We vary the latency L from 0.5 min to 24 min in 0.5 min increments, and calculate the detection accuracy over each of the 61 journeys in turn. For each journey, we take only the first L minutes of data, and match using OE-DTW to the other 60 traces, and calculate the accuracy.

Figure 4.10 shows the variation of Accuracy v/s Latency. As expected, with more latency, the accuracy improves to about 75% in 4 minutes. This indicates that a suitable value of L is 4 minutes.

Analysis of real data from a public bus transportation system in Singapore consisting of 235 bus routes and 2,256,911 trips over a single day shows that the average duration of a bus ride is 14 minutes, while the maximum duration can be as large as 156 minutes. This shows us that the 4 min latency is quite reasonable, especially for longer bus rides.

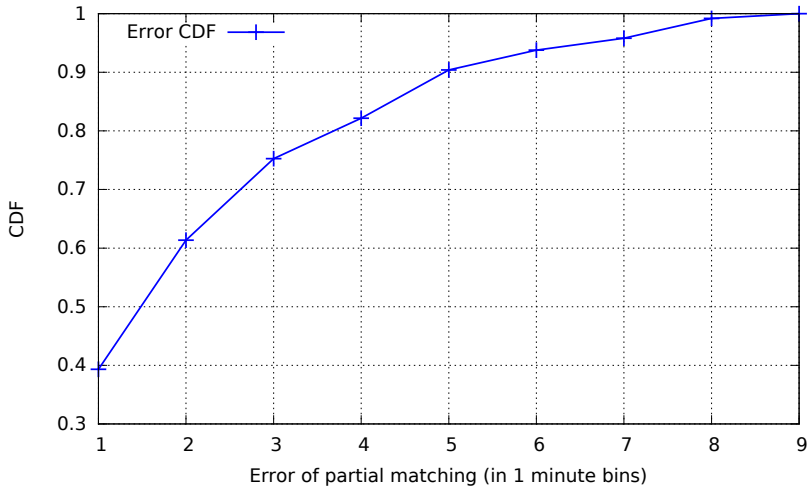


Figure 4.11: CDF of errors for subset matching.

Subset Matching

Out of the 61 traces collected, 33 traces over the 6 public-transport journeys were additionally annotated with ground truth of the bus-stop points. We use the ground truth data to evaluate subset and overlap matching for bus-stop extraction, described in this and the next section.

To evaluate subset matching, for each cluster, we consider all possible pairs of traces. For each pair of traces (note that both belong to the same cluster and hence the same journey), we slice one of the traces, where the slice length is measured in terms of number of bus-stops. Each trace is sliced all possible ways. For example, in a trace containing 5 bus-stops (1, 2, 3, 4, 5), with a slice length of 2 bus-stops, the different possible slices are (1, 3), (2, 4), and (3, 5).

We then match the slice to the other trace (full journey) in the trace pair using subset matching, which yields the positions of the bus-stop points in the full signal. Finally, we calculate the error between the detected point and the ground truth bus-stop point in terms of time along the time-axis in the full journey signal. This process is repeated for all trace pairs in all clusters, and for all possible slice lengths.

Figure 4.11 shows the CDF of errors in the form of 1 minute error bins. On the y-axis, a error of 3 min indicates that the error bin is 2 to 3 minutes. From the CDF, it can be seen that the more that 75% of the time, the error is 3 minutes or less. Note that while the ideal error is 0 min, the baseline error is the time a bus waits at a bus-stop (typically about 0.5 min on average, and goes to a few minutes for peak-hour bus-stops). This is because a bus-stop is in reality not indicated by just one point on the barometer signal, but by several consecutive points when the bus waits at the bus-stop. For example, if a bus waited at a stop for 2 minutes, then this stop corresponds to 2 minutes of consecutive points in the barometer signal.

Note that the slice length counter-intuitively does not affect the error. For example, a larger slice does not mean that the detection is more accurate. In reality, it depends on the features present in the signal. For example, a short slice with prominant signal features can match better than a long slice that has long periods of no height change at its ends. Consequently, we find that the error distribution and average error are almost the same for different slice lengths.

Overlap matching

We evaluate overlap matching in a similar manner as subset matching by slicing traces in different ways. Again, for each cluster, we consider all possible pairs of traces. For each pair, we slice both traces such that they overlap by B bus-stops. For example, if a journey consists of 5 stops (1, 2, 3, 4, 5), and the overlap is 2 bus-stops, then the possible ways to slice each trace pair is {(1,3) , (1,5)}, {(1,4) , (2,5)}, and {(1,5) , (3,5)}.

These overlapping slices are matched using DTW overlap matching and the error is calculated as in the previous section. Similar to the previous result, we find that the overlap length does not effect the error, and the average error and distribution remains similar for different overlap lengths.

Figure 4.12 shows the CDF of errors in the form of 1 minute bins as before. It

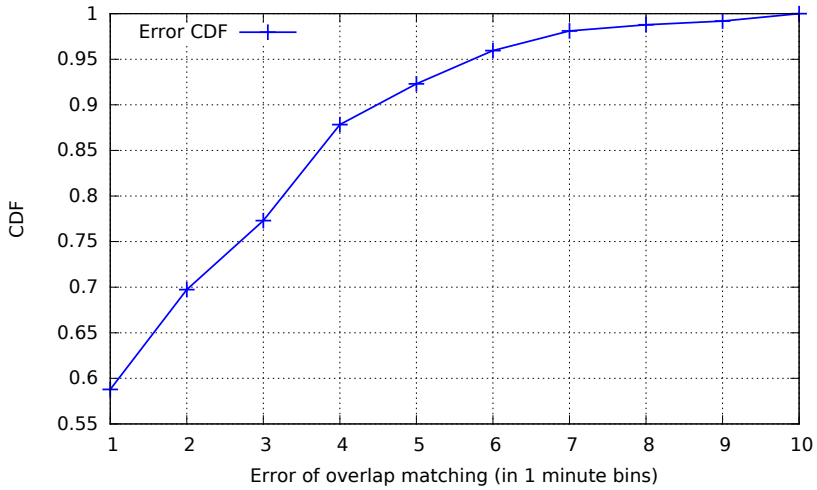


Figure 4.12: CDF of errors for overlap matching.

performs slightly better than subset matching, with more than 75% of errors being 3 minutes or less.

Considering that our technique does not use location at all, we feel that the average error of 2 min in bus-stop detection for subset and overlap matching is quite acceptable.

4.4.2 Simulation

After users in the bus detect their journeys, they not only exchange barometer data to extract bus-stops, but also exchange their historical travel times, useful for arrival-time prediction. While a centralized approach has access to historical travel times of *all* users that have ever taken a particular bus route, the collaborative approach only has access to local data available on the bus itself. In this section, we evaluate, using simulation, whether this local data is sufficient to give a reasonable estimate of journey time.

We run our simulation using 100 days of real-world bus data over 4 bus routes in a busy (and often congested) shopping district in a large urban city, using the data of over 1500 different users travelling on these routes (Only 4 bus routes were chosen in order to keep the simulation scalable). We estimate the journey time as the average of historical

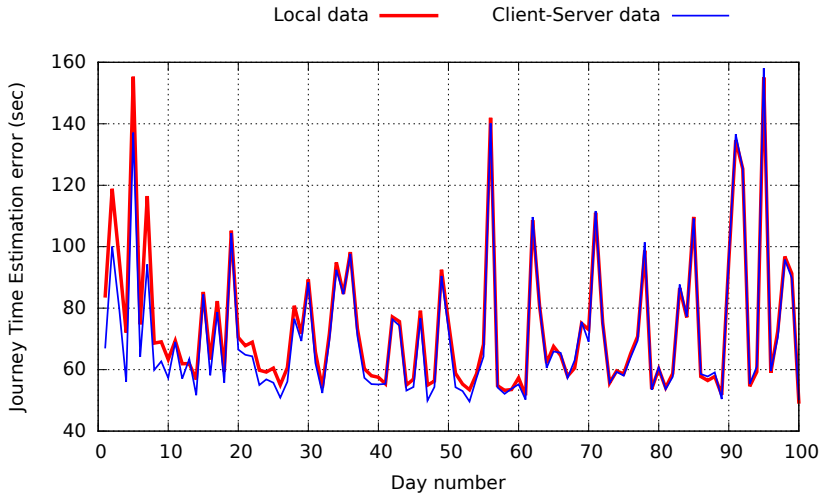


Figure 4.13: Comparison of average prediction error between local data (overall average error 1.22 min) and client-server data (overall average error 1.18 min) over 100 days.

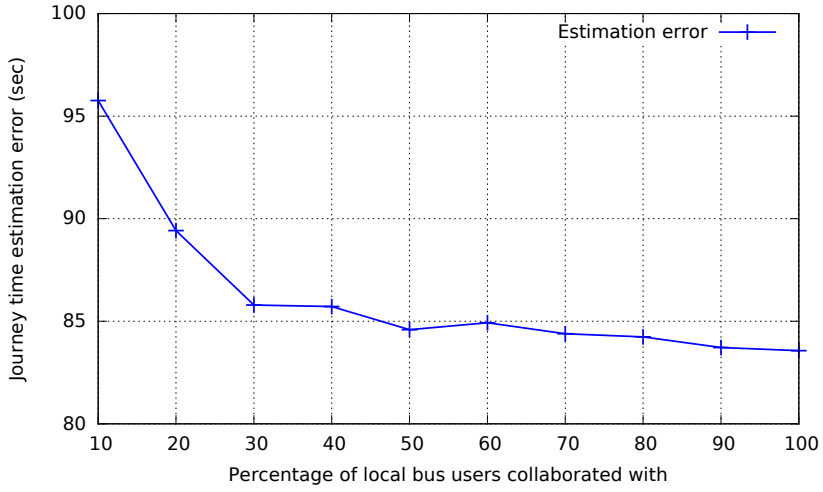


Figure 4.14: Journey time estimation error over 15 days v/s percentage of local users collaborated with on the bus.

travel times, first using a centralized approach that has access to all users' data, and second, using a collaborative approach that has access to only local data available from users in the bus itself.

Arrival-time prediction

Figure 4.13 shows the error in journey time estimation over 100 days for both the centralized as well as the collaborative approach. The collaborative approach initially has higher error than the centralized approach for the first few days. As each user builds up historical data (although at much slower rate than centralized approach), this local data is sufficient to give a reasonable estimate of the journey time, and is within 14% of the client-server estimation in just the first 15 days. The overall average error over 100 days for the collaborative approach is 1.22 min, which is close to the client-server error of 1.18 minutes.

This simulation illustrates that we can use local data from the users on the bus to give a reasonable estimate of journey time without having to use a central server.

