DYNAMIC ROUTE ARRIVAL TIME PREDICTION

By

Brian Michalski

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Major Subject: COMPUTER SCIENCE

Approved:

Mukkai S. Krishnamoorthy, Thesis Adviser

Rensselaer Polytechnic Institute
Troy, New York

April 2011
(For Graduation May 2011)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

The author would like to acknowledge the Parking and Transportation Department, the Vice President for Administration, and their respective staff at Rensselaer Polytechnic Institute for their continued support of the shuttle tracking program and other student initiatives.

ABSTRACT

Predicting the arrival time of a vehicles is an important operation in the mass-transit field. As passengers waiting at stops are increasingly connected to the internet, we seek to provide an estimation of time until the next arrival. Looking specically at non-scheduled transit systems, we explore methods for estimating the arrival time of vehicles providing access to near-real time travel data. To achieve this, we implement an extendable arrival estimation algorithm with adjustable parameters to account for variations in the transit system. In support of this, we develop several components to identify dynamic routes and the vehicles traveling on them. After extensive testing on a campus shuttle system, we found this approach to provide accurate arrival predictions. Further application of these techniques may improve the accuracy and performance of the prediction system.

CHAPTER 1
Route Identification

1.1 Introduction and Historical Review

Non-scheduled transit system vehicles have no expectation of arriving at a stop at a given time. With no published schedule, drivers have no strict stop arrival guidelines to follow and often deviate from the route to serve passenger requests, avoid undesirable traffic patterns, or otherwise get sidetracked. Additionally, drivers have no requirement to self-regulate; without a schedule there is no urgency to speed up after a traffic delay or, more commonly, slow down if they are driving too fast. This lack of schedule frequently results in undesirable shuttle clusters, where multiple buses serving the same route travel in packs around the route.

Transit management personnel often quickly radio route change requests to buses requesting they serve different routes after a break as the demand for service changes. To handle these route service changes, we dedicate significant time to identify what route a vehicle is currently serving in order to correctly select the set of stops it will be arriving at for the estimation algorithm.

1.2 Metrics

To judge the algorithms, we look at their overall ability to perform effectively and integrate well into our existing Ruby on Rails[3] Vehicle Tracking[1] project. Integration plays a particularly key role in this evaluation, as a tight integration into existing code bases will enhance existing open source vehicle tracking software. While these metrics may be bound by the performance constraints of the Ruby on Rails (and underlying) frameworks, we, for simplicities sake, are not comfortable providing a potentially higher performance but less well integrated approach. We evaluate three different route identification strategies in terms of accuracy, efficiency, and implementation.

1.2.1 Accuracy

Realizing the identification of routes is the task as hand, we looked for an alternative way to gather known data to test our methods against. Unable to find large quantities of available route and historic travel data for transit systems, we built a mechanism to leverage the power of the crowd to build a good "known" sample set using our existing data. To evaluate accuracy, we crowd sourced the identification of randomly selected tracking data points and asked users to identify the route of travel, building a data-set of 1000 known point-to-route match ups.

Each point to identify was presented on a map as the tail of a line containing the three previous points of travel. Users were to select the route the point matched or mark the point as unknown if the route could not be determined. To protect against inaccuracy, each point was examined by three different people, only points that all three people agreed on were used for final testing of the algorithms.

1.2.2 Efficiency

Each request passing through the web front-end of our application was logged with timing information in the application log. This information, similar to that in Figure 1.1, is mined using request-log-analyzer[1] to present statistics representing total, mean, standard deviation, min, max, and 95 percentile range. We look for an implementation that, after receiving 1000 requests using the ab[2] tool, per sample point maintained a low total mean response and a low value for the higher bound of the 95 percentile range, representing fast requests in the overall system instead of just database[3] or processing[4] time.

1.2.3 Implementation

The implementation of an approach is the hardest to quantitatively measure, but is key to developing an effective open source tool. An ideal implementation, aside from being well documented and easy to read, will translate easy into Ruby

[1] https://github.com/wvanbergen/request-log-analyzer, Date Last Accessed, 03/17/2011

[2] Apache HTTP server benchmarking tool - http://httpd.apache.org/docs/2.0/programs/ab.html, Date Last Accessed, 04/10/2011

[3] Shown as ActiveRecord in Figure 1.1.

