Blue Team - Traffic Wizard Lab 2 Section 3

**This document is formatted to support being inserted into other documents. Page breaks present in this document are not aligned to class guidelines, since they would change after being put into a group member's Lab 2*

1. Introduction

2. General Description

3. Specific Requirements

The specific requirements for Traffic Wizard involve functional, performance, and non-functional requirements for all aspects of the Traffic Wizard system. Assumptions and constraints are also outlined in this section to define the limitations of the prototype.

3.1. Functional Requirements

The Traffic Wizard product involves multiple complex systems that have functional requirements in order to achieve product features. On the Traffic Wizard server side, requirements must be defined for managing connections, databases, and algorithms. On the client side, requirements must be defined for operation of the smartphone app. The Simulation Console, which is a component unique to the Traffic Wizard prototype only, is a demonstration platform for proving that Traffic Wizard systems work through a series of simulations.

3.1.1. Databases (Kennedy)

The Virtual Checkpoint Database is referenced when an application makes a request for checkpoint data. The speed limit data for the Virtual Checkpoint will be mined from a third-party Speed Limit Database. All information pertaining to an end-user's routes is accessed from the Driver Profile Database.

3.1.1.1. Driver Profile Database (Kennedy)

The Driver Profile Database will store customer information. This information will include authentication credentials, payment information, and customer information. The structure of the database is shown in Figure X. All constraints and structural requirements are listed in Table X. The database must satisfy the constraints and structure as described in the aforementioned figure and table.

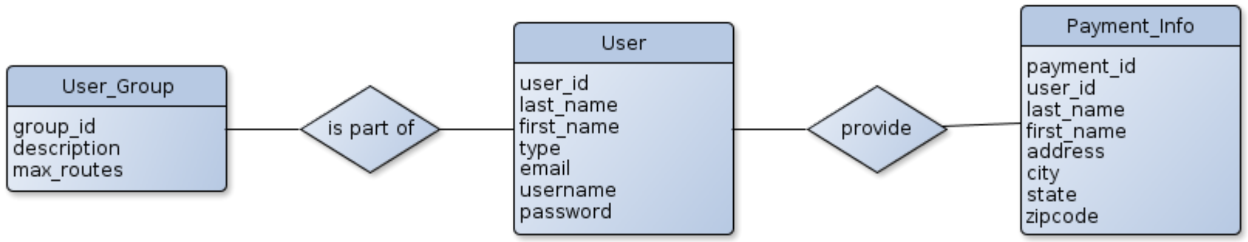


Figure X: Driver Profile ERD

Driver Profile Database					
Table	**Attribute**	**Datatype**	**Key**	**Requirements**	**Null**
User_Group	group_id	int	Primary Key	Must auto-increment starting at 1	Not Null
	description	text			Not Null
	max_routes	int		Must be a positive integer	Not Null
User	user_id	int		Must auto-increment starting at 1	Not Null
	last_name	varchar			Not Null
	first_name	varchar			Not Null
	type	int	Foreign Key to User_Group (group_id)	Must be a positive integer	Not Null
	email	varchar			Not Null
	username	varchar		Must be unique	Not Null
	password	varbinary			Not Null
Payment_Info	payment_id	int		Must auto-increment starting at 1	Not Null
	description	varchar		Must ambiguously identify payment data	Not Null
	user_id	int	Foreign key to User (user_id)	Must be a positive integer	Not Null
	last_name	varchar		Must exist if different from User (last_name)	Null
	first_name	varchar		Must exist if different from User (first_name)	Null

Table X: Driver Profile Database

3.1.1.2. Virtual Checkpoint Database (Kennedy)

The Virtual Checkpoint Database will store all Virtual Checkpoint data. This data includes: information describing each checkpoint and the statistics for each checkpoint. The structure of the database is illustrated in Figure X. The constraints and requirements of the database are listed in Table X. The database must satisfy the constraints and structure as described in the aforementioned figure and table.

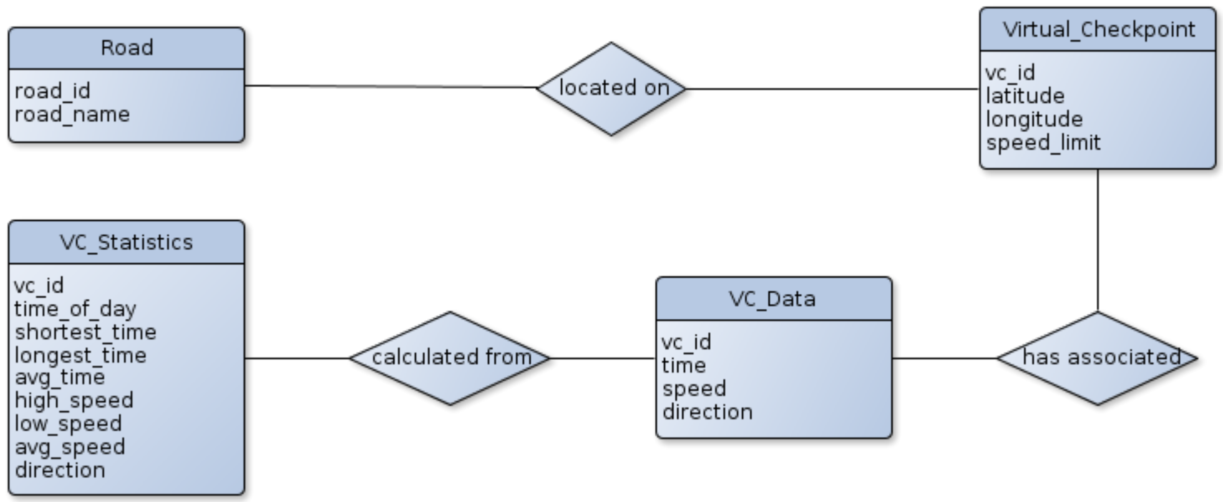


Figure X: Virtual Checkpoint ERD

Virtual Checkpoint Database					
Table	**Attribute**	**Datatype**	**Key**	**Constraints**	**Null**
Road	road_id	int	Primary Key	Must auto-increment starting at 1	Not Null
	road_name	varchar			Not Null
Virtual_Checkpoint	vc_id		Primary Key	Must be a positive integer	Not Null
	latitude	float		Must be a positive real number	Not Null
	longitude	float		Must be a positive real number	Not Null
	road_id	int	Foreign Key to Road (road_id)	Must be a positive integer	Not Null
	speed_limit	float		Must be a positive real number	Not Null
VC_Data	vc_id	int	Foreign Key to Virtual_Checkpoint (vc_id)		Not Null
	time	timestamp			Not Null
	speed	float		Must be a positive real number	Not Null
	direction	char			Not Null
VC_Statistics	vc_id	int	Foreign Key to Virtual_Checkpoint (vc_id)	Must auto-increment starting at 1	Not Null
	time_of_day	varchar			Not Null
	high_speed	float		Must be a positive real number	Not Null
	low_speed	float		Must be a positive real number	Not Null
	avg_speed	float		Must be a positive real number	Not Null
	direction	char			Not Null

Table X: Virtual Checkpoint Database

A Virtual Checkpoint database driver must be implemented to manage the insertion of information into the database and retrieval of information from the database (Sections 3.1.2.2 – 3.1.2.6) . The following functional requirements must be met:

1. All VC_Data must correspond to a pre-defined Virtual Checkpoint (Section 3.1.2.3).

2. All VC_Data must be removed when the corresponding checkpoint is flagged as inactive – cascading delete upon the removal of a checkpoint (Section 3.1.2.3).
3. All VC_Statistics must be removed when the corresponding Virtual Checkpoint is flagged as inactive (Section 3.1.2.3).

3.1.1.3. Speed Limit Database (Kennedy)

The Speed Limit Database will exist as a cache of all information retrieved from an external speed limit database. The structure of the database is explained in Figure X. The constraints and requirements placed upon the database are explained in Table X. The database must satisfy the constraints and structure as described in the aforementioned figure and table.

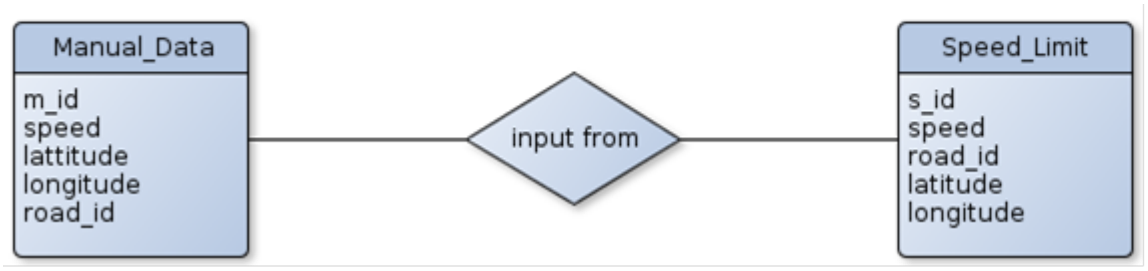


Figure X: Speed Limit ERD

Speed Limit Database					
Table	**Attribute**	**Datatype**	**Key**	**Constraints**	**Null**
Manual_Data	m_id	int	Primary Key	Must auto-increment starting at 1	Not Null
	speed	int		Must be a positive real number	Not Null
	latitude	float		Must be a positive real number	Not Null
	longitude	float		Must be a positive real number	Not Null
	road_id	int		Must be a positive integer	Not Null
Speed_Limit	s_id	int	Primary Key	Must auto-increment starting at 1	Not Null
	speed	int		Must be a positive real number	Not Null
	road_id	int		Must be a positive integer	Not Null
	latitude	float		Must be a positive real number	Not Null
	longitude	float		Must be a positive real number	Not Null

Table X: Speed Limit Database

3.1.2. Algorithms

The Traffic Wizard server software will host a set of algorithms that are used for specific functionality of the Traffic Wizard system. From aggregating speed data to finding blockages in the road, these algorithms give Traffic Wizard its capabilities. Client-side algorithms include algorithms for the smartphone app and the Simulation Console.