Percentage of collaboration

The previous simulation results assume that users collaborate with all other users in the bus. We now evaluate what percentage of users in the bus are sufficient to provide local data for journey time estimation.

Figure 4.14 shows the average estimation error over the first 15 days of simulation versus the percentage of users collaborated with on the bus. As the percentage of users (and hence local data) involved in collaboration increases, the estimation error decreases. However, the trade-off is the power consumption of communicating with many users. A collaboration with 20% of users is a good trade-off, leading to a estimation error that is 21% higher than the centralized approach.

In the worst case, in a bus filled with 100 users, the user may need to collaborate

with 20 users. However, note that this is the total collaboration over the entire duration of the journey. In the next section, we show that even when communicating with 10 users every 5 minutes of the journey, we still consume significantly lower power than the GPS-based approach.

4.5 Phone implementation

We have implemented our system on Android to evaluate the execution time and power consumption, and compare with the traditional GPS-based approach. The Android phone used in our evaluation is Samsung Galaxy S4 running Android 4.3, without any modifications or rooting.

4.5.1 Execution time

In our system, journey detection is performed in real-time when the user is on the bus, and hence must be completed quickly. Clustering is performed once a day when the user is at home and the phone is on charge, and execution time is not as critical. However, unlike other steps in our system, the execution time of clustering is quadratic with increasing number of journeys, and care must still be taken to ensure scalability.

The graph in Figure 4.15 shows how the execution time of clustering and journey detection varies with increasing number of journeys. In our measurements, we use journeys with 1200 samples (about 20 minutes each). Clustering involves $O(N_J^2)$ DTW calculations, where N_J is the number of journeys, since it calculates the dissimilarity between all pairs of journeys. This is not scalable since the execution time can quickly grow to several minutes, which is unacceptable even if the phone is on charge.

However, instead of calculating the dissimilarity matrix from scratch, it can be built incrementally as journeys are added everyday. This is scalable since the user can only go on a limited number of journeys everyday. If the number of journeys per day is

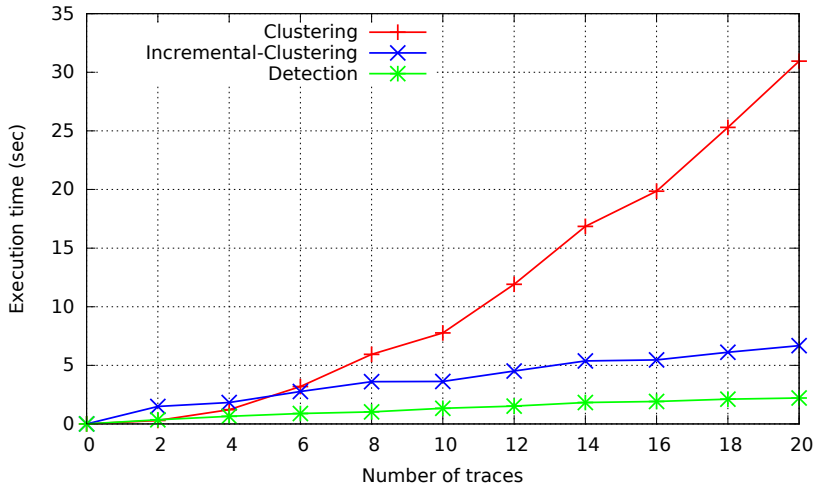


Figure 4.15: Execution Time for Clustering, Incremental clustering and Journey detection.

assumed to be constant (typically 2 to 4 vehicle journeys everyday), then the complexity reduces to $O(N_J)$ DTW calculations. This is much more scalable, as seen in Figure 4.15. Extrapolating the graph, the execution time for incremental clustering is only 25 seconds even for 60 journeys. After this, the system can start throwing away old journeys to add new ones.

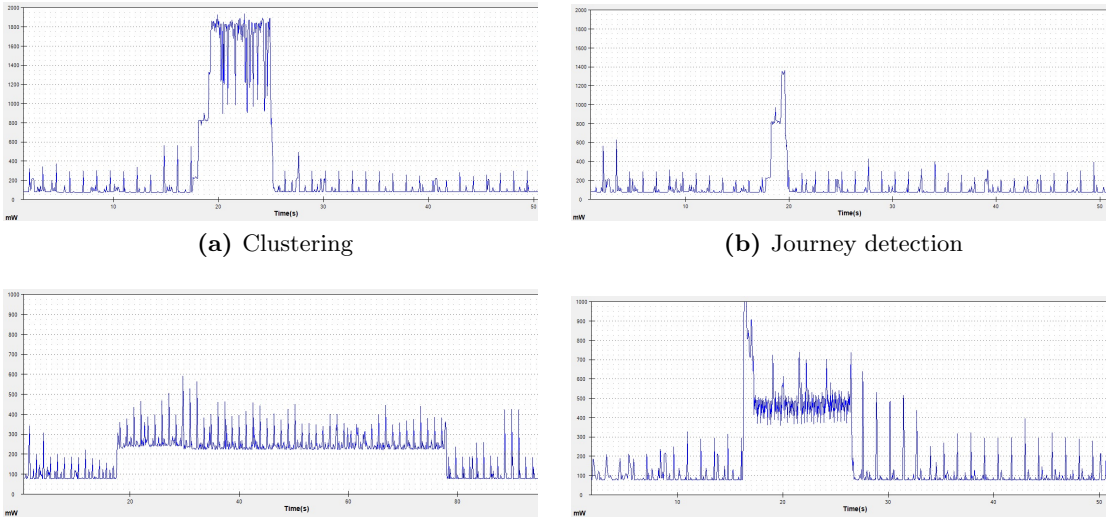
Journey detection involves $O(N_C)$ DTW calculations, where N_C is the number of journey clusters. This is scalable since the user has only a limited number of significant journeys. Figure 4.15 shows that even for the user having 20 journey clusters, the detection calculation can be completed under 2.5 seconds on the bus.

4.5.2 Power consumption

Here, we compare the power consumption of device-to-device collaboration between users in the same bus, as performed in our system, versus a GPS and server-based approach. Power consumption was measured using the Monsoon Power Meter[6], with the phone screen switched off. Wherever possible, other sub-systems and applications on the phone

[6] `http://www.msoon.com/LabEquipment/PowerMonitor/`

irrelevant to the measurement were turned off.



(a) Clustering **(b)** Journey detection

(c) GPS sampling **(d)** 4G Cellular upload

Figure 4.16: Power profiles for Clustering, Journey detection, GPS, and 4G upload.

Table 4.3 shows the power consumption of different phone operations, with duration where applicable. Note that clustering was performed on 10 journeys (involving 45 DTW calculations), detection was performed with 10 cluster representatives (involving 10 DTW calculations), Bluetooth sending includes the connection set-up and sending 30 minutes of barometer journey data to a neighbouring phone, and the 4G upload includes the tail power and upload of 60 seconds of GPS samples. Figure 4.16 shows the power profiles of clustering, detection, GPS sampling, and 4G cellular upload.

Using these power measurements, we calculated the power consumption on the bus for our collaborative-based system and compared it to the traditional GPS-based system. In our system, clustering is performed only once a day at home for a few seconds, and so the large power (about 1.5 W) is acceptable. On the bus, we assume that the phone initiates Bluetooth discovery, and communicates with 10 other users over Bluetooth LE every 5 minutes (chosen as the average time for the bus to travel between bus-stops).

For the GPS-based system, we assume the GPS is sampled for 1 minute, after which

Table 4.3: Power measurements

	Power (mW)	**Duration (sec)**
CPU (Asleep)	28	N/A
CPU (Awake)	116	N/A
Clustering	1522	7.3
Detection	734	1.9
Bluetooth (Discovery)	161	14.7
Bluetooth (Sending)	257	2.1
4G (Upload)	508	10.3
GPS Sampling	254	60.0

Table 4.4: Power consumption on a bus

Technique	**Power (mW)**
Traditional GPS-based	130.0
Collaboration-based	56.5

it is turned off for 2 minutes. After the 1 minute of sampling, the GPS samples are uploaded via 4G to a server. This cycle is repeated every 3 minutes.

Table 4.4 compares the power consumption of the collaborative-based approach versus the GPS-based approach on a bus. Our collaborative approach consumes 56.5% lower power than the GPS-based approach, due to the use of Bluetooth LE and small size of data exchanged.

4.6 Discussion

This chapter proposed a new approach for bus route and bus-stop detection using the phone's barometer and user collaboration, without using the user's location. Using this detection, and exchange of travel time data with other users on the bus, we give an estimate of arrival-time to the destination by averaging historical data.

Several works such as [92, 93, 94] exploit spatial and temporal patterns in historical travel data to give better predictions of journey time. The basic observation is that the travel time on a road segment depends on not only the time of day (temporal patterns),

but also on neighbouring road segments (spatial patterns) [95]. In particular, it depends on the upstream road segments, i.e. on the data of those road segments that lie ahead of the destination.

These works all have global knowledge of the road network, and have access to data from infrastructural sensors or crowd-sourced GPS. Our approach enables the extraction of bus-stops and travel times for road segments beyond the destination, required for better prediction, but in a distributed manner, which can be used to train a prediction model locally on each phone.

However, applying the same prediction techniques in [92, 93, 94] to our collaborative barometer approach is still challenging for two reasons: First, barometer signals are not easily mapped to road segments. Second, real-time arrival-time prediction requires real-time data from users on road segments upstream, who are in different vehicles, and requires vehicle-to-vehicle (V2V) communication. Identification of prediction models suitable for barometer data, that can handle missing data and work together with V2V communication, is the subject of ongoing work.

4.7 Conclusion

In this chapter, we proposed a new technique for bus route and bus-stop detection by matching the smartphone barometer sensor signals between users in the same bus. Unlike existing approaches, our work does not need the user's location, is completely de-centralized, requires no knowledge of bus routes beforehand and does not require an Internet connection. We evaluated our technique using 61 real-world barometer traces of 7 different journeys, as well as in a trace-based simulation of 4 bus routes involving more than 1500 users over 100 days. The results show that we are able to detect bus routes and bus-stops with reasonable accuracy, give an estimate of journey time close to the centralized approach, while consuming 56% lower power than a GPS-based approach.

Chapter 5

On-the-go application deployment

5.1 Introduction

Proximity-based smartphone applications have recently gained increasing popularity. In these applications, users interact with other users around them. Table 5.1 lists some of the popular proximity applications, along with the number of users who downloaded the application, and their average rating out of five. Many of these have more than 10 million downloads, and have high user ratings.