[4] Shown as Views in Figure 1.1.

```
Started GET "/vehicles/current.js" for 128.113.137.39 at 2011-04-15 23:32:37 -0400
  Processing by VehiclesController#current as JS
Completed 200 OK in 5ms (Views: 0.1ms | ActiveRecord: 0.7ms)


Started GET "/vehicles/current.js" for 128.113.17.3 at 2011-04-15 23:32:38 -0400
  Processing by VehiclesController#current as JS
Completed 200 OK in 4ms (Views: 0.1ms | ActiveRecord: 0.7ms)
```

Figure 1.1: Sample of a Ruby on Rails log file.

code. Optimally, implementations will make use of libraries already in use like Arel[5] and ActiveRecord[6] and minimize external extensions and dependencies. Any caching approaches should use existing techniques provided by the Ruby on Rails framework, and all data access should occur though the Active Model accessors.

1.3 Bounding Box

The first route identification strategy explored builds a simple bounding rectangle around the route using the minimum and maximum latitude and longitude values across all points in the route. This box can be constructed by scanning across all points along a route and extracting the minimum and maximum values. After establishing this box, a sample of which is shown in Figure 1.2, we can compute (Appendix A.1) if a point is or is not a member of the route. This simple bounding box approach offers a very fast and cache-able (for non-changing routes) method to compute route membership. Unfortunately, this approach proved too simple and failed on a number of fronts.

First, this approach assumes vehicles are always traveling on the route and not running any special trips in the route region. For a large transit system this might make sense, buses are not very likely to deviate from a prescribed route, however for smaller scale transit organizations vehicles are much more likely to be dispatched on special purpose trips in the same region as a route. The less linear[7] the route the wider the margin of error is.

[5]https://github.com/rails/arel, Date Last Accessed, 02/25/2011

[6]http://api.rubyonrails.org/classes/ActiveRecord/Base.html, Date Last Accessed, 04/19/2011

[7]A straight-line or nearly linear route has a significantly lower margin of error than a route covering an area shaped with large turns. The worst case route for this system to handle would resemble the area covered by an 'L' shape.

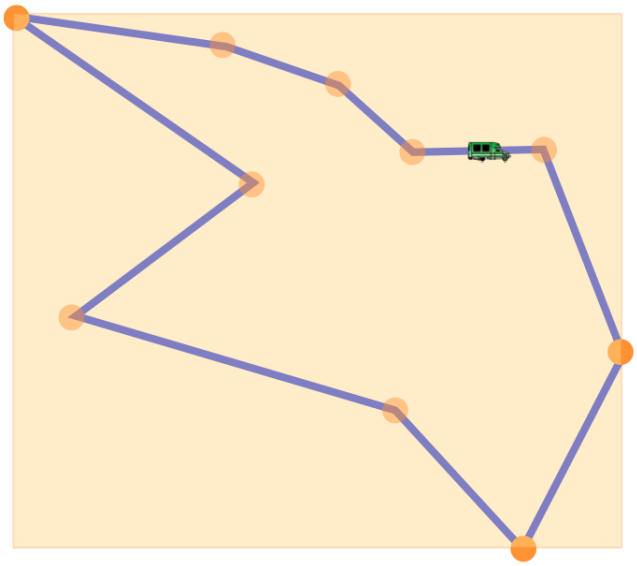


Figure 1.2: Bounding box construction for a sample route.

Overlapping route service areas also cause problems for the bounding box approach. The construction of a bounding box works great for geographically distinct routes but provides no strategy, like a membership score, to identify route membership on routes covering overlapping geographic regions. In the sample we consider at a college campus, both routes have substantial overlap servicing academic areas of campus. During this time, a simple implementation[8] of the bounding box approach is unable to identify which route a vehicle is actually traveling on.

1.4 Ray Casting

Realizing a vehicle in transit is unlikely be traveling exactly on the linear segments that compose a route we looked to provide a tolerance to test if the vehicle was traveling within a boundary built around route. To accomplish this we constructed polygons representing the route, a portion of which is shown in Figure 1.3, providing inner and outer border regions (making a donut shape) for the route. Using a ray casting approach we count (Appendix A.2.2) the number of intersections a vertical line extending up from the current vehicle has with the constructed polygon. An odd number of intersections, like the example in Figure 1.4, indicates the vehicle is

[8] A more advanced implementation could incorporate historic data into the bounding box approach, but this author opted to explore more advanced identification approaches.

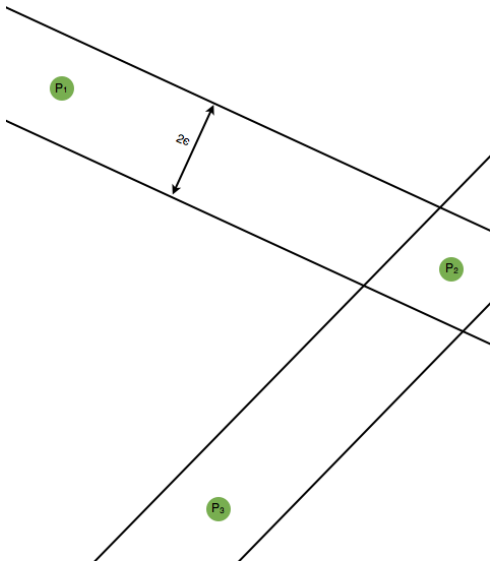


Figure 1.3: Linear construction for three points along a route using tolerance ε.

within the tolerance of the route.

