

# Dynamic Management of Real-Time Location Data on GPS-Enabled Mobile Phones

Sean Barbeau, Miguel A. Labrador, Alfredo Perez, Philip Winters, Nevine Georggi, David Aguilar, and Rafael Perez  
Department of Computer Science and Engineering and Center for Urban Transportation Research  
University of South Florida  
Tampa, Florida 33620  
{barbeau,winters,georggi}@cutr.usf.edu, {labrador,perez,aguilar,perez}@cse.usf.edu

**Abstract**—Advanced location based services have the ability to track users with Global Positioning System (GPS)-enabled cell phones in real-time. These applications share a common problem; the continuous calculation and transmission of GPS fixes from the mobile phone to a server consumes a considerable amount of energy and increases data transmission costs. Therefore, an application-level algorithm is necessary to reduce the number of GPS fixes calculated and transmitted, while continuing to track the user in real-time and record an accurate representation of his or her travel path. In this paper, two complementary algorithms are presented: the Critical Point (CP) algorithm, which filters data points to be transmitted to the server, and the location-aware state machine, which dynamically manages the frequency of the location re-calculation update rate. Both algorithms were implemented in TRAC-IT, a Java Micro Edition (Java ME) application designed to automatically collect user travel behavior; the proposed algorithms allow TRAC-IT to build an accurate representation of the user's path with a considerably reduced number of fixes while significantly extending mobile device battery life.

## I. INTRODUCTION

In today's fast-paced world, the transportation capacity of urban environments serves as an integral function in modern society. Many transportation infrastructures are stretched to their limits as an increasing number of people travel on highways every day. The Transportation Demand Management (TDM) industry seeks to provide a solution to congested thoroughfares by using existing roads more efficiently through the adjustment of commuter travel behavior. This complementary approach to building new roads has the added advantages of reducing congestion, air pollution, and fossil fuel consumption as well as providing mobility solutions for non-drivers.

To provide effective solutions for increased demand, TDM strategies require an in-depth understanding of the local area's transportation patterns. In the past, transportation experts have relied on self-reporting paper and/or phone surveys to analyze trip characteristics including start and end times, duration, distance, origin, destination, purpose, mode, etc. However, these types of surveys have several intrinsic problems. First, the amount of time and effort required to complete an accurate travel survey is significant. As a result, recruiting participants is challenging; often the length of the study must be limited to one or two days to avoid placing an undue burden on respondents. Second, the desired level of detail and accu-

racy are impeded by self-reporting user errors, apathy, and intentional or unintentional omissions, particularly on short trips [1]. Finally, once the surveys are collected, they must be manually post-processed, thus requiring a significant amount of time and effort.

In recent years, modern computing devices including Global Positioning System (GPS)-enabled mobile phones have been evaluated as possible replacements for paper and phone surveys [1]–[4]. GPS-enabled mobile phones can be carried by the user whenever and wherever he or she travels, and provide the opportunity for recording an individual's transportation behavior for any mode of transportation, including travel via public transit, or non-motorized modes such as walking or biking [5]. The objective nature of GPS data, combined with the automated data collection process, can enhance the quality and quantity of collected data. Mobile phones are also capable of transferring GPS data to a central database immediately upon collection, which allows for extended deployment of the survey. Real-time data connectivity also introduces new services to the traveler such as highly targeted traffic alerts based on the user's real-time location and predicted destination. These services help to reduce traffic congestion while providing the user an incentive to allow their travel behavior to be monitored.

One of the most important travel characteristics obtained from GPS data is the user's route (i.e., the path that he or she takes from source to destination). Although route information can be easily obtained without the user's intervention by continuously calculating and sending GPS fixes from the cell phone to a server, this method will quickly drain energy from the cell phone's battery. When tracking for an entire day is desired, power consumption becomes a significant concern that can render a mobile phone inoperable if resources are not managed properly by the application [6]. Furthermore, frequent transmission of unnecessary information not only increases the cost of the user's phone bill, but also utilizes additional network resources. To solve these problems while retaining the ability to continue tracking the user and reconstructing the user's path, new dynamic application-level location management algorithms are required.

