Technical Report Documentation Page Form DOT-F-1700.7

1. Report No. CA(FY)(RPMD#)	2. Government Accession No.	3. Recipient's Catalogue No.
4. Title and Subtitle SMART TRUCK DRIVER ASSISTANT: A COST EFFECTIVE SOLUTION FOR REAL TIME MANAGEMENT OF CONTAINER DELIVERY TO TRUCKS		5. Report Date December 2015 6. Performing Organization code
7. Authors Burkhard Englert, Mehrdad Aliasgari, Shadnaz Asgari		8. Performing Organization Report No.
9. Performing Organization Name and Address California State University Long Beach Department of Computer Engineering Computer Science 1250 Bellflower Blvd. Long Beach, CA 90840		10. Work Unit Number 11. Contract or Grant No. 65A0533 TO 008
12. Sponsoring Agency Name and Address California Department of Transportation Sacramento, CA 95819		13. Type of Report and Period Covered Final. Jan. 1, 2015 to Dec. 31, 2015 14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract The twin ports of Los Angeles and Long Beach are two of the busiest ports in the country. The marine terminals at these ports are under tremendous pressure to enhance their performance levels. One major issue in the terminals' productivity and performance is the uncertainty of upcoming truck arrivals. The longer a truck waits the more it reduces companies' shipping capacity and contributes to air pollution.
In this work, we developed a mobile application through which truck movements near the ports can be tracked. This can provide terminals with a complete overview of upcoming traffic and enables them to sort the containers more efficiently. We use drivers' sensor-rich smartphones to this end, thus eliminating the additional cost of extra equipment. Our phone application provides the drivers with all information necessary to their operation and helps the terminals to foresee the upcoming traffic. Upon collecting the data, our system analyzes it and provides the stakeholders access to real time information about trucks at the ports.

17. Keywords Container terminals, Tracking of Truck movements, truck turn times	18. Distribution Statement No restrictions. This document is available to the public through the National Technical Information Service, Springfield, VA 22161
19. Security Classif. (of this report) Unclassified 20. Security Classif. (of this page) Unclassified	21. No. of Pages 24 22. Page

Disclaimer

Smart Truck Driver Assistant: A Cost Effective Solution for Real Time Management of Container Delivery to Trucks

Final Report

METRANS Project 14-13

February 2016

Burkhard Englert

Mehrdad Aliasgari

Shadnaz Asgari

Department of Computer Engineering Computer Science

California State University Long Beach

Long Beach, CA, 90840

METRANS Transportation Center USC CSULB

Contents

List of figures

Disclosure

Project was funded in entirety under this contract to California Department of Transportation.

1. Introduction

a. Overview of report contents

In section 1.b. we will first describe the problem of tracking trucks as they access the Ports of Los Angeles/Long Beach (POLA/LB). This is followed in 1.c. by a description of the objectives of this project and in 1.d. by the scope of our mobile application implementation. In 2.a. we discuss our methodology and in 2.b. we describe the scope of our solution, including implementation, in more detail. This is followed in 2.c. by our testing procedure and a discussion of problems encountered. We provide conclusions and recommendations in chapter 3. Chapter 4 discusses the deployment and implementation.

b. Problem Statement

The twin ports of Los Angeles and Long Beach are two of the busiest ports in the country, and the marine terminals at these ports are under tremendous pressure to enhance their level of performance [4]. One major measure of productivity and ultimately overall performance of a terminal is the truck visit (turn) time. The truck visit time consists of two main components: 1) queue time, time spent waiting in lines outside the gates; and 2) terminal time, the span of time from a truck's arrival at the terminal to its departure with its container. A 2011 study found that the distributions of truck visit time at the Twin Ports are highly skewed and dependent on the time of visit [5]. For example, for trucks arriving between 17:00 and 18:00, the median visit time in October 2011 was 90 minutes. However, 12% of truck visits took between 2-4 hours. Having long queues of idle trucks waiting to be served at the ports is inefficient in terms of cost, traffic congestion, security and air pollution [2]. The large variability that exists among truck visit times makes planning and scheduling a challenging task. As a result, the reduction of truck visit times is of significant value for the robust management of operations at the Twin Ports. As an approach to achieve this goal, most of the terminals in the Twin Ports have adopted some form of an appointment system, but recent studies have indicated low rates of utilization and an uncertain impact on reducing queuing at terminal gates due to the unpredictability of traffic in the Los Angeles metropolitan area [1]. Therefore, a robust solution to the truck visit time problem not only needs to be cost effective, but it also needs to be real-time to make the implementation of dynamic scheduling decisions feasible.

Aiming at such a goal, a few studies have tried to employ new technologies to implement an accurate measurement of truck visit time. In 2007, Lam et al. used time stamped digital images of trucks captured by a few cameras mounted at various critical points at the Twin Ports (e.g., the entry gate, the bobtail entry, the pedestal, and the exit gate) to obtain several time measures including truck terminal times [3]. However, in the aforementioned project, the matching of truck images was not an automated process, and required several persons to manually record the profile info of each truck entering at a terminal gate. This is a limiting factor in real-time applicability of such an approach. Most recently, Noronha et al. used GPS trackers on 250 trucks to study truck visit times at the Twin Ports [5]. But GPS measurements are subject to multiple errors and cannot provide the fine-grained data needed for such an application. For instance, as acknowledged by Noronha et al., GPS signals do not allow distinguishing between multiple lanes at a terminal gate. Hence, Noronha et al.'s findings were limited by the capability of the GPS equipment, only accounted for instrumented trucks and did not provide a conclusive analysis of turn times at the Twin Ports. These studies and their limitations necessitate development of a cost effective, unbiased (verified and accepted by all parties involved including ports, terminal operators and truck companies) and yet precise data collection method. Such collection of data will provide us with the ability to develop a model of port functionality and such a model in turn can be used for future decision-making.