3.1.2.1. Speed Aggregator (MacLeod)

One of the most important functions within the Traffic Wizard system is how it computes the speed at which cars are travelling at a virtual checkpoint. Assigning the speed of the car that most recently passed would lead to the speed of the checkpoint changing frequently and drastically. Therefore, an algorithm will be created to adjust the checkpoint's speed based on the currently allocated speed, the incoming speed data, and the time since the checkpoint was last updated. The flow of the algorithm is demonstrated in Figure X. The following functional requirements must be met:

1. Must contain a time range in which the old speed data at a checkpoint is considered relevant. The time range is stored within the checkpoint and is determined by how frequently users pass the checkpoint.
2. Must use the old checkpoint speed data if the time passed since it was updated is within the current range
3. Must receive new speed data from user smartphone devices to update
4. Must aggregate multiple inputs into one if more than one input is received between updates
5. Must be able to weight incoming values based on the time passed since the last update
6. Must be able to adjust the update range based on the volume of users passing the checkpoint
7. Must return the new speed of the checkpoint and the time this update occurred
8. Must be able to efficiently detect speed changes at a virtual checkpoint

9. Must be able to reduce or eliminate the effect of outliers

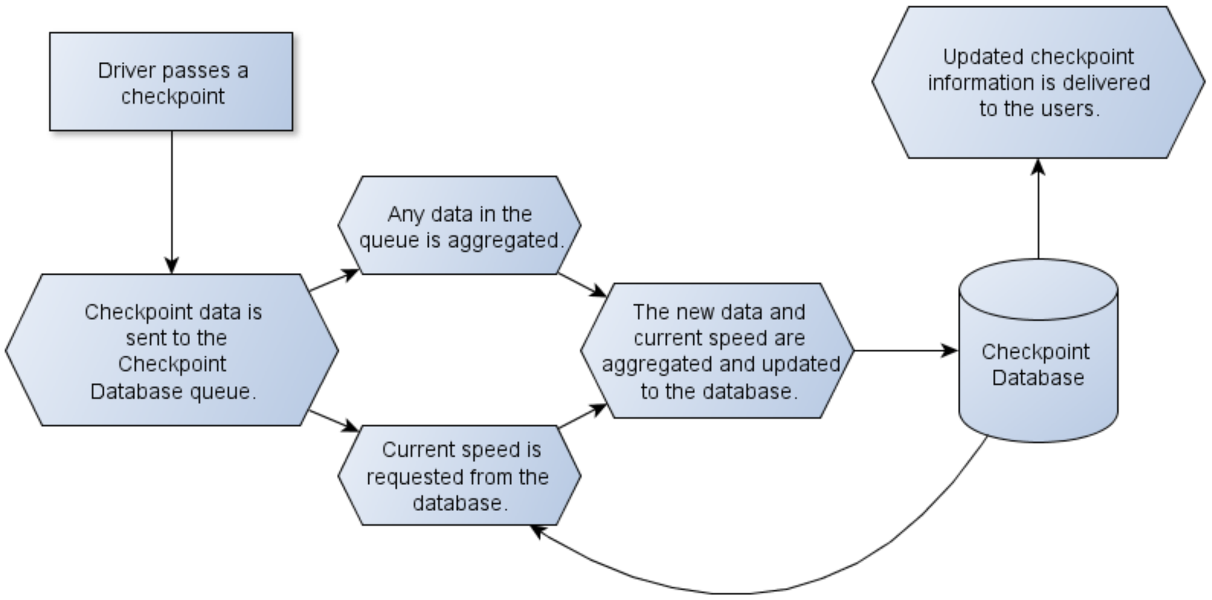

Driver passes a checkpoint
Checkpoint data is sent to the Checkpoint Database queue.
Any data in the queue is aggregated.
Current speed is requested from the database.
The new data and current speed are aggregated and updated to the database.
Checkpoint Database
Updated checkpoint information is delivered to the users.

Figure X: Prototype Features Table

3.1.2.2. Checkpoint Allocator (McKnight)

Creates the initial set of checkpoints for a given region. Using publicly available traffic information from VDOT, the density of checkpoints per road will be determined. Then, the location for a checkpoint will be determined using Google's Geolocation service. Once created, the set is inserted into the Virtual Checkpoint Database.The following functional requirements must be met:

1. Build model of roads in the coverage area
    a. The model shall follow the following design:

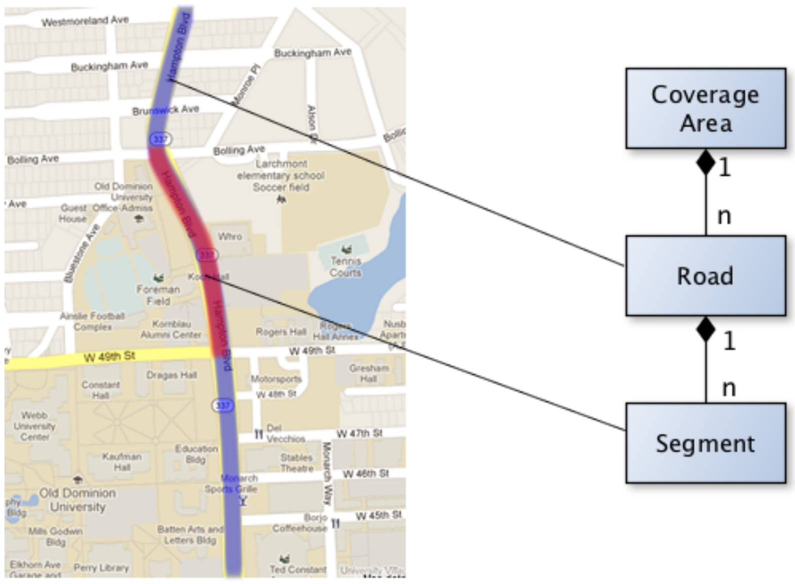


Figure X: Schematic for road model used by Checkpoint Allocator

b. Use Google Geolocation API to determine latitude and longitude positions along roads

c. Divide roads into segments according to traffic volume reports

d. Calculate the lengths of road segments and store in the model

e. Weight roads and corridors according to VDOT traffic volume data

2. Allocate checkpoints to each road segment

   a. Amount *n* of checkpoints per mile is bounded: 1 <= *n*<= 15

   b. Road segment weights are inversely proportional to checkpoint density

3. Iterate through entire model and insert checkpoints into Virtual Checkpoint Database

3.1.2.3. Checkpoint Re-allocator (Dong)

Figure X shows examples of how checkpoints are allocated in changing situation and how checkpoints are allocated by a road(s) speed limit.

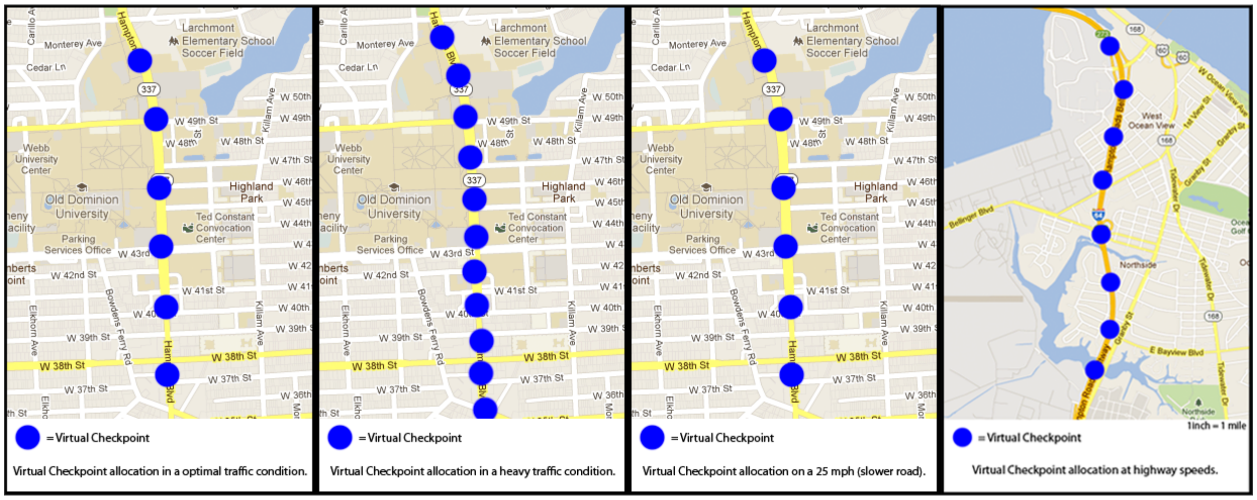

● = Virtual Checkpoint
Virtual Checkpoint allocation in a optimal traffic condition.
Virtual Checkpoint allocation in a heavy traffic condition.
Virtual Checkpoint allocation on a 25 mph (slower road).
Virtual Checkpoint allocation at highway speeds.

Figure X: Checkpoint Allocation Examples

The checkpoint reallocation algorithm will be responsible for dynamically moving the Virtual Checkpoints as traffic conditions change. The flowchart in Figure X demonstrates the checkpoint reallocation's basic logic.

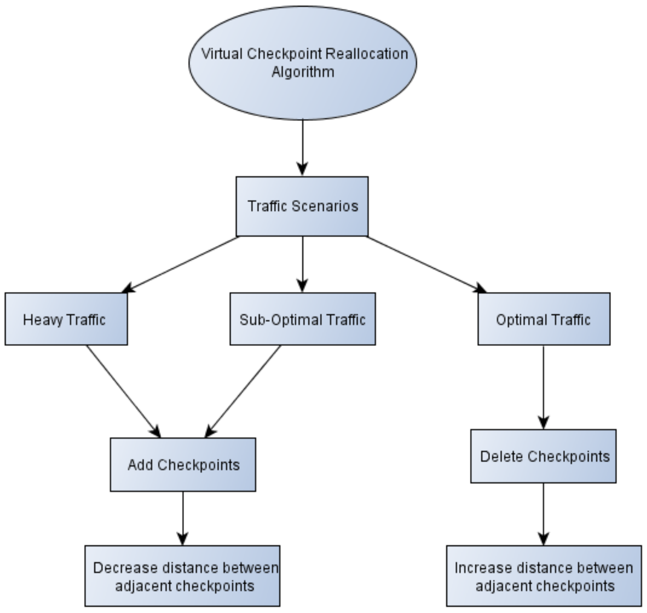

Virtual Checkpoint Reallocation Algorithm
Traffic Scenarios
Heavy Traffic
Sub-Optimal Traffic
Optimal Traffic
Add Checkpoints
Delete Checkpoints
Decrease distance between adjacent checkpoints
Increase distance between adjacent checkpoints

Figure X: Checkpoint Reallocation Flowchart

The following functional requirements must be met:

1. Must use C++/Java to implement algorithm to be supported on the server

2. Must support interfacing to be used by the server when requested
3. Must have the ability to access the Virtual Checkpoint Database (Section 3.1.1.2.)
4. Must have the ability to access the Speed Limit Database (Section 3.1.1.3)
5. Roads with a greater max speed must have fewer checkpoints when compared to roads with a lesser max speed. (Figure 1)
6. Must dynamically adjust the distance between two adjacent Virtual Checkpoints as traffic conditions of a road segment change.
7. Must increase the distance between two adjacent Virtual Checkpoints as traffic becomes optimal. Optimal traffic is reported speeds of Traffic Wizard users that are within 10 mph of the posted speed limit. (Figure 1)
8. Must decrease the distance between two adjacent Virtual Checkpoints as traffic becomes heavy. Heavy traffic is reported speeds of Traffic Wizard users that are less then 10 mph of the posted speed limit. (Figure 1)
9. Must deactivate a checkpoint if there has not been any activity in a defined time. Must reactivate the deactivated checkpoint when an adjacent checkpoint has been utilized.
10. Must update Virtual Checkpoint Database with new Virtual Checkpoint data (Section 3.1.1.2).