In this paper, two complementary algorithms are presented: the Critical Point (CP) algorithm, which is designed to reduce

the number of GPS fix transmissions to a server for tracking applications that provide both real-time and archival recording features, and the location-aware state machine, which dynamically adjusts the position recalculation rate on board the mobile device in order to save energy when frequent position re-calculations are not required. Using components of the architecture described in A General Architecture in Support of Interactive, Multimedia, Location-based Mobile Applications [7], the CP and location-aware state machine algorithms were implemented in TRAC-IT [4], a Java Micro Edition (Java ME) application for GPS-enabled mobile phones that automatically collects travel behavior data. Data resulting from the CP algorithm execution are presented to show the reduced financial and network costs provided by the algorithm. The energy cost savings of the CP algorithm is demonstrated by the results of a battery life benchmarking application in which fixes are periodically transmitted to a server at a fixed update rate until the device battery is exhausted. A similar application that uses fixed position recalculation frequencies is used to illustrate the potential energy savings that can be gained by using the location-aware state machine.

This paper is organized as follows: Section II briefly summarizes existing methods utilized for position updates. Section III describes the Critical Point algorithm in detail, and Section IV discusses the location-aware state machine. Section V includes the evaluation of the algorithms, and finally Section VI concludes the paper.

## II. EXISTING UPDATE METHODS

The concern of this paper is primarily with terminal-based location methods in which the mobile device is the primary position-calculating entity. As such, the device location must be sent to a server to update the system with real-time position information [8].

Many methods for sending mobile device positioning updates to a server have been discussed in existing literature [9]–[12]. In polling, a server pulls the position from the mobile device on a periodic basis, or as requested by the server-side location based application. Polling is useful for occasional position updates such as displaying the current location of devices on a map, but is not efficient for real-time applications with on-board intelligence that relies upon real-time position information, or for detailed route recording. Periodic updates are sent from the mobile device to the server after a fixed interval of time elapses. While it is one of the most commonly used update methods, a significant amount of unnecessary data can be sent to the server when small interval values are used. Additionally, the specific fixed interval must be customized per application. While large time intervals are more efficient, they do not meet the needs of real-time or archival applications. The mobile device can also send position updates based on zones and distance.

In zone-based methods, the mobile device sends GPS fixes when it enters or leaves a particular geographic zone. Distance-based methods trigger a position update after the mobile phone has exceeded a distance threshold. These two methods must

also be customized with a set interval per application, and tend to send more GPS fixes than necessary. Distance-based methods also send unnecessary location updates when the user is traveling in a straight line. Dead reckoning is another method that determines whether to send a new GPS fix based on the most recent location data that were sent to the server, and an estimation function executed simultaneously at the device and the server. For example, if the device detects that its position deviates from what is expected when the estimation function is executed with the most recent server data, it then sends the new position to the server. While this method results in fewer transmissions to the server, it requires the continuous execution of potentially costly estimation functions on both the mobile device and the server. For significantly resource-constrained devices such as those that meet the qualifications of the Java ME Connected Limited Device Configuration (CLDC) [13], these estimation functions may consume significant resources and may not be feasible for real-time operation.

For tracking applications that run in the background on a mobile phone for an extended period of time, none of the above-described positioning update schemes alone provide an efficient means to deliver both real-time Location-Based Services (LBS), as well as an accurate record of the user's travel path within the same application. To achieve highly efficient LBS, an algorithm that combines aspects of several update methods and dynamically adjusts at run-time is required. The Critical Point algorithm presented in Section III can be seen as such a hybrid algorithm. In addition, the device position recalculation rate, which can differ from the location update rate to the server, is also of great importance. All of the update methods described above, with the exception of polling, assume fixed position recalculation rates by the mobile device. Frequent position recalculations will yield high accuracy at the cost of significant energy [14]. Alternately, infrequent position recalculations will increase battery life, but are unable to accurately report real-time device location or the device's travel path while the user is traveling. Therefore, an algorithm is required that intelligently manages the GPS fix request rate to consume as little energy as possible based on real-time application-level needs. The location-aware state machine discussed in Section IV is such an algorithm.