c. Project Objectives

For this purpose, we developed in this project a truck driver mobile application (app) that utilizes the rich environment and features of smartphones. In other words, we use the widely available resources of smartphones (at no additional cost to drivers or terminals) to collect data from trucks and provide the latest information to stakeholders at the touch of a finger. Our methodology can hence be flexibly extended beyond any truck tracking or port monitoring system while providing both capabilities. Moreover the result of our study will lead to a set of reliable criteria that the port community can employ to evaluate the truck queuing and terminal visit times and hence assess the productivity at the Twin Ports.

d. Scope of project

In this project we developed a mobile application that, when employed by truck drivers, will allow all stakeholders at the ports to obtain precise information about the physical location of a truck in real time. This information can then be used to determine truck turn times and to provide stakeholders with precise information about the current condition of the truck segment of the supply chain at or near the ports. We believe that with this information stakeholders will be able to improve efficiency at the ports and increase throughput. Terminals, for example, could obtain real time information about truck locations and assignments, possibly allowing them to sort and prepare containers requested for pick up. This is in turn could significantly reduce truck turn times inside a terminal. Trucking companies on the other hand could use our tool to obtain precise and accurate real-time information about conditions at the ports, allowing them to optimize their operations.
Our application, once installed and authenticated on a driver's smartphone, does not require the driver to enter any information on his phone. The driver can simply ignore our application and the application will automatically collect information about the truck's location and movements. This information will be accessible to the trucking company that provided the current assignment to this truck driver and it can also be made available to other stakeholders such as terminal operators. Terminals, if permitted by the trucking

company, would be able to see which container a truck is dropping off or which container it wants to pick up.
Our system uses role-based access control to ensure that each participant or stakeholder can only access the appropriate or permitted information. If, for example, a truck is on its way to pick up a container from one terminal, only that particular terminal will be able to see this information. Our system also implements the principle of least privilege at all times. Namely all stakeholders have access only to the minimum amount of information they need to complete their current task.

2. Body of Report / Methodology / Technical Discussion

a. Procedures

According to a recent survey, more than 50% of all the individuals in the US own a smartphone [6]. The broad accessibility of smartphones contributes to the practicality and feasibility of our proposed solution: a mobile application (app) that utilizes truck driver-owned smartphones and its sensor-rich environment (network antennas and inertial sensors) in addition to its GPS to obtain a more accurate location readings. Our app has been developed on the most popular mobile platforms (Android and iOS). The GPS location readings along with the smartphone sensors are reliable data sources to determine the truck turn times.

Note that the smartphone's navigation system is far more accurate than a conventional stand-alone GPS navigation system: A GPS device needs to find three or four satellites to locate itself on a map. This often takes a long time and may include errors due to a low power satellite signal or reflection by nearby obstacles. Also inaccurate time stamping from a GPS receiver's clock and atmospheric disturbances can produce errors. Furthermore, the GPS device needs to download information about satellites from above the earth. Given the aforementioned issues, the sole reliance on a GPS device for the purpose of localization does not seem to be an optimal approach. Smartphones, on the other hand, additionally benefit from a technology called Cell ID. Using measurements from different towers of a cellular network, carriers know exactly which sector of a network base station that the smartphone is within and its distances from the nearby antennas. This additional location information is sent back to the smartphone and can be used by any application for a more accurate localization. In other words, applications can request GPS and network-based location readings on smartphones within any time interval (a variable that can be set to as short as milliseconds). Hence, a smartphone's navigation system is far more accurate and easier to use than a conventional stand-alone GPS navigation system in cars.

In addition, smartphones come equipped with inertial sensors. All smartphones have three basic sensors: a compass to detect the direction of a phone, an accelerometer to detect any acceleration (positive when speeding up and negative when decreasing speed through braking) and finally a gyroscope to sense any turning motion. Combined with the GPS techniques, we can not only locate a phone continuously but also detect any change in speed or direction of the phone. This allows us to determine if a truck driver makes any turn to change a lane or more importantly which trucks are stacked up in a queue together within one geographical location. Any acceleration change for a truck at the head of a queue results in a chain of similar changes for all trucks in that queue. Therefore, we can determine the number of trucks in a queue and the exact position of each. When trucks are too close to each other, the GPS systems may wrongly locate them at one single point. But then the accelerometer data that we collect from the smartphone can facilitate distinguishing of the trucks in a queue. In addition, gyroscope readings allow us to estimate which lane the truck is entering or whether a truck is switching lanes. In the future, we could also ask the driver to input the queue lane number for better precision while they are waiting in the queue.

Given that truck drivers are equipped with smartphones, we achieve a high level of precision in data collection at minimum equipment cost because all necessary equipment already exists on smartphones. Our app sends the collected data to a central server using cellular networks. The server uses our algorithms to analyze and process the data and given enough information (a sufficiently large number of participating smartphones) our algorithms can derive a complete model in real time. With enough input data our system can use advanced data mining techniques to detect and eliminate any erroneous information. It produces a complete picture of trucks whose drivers use our App, their precise location and the location of individual trucks waiting in queues or driving on the roads. Our app allows us to indirectly and accurately compute the turn times for trucks whose drivers use our app and also approximate turn times for other trucks that access a terminal together with app-equipped trucks.

Our app requires a one-time registration by the driver. Registrations are made possible through an authorization system that is controlled by trucking companies. For our purpose, we also consider owner operators as trucking companies. At this stage, we collect drivers' registration data from trucking companies and allow drivers' apps to be registered with our system. Each smartphone sends data collected from available sensors of a driver's smartphone to our servers in real time via the cellular network with no identifiable tags. This guarantees that user privacy is respected since only anonymous data is collected from drivers. Our system starts incorporating collected data from trucks into our port analysis once a truck enters an area within a fixed radius of the twin ports. Prior to installing our app on a driver's smartphone, our system asks drivers to read the terms and conditions of this research product. The terms describe what our app aims to do and specify how the app accesses phone sensors and the type of data it collects. We adhere to a policy of complete user privacy. We share no personal data with a third party nor are we using any data in any undisclosed manner. By clicking on agree at installation time, drivers accept the mentioned terms and consent to participate in this research.