The tolerance parameter ε enables the matching polygon to grow and shrink as needed. This parameter can be configured on a per-point basis to provide larger tolerance around areas where a large deviation is likely[9] and conversely restrict the matching region for a very strict portion of a route.

Ray casting provides an effective method to build geographic tolerances around a route, removing the large margin of error presented in the bounding box technique. Routes with overlapping paths still have ambiguous sections, but those sections are limited to the time the vehicle is currently in those portions[10]. This method, while more accurate, also presents a more challenging and less cache-able implementation.

1.5 Nearest Segment Identification

Ray casting provides an effective method to determine if a vehicle is within a polygon representation of a route, but requires the tolerance to be configured or known before any identification can be performed. Looking to replace this re-

[9]Possibly due to a large traffic structure like a multi-lane highway or a low quality signal region for the GPS transmitted where transmitted data is not very accurate.

[10]We can correctly say a vehicle is on route A or B using this method. Bounding Box would have only let us say route A, route B, or some region in between.

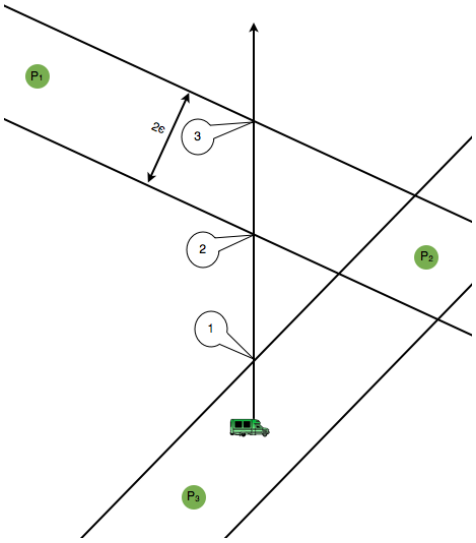


Figure 1.4: Demonstrating the vertical ray intersection with 3 linear equations, 1, 2, 3, confirming the vehicle is within ε of the route.

quirement, we developed an approach that identifies the nearest route segment by measuring the shortest distance, as shown in Figure 1.5, from the current vehicle's reported position to all possible linear segments of the route.

This technique, documented in Appendix A.3, effectively returns the deviation from the route (similar in concept to the tolerance) and enables a degree of membership to be computed. This degree of membership enables an easy ranking of routes to develop how likely the vehicle is to be on a certain route given it's distance from the nearest segment. For ambiguous cases, where a vehicle is equi-distant between two separate routes we have the option of looking at additional route segments before turning to historical data and computing the previous position's degree of membership.

We can optimize this technique for large routes with many segments by reducing the number of comparisons we make with route segments. Using a multi-level tree approach, we divide the large route into smaller route subsets. By performing the deviation calculation on the route subsets we can find the closest subsection of a route and further drill down our analysis to the individual segments or additional subsets as needed. In practice, this approach proved unnecessary given our

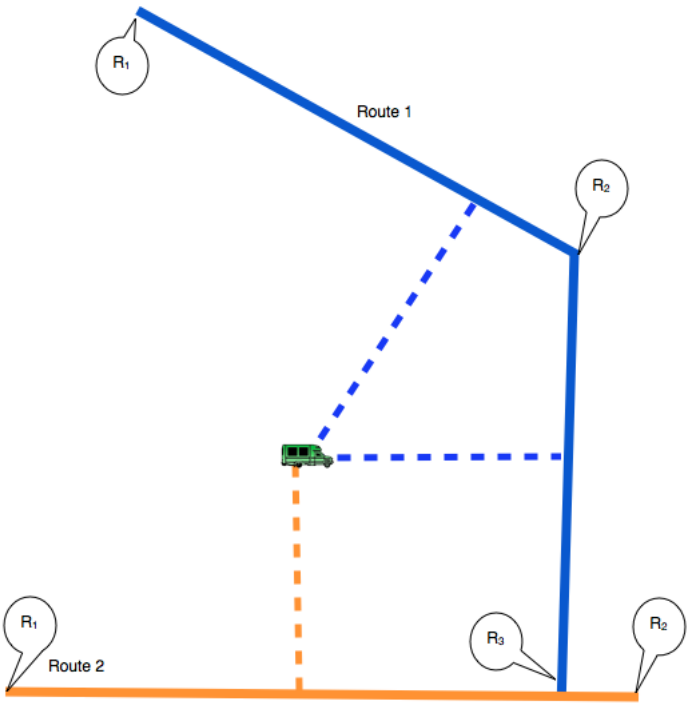


Figure 1.5: Identification of the nearest route segment, $R_2 \rightarrow R_3$, for a vehicle currently near three different segments.

small / simple route size, but given complex route data[11] this approach can reduce computation time by targeting the analysis on the most likely route section.