The rise of proximity applications has sparked an interest in scalable and energy-efficient device-to-device technologies such as LTE-direct and Bluetooth LE beacons. Proximity applications utilizing device-to-device technologies are advantageous over client-

Table 5.1: Examples of Social-Proximity Applications on Android

Application	Description	Downloads	Rating out of 5
Foursquare[1]	Find interesting places nearby, check-in for discounts	10,000,000+	4.2
Badoo[2]	Chatting, dating, making friends with people nearby	10,000,000+	4.5
Groupon[3]	Finding local deals and discounts	10,000,000+	4.6
Skout[4]	Discovering and meeting new people around	10,000,000+	4.1
Circles[5]	Finding people nearby with mutual interests	1,000,000+	4.5
Sonar[6]	Connect with friends and like-minded people nearby	1,000,000+	4.1
GrabTaxi[7]	Finding and booking nearby cabs	100,000+	4.1

server solutions since they are more power efficient, do not require external infrastructure, and do not need access to users' location.

Device-to-device technologies are expanding proximity applications to include not just interaction with people, but with physical places as well, such as bus-stops, stores, theatres, and restaurants. For example, users can check for daily specials in nearby restaurants, and movie combo offers in nearby theatres. In this thesis, we particularly focus on specialized applications services for users in proximity of public-transport transit stops.

Each place of interest is typically associated with its own dedicated application on the phone. Users need to install these apps in order to use them. However, as the number of places of interest grows, installing large number of apps quickly becomes wasteful and annoying to the user. Even for the same place of interest, there may be multiple applications available. For example, as explained in Chapter 1, several specialized public-transport applications are available to check for wheelchair access and navigability, availability of nearby cycle pitstops, and conducting on-the-go satisfaction surveys. Introducing new applications and updating existing applications is problematic as well.

To solve this problem, what is needed is lightweight and convenient installation of proximity applications *when the user is near the place of interest*, in addition to good device-to-device communication. When users go away, the apps should no longer be active nor installed on the phone. This allows users to have highly-localized interactions, with apps engaging users only when necessary, perhaps even only for a brief period of a few minutes. This is especially relevant in the public-transport context, since applications engage users only when they commute, but are unused for the rest of the day.

One possible solution for such highly-localized applications is to deploy native mobile apps 'on-the-go' to users who are in proximity to places of interest over device-to-device communication links. This eliminates the need to install apps beforehand and need for

Internet connection to the server. Delay-Tolerant Networks (DTN) [51] are best suited to deploy apps since they exploit device-to-device technologies. In contrast to Mobile Ad-Hoc Network (MANET) protocols, DTN protocols work even in the face of high user mobility. Unlike client-server solutions, DTN does not require an Internet connection to a central server, nor does it need access to a user's location, being inherently locality-specific.

Installation of native apps 'on-the-go' is however still not lightweight, since the user has to follow the tedious app installation steps. More importantly, users are wary of giving permission to unknown apps to access their phone's storage and private details. Use of web-based applications, as opposed to native applications, solves this problem as web applications run in the browser's security sandbox. Installation is lightweight since it only involves opening a web page. The browser informs the user when a web app attempts to access private details like location, which can be denied. Users are willing to allow such interactions since it is more akin to browsing a website.

While web-based apps do not have the full freedom of native applications, they are still quite powerful, having access to location, camera, and even the phone's sensors. However, they are currently limited to communication over sockets. To enable their full potential, web-based apps *need access to communication over the DTN*, thus making use of upcoming device-to-device technologies.

In this work, we propose and implement a dynamic framework for developing and deploying highly-localized web-based applications written in Javascript. This framework deploys these apps to users near the places of interest. The phone notifies the user of received apps, and if found interesting, can be opened in the mobile browser, and run with support of the framework for communication. After use, the web app can be closed either manually, or automatically when the user leaves the place of interest.

The framework is 'context-aware', using the low-power barometer sensor on the phone to detect when the user is *IDLE*, *WALKING*, or in a *VEHICLE*, as described in Chapter

3.

Context-awareness reduces power consumption of the framework in the following two ways: First, it *restricts deployment* of web apps to only those users in a relevant context (Eg: users walking nearby a store). Second, it *enables automatic switching off* of plugged-in DTN protocols when the user context changes (Eg: when the user gets into a bus). Consequently, context-awareness drastically reduces unnecessary communication between phones, saving power.

We have implemented our framework on Android, and ported it to desktop. It supports both native and web mobile applications. Our analysis of the framework shows that the memory and performance overhead incurred is small. Using real-device measurements, we show that adding context awareness reduces power consumption by at least 53%. In addition, we show via trace-based simulation of real-world public bus transport data that unnecessary communication between phones in a bus is reduced by 87%.

As an example application, we have implemented a simple on-the-go application for bus-stops to help the physically challenged. The app informs users when buses are arriving at the pick-up point, and is customized to physically challenged (wheelchair) users to help them inform bus drivers that they would like to board.

By supporting both Android and web applications, the framework exposes DTN APIs to a large community of developers. Since protocols are plugged in dynamically, it is easy to modify to adapt to current advances in DTN protocols and device-to-device communication without re-compilation of the framework.

This chapter is organized as follows: Section 5.2 describes the design of the framework. Section 5.3 discusses how context-awareness benefits the framework. Section 5.4 describes our sample application. Section 5.5 evaluates the framework. Section 5.6 discusses future work, while Section 5.7 concludes the chapter.

5.2 Design and implementation

In this section, we give a high-level overview of the design and implementation of our framework. As shown in Figure 5.1, it consists of three parts: the framework itself, the deployment application, and the Android/Web applications.

The framework consists of APIs, and protocol components implementing these APIs, all loaded at run-time. To support dynamic loading of code, the middleware uses the Apache Felix implementation of the OSGi specification [96]. It runs as a background (bound) service in Android.

We have written a simple Forwarding Layer API for applications to access routing protocols. This API supports multi-hop message transfers over the DTN. We also have a Link Layer API for one-hop communication, which supports neighbour discovery and connection-oriented communication, implemented by link layer components (Bluetooth, WiFi-direct), and used by forwarding layer components. Dynamically loaded APIs are advantageous since OSGi allows multiple incompatible versions of the API to co-exist without breaking applications.

Although Figure 5.1 shows only two protocols and a single protocol stack, the framework supports multiple protocol stacks, with protocols dependencies arranged in a directed acyclic graph. Every application can potentially load and use its own protocols, or even share protocol stacks. Protocol components are given a user-readable name in their config files. Applications can request for protocols with the specified name. Changing protocols involves loading a different protocol and giving it the same config name.

API components are broken into proxy and stub parts, in accordance with Android's inter-process communication (AIDL). The proxy and stub parts contain logic that shields upper layers from change in underlying protocols at run-time by saving state information, and hides underlying AIDL.

The deployment app is a 'special' DTN application that is used to deploy web apps,

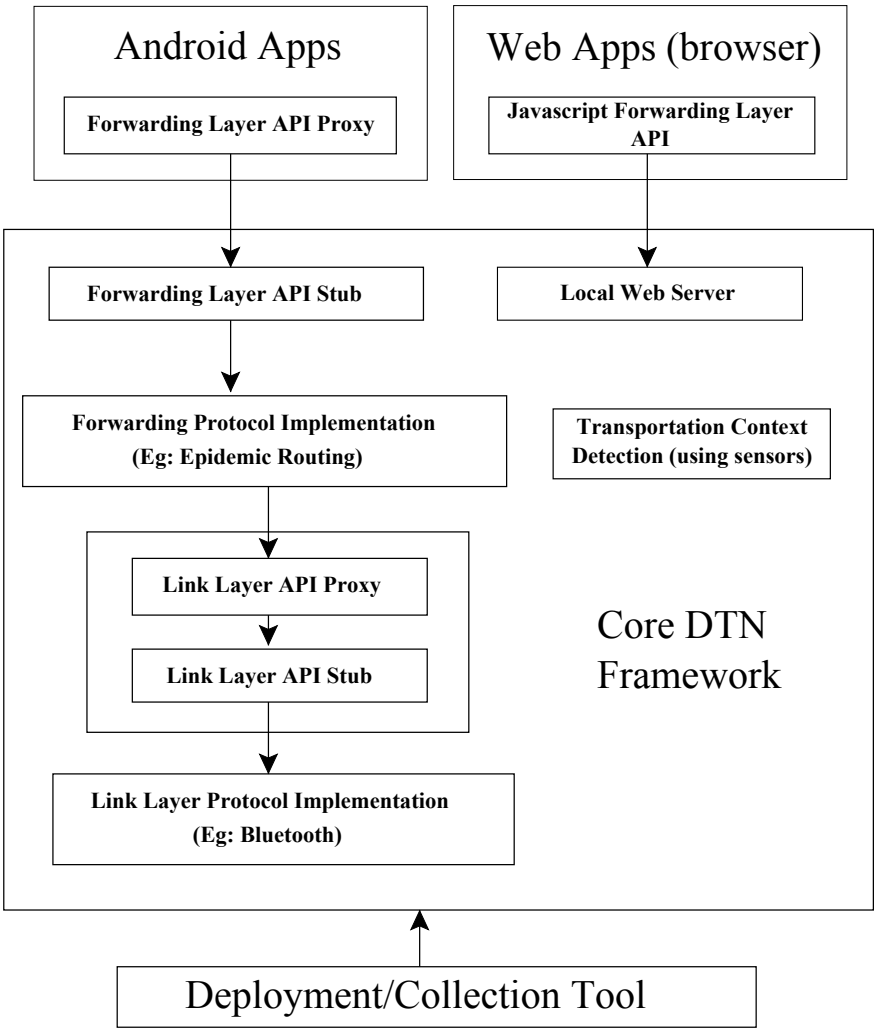


Figure 5.1: Design of the framework

protocols components (`jar` files), and even native applications (it also supports collection of logs over DTN for debugging purposes). The user is notified of received web apps, which are opened in the browser, while protocol stacks are loaded into the framework.

5.2.1 Web app support

Web apps are provided with two Javascript libraries `DtnMessage.js` and `FwdLayerAPI.js`. The first contains convenience methods for creating DTN messages, while the second exposes the Forwarding layer API.

The framework runs a local embedded web server which receives DTN API calls from web apps via AJAX, and translates them into corresponding Java calls. To overcome the same-origin policy restriction, the server supports Cross-origin resource sharing[8]. To enable web apps to receive DTN messages, the Javascript code uses AJAX long polling[9].

The framework runs on Android as well as PCs. A subset of the Android libraries were implemented on the PC so that the framework can compile and run largely without modification. The framework currently has full support for native Android applications, while web app support is in the prototype stage.