In addition to data collection, we are able to create additional capabilities for our app such as the ability to provide feedback and suggestions to the drivers, terminals and ports. It could, for example, notify the terminal of upcoming traffic and estimated arrival times so they could schedule their gates and personnel more efficiently based on the expected traffic. This would lead to better management of terminal time and the ports in general. Our system can also provide estimated waiting times to drivers while approaching the terminals. All of this autonomous data collection and assistance is possible due to sensor-rich smartphones and our efficient and powerful algorithms that can process the data. The driver assistance feature incentivizes drivers to install and use our app.

We have simulated the app's capabilities and tested it using our own cars. Once we have identified a test partner and our app has been deployed, our system will provide an initial analysis of the status of terminals and trucks. However, the developed app can easily be improved in the future to include additional functionalities: Upon availability of further funding, we plan on adding more features. One example is the capability to reduce "trouble tickets". Trouble tickets are issued to trucks by port authorities when there are transaction problems (exceptions from normal processes). This is often due to inaccurate or incomplete information regarding container delivery/pick-up or import/export paperwork failures. These tickets account for 5% of all transactions on average and add about an hour to the turn time. But most trouble tickets can be prevented prior to trucks arriving at terminal gates by providing better communications between truck companies, terminals and ports.

Another extension of our app will be the ability to collect cargo information from truck companies, drivers and terminals. This cargo data could be obtained from barcode scanning using phone cameras and Near Field Communication (NFC) tag reading. Our system then can process data and check the status of the cargo (arrived, on hold, etc.), its estimated delivery/pick-up time and proper paperwork required. It will notify the driver/truck companies of any actions that they need to take prior to entering the long queues. This could reduce the number of trouble tickets and subsequently the turn time.

Another interesting future feature will be a chatting capability that allows drivers to privately chat with each other and their employers to share information about their whereabouts and the status of their cargo and vehicles. In addition, we plan to integrate a terminal appointment system into our application that would allow the drivers and truck companies to make appointments for terminal visits and receive notifications almost instantly on their phones regarding their appointments. Once the application on phones starts exchanging data via Wi-Fi signals, the central servers can derive an even more fine-grained picture of the relative locations of trucks in relation to each other. Additional features could also include an easy turn-by-turn voice-activated navigation system inside terminal facilities for less experienced truck drivers. We could also add alert notifications and information about truck stops and traffic signal alerts to the system. For all these possible future extensions of the proposed application, we intend to aim for larger grants from other sources.

The system is a digital hub for drivers in our freight transportation network. We believe that it will be an extremely valuable asset to the transportation community. Similar approaches in various transportation settings (e.g., buses and public transportation) could also be developed in the future. While our study will mainly focus on the determination and reduction of truck turn times for the Los Angeles/Long Beach Ports, the proposed methodology is applicable to a large number of container ports in the U.S. and abroad.

We note that the availability of smartphones to truck drivers is essential to our system. While not all drivers currently may own smartphones, ComScore recently announced that 163.2 million people in the U.S. at the end of February 2014 owned smartphones [6]. Assuming truck drivers are a good sample of the overall US population, it is reasonable to infer that a large and over time increasing percentage of truck drivers owns or soon will own smartphones. By providing real-time information to drivers, our system can also be used to further incentivize smartphone purchases among drivers. We note that our system will function and return usable results even if, for example, not all drivers in a queue are equipped with a smartphone.

b. Scope Description / Development and Implementation

We have introduced the problem of addressing traffic congestion at port terminals and the need to reduce truck turn times. In the following subsections we will describe our approach, architectural design, development process and the tools and technologies that we used. We also include a higher-level overview of our implementation, which we will discuss with some use cases that determine how to build the server and client logic, the programming languages we used, and how we built the requirements in a modular and incremental fashion. Additionally we include an overview of the server side and client side implementation, which makes use of the MVC (Model View Controller) design pattern. Finally, we will also address testing and validation of the program logic to ensure the delivery or consumption of data.

Analysis

In order to come up with a list of requirements for this project, we first broke down the objectives into use cases. From these use cases we created a set of server-side and client-side requirements. This established the system architecture. One of the use cases requires the user to register their phone with the system. We determined that it would be best if only selected users could register with this system. Hence we designed a mechanism that allowed already authorized users to first pre-register a new user. Since we assume that each user has a unique phone number, the phone number can be used to register the device to the system.

We then decided that a web-based portal would be the best way to pre-register users. The web portal has connections to the database where data is stored. Further inquiry into the ability to register the device then revealed that the system required a RESTful service, ”URL-accessible resource-based service” [7], on the server side component of the overall architecture. With this endpoint the client device would then be able to query the server for a registered user.

From a higher-level overview perspective, it became clear that we need two systems, the server side and the client side. After analyzing use cases, we created a list of contracts to preform a set of functions. These contracts are exposed with the help of secured RESTful endpoints. We then tied the endpoints to private server-side logic, which performs functions such as querying for a pre-registered phone number. After these queries, the endpoint has the ability to deliver a response back to the client requesting the inquiry.

On the other hand, the client-side system needs to have the ability to communicate with the server through the exposed REST endpoints. For user registration to be successful, the client side needs to have the ability to accept the phone number as input from the user and then, upon hitting the register button, use logic written to create a secured connection to the REST endpoint to register the phone number. In our system phone registration happens by fulfilling the endpoint's parameter requirements, and receiving a response from the server, which enables the client device to logically determine the next steps: In our case, either let the user proceed to the next view after a successful registration or deny access to the user.

Design

Our overall system architecture requires a server-side as well as client-side implementation. The server-side architecture makes use of MySQL as the database and an Apache server to handle business logic and deliver HTML to the user. The server's main purpose is to serve RESTful endpoints from which clients would be able to consume data by performing a GET or POST operation. Figure 1 below shows a high-level overview of the server architecture.