1.6 Results

Each of the three previously mentioned techniques were implemented in our testing machine running MRI Ruby 1.9.2[12] and Ruby on Rails 3.0.3. The tests were conducted by sending 10000[13] HTTP requests to a view displaying point information including associated route the algorithm identified.

The bounding box delivered the fastest results of any method tested, but also proved the least accurate. While the simple bounding approach was able to quickly classify points as a member of a route it misidentified 100% of the off-route

[11]This technique may prove particularly useful for routes with lots of small segments simulating an arc. The high precision required to construct these shapes lend well to subset analysis.
[12]Ruby 1.9.2p0 (2010-08-18 revision 29036)
[13]10 requests each for the 1000 sample points.

points, and was unable to disambiguate a substantial portion of the test data in geographically overlapping routes. The simple implementation of this technique did not outweigh it's significant failures.

Ray casting required additional time to configure and determine the most effective tolerance for testing. Initially, we selected a value close to the width of roads[14] under the assumption that most vehicles would be reporting a position on or near a road. After several rounds of testing with different tolerance values we found a tolerance eight times the initial road width[15] provided an acceptable accuracy. The eight-fold difference between our tolerance and the road difference was likely needed to compensate for mistakes in our route construction, which was hand-built using Google Maps[2], or inaccuracies in the GPS data transmitted by the vehicles.

Finally, the nearest segment identification balanced both a fast implementation and accurate results. Using the testing data, we were able to identify an acceptable deviance that make the most sense for our sample data. To choose the best maximum deviation, we tested slow changes in the acceptable deviation against with the sample data to maximize our correct identifications. With a known set of sample data to test against, prior training to identify the nearest segment deviation value that best matches the specific deviations a transit system can offer the highest accuracy without substantially effecting efficiency[16].

[14] 50 feet

[15] 400 feet

[16] An attempt to calculate the maximum acceptable deviation in real time proved inefficient and generally lead to the algorithm recursing out of control

CHAPTER 2
Arrival Time Estimation

2.1 Introduction and Historical Review

While identifying a vehicle's route is beneficial to users, we consider that problem a necessary first step to the larger estimation problem. We aim to predict when a shuttle will arrive at a specific stop; to do so we leverage our knowledge of the defined route path, extensive vehicle history, and current positioning information for the vehicle. Presenting this prediction to users, and making it available to developers via an HTTP API, provides a valuable tool to help counteract some of assumed unpredictability non-scheduled transit systems present.

By presenting the arrival time estimation to transit passengers, we hope to quickly provide enough information to make a "walk" or "wait" decision enabling a passenger to decide if the vehicle is worth the wait or if walking will provide a viable alternative. Additionally, this information can be used to generate a very rough timetable for arrival at a limited number of future stops which may have benefits for passengers riding on larger unscheduled transit systems. To help educate people with the necessary information to make the walk/wait decision, we attempt to include some accuracy or deviance score to quickly convey our confidence in the prediction to users.

2.2 Testing Strategy

To test our arrival time estimations, we turned to our data-set of over 700,000 shuttle positions recorded over a six month period from September 2010 to March 2011. This data set represents the continuous production data for a college shuttle service operating Monday through Friday from 7am to 11pm[4]. Using this data, we can replay historical days and search for a shuttle's arrival at a stop. With that data in hand, we can rewind our data set to an earlier time using Timecop[17] and

[17]https://github.com/jtrupiano/timecop, Date Last Accessed, 09/17/2010

	10ft	50ft	100ft	200ft	300ft	400ft
Student Union	331	584	13120	52008	72710	85429
BARH	64	5290	11624	23912	39699	51441
Sunset Terrace	289	4090	9252	19094	26714	34438
Beman Lane	26	2840	7351	13672	21692	39647
Colonie	598	3935	7339	13706	19771	25342
9th and Sage	238	3159	7545	15990	24753	44215
West Hall	171	3502	8236	18525	28393	37269
Sage	125	4052	9806	20294	30255	42925
Brinsmade	114	3250	6712	13813	23145	35317
Footbridge	107	1655	3576	8723	11908	27510
Polytech	109	1811	4086	7749	10295	12311
Blitman	33	2129	4189	6725	9062	11167
15th and College	103	970	2246	4865	7442	9858

Table 2.1: Arrivals at each stop given by threshold.

test our prediction algorithm comparing the predicted results to the known arrival time.