5.3 Adding context-awareness

5.3.1 Motivation for context-awareness

In this section, we add context-awareness to the framework to solve two practical problems faced during on-the-go deployment of web apps. The first problem is that applications are deployed to *everyone* in proximity of a place of interest. Not only does this waste power, but this unnecessarily disturbs users who may not be a suitable target of the application. For example, a shop that wants to advertise an ongoing special sale using a web app would prefer to deploy the app to users walking nearby, but not to users travelling in vehicles on a nearby road.

The second problem is that once users finish interacting with the web application deployed to them, they may forget to close the browser tab (for example by minimizing the browser and switching to another mobile app), leaving the DTN protocols running in the background even when the user goes away from the place of interest, wasting power.

To solve these two problems, we extend the framework by adding context-awareness (specifically, awareness of the user states *IDLE*, *WALKING*, and *VEHICLE*), determined

[8] `http://en.wikipedia.org/wiki/Cross-origin_resource_sharing`
[9] Use of WebSockets is discussed in Section 5.6

using the low-power barometer sensor on the phone, described in Chapter 3. Context-awareness enables the framework to:

1. **Restrict deployment to relevant users:** Web apps are deployed to only those users in a relevant context. For example, users who have been idle in the same place for a long time, or users travelling in vehicles on nearby roads should not receive web apps.

2. **Automatically switch off DTN protocols:** Once the user goes away from a place of interest, indicated by a change in context (for example leaving the mall by car/bus), the DTN protocols deployed with the application are switched off automatically.

5.3.2 Integrating context into the framework

By restricting deployment and switching off protocols, power consumption of the framework is significantly reduced (see the evaluation in Section 5.5).

First, based on the user state, the deployment app is switched off. Users who are idle for a long time in the same place, and users travelling in vehicles do not receive on-the-go web apps since deployment is switched off. This saves significant power since people often stay indoors (home/office) for extended periods of time, and there is no need to unnecessarily run the deployment app during those periods. Users in vehicles that rapidly pass by places of interest are also not disturbed by deployed apps.

Second, protocols deployed along with web apps are turned off when the user changes state. For example, if the user forgets to close the web app, and goes away from the place of interest by bus/car, the protocols are turned off to prevent unnecessary communication. Applications can control in which states their protocols should be turned off by modifying their configuration files.

Section 5.4.1 describes the usefulness of context-awareness with respect to our sample

application. Section 5.5 evaluates the power saved by adding context-awareness to the framework.

5.4 Sample DTN web application

To demonstrate the usefulness of on-the-go apps, and illustrate how these apps work from the user's perspective, we wrote a simple app for bus-stops and terminals to help the physically disabled as well as regular commuters board buses. This app is a web version of a DTN Android application written by students of the National University of Singapore (the framework has been used for two semesters by student groups in the Wireless and Sensor Networks course to build DTN apps for project work).

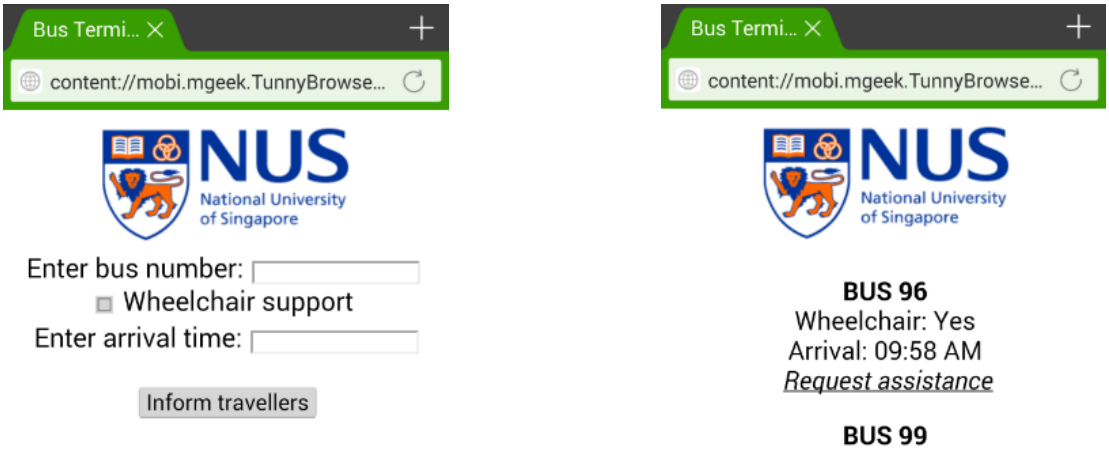
In bus terminals, commuters would like to know when the bus driver has been instructed to go to the pick-up point. Rather than install the LTA (Land Transport Authority) application from the play store beforehand, they can use the web app for a more localized and brief interaction. Physically disabled, such as wheelchair commuters, require assistance to board buses at bus stops and terminals. They need to inform drivers in advance so that they can board first, using a customized version of the app to do this. Our web app uses DTN to enable bus drivers to announce their allotted pick up time, regular commuters to receive this information, and wheelchair commuters to request drivers to assist them while boarding.

Since the framework has been ported to desktop, and supports multiple applications and users on a single device, the web app was developed locally before deploying it to mobile devices. This is especially important since it is easier to debug code using tools available in desktop browsers.

A device (laptop/mobile) located at the bus stop (alternatively can be placed on buses) deploys the web app wirelessly over the DTN to commuters nearby. Users carrying mobile devices running the framework receive the deployed app on-the-fly. In our

prototype, received web apps are displayed in the notification bar. If interested, users can open the app in their browser. The user can choose a customized interface: for example, wheelchair people can choose the web app specialized to help them.

The app for regular users only displays arrival information sent by drivers (Bus driver's interface is shown in Figure 5.2a). Wheelchair people have the additional capability to inform drivers in advance that they would like to board, as shown in Figure 5.2b. Customized DTN protocols can be optionally bundled with the web app, and plugged into the framework at run-time. After usage (i.e. the commuter has boarded), the app can be simply closed in the browser. The framework automatically releases resources used by the app.



(a) For bus drivers

(b) For Wheelchair people

Figure 5.2: Bus Stop Web App

Our sample application demonstrates the advantages of localized web apps: localized interactions, lightweight installation, and secure execution in the browser. Most importantly, these apps exploit device-to-device communication. In the future, our app will be extended to use swipe gestures and audio for the blind.

Table 5.2: Students' applications using the framework (note that these are Android, not web apps)

Application	Description
MaxTix	Last-minute movie ticket sales, re-selling, and ticket transfer
TunePulze	Sharing fitness information and songs during workouts
LiftMeUp	Rapid response and aid to elderly people who have fallen down
DeleCab	Collaboration between users to share the same cab
ChallengeMe	Interactive competitions between people in extreme sports
DisabledPersonTransport	Notifying and helping physically challenged people to board buses
vWant	Multimedia streaming based on crowd-preference
MyRadius	Share and discover information about local events and special sales
SoChat	Share ideas, files, ask questions to students around
WhereToFirst	Collects and displays crowd/queue information of nearby shops

5.4.1 Use of context-awareness

Our sample application targets users who are standing near a bus-stop. However, without context-awareness, the app is also unnecessarily deployed to users travelling in vehicles on the road, or to users in nearby home and office buildings.

Furthermore, once the user finishes using the application and gets into the bus, he/she might forget to close the browser tab (by minimizing the browser), leaving the DTN protocols running in the background. As the bus travels from stop to stop and passengers get in and out, the running protocols unnecessarily waste power by communicating with newly boarded users.

These problems are reduced using the context-awareness of the framework. By turning off deployment in phones of users who are in vehicles, or who have been idle for a long period of time in the same place, deployment to these users are avoided. In addition, by specifying that the protocols must be switched off when the user is in a vehicle, the framework automatically switches off protocols once the user has boarded the bus. This prevents unrequired communication, and consequently saves power. Section 5.5 evaluates the power saved by adding context-awareness to the framework.

Another example of the use of context-awareness is the application *vWant*[10] developed by a student group using the framework to automatically pull music preferences of users sitting around a multimedia screen, and play relevant music based on majority crowd preference. Their application determines whether the user is idle or walking, and does not pull music preferences from those who are walking.

Table 5.2 gives a summary of the Android applications developed by students using our framework. More detailed description of our students' apps, documentation, APIs, and tutorials are available at the framework's website[11].

5.5 Evaluation

In this section, we first compare the use of centralized server versus device-to-device communication with respect to power usage and latency experienced. We then evaluate the performance and memory overhead of our framework implementation. Finally, we analyse the power saved by adding context-awareness to the framework.

5.5.1 Server versus device-to-device

Existing proximity applications have to use a central server to calculate whether a phone (user) is close to a place of interest. The phone uploads its location to the server, which informs it when it is nearby interesting places. Uploading over the cellular network is costly in terms of power. Use of device-to-device technologies can reduce the power consumed, but requires periodic 'device discovery'. In this section, using power profile measurements on the Monsoon power meter, we quantify and compare the power usage of server-based versus device-to-device technologies, and show that there is indeed a power saving in spite of the device discovery process.

The power consumption depends on the frequency of location updates (for server-

[10]Demo video: `https://www.youtube.com/watch?v=DAm9gAY_uAo`
[11]Framework website: `http://www.comp.nus.edu.sg/~kartiks/nusdtn/`

Table 5.3: Monsoon power meter measurements

Operation	Power (mW)
CPU (asleep)	25
CPU (awake)	85
LTE (active)	2000
LTE (tail)	490
WiFi (scan)	300

based solution) and on frequency of device discovery (for device-to-device technologies). The lower the frequency, the lower the power consumed, but at the expense of latency. We assume the device at the 'place of interest' (eg: bus stop) is powered externally, and only focus on the user's phone's power usage here.

We consider the case where location updates to the server occur over the LTE network, while the device-to-device technology used is WiFi-direct. Using the Monsoon power meter, we measured the power profile for sending a small (ping) packet to a server on a Galaxy S3 phone, as well as the power profile for device scanning. Table 5.3 lists the power values measured. Unlike WiFi-direct, LTE suffers from a long tail (more than 12 seconds) after the packet has been sent.