3.1.2.4. Route Matcher (Crossman)

The prototype Traffic Wizard server will use an algorithm to take a set of GPS coordinates (latitude and longitude) as input and return a virtual checkpoint or set of checkpoints that are the closest to those given coordinates. This will be used when the set of virtual checkpoints along a driver's route is unknown or the app does not seem to be registering that the driver is passing any checkpoints after a period of time. The Route Tracer feature of the app utilizes this algorithm in determining checkpoints along the road segments being driven to be saved as a route. This process is illustrated for the Route Tracer feature in Figure X.

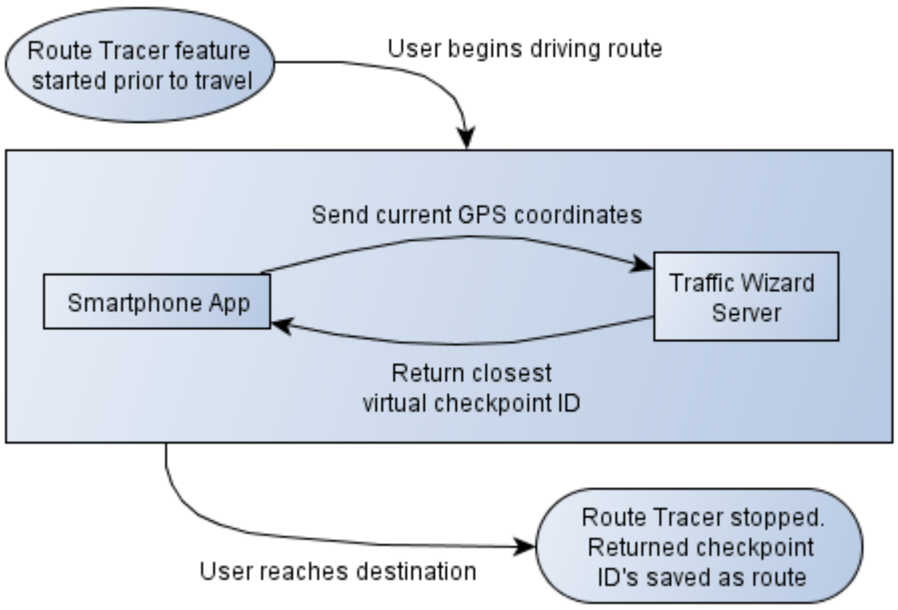

Route Tracer feature started prior to travel
User begins driving route
Send current GPS coordinates
Smartphone App
Traffic Wizard Server
Return closest virtual checkpoint ID
User reaches destination
Route Tracer stopped. Returned checkpoint ID's saved as route

Figure X: Route Matcher Algorithm Flow

The following functional requirements must be met:

1. Must use C++/Java to implement algorithm to be supported on the server
2. Must have the ability to access the Virtual Checkpoint Database (Section 3.1.1.2.) to determine what GPS coordinates to which the checkpoints are assigned
3. Must support GPS coordinate parameters:
    1. Input parameters for single GPS coordinate are floating point values for latitude and longitude
    2. Input parameter for GPS coordinate set (multiple coordinates) is a list structure with GPS coordinate objects (with latitude and longitude)
4. Must return ID of closest virtual checkpoint within a 100 foot radius to the smartphone device for a single GPS coordinate request

5. Must return a set of ID's of closest virtual checkpoints along the same road segment to the smartphone for a multiple GPS coordinate request
6. Must return that no checkpoint is close enough (within a 100 foot radius from the user) if the algorithm does not find a checkpoint within range

3.1.2.5. Route Analyzer (Kennedy)

The Route Analysis Algorithm will request checkpoint congestion data from the server. Calculations will be performed to determine level of congestion on the analyzed route. Figure X illustrates the flow of data between the smartphone application and the Route Analysis Algorithm.

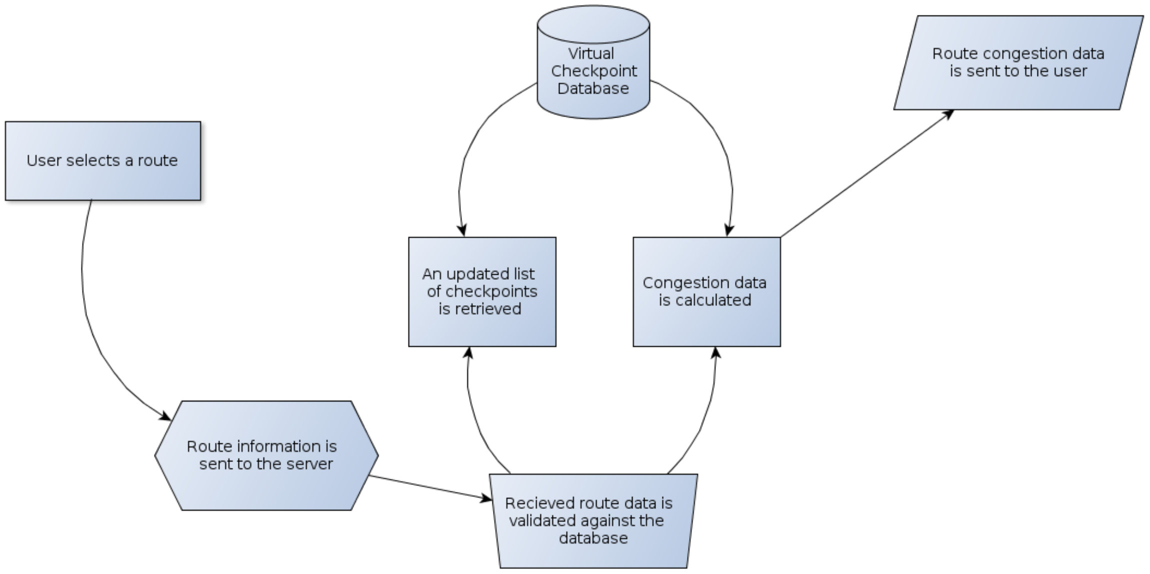

Virtual Checkpoint Database
Route congestion data is sent to the user
User selects a route
An updated list of checkpoints is retrieved
Congestion data is calculated
Route information is sent to the server
Recieved route data is validated against the database

Figure X: Route Analyzer Data Flow

The following functional requirements must be met:

1. The route information received from an end-user must be parsed to identify which Virtual Checkpoints must be analyzed.
    1. The parsed Virtual Checkpoint "v_id" values must be validated against the "v_id" values in the Virtual Checkpoint Database.

    2. When a "v_id" corresponds to an inactive Virtual Checkpoint, the appropriate procedure for the specific scenario must occur.
        1. When a Virtual Checkpoint is removed, an alert message must be passed to the smartphone application.
        2. When one or more new Virtual Checkpoints exist along a route the "v_id" values must be passed to the smartphone application.
2. Current congestion conditions must be calculated from the VC_Statistics data.
    1. When the available congestion data is not considered relevant – more than ten minutes has elapsed since the last data entry – an appropriate message shall be sent to the smartphone application.
    2. The calculations must be split into groups when the length of a route necessitates a large number of congestion calculations.
    3. The congestion data from each group of calculations must be sent to the end-user upon completion of aforementioned calculations.
        1. The groups shall be ordered based upon the distance from the driver's current position.
        2. The data from the congestion groups must be returned in the order determined in the previous requirement.
3. Outdated traffic congestion data must be returned to the smartphone application.
    1. A timestamp shall specify the time at which the statistics were most recently updated.
    2. The display of the congestion data is to be determined by the smartphone application (Sections 3.1.4.1.2 and 3.1.4.1.6).

3.1.2.6. Blockage Finder (Godavarthi)

- Construction in some areas may be planned, in that way before the travelling the driver will provide the address in the smartphone application. Possibilities of blockages may likely be closed areas ahead, reduction of travel lanes, lane diversions.
- The app is designed as when drivers move towards their destinations or respective travelling roads, checkpoint allocator algorithm is used for the virtual checkpoints to their initial latitude and longitude coordinates for a specific region. They will upload their travel metadata (such as time, speed, and direction) to the server for further analysis.
- If travelling speed is less than actual speed, the app will locate the blocked area; re-allocator algorithm will be effective which sends the data to the server. Or else the driver will continue to the destination point along the same path.