Valid arrivals were identified by searching for historical updates that logged a location within 100 feet of a defined stop. The 100 feet threshold provides over 95000 stop arrivals across the system, representing 12% of the total data set. Increasing the threshold to larger values, shown in Table 2.1, started to drastically reduce the non-stop sample set.

2.3 Proportional Prediction

Using the nearest segment identification method from Section 1.5, we compute a rough linear estimation of the vehicle to a destination stop. We assume, knowing the route of travel and the closest segment, the vehicle is in progress of traveling on the route segment it is closest to. From this, we add up the distance traveled from each subsequent route segment to the destination stop. Needing to account for the current route segment in progress, we add the distance from the vehicle's current position to the end of the nearest segment. While the vehicle is unlikely to actually travel in in a straight line to get back onto the route at the nearest point this distance represents a much closer approximation than traveling in a straight

line to the closest point on the route (mid segment) and then resuming the rest of that segment. Additionally, the use of nearest segment identification supports our belief that any deviation from the route is likely to be minimized. Knowing the total distance the vehicle has to travel in order to reach the stop, we can simply divide the distance by the vehicle's current speed to gain a very rough approximation of its arrival.

This technique assumes a constant speed and works well for predicting the arrival at very close stops. Close stops are likely to be reached by the vehicle while it's traveling a near constant speed, the speed of travel for stops further away is much less likely to be constant and as a result the accuracy of this simple measure declines. By combing this simple metric with additional prediction techniques we can improve accuracy for non short term predictions and maintain the accuracy for near-stop arrival estimations

2.3.1 Integral Feedback

To help smooth the sudden jumps in the simple proportional prediction resulting from the discontinuous flow of data[18] we added an integral-style component. Borrowing the technique from PID[5] control systems, we examine a previous number of predictions, subtract the time since they were made from their estimation[19] and factor them in declining weight[20] to our current prediction.

In our trials, varied the number of previous results averaged in with the simple proportional prediction and also varied the weight at which each trial was counted. Based on our simulations, we found including more than 5 previous predictions into the system did not dramatically increase accuracy at all. Furthermore, including more than 10 previous predictions actually decreased the accuracy of the system. We attribute this to the outdated of old predictions that, despite being weighted so low, still were able to still effect the proportional prediction when pooled. An effective balance seemed to be reached by including 3-5 previous predictions in addition to the current proportional prediction.

[18] Our vehicles are suppose to report data every $\frac{1}{4}$ mile they travel.
[19] A prediction of 5 minutes made 2 minutes ago now becomes 3 minutes.
[20] The most recent prediction is more valuable than the one 5-10 predictions ago.

2.4 Historic Prediction

Having a large historical data set enables us to mine past data to not only test our predictions but also enhance current predictions. Applying the reverse strategy to the testing methodology described in Section 2.2, we search historical data for records with a similar location to the present vehicle's location and scan forward in that data to calculate how long it takes a vehicle to arrive at a given stop. We can apply similar weighing techniques to those described in the previous *Integral Feedback* section and weight most recent historical predictions higher than older ones.

Additionally, we found accuracy in the historical predictions could be increased by weighting also to match time of day and day of week, though these records are often only minutely boosted due to their outdated nature. These methods slightly increase the value of certainly historical points, but do not significantly outweigh the most recent data collected in the previous few hours.

2.4.1 Discrete Data Challenges

While theoretically sound, much of the historical arrival data struggles due to data inconsistencies. Our GPS hardware does not consistently transmit data uniformly while a vehicle is in motion. As a result, there are many instances where it takes several loops around the route for a start position and stop position to match up properly. To handle these cases, we introduced a cutoff threshold that prevents historical data from being included in calculations if the historically calculated arrival is more than double the integral feedback prediction. This cutoff successfully disregards inaccurate historical records when routes suddenly change slightly, perhaps adjusting to an inclement weather route, and when weighted in with integral and proportional components, can help fine tune to the arrival prediction.

CHAPTER 3
Presentation and Display

Initially, presentation of the arrival time prediction was considered a trivial task, one that would be quickly implemented and passed over as a small detail in the program. We, however, realized there are significant differences in interpretation of the estimation based on the presentation and formatting of this data. While our user interactions studies are by no means exhaustive and complete, we explore the two options considered and their various ramifications.

3.1 Arrival Countdown

The first method initially used in mockups like Figure 3.1 indicated the shuttle would be arriving in X minutes. When making the previously mentioned walk / wait decision, this number helps potential riders quickly decide how long they are willing to wait for an arrival. Most people can use this number to quickly estimate how long an alternate method, such as walking, might take them and base their decision off that quick mental comparison.