The client-side architecture, based on requirements, needs to have the ability to register a user, get the location of the user, and post the location of the user back to the server. The client application requires a local database known as Core Data, which is specific to the iOS operating system and the use of geolocation services provided by iOS. The client also needs the ability to create secure connections to the server's endpoints. Figure 2 below shows a representation of this architecture.

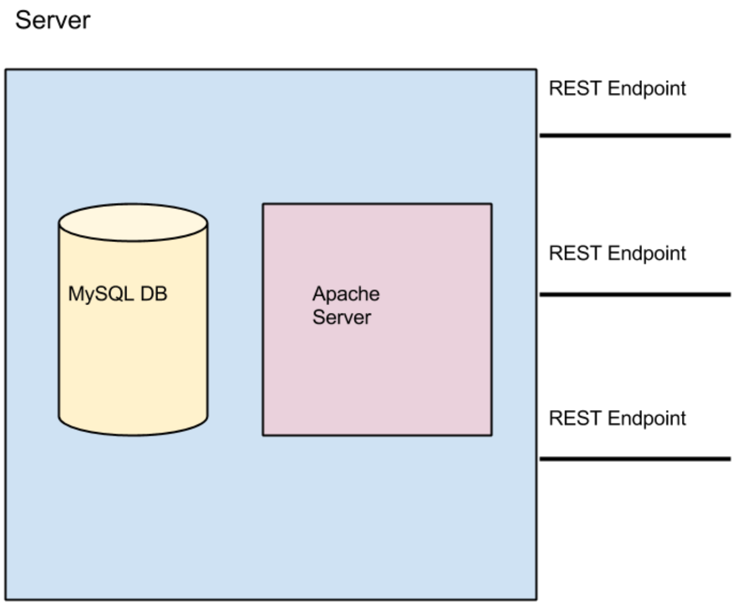


Figure 1. Server architecture with REST endpoints.

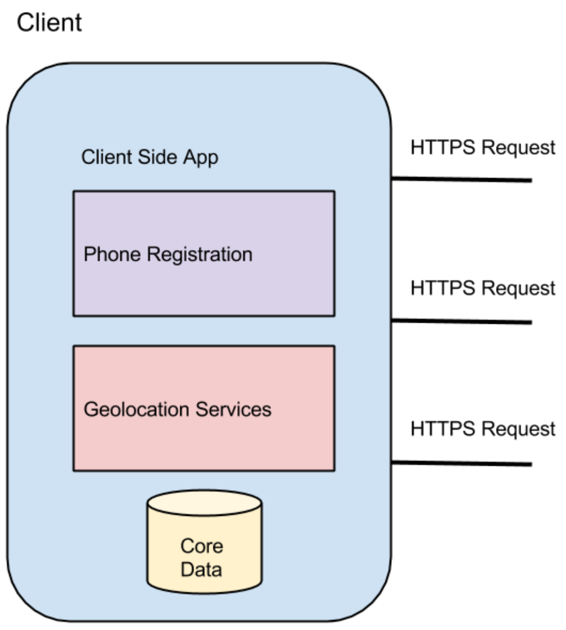


Figure 2. Client architecture with secured request connections for server endpoints.

Tools and technologies used

We used two different sets of technologies and tools to develop the overall system architecture. Server-side and client-side implementation both require a different set of integrated development environments (IDEs), languages, frameworks, and hardware. We used version control on both architectures but separately and each with their own repositories.

To implement the server-side architecture, we also needed an online hosting service provider. We selected DigitalOcean. DigitalOcean provides a cloud infrastructure solution where the MySQL database and Apache server can live. At the lowest tier provided by DigitalOcean, a client receives 512Mb of memory, a single core processor, 20Gb of solid state drive (SSD) disk space and 1Tb of data transfer bandwidth [7].
We then selected a framework and a set of languages for the server-side implementation of the system. Laravel, a PHP framework with the ability to make use of the MVC (Model View Controller) paradigm, allows one to create web applications at a fast pace [8]. Laravel has a large active community and the documentation is extensive and thorough. As mentioned earlier, Laravel requires the use of PHP as the scripting language of choice. Communication between the web application and the database can be done through MySQL queries as well as through the Eloquent object relational model (ORM) [9]. This would all constitute a LAMP stack (Linux, Apache, MySQL and PHP) [10].
To access files on the DigitalOcean server and to update code, we selected Aptana Studio 3 as the IDE of choice. Aptana is open source and free to use [11].

On the client side of the system architecture, we developed in parallel an Android-based and an Apple iOS-based application. We will describe the Apple-based application. We used the Xcode IDE to develop iOS applications. Xcode requires the latest version of the Mac operating system and runs on Apple hardware [12]. We wrote the client application in objective-c. To run programs written in Xcode we used an iPhone with the latest version of iOS. Another option of running applications is on the simulator provided by Xcode [13].

We also used the git version-control system. Git is open source, light weight [14], and is important in making sure that different versions of code through a development period are saved. This gives the developer the ability to revert back to an older version if necessary. Since there are two entities to the system architecture, we used separate git repositories. We also used a remote version of the git repository to save code. Once we made local commits to the local git repository, we were able to push code to the remote repository. We chose Bitbucket as the remote git repository [15]. Bitbucket allows for unlimited free private repositories as long as the development team does not exceed a fixed upper bound [16].