Based on these measurements, we calculated the power consumption at different frequency of location updates/device scans, shown in Figure 5.3. For the same latency, server-based approaches would consume higher power. For example, at a (reasonable) 20 second worst-case latency, the power saving of using device-to-device technologies is 86%. Thus, use of device-to-device communication can benefit future proximity applications by being more power-efficient. Although the power to transfer web apps is not included in Figure 5.3, we expect that the higher bandwidth between devices would make such

[5] `https://play.google.com/store/apps/details?id=com.joelapenna.foursquared`
[6] `https://play.google.com/store/apps/details?id=com.badoo.mobile`
[7] `https://play.google.com/store/apps/details?id=com.groupon`
[8] `https://play.google.com/store/apps/details?id=com.skout.android`
[9] `https://play.google.com/store/apps/details?id=com.discovercircle10`
[10] `https://play.google.com/store/apps/details?id=me.sonar.android`
[11] `https://play.google.com/store/apps/details?id=com.grabtaxi.passenger`

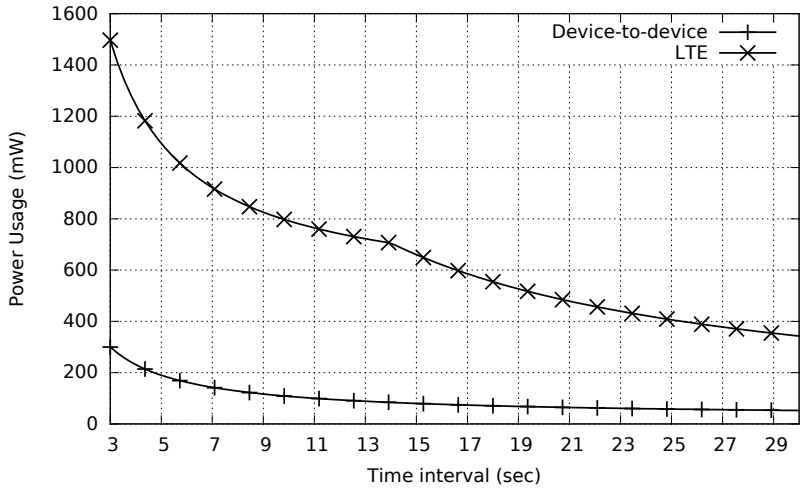


Figure 5.3: Power of server (LTE) v/s device-to-device (WiFi)

transfers faster and lower power than LTE as well.

5.5.2 Deployment latency

The latency between a device arriving at a place of interest and receiving the deployed web app is important to users. As explained earlier, this is a function of the discovery interval used (set to 10 seconds in our deployment tool). We measured the deployment latency and found it to be 6.4 seconds on average, which is reasonable. This can be modified to tradeoff savings in power (Figure 5.3).

5.5.3 Performance overhead

In DTN, devices exchange information when they come into range of one another. It is critical that data is transferred as quickly as possible during the limited contact duration time. Here we measure the performance overhead introduced by the framework during the data transfer.

Two aspects of the framework cause overhead during communication: the Inter-process communication (IPC) where data is copied from the DTN application to the

Table 5.4: Breakdown of Framework Overhead during File Transfer (time is in millisecs)

File size	Metadata size	Proxy	IPC	Stub	Transfer time	Overhead%	Overhead% (no IPC)
256 kB	32 kB	5.66	81.03	8.40	581.12	16.36	2.42
512 kB	32 kB	8.93	25.03	8.20	752.50	5.60	2.28
1 MB	32 kB	6.53	76.09	6.75	1464.98	6.10	0.91
2 MB	32 kB	7.32	25.04	36.18	2137.29	3.21	2.04
4 MB	32 kB	13.47	25.67	8.66	4309.97	1.11	0.51

framework, and the use of API Proxy/Stub (see Figure 5.1). We expect the Proxy/Stub overhead to be independent of data size, since it does not involve any data copying. We expect IPC overhead to vary linearly with data size. Note that IPC occurs only when data is initially passed from the DTN app to the routing protocol. After the initial copy, it is buffered in the framework for forwarding to other devices opportunistically.

To reduce IPC overhead for large data (such as audio/pictures), the framework allows data to be transferred from the app via files in the phone's storage. This removes the need for data copy, and is more convenient for the app. Only (optional) 'metadata' needs to be copied via IPC. For example, a mall application advertising a special sale would transfer product photos via files, while smaller textual data like name and price would be transferred via IPC.

Figure 5.4 shows the overhead involved for file transfer between two phones running the framework. The transfer was done using TCP over a 802.11b interface. Each data point is an average of 30 trials. The overhead is small compared to the transfer time, especially for moderate to large file sizes. Table 5.4 shows a breakdown of the overhead. As expected, Proxy/Stub overhead is independent of data size. IPC overhead is due to the large metadata size (32 kB) used in the experiment, but is independent of the file size. Overhead is 6% and lower for moderate to large file sizes. If IPC is not involved (i.e. if data is already buffered in the framework), then the overhead is even lower.

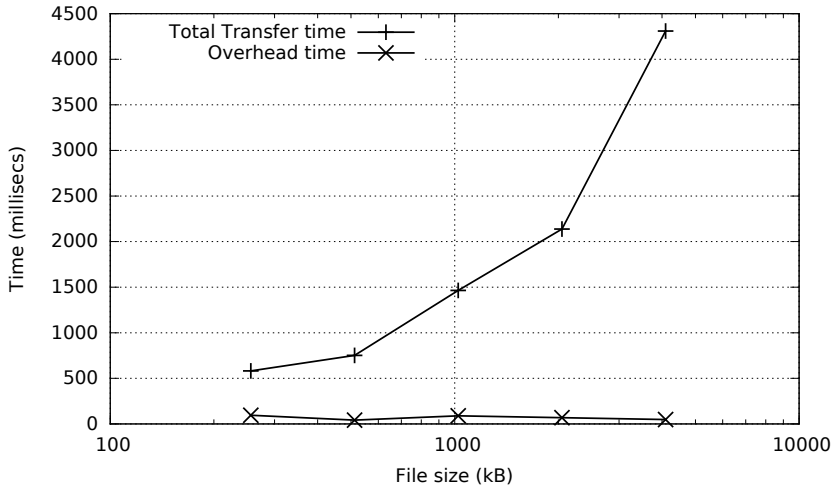


Figure 5.4: Overhead during file transfer

Table 5.5: Memory Overhead

Part of the Framework	Memory
API Proxy (application-side)	1.1 MB
Middleware Service (nothing plugged in)	8.9 MB
Middleware Service (2 APIs + 2 Protocols, no messages)	9.1 MB

5.5.4 Memory overhead

Here we measure the extra memory used by the framework. In our implementation, the API Proxy class occupies memory in the application memory space. The middleware itself runs as a service, and occupies memory separately from the application. Table 5.5 shows the memory overhead, evaluated using the Eclipse Memory Analyser.

Android imposes a limit on the amount of heap occupied. This limit varies from one Android version to another, and with RAM size. Assuming a limit of 32 MB, this leaves 23 MB of heap for message buffering. If each message is 1 MB, we can buffer 20 messages, which is too few. However, bulk of data is in the form of pictures/audio stored as files on the `sdcard`, and not in heap memory. The buffer contains only the message's metadata. If metadata is 32 kB, then the phone can buffer about 700 messages. As

newer phones have more and more RAM, we do not expect the 9 MB overhead to be significant.

5.5.5 Evaluation of context-awareness

In this section, we evaluate the context-awareness added to the framework. First, using real-device measurements, we analyse the power saved by adding context awareness to the framework to restrict deployment and turn off protocols. Second, using real-world bus transport data, we estimate the reduction in number of users involved in communication by switching off protocols.

Power saved using context-awareness

As explained previously in Section 5.3, we add context-awareness to save power by restricting deployment and switching off protocols, reducing unnecessary phone-to-phone communication. This is particularly significant when users are idle in the same place for extended periods, such as in home or office, and phones should not be running the deployment app and application protocols. However, performing context detection all the time adds to the power consumption of the system.

In this section, using real-device measurements, we evaluate the additional power consumption overhead of running a framework with context detection against a framework running without context awareness, and calculate the energy savings.

Majority of the power consumption in the framework running without context awareness is due to neighbour discovery and phone-to-phone communication. Since the amount of communication involved (and hence the power saving of context awareness) depends on the traffic generated by the mobile web application, we measure the power consumption in two cases when the framework runs without context-awareness: without any communication traffic (only neighbour discovery runs every minute), and with communication traffic (each device sends a 10 kB message every 2 minutes), which act as two

baselines for comparison.

In this experiment, we use 5 Android phones (1 Galaxy S4, 3 Galaxy S3, and 1 Nexus 4) deployed in different locations in a room, with the Galaxy S4 phone connected to the Monsoon power meter. The phones perform neighbour discovery and phone-to-phone communication using Bluetooth LE, while running the framework and a traffic generating application. Other subsystems, including screen and WiFi, are turned off during the experiment. Neighbour discovery is performed once every minute, and in the case where traffic is generated, each device sends a 10 kB message to all other phones every 2 minutes. Note that using additional phones or higher traffic would only increase the power consumption of the baselines.

We measure the average power consumption over 30 minutes for each of the following three cases:

1. Framework running *without* context awareness (*no traffic*) [Baseline 1]: Since no traffic is generated, we expect majority of the power consumption to be from neighbour discovery at the link layer protocol.

2. Framework running *without* context awareness (*with traffic*) [Baseline 2]: Since traffic is generated by the application, we expect a larger power consumption due to phone-to-phone communication in addition to neighbour discovery.

3. Framework *with* added context awareness (*with traffic*): We expect the context detection to report that the user is idle, and protocols and deployment to be switched off automatically. So, the power consumption is expected to be largely due to the overhead of context detection.

Table 5.6 lists the power consumption from measurements on the power meter for the three cases above (excluding the CPU base power of 95 mW).

From the power measurements, we can see that by using context awareness, we save 53% power of the first baseline, and 71% power of the second baseline running without

Table 5.6: Comparison of power consumption of framework using context awareness versus without context awareness

	Energy (mJ)	**Duration (sec)**	**Power (mW)**
Without context awareness (no traffic) [Baseline 1]	84620	1800.42	47
Without context awareness (with traffic) [Baseline 2]	136833	1800.43	76
With added context awareness (with traffic)	39616	1800.72	22

context awareness. Thus, by using context, we achieve significant savings in the power consumption by switching protocols and deployment off, especially important when the user is at home or office for several hours.

Reducing unnecessary communication

While the previous section evaluated the power savings using real-device measurements, in this section we evaluate using trace-based simulation of real world bus transport data how context-awareness reduces unnecessary communication between phones with respect to our sample application, which is deployed to users waiting at a bus-stop.

Once users have finished interacting with the application, they board the bus. However, users may forget to close the application, leaving it running the background while they are in the bus. As the bus travels from stop to stop, and other users get in and out, the protocols unnecessarily waste power by communicating with newly boarded users. If the bus is crowded, this can lead to a large power wastage. However, by automatically switching off protocols using context-awareness, this problem can be avoided.