This display method, in the author's eyes, is more strongly associated with a countdown in most people's minds. At the next page load, they expect the arrival time to decrease by approximately how long they had been waiting on the page. The estimation algorithm does factor this adjustment into it's integral step, but only so much as to make sure all the previous estimates have been decremented appropriately. These steps attempt to reduce sudden jumps in the arrival time, but there is no way to falsify the prediction to make the number decline at a more

Figure 3.1: Displaying an arrival countdown for a vehicle approaching a stop serving two routes.

Figure 3.2: Displaying an arrival time for a vehicle approaching a stop.

gradual or expected pace.

3.2 Arrival Clock

Adding the estimated time to arrival to the current time, we can display an clock-based estimate of when the next vehicle is expected to arrive at the stop. This method, shown in Figure 3.2, does not enable potential riders to quickly determine how far away the vehicle is without knowing the current time, but does provide a display much more akin to the scheduled display most transit systems use. Displaying the arrival time on a clock slightly obfuscates the arrival adjustments as a result of a delay or speed up in the transit system, as changes in the time aren't as noticeable as adjustments in an integer like the countdown method.

The use of a clock display does not require constant updating like the countdown does, after 1-2 minutes the countdown display will be no longer but the clock may still be. As a result, we opt to use a clock display in many of our displays where constant updating is not an option. We retain a countdown implementation exposed via an API for interested developers.

CHAPTER 4
Discussion and Conclusions

Extensive testing was conducted using the historical data set recording positions from six months at a small scale university transit system. Additional feedback, perhaps via a thumbs up/down voting system exposed to users could provide additional clues to improve the system's accuracy; due to time constraints to gather a substantially large sample size, data to this effect was not available at the time of writing. In addition to the work in arrival prediction, many of the tools and concepts developed here have can be used auditing transit systems to, among other things, verify accurate route markup, stop placement, or vehicle distribution.

We successfully identified, implemented, and tested three different techniques for identifying a current route of travel given an expected route path and a current vehicle's position. Using the *Nearest Segment Identification* approach, we are able to correctly identify the route a vehicle is traveling on and select the correct set of stops for estimation purposes. Additionally, this identification information can be presented to users, in our instance via colour coding, to disambiguate the route of travel during segments of overlapping routes.

Using the route identification to select a subset of stops, we expand on a simple proportional technique borrowing a Proportional-Integral control style technique which can be combined with mining thousands of historical data points to estimate a vehicle's arrival at future stops. We present this data to users though an *Arrival Clock* displaying the time when the next vehicle will be arriving. This estimation data is also made available to developers, like those developing the Mobile Shuttle Tracker[21] iPhone and Android application, for inclusion in both arrival time and arrival countdown formats.

[21] http://rcos.rpi.edu/projects/mobile-shuttle-tracker/, Date Last Accessed, 04/20/2011

BIBLIOGRAPHY

[1] August Fietkau, Reilly Hamilton, Brian Michalski, and Zach Rowe. Vehicle Tracking. `https://github.com/wtg/shuttle_tracking`, Dec 2010. Date Last Accessed: April 03 2011.

[2] Google. Google Maps. `http://maps.google.com`, August 2010. Date Last Accessed: April 14 2011.

[3] David Heinemeier Hansson. Ruby on Rails. `http://www.rubyonrails.org`, Nov 2010. Date Last Accessed: April 18 2011.

[4] Jason Jones. RPI Red Hawk Shuttle Service. `http://www.rpi.edu/dept/parking/shuttle.html`, April 2011. Date Last Accessed: April 18 2011.

[5] Nicolas Minorsky. Directional stability of automatically steered bodies. J. Amer. Soc. Naval Eng., 1922. 34 (2): 280 - 309.

APPENDIX A
Route Identification

A.1 Bounding Box

To computer a point P's membership in route R, we compare the latitude (lat) and longitude (long) values for the point and route minimum and maximum values.

$$\{P \in R : (\min R_{latitude} \leq P_{latitude} \leq \max R_{latitude}) \wedge (\min R_{longitude} \leq P_{longitude} \leq \max R_{longitude}) \tag{A.1}$$

A.2 Ray Casting

A.2.1 Ray Construction

We construct lines bounding the route with a given tolerance ε. For this example ε is uniform across the entire route though this value could be customized on a per route point basis to account for large traffic structures.

$$\Delta X = P_{i+1_{latitude}} - P_{i_{latitude}}. \tag{A.2}$$

$$\Delta Y = P_{i+1_{longitude}} - P_{i_{longitude}}. \tag{A.3}$$

$$\theta = \arctan \frac{\Delta Y}{\Delta X}. \tag{A.4}$$

$$Y = \frac{\Delta Y}{\Delta X}(X - (P_{i_{latitude}} \pm \varepsilon \cos\theta)) - (P_{i_{longitude}} \pm \varepsilon \cos\theta) \tag{A.5}$$