Server implementation

Before we were able to write any server-side logic, we had to set up the cloud infrastructure with a MySQL database along with an Apache server. We acquired a secure sockets layer (SSL) certificate and put it in place for HTTPS secure connections. This ensures state-of-the-art eavesdropping protection for all data transmitted over the public network by our applications. We then installed the Laravel framework and initialized Git and set up at the root level of the Laravel folder for version control. With our LAMP stack now in place, our scripting languages of choice were PHP, HTML, CSS along with the MySQL query language for database queries. The Laravel 4 framework has features that are essential to creating a backend service. Laravel 4 has the ability to create users, user groups, prevent brute force logins, and comes with a set of prewritten pages, such as the 404 forbidden access page and the 500 internal server error page [17]. We then built the server-side logic in a modular fashion by separating out the different required REST endpoints and building upon each endpoint one at a time. Each endpoint performs a single function for which several private methods take place to deliver the function. For example, an objective of one of the endpoints is to deliver a set of terminals, which is achieved by building a HTTP GET terminals endpoint. Each terminal object provides a name, latitude, and longitude. Using Laravel's Eloquent ORM, a model with a relationship to the database table holds the list of terminals created. We then wrote business logic into the controller file as a function to connect to the terminals model and to retrieve a list of all terminals. Since Laravel requires the creation of a route to access the controller [18], we needed one more action to be implemented by making an update to the route file. We created a route with the ability to do a HTTP GET. This route points to the public function of the terminals controller with the ability to retrieve a list of terminals. The endpoint mentioned only has the ability to retrieve a set of terminals and does not accept data.

We created endpoints to accept data through a HTTP POST, for example one endpoint that allows location data to be saved on the server. We then created a location model file with access to the database's locations table. Next we created a controller file with a function using the Eloquent ORM to do inserts into the locations table through the model. Using the Eloquent ORM is safe as it prevents SQL injection attacks and sanitizes user input data [19]. Finally we updated the route file with a route that allows a HTTP POST and points to the function in the controller, which performs inserts of locations into the database. The remaining list of endpoints include the ability to register a user, the ability to get a single trip, the ability to post the timestamp of when a user has reached the in-queue of a terminal, the in-gate of a terminal, the out-queue of a terminal, and the out-gate of the terminal.

Client implementation

We implemented client-side applications for both Google's Android OS and Apple's iOS. These are currently the two dominating mobile operating systems. We will limit our discussion to iOS but a similar process applied to the Android version of the application. We used Xcode as the IDE and objective-c as implementation language.

We wrote all our code in a modular fashion. We built all modules based on functionalities that would cater to the servers REST endpoints described above. The functions have the ability to either consume data coming from the endpoints or to post data to the server's endpoints. The registration and login process on the client side requires the use of the phone number and pin code to register the device on the server. A driver will obtain his/her pin code from his/her employer (trucking company). This ensures that even if users who are not truck drivers download the app they will not be able to use it and our system will not consider this "false" data. Once the server accepts the phone number and pin code it delivers a private key to the device with which the device can then make authorized queries to the server through the public facing endpoints. This requires a model represented through a database table, in this case the phone's core data database, to save the phone number and the private key once it has been received in a success response. We also need a controller to make a connection between the phone's database and the user inputs through a function, which creates a secured connection to the server's registration endpoint. This function performs an asynchronous HTTP POST and listens to the results within a predetermined timeframe. Lastly we need a view to retrieve user inputs and to display them to the user.

Communication with the endpoint has a set of responses. If an error occurs, the user is notified of an error and asked to try again. If a failure of registration occurs, such as an invalid phone number then the user receives a failure notification and is asked to try again. Upon successful registration the user is sent to the next view and allowed to use its functions.

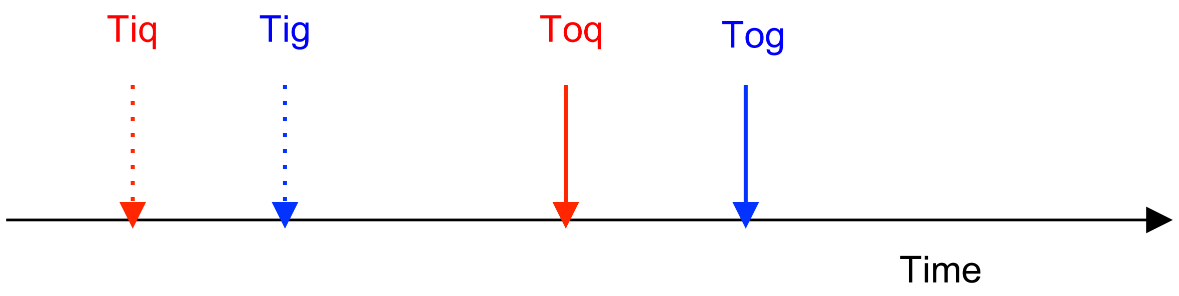
We also developed a function that implements the ability for the client application to send to the server the time and location of when and where the phone device reached a gate at one of the terminals. This requires the client to have the ability to recognize when it enters a registered geofenced area and send out a timestamp to the server of when that crossing of the geofence occurred. These processes are done asynchronously in the background. The system registers geofenced locations of terminals to the application after it receives a list of the terminals from the endpoint that delivers that data. Once this is done, the application listens for a geofence crossing while the application is in the background or foreground. As a geofence is crossed the application then sends an HTTP POST to the server and thence completes one of its functions.

The remaining list of functions of the client application include the ability to perform a HTTP request to get a trip for the day, the ability to post the user's current location to the server, and the ability to post the timestamp for when the user enters the in-queue, or out-queue of a terminal.

Testing and validation

While developing our app and server application we also built test cases in parallel to ensure the results delivered by the application met the requirements. We used this approach on both the client side as well as the server side. To ensure that the client receives accurate information, we separately tested every REST endpoint on the server by creating a separate test controller and view. An example of how an end point that functions correctly is the phone registration endpoint; we ensured that new phone numbers are validated and in response receive a private key. The phone device then consumes and uses the private key to authenticate further transactions with the server. With the endpoint's test page the user is able to enter a pre-registered phone number and a pin code. The response to this is a generated JavaScript Object Notation (JSON) structure with either success or error codes. JSON is an easily readable object by both machine parsers and humans [20]. A success code is then delivered with the private key needed while an error code would have several cases explaining which error occurred. Examples of such error codes include an invalid phone number or pin code.