## III. THE CRITICAL POINT ALGORITHM

Critical Point (CP) algorithm filters non-critical position data out of a continuous stream of real-time location data. Location data points are defined as a set of measurements containing latitude, longitude, and time-stamp information that represents the location of a mobile device at a particular moment in time. Data points may also include other useful information such as altitude, estimated accuracy uncertainty, speed, and heading. Non-critical data points are redundant or useless data that do not contribute to the knowledge of the device path. The goal of the CP algorithm is to reduce battery consumption (i.e., energy costs to the device), bandwidth consumed (i.e., cost to the network), and number of bytes

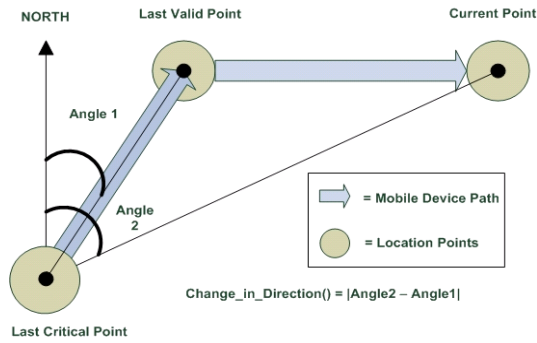


Fig. 1. Calculating changes in direction.

transferred (i.e., cost to the user) by transferring only critical points between entities in a LBS system.

The CP algorithm essentially reduces a stream of location data points into a series of connected lines. The vertices joining these lines (i.e., critical points) are then transferred to the server in real-time. In other words, points along the line are discarded since they do not contribute additional path information. Since the simplest path is a straight line, a path will always have at least two critical points, the starting and ending points. Non-critical points can lie directly between two critical points so that if a line was drawn between the two critical points, it would nearly intersect the non-critical points between them. Non-critical points can also include information gathered while a device is standing still (i.e., redundant location data), or data from fallback positioning technologies such as Cell ID (i.e., the center point location area of current cellular coverage) when GPS is not available. Since positioning systems always include some uncertainty in measurements, there must be a filter to remove location data points that are close in proximity but contain the same basic position information.

Changes in direction, measured by difference in azimuth values, are used to distinguish a critical point (Figure 1). If a device is traveling in a straight line and changes direction, a new critical point will be recorded when the change in direction surpasses a defined threshold. A speed threshold is used to determine the appropriate minimum speed that should be used to filter out data when the device is not moving. Additional speed thresholds are used to approximate the current mode of transportation, which are used to dynamically adjust the azimuth thresholds. A large azimuth threshold value may be useful to optimize the algorithm for walking speeds (i.e., less than five meters per second), and a smaller azimuth threshold may be useful for vehicle speeds (i.e., greater than five meters per second).

The CP algorithm can be viewed as a real-time, dynamic variation of static line simplification algorithms as presented in the field of Geographic Information Systems (GIS) [15]. Specifically, the CP algorithm is similar in concept to the perpendicular distance routine, a local processing routine that has an order of  $n$  time complexity, and therefore can be adapted to run in real-time on board a mobile device with

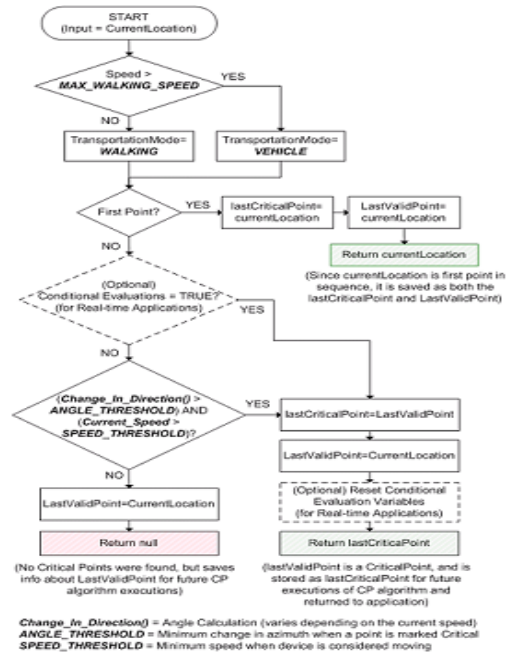


Fig. 2. The critical point algorithm.