[Space intentionally left blank]

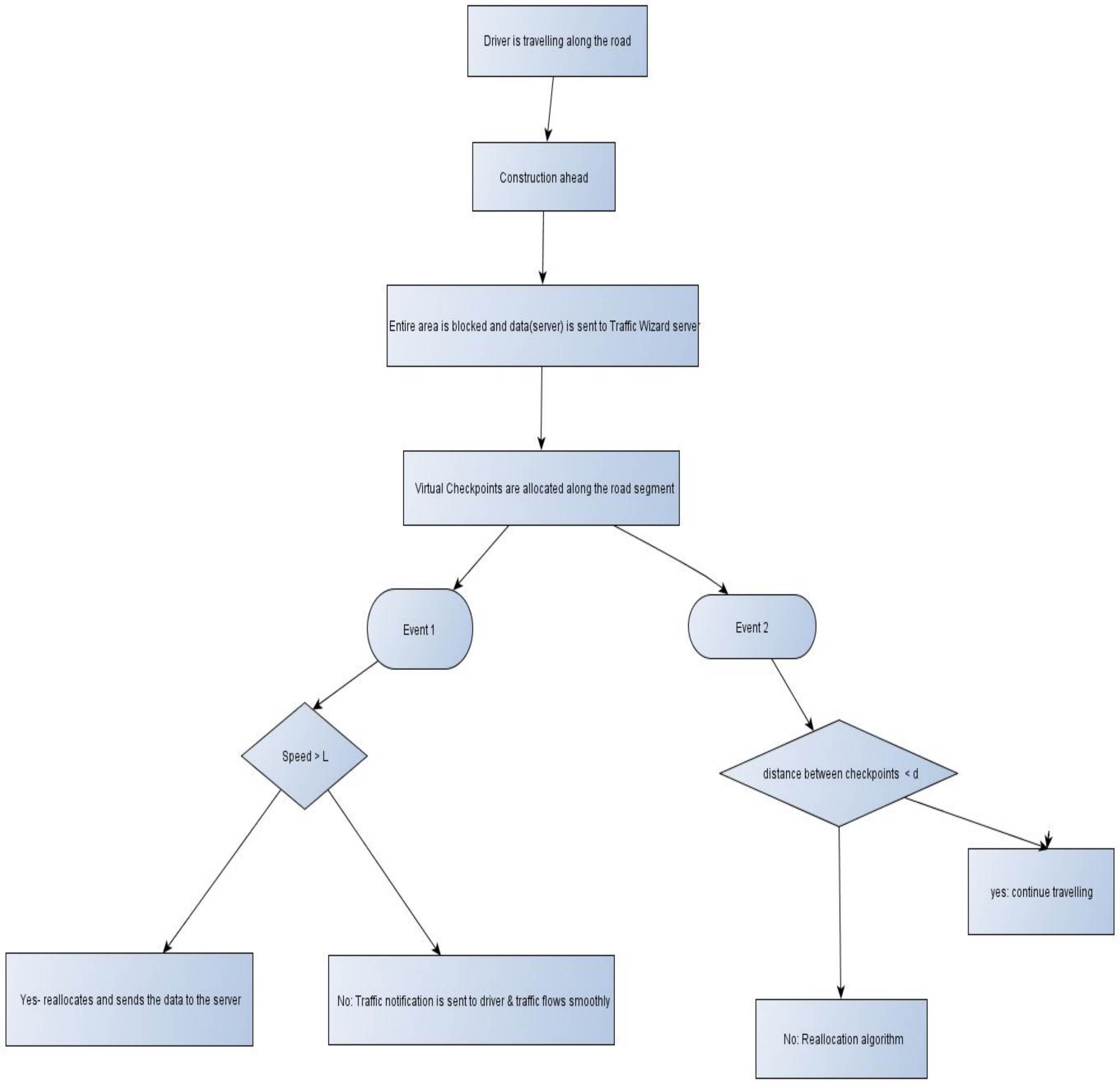

Driver is travelling along the road
Construction ahead
Entire area is blocked and data(server) is sent to Traffic Wizard server
Virtual Checkpoints are allocated along the road segment
Event 1
Event 2
Speed > L
distance between checkpoints < d
yes: continue travelling
Yes- reallocates and sends the data to the server
No: Traffic notification is sent to driver & traffic flows smoothly
No: Reallocation algorithm

Figure X: Blockage Finder Algorithm Flow

The following functional requirements must be met:

1. Must use C++ /Java to implement algorithm to be supported on the server.
2. Must support interfacing to be used by the server when requested.
3. Must have the ability to access the Virtual Checkpoint Database (Section 3.1.1.2.).
4. Must be able to locate the geographic area using GPS using Google Maps.

5. Must be in a time range to know the location of virtual checkpoint and retrieve the information.
6. The Route Analysis Algorithm will request checkpoint congestion data from the server and calculations are estimated to the traffic flow.
7. Approaching slow traffic speed, possibility of having a blockage wherein the Virtual Checkpoints are placed along the road segments will send the data to the server.
8. The data is sent to the Traffic Wizard application while travelling.

3.1.2.7. Next Checkpoint Estimator (McKnight)

Determines time until user reaches the next Virtual Checkpoint on their trip. The calculated information describes how long certain processes in the application may temporarily pause to conserve battery life and data usage.

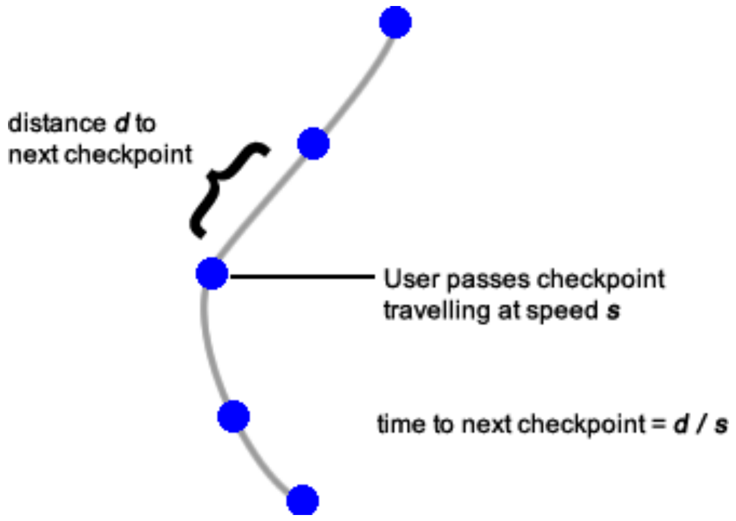

distance *d* to next checkpoint
User passes checkpoint travelling at speed *s*
time to next checkpoint = *d* / *s*

Figure X: Next Checkpoint Estimator schematic

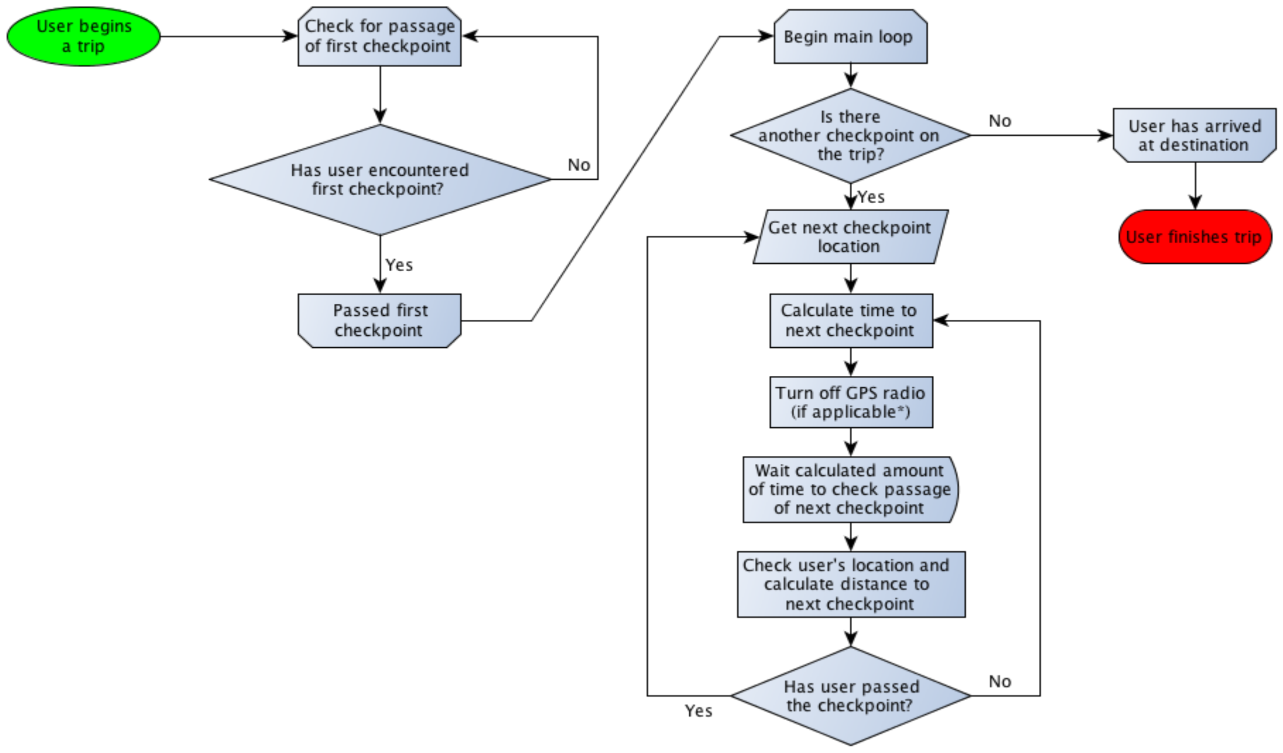

User begins a trip
Check for passage of first checkpoint
Has user encountered first checkpoint?
No
Yes
Passed first checkpoint
Begin main loop
Is there another checkpoint on the trip?
No
Yes
User has arrived at destination
User finishes trip
Get next checkpoint location
Calculate time to next checkpoint
Turn off GPS radio (if applicable*)
Wait calculated amount of time to check passage of next checkpoint
Check user's location and calculate distance to next checkpoint
Has user passed the checkpoint?
Yes
No

Figure X: Next Checkpoint Estimator execution flow

The following functional requirements must be met:

1. obtain a Location object from the iOSCoreLocation API
2. determine if user has left route
    1. Find distance n to the next checkpoint from current location
    2. Find distance b between the next checkpoint and last checkpoint
    3. If n > b, it shall be determined that user is no longer travelling on their prescribed route
    4. Generate an event to pass program execution to the calling function so the appropriate action may be taken by the user
3. use speed and Euclidean distance to next checkpoint to calculate time
4. return the time in milliseconds

3.1.3. Simulation Console (Crossman)

The platform used to demonstrate the Traffic Wizard prototype is the Simulation Console. This is a standalone application, written in C# using the .NET 4.0 platform, that will maintain communication with the Traffic Wizard server instead of user's phones in the real world product. This component of the prototype will not be used in the final product as it is just a means of proving Traffic Wizard's features using a set of simulations. The Simulation Console will support the selection of multiple region sizes and traffic scenarios to be used as part of the simulation and will be able to animate virtual drivers as they drive their route. Virtual checkpoint operations can be observed and the data exchange can be simulated using this platform. Each component of the Simulation Console has its own criteria for requirements.