A.2.2 Intersection Detection

for all *point* in route R points **do**
  Construct linear equation f as perscribed in equation A.5
  Compute $f(vehicle\, point_{latitute})$
  if $f(vehicle\, point_{latitute}) \geq vehicle\, point_{longitude}$ **then**
    ray intersections++
  end if

end for

$$\{P \in R : \sum (ray\ intersections)\ is\ odd\} \tag{A.6}$$

A.3 Nearest Segment Identification

We compute the nearest route segment by measuring the shortest distance from the current point P to each route segment designated with points R_i and R_{i+1}. Recognize the small-scale nature of transit systems likly to use this system, we opt not to account for the curvature of the earth in our calculations.

$$\begin{aligned} S = & \frac{(P_{latitude} - R_{i_{latitude}})(R_{i+1_{latitude}} - R_{i_{latitude}})}{\|R_{i+1} - R_i\|^2} \\ + & \frac{(P_{longitude} - R_{i_{longitude}})(R_{i+1_{longitude}} - R_{i_{longitude}})}{\|R_{i+1} - R_i\|^2} \end{aligned} \tag{A.7}$$

$$\Delta X = |R_{i_{latitude}} + S(R_{i_{latitude}} - R_{i+1_{latitude}}) - P_{latitude}| \tag{A.8}$$

$$\Delta Y = |R_{i_{longitude}} + S(R_{i_{longitude}} - R_{i+1_{longitude}}) - P_{longitude}| \tag{A.9}$$

$$deviation = \sqrt{\Delta X^2 + \Delta Y^2} \tag{A.10}$$

APPENDIX B
Arrival Time Estimation

B.1 Proportional Prediction

Computing the simple porportional prediction is done by identifying the closest route segment (See Appendix A.3), summing the distance to the stop, and dividing by the current rate of travel as described below. The values below assume calculations near latitude 42, the specific mile conversion multipliers (70 and 51) may change slightly.

R_{next} = end of nearest route segment

D_{next} = distance to nearest route segment

$distance = D_{next}$

for all *point* in route R from R_{next} to stop S **do**

$d = \sqrt{(\Delta point_{latitude} * 69.0175)^2 + (\Delta point_{longitude} * 51.4809)^2}$

$distance \leftarrow distance + d$

end for

$Arrival = \frac{distance}{V_{current}}$

B.1.1 Integral Feedback

$$w = Weight\ of\ arrival\ times\ e.g.\ (0.25, 0.15, 0.10) \tag{B.1}$$

$$Integral\ Arrival = \int_{n=0}^{i} w_n * Stop\ Arrival_n \tag{B.2}$$

B.2 Historical Records

$w = Weight\ of\ arrival\ times\ e.g.\ (0.25, 0.15, 0.10)$

Find a historical position P_h within n distance of the current position $P_{current}$.

while $(P_{next_{time}} - P_{current_{time}}) \leq (2 * integral\ feedback\ estimate)$ **do**