little impact on device resources. The CP algorithm differs from the perpendicular distance routine in that it utilizes angle measurement values instead of the distance of a point from a line to detect critical points. The CP algorithm also runs in real-time, processes data from multiple positioning technologies (including assisted GPS), and is capable of dynamically adjusting the angle threshold values and filtering non-critical points based on real-time velocity measurements. The CP algorithm is executed each time a new position is calculated on board a mobile device (Figure 2). The algorithm is stateful, and therefore retains memory of past input to determine whether or not a critical point should be returned to the application. Since the algorithm is stateful, the input location may not necessarily be the same as the critical point location most frequently returned to the application.

#### A. Sample execution

An LBS application starts and begins to calculate its position at a fixed interval of every 4 seconds. The CP algorithm executes every time a new position is calculated, beginning with the first fix. The first fix is determined critical, and information about this fix is then saved within the algorithm and sent to the server by the application. The device then calculates a new position four seconds later, and the data are entered in the CP algorithm by the application. The algorithm generates a null output, and no data are sent to the server as it does not have enough information to determine whether this second point is a critical point. However, it saves information about this fix for future calculations which may determine that the point is critical. A third fix is then calculated and entered into the algorithm. If the difference in azimuth between the

first and second fix and the first and third fix (Figure 1) exceeds an angle threshold value, and the speed value for the third fix exceeds a speed threshold (i.e., the device is not stationary), the second fix is then established as a critical point and is returned by the algorithm. Information about the second and third fixes is then saved for future calculations. The application then sends the second fix to the server. If the angle or speed thresholds are not exceeded, the algorithm saves information about the second and third fixes, and generates a null result. In this case, no fixes are sent to the server. A fourth fix is then calculated by the device. If the difference in azimuth between the second and third fix and the second and fourth fix exceeds the angle threshold value, and the speed value for the fourth fix exceeds the speed threshold, then the third fix is established as a critical point, returned by the algorithm, and sent to the server. Information about the third and fourth fixes is saved for future critical point calculations. If the thresholds are not exceeded, the application saves the information about the third and fourth fixes and returns a null result. Again, no fixes are sent to the server. This process continues until the final fix is calculated and sent to the server as the final vertex in the created line.

The CP algorithm can be used along with the location-aware state machine algorithm (Section IV) that varies the location recalculation interval based on a condition such as the success or failure of a fix attempt, or a criterion such as the value of estimated accuracy uncertainty. This algorithm further increases the possibility of increasing the typical battery life of a mobile device running location-aware software.

### B. Conditional evaluations

When the CP algorithm is used in real-time LBS applications, several optional conditional evaluations can be activated by the mobile application. This allows the CP algorithm to meet real-time constraints while maintaining the more efficient model when real-time features are not active (e.g., the user is inside a building and is not actively traveling). For example:

- **Has the Timer Exceeded the Threshold?** A timer can also be added so that a point is established as critical and sent to the server after a certain amount of time elapses. This ensures that a position is reported at a minimum given interval, in case the device is stationary for long periods of time or traveling in a straight line for an extended period of time. For example, after 30 seconds, if a critical point has not been determined, then the next point is considered a critical point. This can be viewed as a dynamic transition to a periodic location update method.
- **Has the Distance Counter Exceeded the Threshold?** A distance counter can also be established which starts after a critical point is found. While the device is traveling in a straight line, distance is increased upon each position update. Once the device exceeds a distance threshold, it declares the next point to be critical and sends this point to the server. This method ensures that the server will receive frequent position updates for a device, even if it is traveling in a straight line for an extended period of

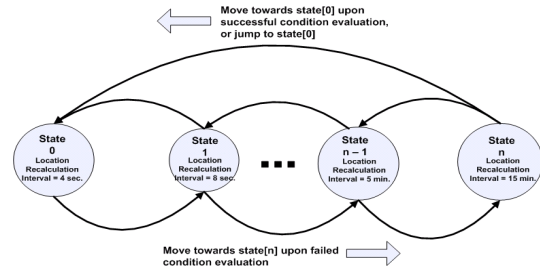


Fig. 3. The location-aware state machine.

time. This can be viewed as a dynamic transition to the distance-based location update method.