In our study, we developed our system for specific terminals. We called such specific terminals target locations. In addition, we needed a central service to store and analyze all the collected data. This central service is called the server. The server is basically a computer program that accepts data from specific devices through secure Internet connections (using TLS v 1.1 protocol). We also developed an iPhone as well as an Android mobile application for truck drivers' smartphones. The application requires permission to access a user's location data when the driver is on duty. Once a driver enters a fenced area around target locations the mobile application records the time of entry. This is carried out using geofencing technology that is available on both iPhone and android devices. In geofencing, an imaginary geographic circle is assumed around a central location. In our case, the mobile application could receive information about the center of each geofence along with its radius. We note that each terminal has two gates, one to enter the terminal (entry gate) and one to exit (exit gate). We created two geofenced areas for entry gate with 300 ft. and 1 mi radiuses. We also created two geofences for exit gates with 300 ft. and 0.5 mi radiuses. As a result, there are four geofenced areas for each terminal (one small and one large geofence per gate). For the geofenced areas around an entry gate, we are concerned about when a truck enters each of these areas from the outside of a terminal. As for the exit gate, we are concerned when the truck enters those small and large geofenced areas from inside of the terminal. We represent the times at which a truck enters the large geofence of the entry gate and the small geofence of the entry gate by **Tiq** and **Tig,** respectively. Similarly, we represent the times at which a truck enters the large geofence of an exit gate and the small geofence of an exit gate by **Toq** and **Tog.** Note that **Toq** and **Tog** are only recorded after **Tig** has been recorded (a truck first enters a terminal and then it leaves it).



Entrance queue to exit gate	Tog - Tiq
Entrance queue to exit queue	Toq - Tiq
Entrance gate to exit gate	Tog - Tig
Entrance gate to exit queue	Toq - Tig

The distance between small and large geofenced areas of any gate is about 1 mi. If there is no traffic and assuming a speed limit of 20mph then Tig - Tiq will be approximately equal to 3 minutes. If this is the case, we set **Tiq**= **Tig** otherwise **Tig - Tiq** is equal to the queue time to enter the terminal. Similarly, if **Tog - Toq** = 1.5 minutes (approximately the time to travel 0.5 mil within a terminal with no traffic) then we set **Tog**= **Toq** otherwise **Tog- Toq** represents the queue time to exit the terminal.

The mobile application starts sending the location of the driver to the server periodically after the driver has entered the large geofenced area of the entrance gate. This is to record the movement of a truck within and around a terminal. As the first phase of testing, we tested our system within a fictional terminal. CSULB campus was chosen as our fictional terminal with entrance gate and exit gate located at opposite sides of the campus. To simulate the movement of a truck, a student walked to the entrance gate, campus quads and then left the campus through the exit gate. We observed that **Tiq**, **Tig**, **Toq** and **Tog** were captured with a precision of one minute. Throughout the experiments, we didn't notice any noticeable increase in smartphone battery consumption. In addition, the bandwidth consumption to connect and send data to the server was negligible.

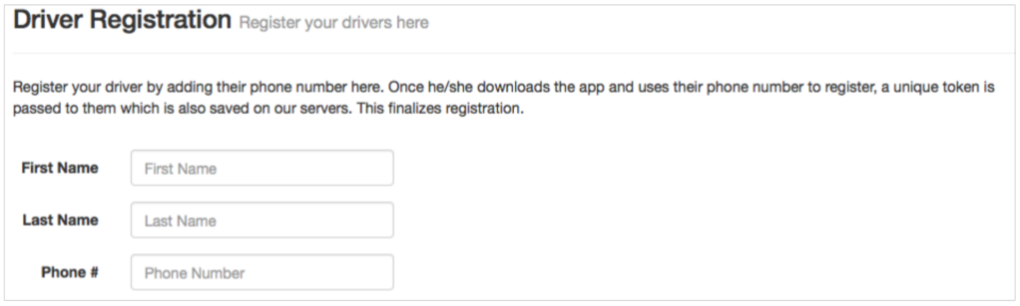
We conclude that motor carriers with minimal cost and maintenance can use our newly developed application. The collected data can be shared with all stakeholders. This data can be reported on an hourly, weekly, monthly and yearly basis as required. It can also be reported exclusively to each motor carrier, terminal or port. Further statistical analysis can be conducted, as needed.

c. Problems and testing

We conducted several experiments to check the results produced by the system. The goal was for the application to be able to gather timestamp data when a phone has entered the in-queue, in-gate and then, after going through the terminal, hitting the out-queue and eventually the out-gate.

The experiment started off by pre-registering a phone number on the application. The user then entered a phone number and received a pin code on the registration device. Figure 3 shows the driver registration view with the phone number pre-registration process.

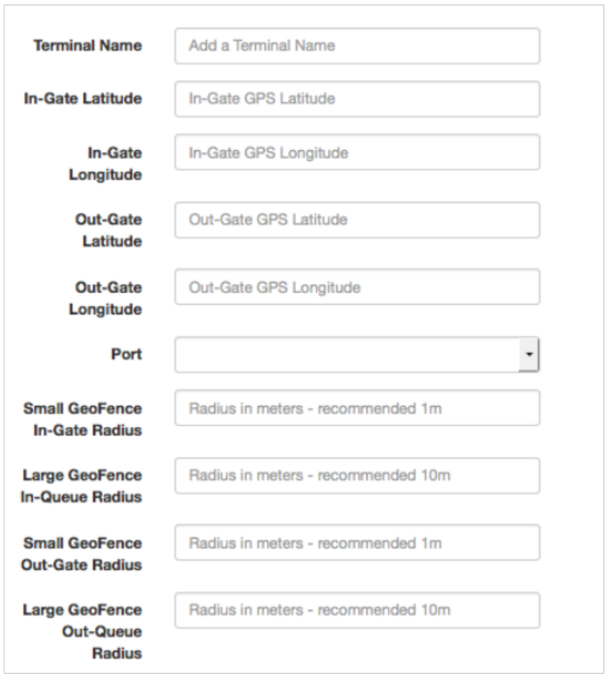
We created a terminal with latitude and longitude points for the in and out gates. We then introduced separate radii to create geofences for both the in-gate and out-gate. We also introduced radii to define the geofences of the in-queue and out-queue. Figure 4 shows the terminal create view.