3.1.3.1. Region Selection (Crossman)

The option to choose what region size to perform the simulation on must be available from the Simulation Console. To demonstrate that Traffic Wizard systems work effectively on varied scales, three geographical areas will be defined to use as the three options for regions. Each of these regions will be associated with a Google map to visually represent the area as well as a set of parameters to impact the generation of virtual drivers in the Driver Generator. The following functional requirements must be met:

1. Must support three different geographical areas to be labeled as small, medium, and large:
    1. Small: Old Dominion University and surrounding roads
        1. Upper-left boundary: W 49th St. and Powhatan Ave. intersection
        2. Lower-right boundary: Monarch Way and W 41st St. intersection
    2. Medium: Downtown Norfolk area

        1. Upper-left boundary: W Princess Anne Rd. and Hampton Blvd. intersection
        2. Lower-right boundary: I-264 at Grandy Village
    3. Large: Norfolk / Portsmouth area with Downtown Tunnel
        1. Upper-left boundary: Craney Island US Naval Supply Center in Portsmouth
        2. Lower-right boundary: Indian River Rd. and Lynnhaven Pkwy. intersection

2. Must have an associated Google Maps satellite image for each of the three areas. Boundaries for the image must align with the region bounds as defined by the region size (See requirement 3.1.3.1.1)
3. Must be selectable from the main dashboard of the Simulation Console to be changed as necessary during demonstration
4. Must define specific arrival entry points on inbound roads from selected region map to use in the Driver Generator (Section 3.1.3.3.):
    1. Small: Old Dominion University and surrounding roads
        1. Powhatan Ave., from North and South
        2. Hampton Blvd., from North and South
        3. Bluestone Ave, from North
        4. W 49th St., from East
        5. W 43rd St., from East
    2. Medium: Downtown Norfolk area
        1. W Brambleton Ave., from West

        2. W Princess Anne Rd., from West
        3. Llewellyn Ave., from North
        4. Monticello Ave., from North
        5. Church St., from North
        6. Tidewater Dr., from North
        7. E Princess Anne Rd., from East
        8. E Virginia Beach Blvd., from East
        9. Ballentine Blvd., from East
        10. I-264, from East and South
        11. Campostella Rd., from South
    3. Large: Norfolk / Portsmouth area with Downtown Tunnel
        1. I-64, from North and South, and from Chesapeake Blvd., Northhampton Blvd., Easton Forest, and Indian River Rd. interchanges
        2. Northhampton Blvd., from Northeast
        3. I-264, from East and West, and from N Military Hwy, Diamond Springs Rd., Effingham St., Tidewater Dr., and Portsmouth Blvd. interchanges
        4. I-464, from South, and from Bainbridge Blvd. and Berkley interchanges
        5. Martin Luther King Fwy, from West
5. Must define multiple specific destination points for each map for assignment during virtual driver generation:
    1. Small: Powhatan parking lot, Lot 42 (Powhatan Ave. and W 43rd St.), Elkhorn Ave. Garage, 43rd St. Garage, BAL parking lot (Hampton Blvd. and W 43rd St.), Constant Center North Garage, Constant Center South Garage, University

    Bookstore, 7-11 area, Rogers Hall, 49th St. Stadium Garage, and Library parking lot (between Kaufman Hall and Perry Library)
2. Medium: All entry points, Barraud Park, Middle Towne Arch, Norfolk State University, Elmwood Cemetery, MacArthur Center, Chrysler Museum of Art, Ghent Square, Town Point Park, and Harbor Park Stadium
3. Large: All entry points also act as destination points

3.1.3.2. Traffic Scenario Selection (Crossman)

The option to choose what traffic scenario to simulate must be available from the Simulation Console. To demonstrate that Traffic Wizard systems work for a range of possible traffic-related scenarios, different schemes of producing virtual drivers will be utilized to model various traffic conditions that a user would experience on the road. Each traffic scenario has a specified set of parameters to impact virtual driver generation in the Driver Generator. The following functional requirements must be met:

1. Must support different data sets for representing distinct and independent instances of certain traffic conditions:
    1. Low volume, low congestion, rare blockages
    2. Low volume, medium congestion, few blockages
    3. Low volume, medium congestion, many blockages
    4. Medium volume, low congestion, rare blockages
    5. Medium volume, medium congestion, few blockages
    6. High volume, low congestion, few blockages
    7. High volume, medium congestion, few blockages
    8. High volume, high congestion, many blockages

2. Must structure each traffic scenario with its associated arrival distribution for the Driver Generator (Section 3.1.3.3.)
3. Each traffic scenario must be scalable to be compatible with each of the three region sizes (Section 3.1.3.1.)
4. Must be selectable from the main dashboard of the Simulation Console to be changed as necessary during demonstration

3.1.3.3. Driver Generator (Crossman)

The software component responsible for the creation and management of virtual driver entities is the Driver Generator. This component uses simulation time to simulate arriving drivers to an area by using a mathematical distribution to determine the rate of arrivals. The selected region size and traffic scenario have an appropriate impact on the generation of drivers, and may change certain parameters in the algorithm as well. The following functional requirements must be met:

1. Must utilize mathematical probability distributions to define distribution of arrivals over time in reference to region size and traffic scenario. Variables to be utilized include:
    1. Arrival rate must support using a mathematic distribution with specified parameters
        1. Exponential (parameters: mean)
        2. Normal (parameters: mean, standard deviation)
        3. Triangular (parameters: minimum, mode, maximum)
        4. Weibull (parameters: beta, alpha)
    2. Percentage of arriving virtual drivers that are using Traffic Wizard to define drivers with re-routing abilities

2. Must be selectable from the main dashboard of the Simulation Console to allow changing the percentage of virtual drivers with re-routing capabilities (using Traffic Wizard)
3. Must create virtual driver objects according to specific arrival distribution for the selection region and traffic scenario as simulation time advances
4. Must assign random arrival entry point to newly created virtual drivers from list of available entry points for the region map to serve as starting location (Section 3.1.3.1.)
5. Must assign random destination point to newly created virtual drivers from list of available destinations for the region map to serve as ending location (Section 3.1.3.1.)
6. Must have the capability to send the virtual driver object into the system after creation and assigning arrival/destination attributes

3.1.3.4. Simulation Runtime Execution (Crossman)

Certain functional requirements exist for the runtime execution of simulations in the Simulation Console. These correspond to configuration of the demonstration and requirements for the simulation itself. The following functional requirements must be met:

1. Must require that a region and traffic scenario have been selected in the main dashboard before executing
2. Must advance simulation time using a floating point variable to represent the time-step at each iteration
3. Must contain user interface controls to Start, Pause, and Stop the current simulation setup
4. Must accept a time frame ranging from 1 minute to 15 minutes to set as the terminating condition for the simulation if it is not stopped
5. Must support two types of virtual driver objects arriving to the system:

   1. Drivers not using Traffic Wizard have no ability to alter the set of road segments they take once they set out on their route (no re-routing functionality)
   2. Drivers using Traffic Wizard have the ability to take advantage of traffic updates by the server and change their route to avoid congestion (re-routing functionality)

3.1.3.5. Traffic Activity Display (Crossman)

In order to effectively demonstrate the Traffic Wizard system at work, the simulations must have a visual element that lets viewers see how the system works. By displaying the map associated with the selected region size (Section 3.1.3.1.), the demonstration can show the area that will be used in the simulation. Once the simulation is started, virtual driver entities will visually appear on the map and animate to move along the roads according to simulation time. The following functional requirements must be met:

1. Must display designated region map to take up Simulation Console window based on chosen region (Section 3.1.3.1.)
2. Must display virtual driver entities in the simulation as colored circles on the region map that are 16x16 pixels in size and colored according to the type of virtual driver
   1. Drivers with re-routing functionality (using Traffic Wizard) are marked as blue
   2. Drivers without re-routing functionality (standard driver) are marked as white
3. Must animate virtual driver objects by moving their positions as the simulation time advances according to Simulation Runtime Execution (Section 3.1.3.4.)
4. Must denote found blockages with a 8x18 pixel red rectangle on the map at the coordinates that the blockage occurred
5. Must represent virtual driver objects that are within near proximity on the same road so that they do not overlap each other (circle edges cannot intersect other circles)

3.1.4. GUI

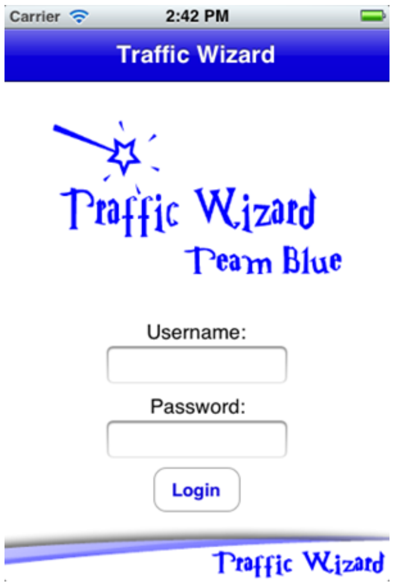
The Traffic Wizard system uses two different sets of graphical user interfaces on two separate platforms. A prototype smartphone app was developed for iOS that contains a mapping of several GUI screens. The Simulation Console, the demonstration platform for Traffic Wizard simulations, uses its own set of GUI screens to achieve its functionality.

3.1.4.1. Smartphone Application Interface (McKnight)

The client smartphone application must be an intuitive, robust interface. Unexpected input must be guarded against, and navigation between views must be strictly controlled to prevent unauthorized use and tampering. While in Drive Mode—either using Route Tracer or driving on a saved route—visual and touch interaction with the smartphone must be minimized.

3.1.4.1.1. Login (McKnight)

Upon opening the application, the first screen the user sees is a login screen.