To analyse the impact of using context-awareness in the real world, we have written a trace-based simulator using a day of real-world transport data from public buses in Singapore, containing data of over 1,000,000 users, and over 2000 buses. The transport data contains the timestamped information of when buses arrive at every bus-stop, and information of users getting in and out (using their transport ezlink card). The simulator uses this data to keep track of which users are present in what buses, and how crowded the buses are at each stop.

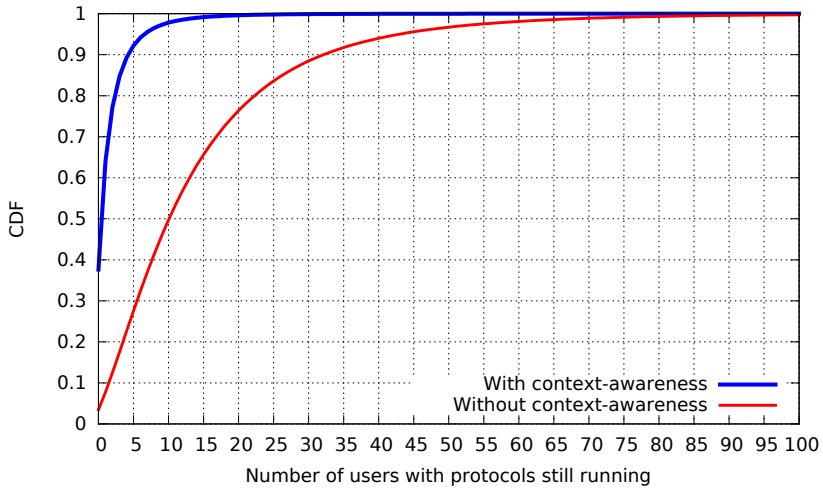


Figure 5.5: CDF of number of users on a bus with protocols still running

Figure 5.5 shows the CDF of the number of users in a bus with protocols unnecessarily still running without context-awareness when the bus arrives at a bus-stop. Without switching off protocols, these users unnecessarily communicate with other newly boarded users, wasting power.

Protocols are switched off once the bus starts moving. Figure 5.5 shows the CDF of number of users on the bus with protocols still switched on after using context-awareness. We find that the number of such 'active' users is drastically reduced compared to the case without context-awareness. On an average, there is an 87% reduction in number of users involved in unnecessary communication.

Thus, we can see that in a real-world bus system, context-awareness and automatically switching off protocols drastically reduces the unnecessary communication between users in a bus. Since the simulation does not consider users walking outside the bus, the communication saving is expected to be even larger in the real-world compared to simulation.

5.6 Discussion

- **Use of WebSockets instead of AJAX:** We will be re-writing our Javascript code to use WebSockets, now increasingly supported in mobile browsers, suitable for push-based notifications of messages received, to replace the AJAX long polling currently used.

- **Security in protocol components:** While apps run in the browser sandbox, protocol components loaded in the framework have dangerous access to Android libraries. OSGi enables fine-grained control over the libraries accessible by loaded components. In the future, we will use this to restrict a protocol's access to the phone, following the permission model of Android but on a protocol level.

5.7 Conclusion

In this chapter, we proposed a dynamic framework for deployment of on-the-go applications written in Javascript. These apps free users of the burden of installing multiple native proximity applications on the phone. They are lightweight, easy to open/close in a browser, and operate only when the user requires proximity interactions. To demonstrate their usefulness, we wrote a simple app for bus stops to help the physically disabled. We evaluated our framework and found that it has low overhead, and that adding context-awareness saves significant power. In the future, we will be enhancing our framework's web app support and analyse the performance of localized web apps in a real setting.

Chapter 6

Conclusion and future work

In this chapter, we first summarize the research contributions of this thesis. Our contributions lead to new research ideas that we are currently working on, which are discussed here as future work.

6.1 Contributions

The goal of this thesis is to develop new techniques for smartphone-based public-transport applications deployed on-the-go that make fewer assumptions on infrastructure than existing solutions, enabling them to work even in cities of developing countries. At the same time, our techniques also provide location privacy and reduce power consumption when used in cities of developed countries.

To do this, we proposed new techniques to tackle the core challenges of context detection, bus route and bus-stop detection, and deployment of specialized application services, required to provide useful public-transport application functionality, while addressing the limitations faced by existing solutions. In this thesis, we made the following contributions:

1. **Barometer-based transportation context detection:** We proposed and

implemented the first research work in literature that uses only the barometer for low-power transportation context detection of the states *idle*, *walking*, and *vehicle*. Unlike existing approaches that have high user dependence and require extensive training, our barometer-based approach is inherently independent of the user. It has similar detection accuracy while consuming lower power. In the situations where the user is waiting for a bus, and while travelling on smooth vehicles, existing accelerometer-based techniques have less than 25% detection accuracy, while our approach has almost 100% accuracy.

2. **Barometer-based vehicle context detection:** We proposed novel techniques for using only the barometer and local collaboration of users in the same bus for bus route and bus-stop detection. Our approach is the first research work that makes the fewest assumptions on infrastructure and availability of route maps, enabling it to work in even in developing countries where other approaches fail, since it is decentralized and does not require an Internet connection. In developed countries, it provides location privacy and reduces power consumption. This work is the first to demonstrate a novel use of the barometer sensor for bus route and bus-stop detection.

3. **On-the-go deployment of applications:** We proposed and implemented a dynamic framework on Android for deploying on-the-go applications written in Javascript to users in proximity of places of interest, such as near public-transport stops. Users are notified of received apps, and can run them in the browser with support of the framework for communication without the Internet. After usage, the application is stopped by closing the browser tab, or automatically when users leave the place of interest. Our deployment framework removes the need for users to install multiple applications beforehand on their phone, and allows users to choose 'specialized' applications tailored to their needs. To illustrate this, we implemented

a simple on-the-go application to help the physically challenged (wheelchair) people board the bus.

We evaluated our techniques using over 60 hours of real-world barometer transportation traces from 3 countries and 15 volunteers during their daily commute, as well as using over 900 km of elevation data of 5 cities from Google Maps. We also implement a trace-based simulation of 4 bus routes involving more than 1500 users over 100 days. The results showed that our context detection algorithm achieves 23% better accuracy than Google's Activity Recognition, while consuming 26% lower power. Our route clustering technique has a precision of 87% and recall of 81%, and we can detect the destination bus-stop with an average error of 2 minutes. Finally, we showed that collaboration between users in the same bus can be used to predict remaining travel time with an average error of 1.22 minutes.

We implemented our techniques on Android for power and execution time measurements. In addition, we provided context detection and DTN framework APIs to students of the CS4222 Wireless and Sensor Network course in NUS, for use in their course projects.

This thesis has two key insights. First, the barometer can act as a power-efficient alternative to absolute location, while still respecting location privacy. It is unaffected by hand movement, yielding clean and stable signals. Second, the users in the same bus have a rich set of local data that can be used to extract points of bus-stops and give an estimate of arrival time. These insights can be used to create a decentralized and location-'less' application.

Our contributions lead to new research ideas, which are discussed in the next section as future work.

6.2 Future work

Our contributions open up new research ideas that we are currently working on. Here, we discuss them as future work. These research ideas are also important steps towards our goal of building a location-'less' transport application using smartphones.

1. **Semi-supervised activity diary using barometer:** In this thesis, we used barometer to detect the user states *idle*, *walking*, and *vehicle*. A more challenging and important problem in the literature is detection of high-level user activities, such as home, office, shopping, school, meetings, etc., and commute between these places. Existing works track user location using smartphones [97], and reverse geocode to guess high-level activities, typically using supervised machine learning techniques when the place name is not sufficient to identify the activity [98]. However, this requires sufficient training data. Supervised detection models are also difficult to tailor to each user's unique set of activities.

   An alternative to supervised learning is semi-supervised learning, where the user is involved in labeling his/her activities for a few data points (for example, Google asks the user to label the locations of home and work after installing Google Now, and to specify if they travel by car or public-transport). This has two advantages: the need for training is removed, and activities can be tailored since the labeling is done by that specific user.

   Semi-supervised approaches are more commonly used in detecting fine-grained user activities at the same place, such as sleeping, sitting, bathing, etc. in a user's home [99, 100, 101]. They use low-power motion sensors like accelerometer and devices like RFIDs. However, these low-power sensors cannot be used for creating activity diaries where the user goes to several places, and existing semi-supervised techniques for making these travel diaries still depend on location sensors.

   In this future work, we intend to use the barometer for semi-supervised detection

of user activites, even when the user travels between different places. From data we have collected, we find that the barometer signal, observed over the entire day, exhibits characteristic patterns in the signal while the user travels between places.

Semi-supervised techniques have been used previously in speech recognition to recognise frequently used words and phrases in a new language [102]. We intend to apply these techniques to extract frequent patterns occurring in the barometer signal, which can be labeled by the user. An important advantage of applying speech recognition techniques is that context of the phrase is important - that is, not just the word, but how the word is placed in the sentence is also significant. This applies to activity diaries as well, where 'taking bus 241' can be a part of different journeys to different places.

The main advantage of using barometer is the power efficiency w.r.t. GPS. New phones come with sensor batching, where barometer data can be buffered for several minutes without waking up the main processor. Another advantage is that barometer signals are clean and stable, unlike signals from accelerometer and GPS that are noisy.

If this research work is successful, we intend to replace the context-detection component in this thesis with this system, to yield a richer set of user activities.

2. **Fully-fledged public-transport app functionality:** In this thesis, we implemented new techniques catering to a subset of the features of public-transport applications, namely bus-stop countdown and ETA. However, we did not focus on other important application features, such as bus service number identification and next bus arrival.

   In this thesis, although we distinguish different journeys from one another and detect bus-stop points, we do not identify the bus service number nor the bus-stop IDs, since we assume the lack of route maps. In the case where route maps

are available, barometer data could potentially be used to guess the bus service number and identify bus-stop IDs as well.

Using bus route maps, which in turn contain locations of bus-stops, we pulled terrain data for over 150 bus routes in Singapore using Google Maps. Our preliminary results show that by just using the barometer signal trace and the Google Map data, we are able to guess the top likely bus service numbers and bus routes that the user has taken, provided that the search space is not made too large.

Terrain data pulled from Google Maps for that bus route can be annotated with bus-stop IDs, and this can be used for bus-stop detection, instead of relying on collaboration with other users.

Two improvements this would provide over the current system are: First, bus service numbers and bus-stop IDs and names can be displayed to the user, which is more informative. Second, this method uses data from Google Maps, which is easy to obtain compared to users travelling on buses and collecting barometer data.

Pushing information to users waiting at bus-stops without cellular data connectivity, and without vehicle-to-vehicle communication, is impossible. However, if data connectivity is available, then collaboration can potentially be done at a server instead of on the bus. Such a system would be very power efficient since the bulk of the power consumption caused by neighbour discovery during collaboration would be removed and the location sensor is not used.