Driver Registration Register your drivers here

Register your driver by adding their phone number here. Once he/she downloads the app and uses their phone number to register, a unique token is passed to them which is also saved on our servers. This finalizes registration.

First Name: First Name
Last Name: Last Name
Phone #: Phone Number

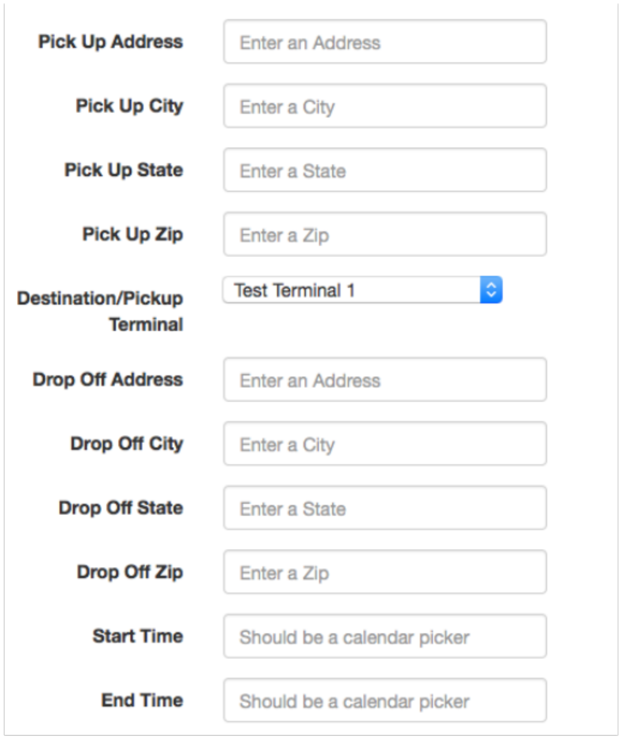
Figure 3. The driver registration view.



Terminal Name	Add a Terminal Name
In-Gate Latitude	In-Gate GPS Latitude
In-Gate Longitude	In-Gate GPS Longitude
Out-Gate Latitude	Out-Gate GPS Latitude
Out-Gate Longitude	Out-Gate GPS Longitude
Port	
Small GeoFence In-Gate Radius	Radius in meters - recommended 1m
Large GeoFence In-Queue Radius	Radius in meters - recommended 10m
Small GeoFence Out-Gate Radius	Radius in meters - recommended 1m
Large GeoFence Out-Queue Radius	Radius in meters - recommended 10m

Figure 4. Adding a terminal view.

After completing a terminal registration, we created a trip with the designated start times and end times. The user then chooses the created terminal and the new user is selected as the driver for this trip. Figure 5 shows the trip create view.



Pick Up Address	Enter an Address
Pick Up City	Enter a City
Pick Up State	Enter a State
Pick Up Zip	Enter a Zip
Destination/Pickup Terminal	Test Terminal 1
Drop Off Address	Enter an Address
Drop Off City	Enter a City
Drop Off State	Enter a State
Drop Off Zip	Enter a Zip
Start Time	Should be a calendar picker
End Time	Should be a calendar picker

Figure 5. Adding a trip view.

On the client (driver) side, a driver downloads the iOS (or Android) version of our app onto a smartphone before starting the application. The application then prompts the user with a registration view to enter the device's phone number and pin code. Figure 6 below shows the client-side registration view.

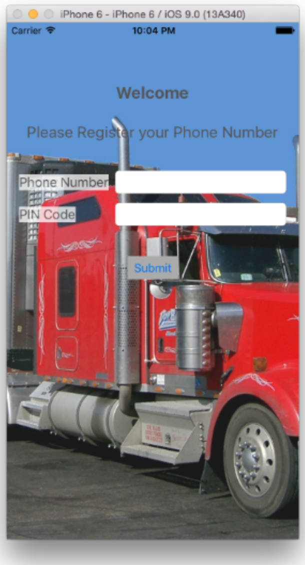
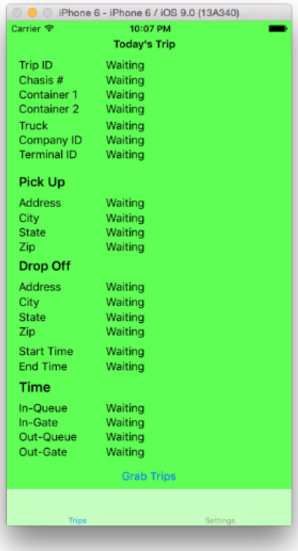

iPhone 6 - iPhone 6 / iOS 9.0 (13A340)
Carrier
10:04 PM
Welcome
Please Register your Phone Number
Phone Number
PIN Code
Submit

Figure 6. Client phone registration view.

Upon hitting submit, a successful registration redirects the user to the trip view where the user is allowed to get a single trip for the day. After getting a single trip, a view is populated with the data while in the background, a set of processes has taken place to register the user and to listen to the geofences, and the user's location. Figure 7 below shows the client- side trip view before querying the server for a trip.



Today's Trip

Trip ID	Waiting
Chasis #	Waiting
Container 1	Waiting
Container 2	Waiting
Truck	Waiting
Company ID	Waiting
Terminal ID	Waiting

Pick Up

Address	Waiting
City	Waiting
State	Waiting
Zip	Waiting

Drop Off

Address	Waiting
City	Waiting
State	Waiting
Zip	Waiting
Start Time	Waiting
End Time	Waiting

Time

In-Queue	Waiting
In-Gate	Waiting
Out-Queue	Waiting
Out-Gate	Waiting

Figure 7. Client trip view.