- **Received Location Probe?** The CP application can also have an optional section that if probed by a server, the next point is automatically determined to be a critical point and sent to the server. This can be viewed as a dynamic transition to a polling location-update method.

## IV. THE LOCATION-AWARE STATE MACHINE

Another common problem found in LBS applications is the continuous querying of the positioning system when the system is temporarily unavailable, or when the user is immobile for extended periods of time. For example, if the application continues to request GPS fixes from the mobile phone when GPS signal levels are not sufficient to calculate a fix (e.g., the user is indoors), the phone will waste significant resources such as battery energy and CPU cycles before a time-out value is reached. Alternately, if a user is inside a building and a high-sensitivity GPS receiver is able to continuously calculate a fix, significant battery energy is expended to repeatedly calculate the same general position. Therefore, in both situations, it is desirable to increase the amount of time between new location requests when it is apparent that GPS is not currently available, or that the user is not moving for an extended amount of time.

Figure 3 shows a location-aware state machine that has been implemented in the TRAC-IT Java ME application. It regulates the rate at which position information is requested from the GPS hardware by the mobile application. If invalid location information (i.e., non-GPS data or GPS data with poor estimated accuracy) is repeatedly obtained by the application, or if speed values are less than a threshold for an extended amount of time, it gradually increases the amount of time between new position requests until a maximum value is reached at the right-most state, also called “state n” (e.g., a position request every eight minutes). When valid location information is obtained, the amount of time between new position requests slowly decreases until a minimum value is reached at the left-most state, called “state 0” (e.g., a position request every four seconds). The state machine also supports the functionality of snapping back to the most frequent recalculation rate when the first valid location data are obtained with speed values greater than a threshold. TRAC-IT utilizes this functionality since tracking applications ideally begin monitoring position information at a rapid rate as soon as the user is moving.

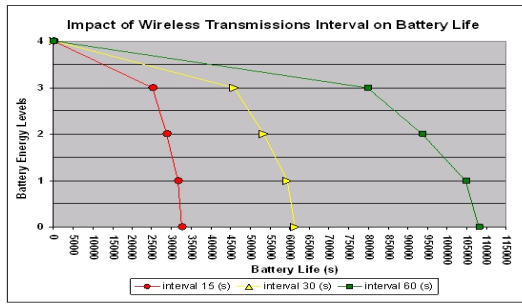


Fig. 4. Battery life benchmarking.



Fig. 5. Trip example showing the reduction in points.

In its current implementation, the TRAC-IT system manipulates the time values related to the frequency of location requests. The values related to the timing are the Interval (the time between fix requests), Timeout (time allotted for the retrieval of a valid fix), and Maximum Age (the length of time a fix may remain current on the phone before a new location fix must be calculated). The state machine may also be configured to change positioning technologies (i.e., Cell ID or Advanced Forward Link Trilateration) or manipulate other variables. It should be noted that properties other than a valid or invalid fix and speed threshold can be used as state transition triggers. For real-time LBS applications, a trigger could be the distance to a mobile or stationary target.

## V. EVALUATION

The CP algorithm and the location-aware state machine were implemented in TRAC-IT, a Java ME application that runs on GPS-enabled cell phones to automatically collect user travel behavior. While the CP algorithm and the location-aware state machine can be executed simultaneously for the greatest resource savings, the two algorithms are evaluated separately in this paper to demonstrate the effect that each has on device resources. The JSR179 Location Application Programming Interfaces (API) [16] was utilized to request location updates from the mobile phone. More information regarding the management of location retrieval using the JSR179 Location Listener can be found in Location API 2.0 for J2ME - A New Standard in Location for JAVA-enabled Mobile Phones [17]. A Sanyo SCP-7050 mobile phone with a Sanyo SCP-22LBPS 3.7V Lithium Ion 1000 milliampere-hour (mAh) battery on the Sprint-Nextel Code Division Multiple Access (CDMA) 1 x Radio Transmission Technology (RTT)

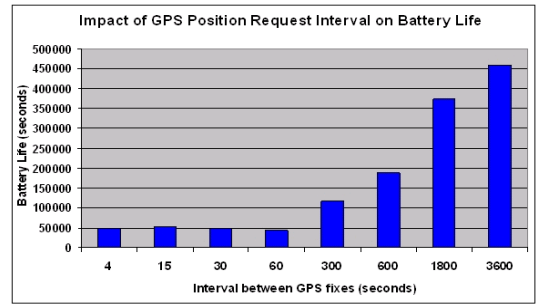


Fig. 6. Battery life benchmarking.