if P_{next} is within n distance of the desired stop S **then**

$S_{estimate} = P_{next_{time}} - P_{h_{time}}$

else

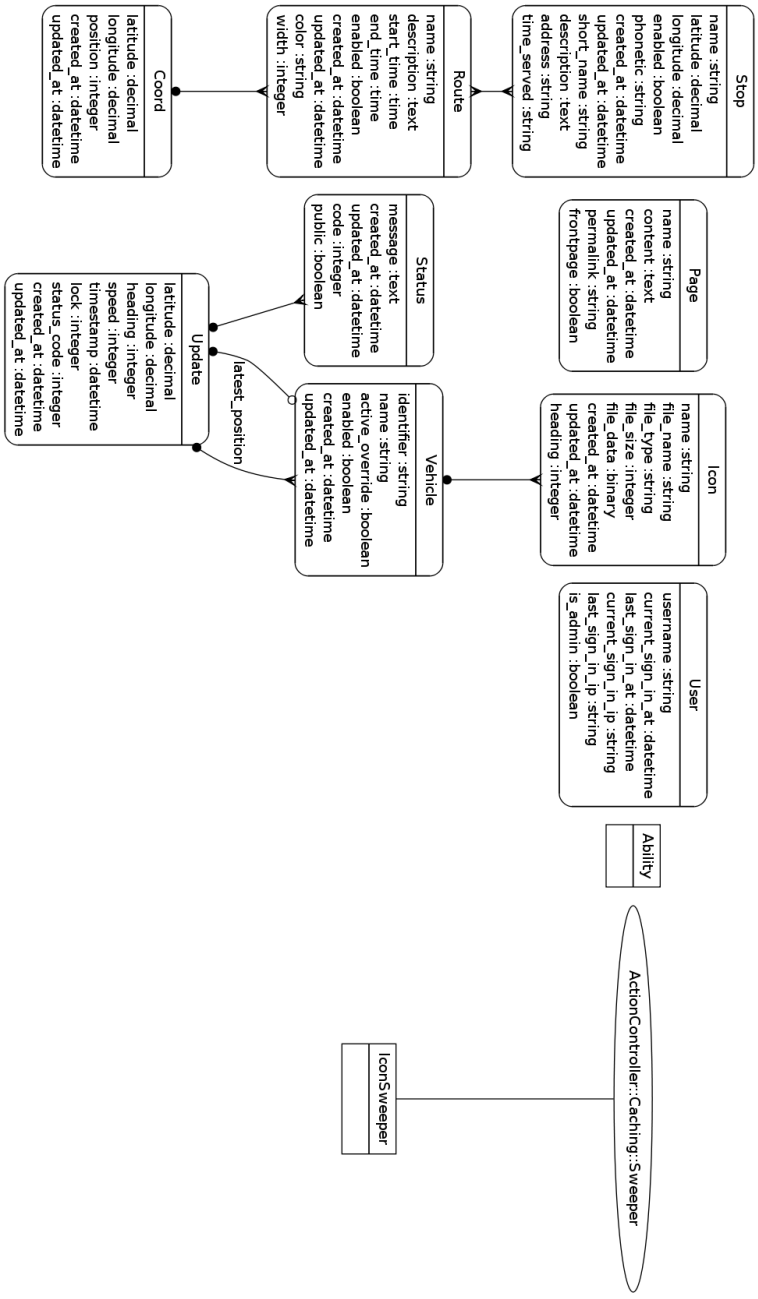
$P_{next} \leftarrow$ next position following P_{next}

end if

end while

APPENDIX C
Model Diagram

C.1 Ruby on Rails Models



APPENDIX D
Included Code

We include several code references when appropriate to present approach samples. For all code, including documentation and sample data, see `https://github.com/bamnet`.

D.1 Route Identification

D.1.1 Bounding Box

```
# AREL implementation (ORM-agnostic)
s = Stop.arel_table
u = Update.arel_table
Update.where((u[:latitude].lt(s[:latitude].maximum)).and(u[:
    latitude].gt(s[:latitude].minimum)).and(u[:longitude].lt(
    s[:longitude].maximum)).and(u[:longitude].gt(s[:longitude
    ].minimum)))

# ActiveRecord Optomized (cache-friendly)
r = Route.first #As an example, test against the first route
Update.where(:latitude => r.stops.minimum(:latitude)..r.
    stops.maximum(:latitude))
```

D.1.2 Nearest Segment Distance

```
Route.coords.each do |coord|
  p1 = coord # So we don't have to type so much
  p2 = coord.next # The next point along the route

  # Figure out the equation of a line
  # in the form aX + bY + c = 0
  a = p2.longitude - p1.longitude
```

```
  b = p2.latitude - p1.latitude
  c = p1.latitude*p2.longitude - p2.latitude*p1.longitude

  # Compute the distance from current pont (cur) to the line
  distance = (a*cur.latitude + b*cur.longitude + c).abs /
     Math.sqrt(a**2 + b**2)
end
```

D.2 Arrival Time Estimation

D.2.1 Proportional

```
segments.sort! do |s1, s2|
  s1.distance_to(cur) <=> s2.distance_to(cur)
end
distance = segments.first.distance_to(cur)
segments_to_stop(segments.first, stop).each do |segment|
  distance += segment.distance
end
return distance / cur.speed
```

D.2.2 Integral

```
estimates = Estimates.order('created_at').limit(n)
estimates.each_with_index |estimate, i|
  estimate.arrival_time -= (Time.now - estimate.update_at)
  estimate.save!
  integral_estimate += estimate.arrival_time * weight(i)
end
return integral_estimate
```

D.2.3 Historical

```
u = Updates.arel_table
updates = Update.where((u[:latitude].lt(cur.latitude+
    threshold)).and (u[:latitude].gt(cur.latitude-threshold))
    .and (u[:longitude].lt(cur.longitude+threshold)).and (u[:
    longitude].gt(cur.longitude-threshold))). order(:
    created_at). limit(n)
updates.each do |update|
  stop = next_stop(update)
  next_update_near_stop = (update, stop)
  if (next_update_near_stop.timestamp - update.timestamp) <
     (integral_feedback * 2)
    weight = weight(Time.now - update.timestamp)
    historic_prediction = (next_update_near_stop.timestamp -
        update.timestamp) * weight
  end
end
```