As the user drives around and passes through the in-queue and eventually the in-gate, the application delivers a timestamp to the server. Here are the results for one of our experiments.

In-queue time: 2015-09-27 16:59:15. In-gate time: 2015-09-27 16:59:15.

Upon passing first through the out-queue, then the out-gate, the observed results are as follows.

Out-queue time: 2015-09-27 17:45:03. Out-gate time: 2015-09-27 17:45:03.

At the speed of the vehicle passing through the queues and gates, no delay is noticeable between the queue and the gate. However, there is a noticeable delay between the in- and out-gates and queues. In another experiment we conducted the in- and out-gates were close to each other. The following are the results for that experiment.

In-queue time: 2015-09-13 23:40:52. In-gate time: 2015-09-13 23:40:52.
Out-queue time: 2015-09-13 23:41:03. Out-gate time: 2015-09-13 23:41:03.

There is a difference between in- and out-gate times here as well but since the queue radii were small there was no difference in time between the queues and gates. A queue with a larger radius would be required to see a significant difference between queue and gate entry times. The application ran on both the client side as well as the server side with the client sending out data and the server side receiving the data, while the client application is running in the background.

3. Conclusions and Recommendations

We developed a mobile application to track the truck movements near the POLA/LB. This mobile application can provide terminals with a complete overview of upcoming traffic and enables them to sort the containers more efficiently. To eliminate any additional cost of extra equipment, our proposed solution leverages on the drivers' sensor-rich smartphones and provides the drivers with all the information necessary for their operation. On the other hand, our mobile application helps the terminals to foresee the upcoming traffic and plan accordingly. Upon collecting the data, our developed system analyzes the data and provides the stakeholders access to real-time information about trucks at the ports. In addition, terminals can utilize the upcoming truck traffic data to optimize storage and re-handling of containers. The fewer container shuffling, the lower the risk of accidents and the faster cargo can be loaded /unloaded. An algorithm could furthermore use the truck traffic data to optimize movements of containers to reduce the truck turnaround time. Thus, such an approach can potentially increase the performance of terminals and ports. This problem is left as a future work.

4. Deployment and Implementation

We have already completed several tests at the mock terminals successfully. The test results show that our app is fully functional and provides the expected outcomes. As of the time of this writing, we are in discussion with several stakeholders to conduct a field test of our application. So far, we have received very positive responses from trucking companies and a few ports. They all believe that our approach is very promising and should be pursued. We expect that within the next few months (early 2016) we will be able to test our application with a trucking company at the ports. Therefore, more results are expected to follow in the near future.

5. References

[1] Giuliano G., O'Brien T. (2007) Reducing Port Related Truck Emissions: The Terminal Gate Appointment Systems at the Ports of Los Angeles and Long Beach. Transportation Research Part D, 12, 460-473.

[2] Giuliano G., Hayden S., Dell 'Aquila P., and O'Brien T. (2008) Evaluation of the Terminal Gate Appointment System at the Los Angeles/Long Beach Ports. METRANS Final Report Project 4-06.

[3] Lam S., Park, J. and Pruitt C. (2007) An Accurate Monitoring of Truck Waiting and Flow Times at a Terminal in the Los Angeles/Long Beach Ports. METRANS AR 05-01.

[4] Lam S., Englert B. and Chassiakos A. (2008) On Sequencing of Container Deliveries to Over the-Road Trucks from Yard Stacks METRANS Final Report Project 07-12.

[5] Noronha V. (2011) Taking the Pulse of the Ports: Duration of Truck visits to Marine Terminals, Final Report, Digital Geographic Research Corporation, www.metris.us/services/turntime/ Retrieved February 16, 2016.

[6] ComScore reports February2014 US Smartphone Subscriber Market Share, https://www.comscore.com/Insights/Press_Releases/2014/4/comScore_Reports_February_2014_US_Smartphone_Subscriber_Market_Share Retrieved February 16, 2016.

[7] M. Bleigh, "Rest isn't what you think it is, and that's ok."https://www.mobomo.com/2010/04/rest-isnt-what-you-think-it-is/, April 2010. Retrieved February 16, 2016.

[8] "Simple pricing. digitalocean." https://www.digitalocean.com/pricing/. Retrieved February 16,

[9] "Laravel." http://laravel.com/. Retrieved February 16, 2016.

[8] "Eloquent orm. laravel." http://laravel.com/docs/5.1/eloquent . Retrieved February 16, 2016.

[10] "Lamp stack." https://www.turnkeylinux.org/lampstack. Retrieved February 16, 2016.

[11] "Apatana studio 3." http://www.aptana.com/. Retrieved February 16, 2016.

[12] "Xcode ide." https://developer.apple.com/xcode/ide /. Retrieved February 16, 2016.

[13] “ios simulator. xcode features.” https://developer.apple.com/xcode/
features/. Retrieved February 16, 2016.

[14] “Git version control system.” https://git-scm.com/. Retrieved February 16, 2016.

[16] “Bitbucket.” https://bitbucket.org/. Retrieved February 16, 2016.

[17] “Pricing. bitbucket.” https://bitbucket.org/product/pricing. Retrieved February 16, 2016.
[18] Snipe, “Features. laravel 4 bootstrap 3 starter site.”
https://github.com/andrewelkins/Laravel-4-Bootstrap-Starter-Site. Retrieved February 16, 2016.

[19] “Http routing. laravel.” http://laravel.com/docs/5.1/routing. Retrieved February 16, 2016.

[20] S. Mathew, “Laravel’s query builder and eloquent orm.”
http://cubettech.com/blog/laravels-query-builder-and-eloquent-orm/, January 2015.
Retrieved February 16, 2016.

[21] “Json.” http://www.json.org/. Retrieved February 16, 2016.