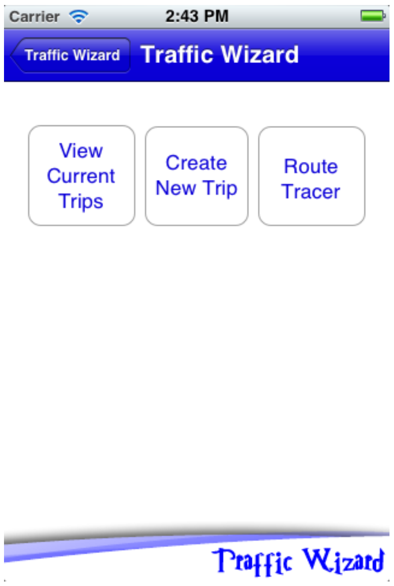


Figure X: Traffic Wizard Login (left) and Main Screen after Login (right)

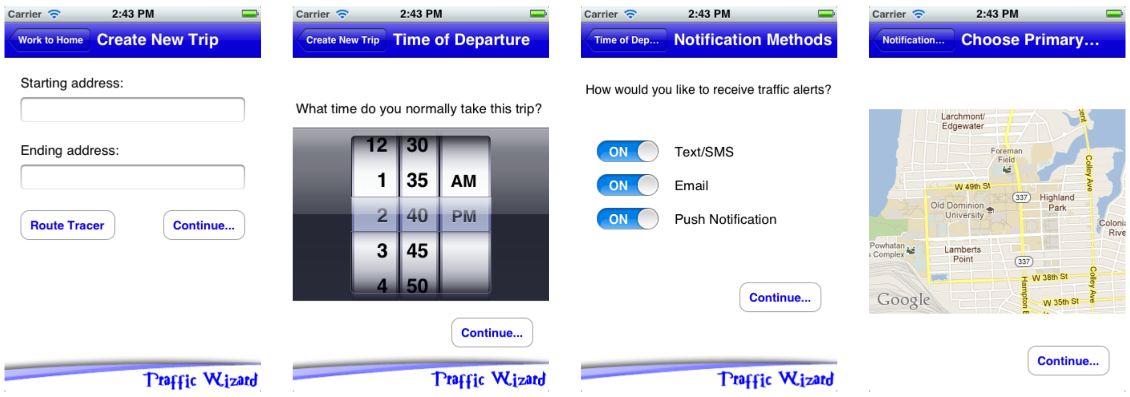
The following functional requirements must be met:

1. User must input a username and a password to enable the “Login” button

2. Username and passwords must be validated to only contain alphanumeric characters
3. Login info is matched to data in Driver Profile Database
4. Either a ‘success’ or ‘fail’ signal is sent and appropriate action is taken:
    i. Unsuccessful login attempts keep user at login screen, displaying alert
    ii. Upon successful login, user is taken to the main screen

3.1.4.1.2. New Trip (McKnight)

Users must be able to create new Trips to take full advantage of the customized profile feature. A series of screens will take the user through all of the options to set for a Trip.



What time do you normally take this trip?

How would you like to receive traffic alerts?

ON Text/SMS

ON Email

ON Push Notification

Figure X: New Trip Menu Screens

The following functional requirements must be met:

1. Name must be checked for illegal characters and for existing match in local database
2. Start and End addresses must be validated and stripped of illegal characters
    1. Addresses must exist in the Google Geocoding API
    2. User may employ the Route Tracer (Section 3.1.4.1.3)
3. Time of departure will begin on a default value, so it will never be undefined
4. Notification methods are all disabled by default
5. Zero or more notification methods may be selected

6. User is presented with a list of choices for primary route, of which only one at a time is selectable
7. All information is written to the local database

3.1.4.1.3. Route Tracer (McKnight)

The Route Tracer environment may be accessed independently to define a Trip to be saved or from the New Trip screen (Section 3.1.4.1.2). After completing usage, further options are presented to the user similar to new trip creation.

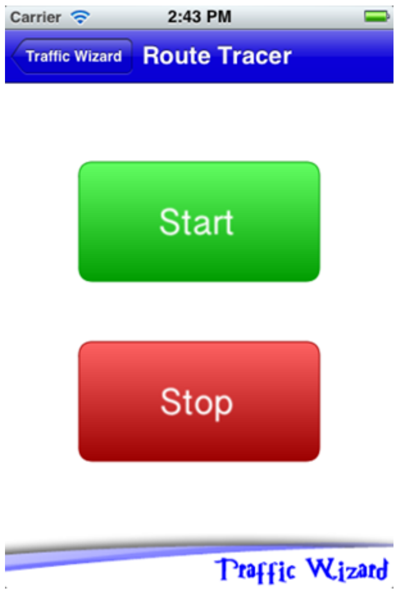


Figure X: Route Tracer Screen

The following functional requirements must be met:

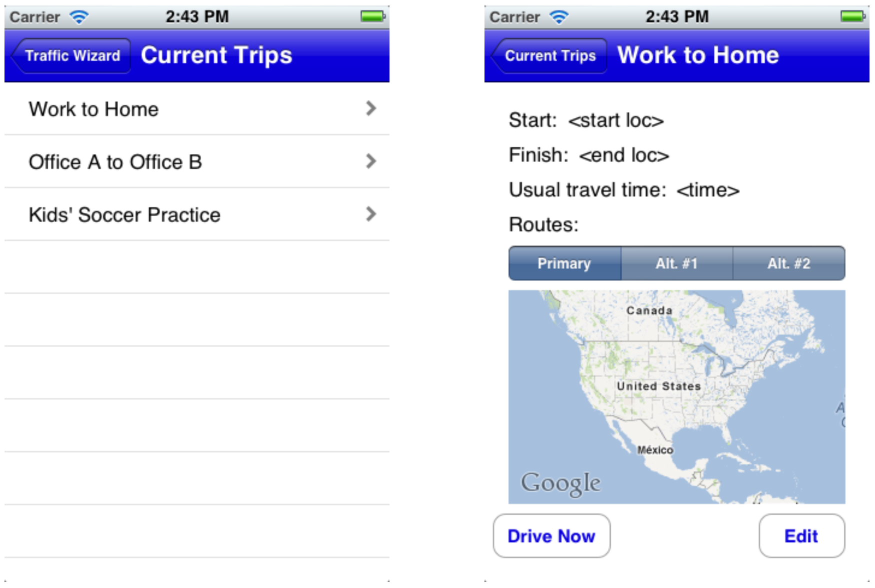
1. Initially, only Start button is enabled
    1. After Start button is pressed, it must be disabled
    2. Stop button must be enabled after Route Tracer has begun
2. Pressing the Start button begins the Route Tracer algorithm in a separate execution thread
    1. User may navigate elsewhere in the program or smartphone during Route Tracer execution

2. Only one instance of Route Tracer must be allowed to run concurrently

3. Pressing stop button stores lat/long coordinate vector in local memory and reverts program control to New Trip Creation (Section 3.1.4.1.2)

3.1.4.1.4. Edit Trip (McKnight)

Users may edit existing Trips in their profile to change all the same options available upon Trip creation. Trip editing must be initiated from the Current Trips screen.



Start: <start loc>
Finish: <end loc>
Usual travel time: <time>
Routes:

Figure X: Current Trips Menu (left) and Trip Details Screen (right)

The following functional requirements must be met:

1. Every Trip a user has programmed must be editable
2. See Section 3.1.4.1.2 for specifications on modifiable options

3.1.4.1.5. End of Trip (McKnight)

Upon completion of a Trip, the user must be alerted that Traffic Wizard has ceased Drive Mode. The user is presented with statistics and the opportunity to edit the trip.

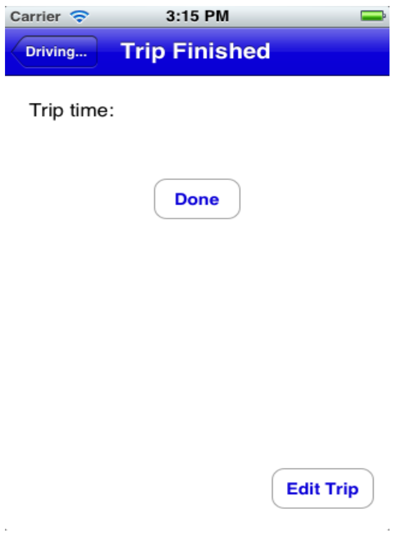


Figure X: End of Trip Screen

The following functional requirements must be met:

1. This screen must only be reachable from Drive Mode
2. Option to edit trip must be presented to the user

3.1.4.1.6. Delay Notification (McKnight)

Users may manually analyze a route or set a time to automatically analyze a saved route. In the event an excessive delay is found, the user must be notified to take appropriate action. In the event that analysis was performed automatically, users must be notified by Push Notification.

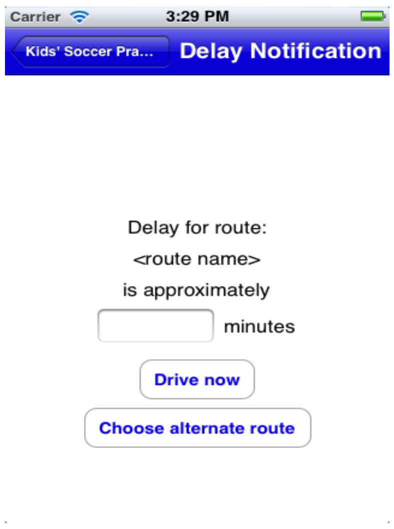


Figure X: Delay Notification Screen

The following functional requirements must be met:

1. The delay time must not be a negative number
2. If user is currently using Traffic Wizard in Still Mode when notification is sent, they must be taken to this screen immediately
3. If user is in Drive Mode, a voice alert must be generated
4. Otherwise, a push notification is sent to the smartphone

3.1.4.2. Simulation ConsoleInterface (Crossman)

The graphical interface layout for the Simulation Console will consist of a Main Menu that has options to experience the different aspects of the Traffic Wizard prototype. Each component of the console is a demonstration of a specified Traffic Wizard feature.