However, aggregating the barometer data at a global level at the server is more challenging than at a local level in the bus. The challenge lies in identifying which users are in the same bus route, in order to suitably piece together barometer data. Addressing this challenge is left as future work.

Bibliography

[1] Land and Transport Authority of Singapore: Publications and research. http://www.lta.gov.sg/content/ltaweb/en/publications-and-research.html.

[2] IBM traffic congestion: Overview. http://www.ibm.com/smarterplanet/sg/en/traffic_congestion/ideas/.

[3] Report on study of road traffic congestion in Hong Kong, December 2014.

[4] How smartphones can improve public transit. http://www.wired.com/2011/04/how-smartphones-can-improve-public-transit/.

[5] Smartphones finally overtook dumbphone sales. http://techcrunch.com/2013/08/14/gartner-q2-smartphone/.

[6] Lta mytransport application. http://www.mytransport.sg.

[7] Moovit application. http://moovitapp.com.

[8] Google now. https://www.google.com/landing/now/.

[9] Google's Activity Recognition API. http://developer.android.com/google/play-services/location.html.

[10] Yu Zheng, Quannan Li, Yukun Chen, Xing Xie, and Wei-Ying Ma. Understanding mobility based on gps data. In *Proceedings of the 10th International Conference on*

Ubiquitous Computing, UbiComp '08, pages 312–321, New York, NY, USA, 2008. ACM.

[11] Sasank Reddy, Min Mun, Jeff Burke, Deborah Estrin, Mark Hansen, and Mani Srivastava. Using mobile phones to determine transportation modes. *ACM Trans. Sen. Netw.*, 6(2):13:1–13:27, March 2010.

[12] Jason Ryder, Brent Longstaff, Sasank Reddy, and Deborah Estrin. Ambulation: A tool for monitoring mobility patterns over time using mobile phones. In *Proceedings of the 2009 International Conference on Computational Science and Engineering - Volume 04*, CSE '09, pages 927–931, Washington, DC, USA, 2009. IEEE Computer Society.

[13] Sarfraz Nawaz, Christos Efstratiou, and Cecilia Mascolo. Parksense: A smartphone based sensing system for on-street parking. In *Proceedings of the 19th Annual International Conference on Mobile Computing & Networking*, MobiCom '13, pages 75–86, New York, NY, USA, 2013. ACM.

[14] Shuangquan Wang, Canfeng Chen, and Jian Ma. Accelerometer based transportation mode recognition on mobile phones. In *Proceedings of the 2010 Asia-Pacific Conference on Wearable Computing Systems*, APWCS '10, pages 44–46, Washington, DC, USA, 2010. IEEE Computer Society.

[15] Samuli Hemminki, Petteri Nurmi, and Sasu Tarkoma. Accelerometer-based transportation mode detection on smartphones. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, page 13. ACM, 2013.

[16] James Biagioni, Tomas Gerlich, Timothy Merrifield, and Jakob Eriksson. Easytracker: Automatic transit tracking, mapping, and arrival time prediction using smartphones. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, SenSys '11, pages 68–81, New York, NY, USA, 2011. ACM.

[17] 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*, SenSys '10, pages 85–98, New York, NY, USA, 2010. ACM.

[18] 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 *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, pages 85–98, New York, NY, USA, 2009. ACM.

[19] Pengfei Zhou, Yuanqing Zheng, and Mo Li. How long to wait? predicting bus arrival time with mobile phone based participatory sensing. *Mobile Computing, IEEE Transactions on*, 13(6):1228–1241, June 2014.

[20] Pengfei Zhou, Shiqi Jiang, and Mo Li. Urban traffic monitoring with the help of bus riders. In *Distributed Computing Systems (ICDCS), 2015 IEEE 35th International Conference on*, pages 21–30, June 2015.

[21] Arvind Thiagarajan, Lenin Ravindranath, Hari Balakrishnan, Samuel Madden, and Lewis Girod. Accurate, low-energy trajectory mapping for mobile devices. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 267–280, Berkeley, CA, USA, 2011. USENIX Association.

[22] Moves application. `http://www.moves-app.com/`.

[23] Carlos Carrion, Francisco Pereira, Rudi Ball, Fang Zhao, Youngsung Kim, Kalan Nawarathne, Naijia Zheng, Chris Zegras, and Moshe Ben-Akiva. Evaluating fms: A preliminary comparison with a traditional travel survey. In *93rd Annual Meeting of the Transportation Research Board*, 2014.

[24] Caitlin D Cottrill, Francisco Camara Pereira, Fang Zhao, Ines Ferreira Dias, Hock Beng Lim, Moshe E Ben-Akiva, and P Christopher Zegras. Future mobility survey: Experience in developing a smartphone-based travel survey in singapore. *Journal of the Transportation Research Board*, 2354:59–67, 2013.

[25] OzlemDurmaz Incel, Mustafa Kose, and Cem Ersoy. A Review and Taxonomy of Activity Recognition on Mobile Phones. *BioNanoScience*, 3(2):145–171, 2013.

[26] Cheng Bo, Xiang-Yang Li, Taeho Jung, Xufei Mao, Yue Tao, and Lan Yao. Smartloc: Push the limit of the inertial sensor based metropolitan localization using smartphone. In *Proceedings of the 19th annual international conference on Mobile computing & networking*, pages 195–198. ACM, 2013.

[27] Ian Anderson and Henk Muller. Practical activity recognition using gsm data. Technical Report CSTR-06-016, Department of Computer Science, University of Bristol, July 2006.

[28] Timothy Sohn, Alex Varshavsky, Anthony LaMarca, MikeY. Chen, Tanzeem Choudhury, Ian Smith, Sunny Consolvo, Jeffrey Hightower, WilliamG. Griswold, and Eyal Lara. Mobility detection using everyday gsm traces. In Paul Dourish and Adrian Friday, editors, *UbiComp 2006: Ubiquitous Computing*, volume 4206 of *Lecture Notes in Computer Science*, pages 212–224. Springer Berlin Heidelberg, 2006.

[29] Jeffrey Hightower, Sunny Consolvo, Anthony LaMarca, Ian Smith, and Jeff Hughes. Learning and recognizing the places we go. In *Proceedings of the 7th International Conference on Ubiquitous Computing*, UbiComp'05, pages 159–176, Berlin, Heidelberg, 2005. Springer-Verlag.

[30] Gerald Bieber, Philipp Koldrack, Christopher Sablowski, Christian Peter, and Bodo Urban. Mobile physical activity recognition of stand-up and sit-down tran-

sitions for user behavior analysis. In *Proceedings of the 3rd International Conference on PErvasive Technologies Related to Assistive Environments*, page 50. ACM, 2010.

[31] Apiwat Henpraserttae, Surapa Thiemjarus, and Sanparith Marukatat. Accurate activity recognition using a mobile phone regardless of device orientation and location. In *Body Sensor Networks (BSN), 2011 International Conference on*, pages 41–46. IEEE, 2011.

[32] Toshiki Iso and Kenichi Yamazaki. Gait analyzer based on a cell phone with a single three-axis accelerometer. In *Proceedings of the 8th conference on Human-computer interaction with mobile devices and services*, pages 141–144. ACM, 2006.

[33] Jun Yang. Toward physical activity diary: motion recognition using simple acceleration features with mobile phones. In *Proceedings of the 1st international workshop on Interactive multimedia for consumer electronics*, pages 1–10. ACM, 2009.

[34] Pekka Siirtola and Juha Röning. Recognizing human activities user-independently on smartphones based on accelerometer data. *International Journal of Interactive Multimedia & Artificial Intelligence*, 1(5), 2012.

[35] Pravein Govindan Kannan, Seshadri Padmanabha Venkatagiri, Mun Choon Chan, Akhihebbal L Ananda, and Li-Shiuan Peh. Low cost crowd counting using audio tones. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, pages 155–168. ACM, 2012.

[36] Chengwen Luo and Mun Choon Chan. Socialweaver: collaborative inference of human conversation networks using smartphones. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, page 20. ACM, 2013.

[37] Pengfei Zhou, Yuanqing Zheng, Zhenjiang Li, Mo Li, and Guobin Shen. IOdetector: A generic service for indoor outdoor detection. In *Proceedings of the 10th acm conference on embedded network sensor systems*, pages 113–126. ACM, 2012.

[38] Chengwen Luo, Hande Hong, and Mun Choon Chan. Piloc: A self-calibrating participatory indoor localization system. In *Information Processing in Sensor Networks, IPSN-14 Proceedings of the 13th International Symposium on*, pages 143–153. IEEE, 2014.

[39] Anshul Rai, Krishna Kant Chintalapudi, Venkata N Padmanabhan, and Rijurekha Sen. Zee: zero-effort crowdsourcing for indoor localization. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, pages 293–304. ACM, 2012.

[40] Jieying Zhang, E. Edwan, Junchuan Zhou, Wennan Chai, and O. Loffeld. Performance investigation of barometer aided gps/mems-imu integration. In *Position Location and Navigation Symposium (PLANS), 2012 IEEE/ION*, pages 598–604, April 2012.

[41] M. Tanigawa, H. Luinge, L. Schipper, and P. Slycke. Drift-free dynamic height sensor using mems imu aided by mems pressure sensor. In *Positioning, Navigation and Communication, 2008. WPNC 2008. 5th Workshop on*, pages 191–196, March 2008.

[42] S. Vanini and S. Giordano. Adaptive context-agnostic floor transition detection on smart mobile devices. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on*, pages 2–7, March 2013.

[43] Kartik Muralidharan, Azeem Javed Khan, Archan Misra, Rajesh Krishna Balan, and Sharad Agarwal. Barometric phone sensors–more hype than hope! *15th International Workshop on Mobile Computing Systems and Applications*, 2014.

[44] Wei Shen, Yiannis Kamarianakis, Laura Wynter, Jingrui He, Qing He, Rick Lawrence, and Grzegorz Swirszcz. Traffic velocity prediction using gps data: Ieee icdm contest task 3 report. In *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*, pages 1369–1371. IEEE, 2010.

[45] Dihua Sun, Hong Luo, Liping Fu, Weining Liu, Xiaoyong Liao, and Min Zhao. Predicting bus arrival time on the basis of global positioning system data. *Transportation Research Record: Journal of the Transportation Research Board*, (2034):62–72, 2007.

[46] Dalia Tiesyte and Christian S Jensen. Similarity-based prediction of travel times for vehicles traveling on known routes. In *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, page 14. ACM, 2008.