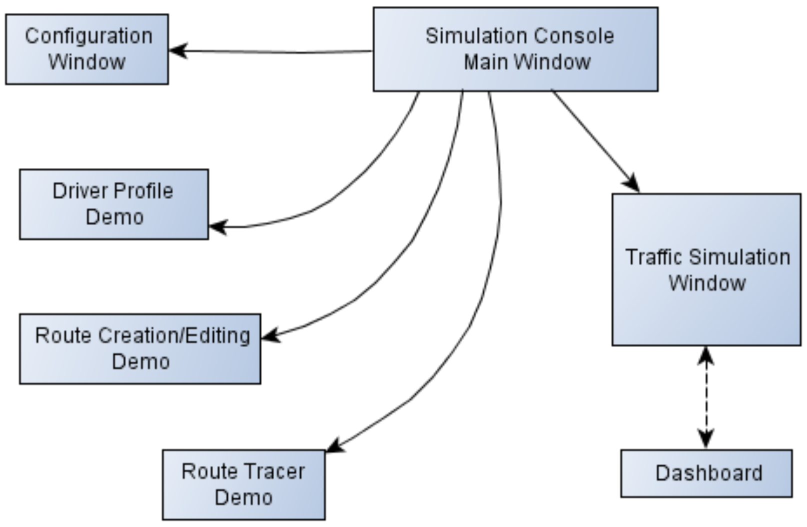

Configuration Window
Simulation Console Main Window
Driver Profile Demo
Traffic Simulation Window
Route Creation/Editing Demo
Route Tracer Demo
Dashboard

Figure x: Simulation Console Sitemap

3.1.4.2.1. Main Menu (Crossman)

When the Simulation Console launches, the Main Menu is the first screen that appears and requires interaction. This window acts as the main portal to the various Traffic Wizard demonstrations for each innovative feature. The following functional requirements must be met:

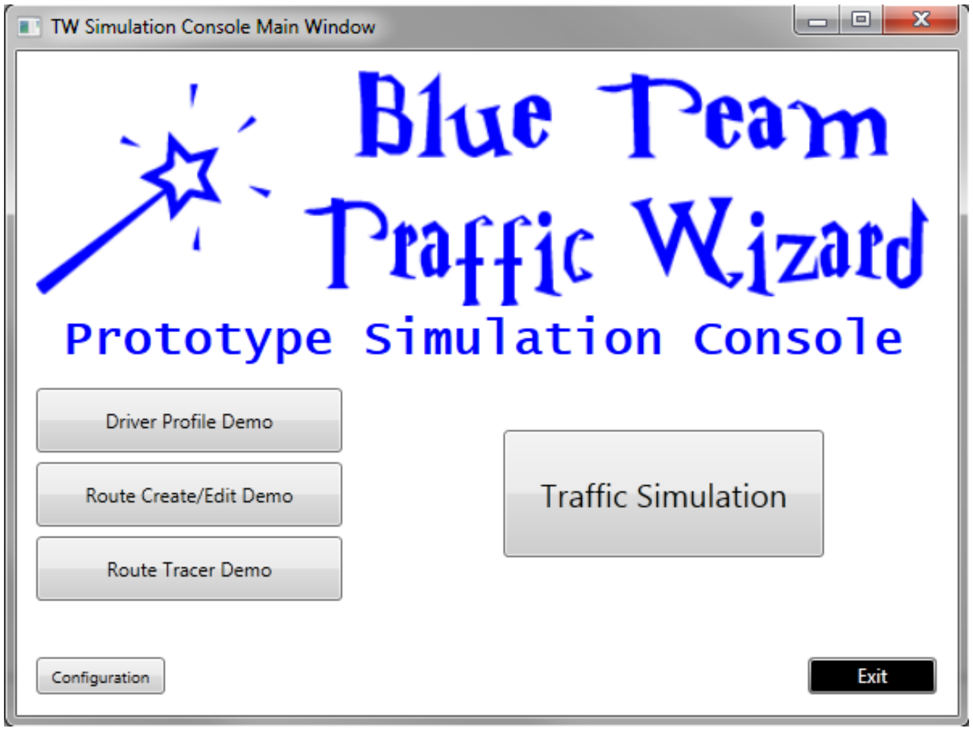


Figure X: Simulation Console Main Menu

1. Must contain Traffic Wizard/Blue Team logo
2. Must contain buttons to open each of the demo screens:
    - Driver Profile Demo
    - Route Creation/Editing Demo
    - Route Tracer Demo
    - Traffic Simulation
3. Must contain a button to access configuration options for using the Simulation Console
4. Must contain an Exit button to quit the application

3.1.4.2.2. Configuration Window (Crossman)

Options for using the Simulation Console are present in this window. From here, a connection to the Traffic Wizard server can be established or verified. An About Traffic Wizard option will also be visible that brings forth version information on the software. The following functional requirements must be met:

1. Must provide options for:
    1. Connect to Server: Entering new server information to connect (IP address, Port)
    2. Verify Connection: Verifying server connection is established
    3. About Traffic Wizard: Showing version information on Simulation Console software and names of developers
2. Must be able to return to Main Menu when configuration is done

3.1.4.2.3. Driver Profile Demo (Crossman)

This window provides a text summary of how Traffic Wizard driver profiles work and shows each feature that is associated with a driver profile. Various GUI images of the Traffic Wizard smartphone app will be used here to show the process. The following functional requirements must be met:

1. Must describe all features of driver profiles as outlined in Section 2.2.
2. Must use smartphone app GUI from Section 3.1.4.1. as foundation for images
3. Must be able to return to Main Menu at any time

3.1.4.2.4. Route Create/Edit Demo (Crossman)

This window provides an overview of how Traffic Wizard users can create routes to be stored on their phone as part of their driver profile. Created routes can also be edited in the future if the user wishes to do so. Various GUI images of the route creation screens for the smartphone app will be used here to show the process. The following functional requirements must be met:

1. Must describe all fields required for creating a new route manually as outlined in Section 3.1.4.1.3.
2. Must use smartphone app GUI from Section 3.1.4.1. as foundation for images
3. Must be able to return to Main Menu at any time

3.1.4.2.5. Route Tracer Demo (Crossman)

This demonstration will describe the ability for Traffic Wizard to trace a driven route when a user wishes to enter their route into their profile by driving it. When enabled, the Route Tracer will ping GPS coordinates while the user drives in order to define the route they are taking. At the end of the trip, the app defines the route in more definite terms and saves it to the user's profile. The following functional requirements must be met:

1. Must describe entire process of locating, initializing, using, and closing the Route Tracer feature from the app
2. Must use smartphone app GUI from Section 3.1.4.1. as foundation for images
3. Must be able to return to Main Menu at any time

3.1.4.2.6. Traffic Simulation Window (Crossman)

This window is the main focus of the Traffic Wizard demonstrations and acts as a stage for the system to be demonstrated upon. All activity in this window will appear on the central Google map once a region size and traffic scenario has been chosen and the simulation has been started. The following functional requirements must be met:

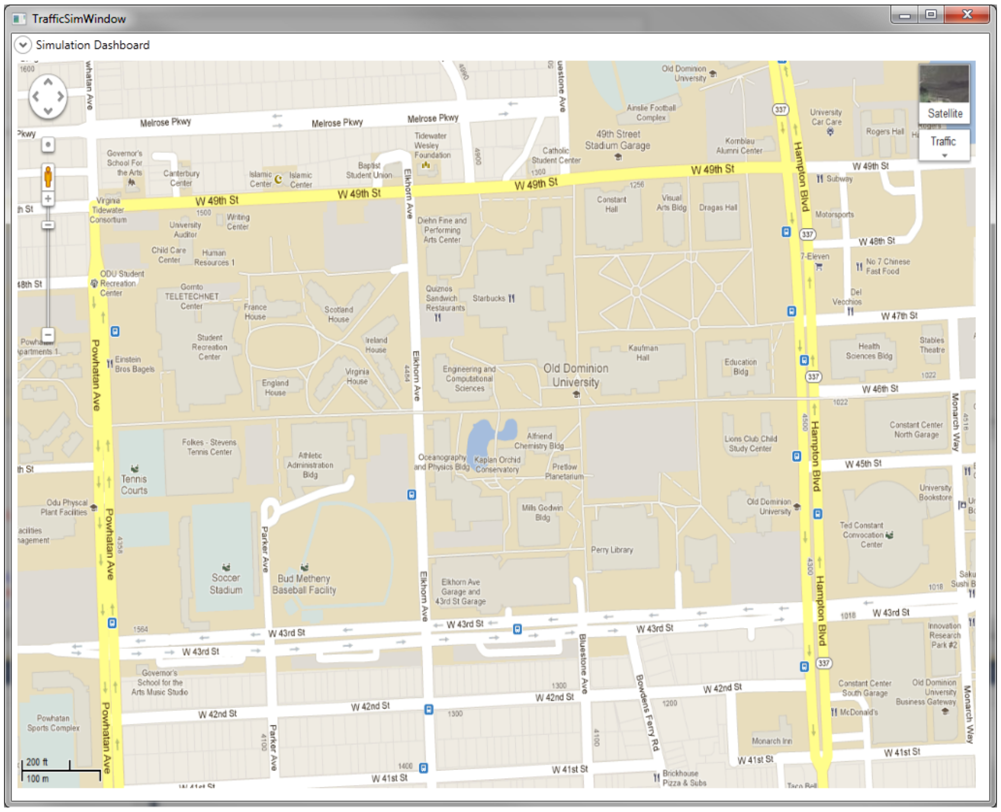


Figure X: Traffic Simulation Window

1. Must show the Small region size map by default if another region has not been chosen
2. Must display entire map segment that is defined by the boundaries outlined in Section 3.1.3.1.
3. Must show virtual driver entities and virtual checkpoint statuses with animation support as outlined in Section 3.2.5.

3.1.4.2.7. Dashboard (Crossman)

The Dashboard is an expandable menu from the top bar in the Traffic Simulation Window labeled as the Simulation Dashboard. This menu acts as a control panel for the simulation being demonstrated by presenting options to change the environment of the simulation. Options to change the region size map, traffic scenario, and percentage of users using

Traffic Wizard (out of arriving virtual drivers) must be available to be changed to create an altered simulation. The following functional requirements must be met:

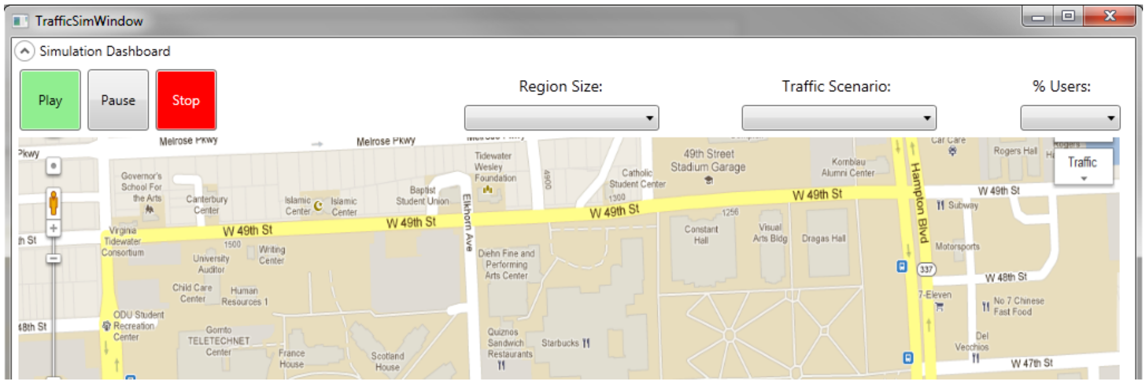


Figure X: Simulation Dashboard Expanded (Top of Traffic Simulation Window)

1. Must have expand and collapse capability that overlaps the region map
2. Must have functional control buttons to Play, Pause, and Stop the current simulation
    1. Play: Start a new simulation with current options, or resume a paused simulation
    2. Pause: Freeze animation and stop computation for current simulation
    3. Stop: End current simulation and prompt user to confirm if they want to clear the map of virtual driver entities
3. Must provide a drop-down box to select the region size (regions in Section 3.1.3.1)
4. Must provide a drop-down box to select the traffic scenario (scenarios in Section 3.1.3.2)
5. Must provide a drop-down box to select a percentage amount of Traffic Wizard users to be sent to the Driver Generator (Section 3.1.3.3.)
6. Must be able to return to Main Menu from Traffic Simulation Window at any time where the simulation is paused or stopped.
7. Must not allow returning to Main Menu if a simulation is playing (running).

3.2. Performance Requirements

Performance is a very important aspect of the Traffic Wizard system since the nature of the data being observed is very temporal. Server operations and database activity must be performed in the most efficient manner in order to provide accurate and timely updates on traffic conditions. For the prototype, this is still true since the server must be able to handle data being sent from the virtual drivers from the Simulation Console. In order to fluently execute Traffic Wizard features, database read/write speed, wireless transmissions speed, algorithm efficiency, and smartphone resources must all be optimized.

3.2.1. Database Read/Write Speed (MacLeod)

The speed at which the Traffic Wizard database is able to be accessed is integral to the design of the program. It determines how often the checkpoint speeds will be updated as well as how often each user can be sent the updated information. The speed of the database also limits the user load that can be placed on the server. The database has these performance requirements.

1. Must update checkpoint speeds frequently enough for the information to be relevant
2. Must send users updated information every time it is requested before the same user requests information again
3. Must minimize the total amounts of reading and writing whenever possible
4. Must queue incoming checkpoint information to be updated when unavailable to do so immediately
5. Must always be available to read

3.2.2. Smartphone Resource Management (Godavarthi)

A smartphone application, Traffic Wizard is developed in such a way that the product can be used by drivers that will assist them in heavy traffic along their typical and new routes. This application will bring efficiency to the drivers and can be in profit to the market industry. Virtual Checkpoint System (VCS) updates the current real-time traffic conditions and route analysis engine. The VCS is a method where GPS labeling of specific latitudes and longitudes coordinates along roads which gives the traffic status in the current area and provides the exchanged data to the driver. Every user or driver have their own personal travel profiles. The application is based on iPhone support, receiving notification alerts, turn-by-turn GPS directions and time predictions. In this process, an important feature is introduced usage reduction of battery where in other apps don't include this feature.

The Virtual Checkpoint System (VCS) is a process to determine thee updates when the user or driver is sent and travel information is obtained from the server. This information is sent to the driver who is travelling. With the simulated GPS roads these virtual checkpoints play an important role and are reallocated. This way, the usage of smartphone battery can be minimized. The network usage will be also be used properly, thereby people do not have pay more for the network usage. Interaction with an app while driving is high distraction risk. This will be mitigated with an interface that assists the driver with likely a physical interface with the device. This is a risky posture for the driver to operate the app while driving. With the Traffic Wizard features that won't be a problem. Drivers or users can preset the directions before travel. Traffic Wizard will deliver timely, accurate traffic information to drivers and enabling them to make the best possible travel decisions.

3.2.3. Wireless Data Transmission/Fidelity (Dong)

Cellular wireless data is a must for the client side of Traffic Wizard. Data must be bi-directional from the server and Traffic Wizard application.

1. Wireless data transmission must be kept to a minimum.

2. Wireless data shall be sent via the UDP protocol.
3. Wireless data must be checked for integrity within the server.
4. Connections must be only open when data needs to be transferred.
5. Connections shall be closed when data transfer is complete.
6. Connections must not be constant or "always-on."
7. Wireless data must be sent via Wi-Fi if the smartphone is connected to a hotspot.
8. Wireless 3G must be available on the smartphone.

3.2.4. Algorithm Efficiency (McKnight)

The server algorithms (and to a lesser extent, client algorithms) are a large part in the chain of data flow. To ensure that data sent from one user is available to other users in as short a time as possible, the algorithms must be optimized.

1. Space complexity must be sacrificed to make any possible gains in computational complexity
    1. Use memorization and caching techniques to avoid recalculations
2. Worst case running times must not exceed O(n2)
3. Implementation in SQL must be maximized to take advantages of databases' optimized algorithms

3.2.5. Simulation Demonstration (Crossman)

Demonstration of how the Traffic Wizard system works in various traffic scenarios is the foundation for the Simulation Console presentations. Through proper animation during Traffic Simulation (Section 3.1.3.5.), viewers can experience a visual representation of traffic flow and the various components of the Traffic Wizard system. Virtual drivers must be animated across the roads, and virtual checkpoint statuses must be displayed in order to prove concept.

1. Must iterate between available menu options with no input delay and selected items must notate that they have been selected (e.g., button pressed in when selected)
2. Must assign region and traffic scenario to the simulation when selected from the dashboard with a maximum loading time of 5 seconds
3. Must be able to communicate with the server to send data, receive algorithm output, and define virtual checkpoint traffic status during each time-step
4. Must advance time-step at a fixed pace of one time-step per second when Play is selected from the dashboard
5. Must animate virtual driver objects based on server output for that time-step by moving their position before the next time-step occurs
6. Must be able to pause the simulation and keep the status of all components in the system the same (number of virtual drivers, checkpoint statuses, and driver position) to be still and visible on the Traffic Display
7. Must be able to resume the simulation from a paused state
8. Must be able to manually stop the simulation gracefully when Stop is selected from the dashboard by pausing the simulation and prompting the user to clear the map of current virtual checkpoints and drivers

3.3. Assumptions and Constraints

The Traffic Wizard prototype will operate on a set of assumptions and constraints that will act as boundaries for the prototype functionality. Each assumption is a pre-defined factor that will be considered when developing and demonstrating the product. These limitations are outlined in Table X.

Type	**Condition**	**Effects on Requirements**

Assumption	All sent data from a driver smartphone arrives at the server.	Performance requirements will not reflect real-world potential data loss through a 3G Internet connection.
Assumption	There will be varying percentages of users on the road using the Traffic Wizard app.	Simulation Console Traffic Scenarios will reflect the percentage of users during automatic driver generation.
Assumption	There will be limited geographical coverage available for analysis by the Traffic Wizard server.	Simulation Console Region selections will be scaled-down to select geographical regions in Hampton Roads, VA (chosen by Team Blue).
Assumption	The operating system handles GPS usage to maximize battery life.	Not necessary to handle turning the GPS on and off.
Assumption	Database implementations of algorithms for sorting, etc. are as efficient as possible.	In addition to using C++/Java in algorithm development, as much implementation shall be done using SQL as possible
Assumption	GIS data exists that discretizes roads into segments defined by latitude and longitude coordinates.	The Checkpoint Allocator implements a specific algorithm to calculate road segment lengths using Euclidean distances between coordinate points.
Constraint	User authentication will support a small number of users.	Driver Profile database will only contain entries for select test users (members of Blue Team).
Constraint	Will only support the iOS platform.	No Android or Windows Phone prototype will be developed.
Constraint	Blockage types will not be determined.	Only the fact that there is a slowdown/blockage will be displayed.
Constraint	New user creation will not be supported.	Several pre-generated profiles will need to be created for the simulation.
Constraint	Application settings will not be modifiable by user through a settings screen.	Only one configuration regarding how to transmit data or perform certain operations on the phone need be programmed.

Table x: Assumptions and Constraints Table

3.4. Non-functional Requirements (Godavarthi)

The non-functional requirements address features of the prototype that are connected with the functional requirements of the Traffic Wizard Prototype. Easy use is important as drivers and customers will need when downloading the application. To start using the application, they have to register and proceed by entering the routes and storing personal information. Non-functional requirement include security, maintainability, and reliability.

Security is an important component to keep the integrity of the Smartphone application. Maintainability and reliability are important in any application and an important aspect for the customer satisfaction.

3.4.1. Security (Godavarthi)

Security is important to the prototype because it will improve and protect the driver profile database information. Security measures should be taken to keep the information safe from unauthorized users. Only the admin users will have administrative privileges if new changes occur. Potential users will download the application from the app market to their respective smart phones. Once installed and run through the phone they can either sign in with an existing account or create a new one. Authentication will be very basic, consisting of username and password. Once registered, users will have access to their own personal driver profiles. This feature allows store their most frequent or favorite routes that they travel to be saved on their device. Stored routes can be pre-analyzed by the application.

3.4.2. Maintainability (Godavarthi)

The Traffic Wizard will require network maintenance and should handle properly. This application depends on maintenance and software to keep current updates and other required software's running in the background process. Maintained of all the database information and server information should be secure. The designed server and network connecting drivers must be maintained properly. As the application is likely built on these factors, the hardware and software testing should be ensured to provide the best for the smartphone application. It will improve the quality and interest of the customer if the application has no error issues.

3.4.3. Reliability (Godavarthi)

The advantage of using the Traffic Wizard smartphone application is reducing commute time for drivers along the planned route and eventually save money. The features include having personal profile, saving routes, less distraction, providing minimum usage of battery and network usage. This application is reliability to people, drivers, customers, taxi drivers living in large metropolitan areas.

3.4.3.1. Server Uptime/Redundancy (Godavarthi)

The configuration and design of the server will be distributed and data maintained properly. Performing the calculations for route analysis in the most efficient and productive manner will result in best real-time traffic updates. There will be an overload of information sent to the servers sometimes, so the server should be maintained carefully by providing the ongoing information to the phone. As this application involves receiving real-time traffic updates to pre travel or while driving. Any disturbances to the server will disturb the entire process and the risk mitigation should be immediately taken into action.