[47] Sarfraz Nawaz and Cecilia Mascolo. Mining users' significant driving routes with low-power sensors. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, SenSys '14, pages 236–250, New York, NY, USA, 2014. ACM.

[48] Yan Michalevsky, Gabi Nakibly, Aaron Schulman, and Dan Boneh. Powerspy: Location tracking using mobile device power analysis. *arXiv preprint arXiv:1502.03182*, 2015.

[49] Jingyu Hua, Zhenyu Shen, and Sheng Zhong. We can track you if you take the metro: Tracking metro riders using accelerometers on smartphones. *arXiv preprint arXiv:1505.05958*, 2015.

[50] Haibo Ye, Tao Gu, Xianping Tao, and Jian Lu. Crowdsourced smartphone sensing for localization in metro trains. In *A World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on*, pages 1–9. IEEE, 2014.

[51] Kevin Fall. A delay-tolerant network architecture for challenged internets. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '03, pages 27–34, New York, NY, USA, 2003. ACM.

[52] Jörg Ott and Dirk Kutscher. Bundling the web: Http over dtn. *Proceedings of WNEPT*, 2006.

[53] Aruna Balasubramanian, Yun Zhou, W Bruce Croft, Brian Neil Levine, and Aruna Venkataramani. Web search from a bus. In *Proceedings of the second ACM workshop on Challenged networks*, pages 59–66. ACM, 2007.

[54] Jay Chen, Lakshminarayanan Subramanian, and Jinyang Li. Ruralcafe: Web search in the rural developing world. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 411–420, New York, NY, USA, 2009. ACM.

[55] Mikko Pitkanen, Teemu Karkkainen, and Jörg Ott. Opportunistic web access via wlan hotspots. In *Pervasive Computing and Communications (PerCom), 2010 IEEE International Conference on*, pages 20–30. IEEE, 2010.

[56] Anders Lindgren. Social networking in a disconnected network: fbdtn: facebook over dtn. In *Proceedings of the 6th ACM workshop on Challenged networks*, pages 69–70. ACM, 2011.

[57] L Peltola. Dtn-based blogging. *Helsinki University of Technology, Department of Communications and Networking*, 2007.

[58] HTML5 DeviceOrientation event specification. `http://www.w3.org/TR/orientation-event/`.

[59] Erik Nordström, Per Gunningberg, and Christian Rohner. Haggle: a data-centric network architecture for mobile devices. In *Proceedings of the 2009 MobiHoc S3 workshop on MobiHoc S3*, MobiHoc S3 '09, pages 37–40, New York, NY, USA, 2009. ACM.

[60] M. Skjegstad, F.T. Johnsen, T.H. Bloebaum, and T. Maseng. Mist: A reliable and delay-tolerant publish/subscribe solution for dynamic networks. In *New Technologies, Mobility and Security (NTMS), 2012 5th International Conference on*, pages 1–8, 2012.

[61] A. Petz and C. Julien. The madman middleware for delay-tolerant networks. In *Poster at HotMobile 2010 (Proceedings of the 11th workshop on Mobile computing systems and applications)*, 2010.

[62] Mauro Caporuscio, Pierre-Guillaume Raverdy, Hassine Moungla, and Valerie Issarny. ubisoap: A service oriented middleware for seamless networking. In *Proceedings of the 6th International Conference on Service-Oriented Computing*, ICSOC '08, pages 195–209, Berlin, Heidelberg, 2008. Springer-Verlag.

[63] Anna-Kaisa Pietiläinen, Earl Oliver, Jason LeBrun, George Varghese, and Christophe Diot. Mobiclique: middleware for mobile social networking. In *Proceedings of the 2nd ACM workshop on Online social networks*, WOSN '09, pages 49–54, New York, NY, USA, 2009. ACM.

[64] Frdric Guidec and Yves Maho. Opportunistic content-based dissemination in disconnected mobile ad hoc networks. In *International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, UBICOMM 2007, pages 49–54, 2007.

[65] H. Ntareme and S. Domancich. Security and performance aspects of bytewalla: A delay tolerant network on smartphones. In *Wireless and Mobile Computing, Networking and Communications (WiMob), 2011 IEEE 7th International Conference on*, pages 449–454, 2011.

[66] Hao Zhuang, Hervé Ntareme, Zhonghong Ou, and Björn Pehrson. A service adaptation middleware for delay tolerant networks based on http simple queue service. In *Proc. of the 6th Workshop on Networked Systems for Developing Regions (NSDR'12)*, 2012.

[67] Romain Rouvoy, Paolo Barone, Yun Ding, Frank Eliassen, Svein Hallsteinsen, Jorge Lorenzo, Alessandro Mamelli, and Ulrich Scholz. Software engineering for self-adaptive systems. chapter MUSIC: Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments, pages 164–182. Springer-Verlag, Berlin, Heidelberg, 2009.

[68] Darren Carlson, Bashar Altakrouri, and Andreas Schrader. Ambientweb: Bridging the web's cyber-physical gap. In *Internet of Things (IOT), 2012 3rd International Conference on the*, pages 1–8. IEEE, 2012.

[69] Cover is an android-only lockscreen that shows apps when you need them. `http://techcrunch.com/2013/10/24/cover-android/`.

[70] Google's Fused Location API, Google I/O 2013. `https://www.youtube.com/watch?v=URcVZybzMUI`.

[71] Apple M7. `http://en.wikipedia.org/wiki/Apple_M7`.

[72] Google adds low-power step counting to android 4.4. `http://mobihealthnews.com/26977/google-adds-low-power-step-counting-to-android-4-4/`.

[73] Weathersignal. `https://play.google.com/store/apps/details?id=com.opensignal.weathersignal`.

[74] Pressurenet. `https://play.google.com/store/apps/details?id=ca.cumulonimbus.barometernetwork`.

[75] Semefab Limited. MEMS Pressure Sensors: Technologies and Fabrication. 2011 Whitepaper.

[76] Stephen Ming-Chang Hou. *Design and fabrication of a MEMS-array pressure sensor system for passive underwater navigation inspired by the lateral line*. PhD thesis, Massachusetts Institute of Technology, 2012.

[77] *Hardware and software guidelines for use of the LPS331AP*, November 2012.

[78] Greg Milette and Adam Stroud. *Professional Android Sensor Programming*. Wiley, 2012.

[79] Kartik Sankaran, Minhui Zhu, Xiang Fa Guo, Akkihebbal L. Ananda, Mun Choon Chan, and Li-Shiuan Peh. Using mobile phone barometer for low-power transportation context detection. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, SenSys '14, pages 191–205, New York, NY, USA, 2014. ACM.

[80] Bo-Jhang Ho, Paul Martin, Prashanth Swaminathan, and Mani Srivastava. From pressure to path: Barometer-based vehicle tracking. In *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*, pages 65–74. ACM, 2015.

[81] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 26(1):43–49, 1978.

[82] Eamonn J Keogh and Michael J Pazzani. Scaling up dynamic time warping for datamining applications. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 285–289. ACM, 2000.

[83] Michel Marie Deza and Elena Deza. *Encyclopedia of distances*. Springer, 2009.

[84] AnYuan Guo and Hava Siegelmann. Time-warped longest common subsequence algorithm for music retrieval. 2004.

[85] Stan Salvador and Philip Chan. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5):561–580, 2007.

[86] Toni Giorgino. Computing and visualizing dynamic time warping alignments in r: the dtw package. *Journal of statistical Software*, 31(7):1–24, 2009.

[87] Samuli Hemminki, Petteri Nurmi, and Sasu Tarkoma. Accelerometer-based transportation mode detection on smartphones. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, SenSys '13, pages 13:1–13:14, New York, NY, USA, 2013. ACM.

[88] Leonard Kaufman and Peter Rousseeuw. *Clustering by means of medoids*. North-Holland, 1987.

[89] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[90] Paolo Tormene, Toni Giorgino, Silvana Quaglini, and Mario Stefanelli. Matching incomplete time series with dynamic time warping: an algorithm and an application to post-stroke rehabilitation. *Artificial intelligence in medicine*, 45(1):11–34, 2009.

[91] Yasushi Sakurai, Christos Faloutsos, and Masashi Yamamuro. Stream monitoring under the time warping distance. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 1046–1055. IEEE, 2007.

[92] Muhammad Tayyab Asif, Justin Dauwels, Chong Yang Goh, Ali Oran, Esmail Fathi, Muye Xu, Menoth Mohan Dhanya, Nikola Mitrovic, and Patrick Jaillet. Spatial and temporal patterns in large-scale traffic speed prediction. *Intelligent Transportation Systems, IEEE Transactions on*, 15(2):794–804, 2014.

[93] Wanli Min and Laura Wynter. Real-time road traffic prediction with spatio-temporal correlations. *Transportation Research Part C: Emerging Technologies*, 19(4):606–616, 2011.

[94] Wei Shen and L. Wynter. Real-time road traffic fusion and prediction with gps and fixed-sensor data. In *Information Fusion (FUSION), 2012 15th International Conference on*, pages 1468–1475, July 2012.

[95] Ella Bolshinsky and Roy Freidman. Traffic flow forecast survey. *Technion–Israel Institute of Technology.–2012.–Technical Report.–15 p*, 2012.

[96] OSGi Alliance. About the osgi service platform. 2007 Technical Whitepaper.

[97] Dan Feldman, Andrew Sugaya, Cynthia Sung, and Daniela Rus. idiary: From gps signals to a text-searchable diary. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, page 6. ACM, 2013.

[98] Fang Zhao, Ajinkya Ghorpade, Francisco Câmara Pereira, Christopher Zegras, and Moshe Ben-Akiva. Stop detection in smartphone based travel surveys. In *10th International Conference on Transport Survey Methods, Australia*, 2014.

[99] Tâm Huynh, Mario Fritz, and Bernt Schiele. Discovery of activity patterns using topic models. In *Proceedings of the 10th international conference on Ubiquitous computing*, pages 10–19. ACM, 2008.

[100] Donald J Patterson, Dieter Fox, Henry Kautz, and Matthai Philipose. Fine-grained activity recognition by aggregating abstract object usage. In *Wearable Computers, 2005. Proceedings. Ninth IEEE International Symposium on*, pages 44–51. IEEE, 2005.

[101] Liang Wang, Tao Gu, Hanhua Chen, Xianping Tao, and Jian Lu. Real-time activity recognition in wireless body sensor networks: from simple gestures to complex activities. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2010 IEEE 16th International Conference on*, pages 43–52. IEEE, 2010.

[102] Alex S Park and James R Glass. Unsupervised pattern discovery in speech. *Audio, Speech, and Language Processing, IEEE Transactions on*, 16(1):186–197, 2008.