TABLE I

SAVINGS OF DATA CHARGES USING THE CRITICAL POINT ALGORITHM.

Trip	Total Number Points	Number Critical Points	% Fixes Transmitted	Bytes Saved	Financial Savings
1	73	26	35.61	5593	\$0.17
2	363	56	15.42	36533	\$1.10
3	489	65	13.29	50456	\$1.50
4	208	73	35.09	16065	\$0.48
5	357	62	17.37	35105	\$1.05
6	2330	159	6.8	257159	\$7.71
7	1022	139	13.60	105077	\$3.15
8	811	137	16.89	80206	\$2.40

cellular network was utilized in testing.

TRAC-IT was initially designed to transmit GPS fixes to the server periodically, every four seconds. However, it became clear that while the user's route could accurately be determined, the massive amount of generated data consumed significant resources on the mobile device and the server. Figure 4 illustrates the difference in device battery life for an application utilizing the User Datagram Protocol (UDP) to transmit the GPS fixes to the server at fixed transmission intervals of 15, 30, and 60 seconds until the device's battery was completely depleted. It is clear that battery life is directly proportional to the length of transmission interval, meaning that less frequent wireless transmissions result in a significant increase in battery life. By increasing the interval from 15 to 30 seconds, battery life is extended from approximately 9 hours to almost 17 hours. If the interval is increased further to 60 seconds, battery life reaches approximately 30 hours. Figure 5 and Table I show results obtained with the CP algorithm. Figure 5 clearly demonstrates that the CP algorithm is able to detect changes in the direction of the user, and establish critical points at appropriate places. Table I provides quantitative results on a set of trips, and an estimate of the cost savings to the mobile phone user produced by the algorithm. These results were obtained running the CP algorithm without any optional evaluations, and without the location-aware state machine. As shown in Table I, on average, the CP algorithm sent less than 20% of the GPS fixes to the server.

The location-aware state machine was also evaluated by benchmarking battery life on the Sanyo SCP-7050. Figure 6 shows the clear benefit of adjusting the GPS recalculation interval to values above 60 seconds. Battery life is extended

from under 14 hours using an interval of 60 seconds, to over 33 hours when using an interval of 5 minutes. While small intervals such as four seconds, are required to accurately represent the user's path as well as provide real-time services, significant savings can result from increasing the interval when the user is indoors, obstructed by tall buildings in a metropolitan area, or immobile for extended periods of time. Since the average daily travel time per person in the United States is only 82.3 minutes [18], utilizing this algorithm results in significant savings over the entire day when compared to a static interval of four seconds required for real-time tracking, but is not useful when the user is indoors.

While the length of battery life reported in the above tests may seem sufficient for a full day of operation when only one of the two algorithms is utilized, it should be noted that for each set of tests, the benchmarking application was executed while the phone performed no other operations (e.g., phone calls, screen display, etc.). In order to allow a location-aware application to execute on a mobile phone without a noticeable impact to the end user, it is important that both algorithms be utilized together in order to offset the additional energy costs resulting from normal mobile phone operations, as well as the simultaneous use of GPS and wireless data transmissions during real-time tracking.

## VI. CONCLUSIONS

In this paper, the Critical Point algorithm, an application-level algorithm designed to reduce the number of location fixes that a GPS-enabled cell phone needs to transmit to the server to save energy and communication costs, while allowing the application to continuously track the user in real time is presented. An additional complementary algorithm, a location-aware state machine that dynamically adjusts the GPS position recalculation rate based on environmental conditions, is also described. Both algorithms were implemented in TRAC-IT, a Java ME application for GPS-enabled mobile phones that automatically collects user travel behavior while providing the user with real-time, location-based services. The test results show that the Critical Point algorithm achieves its objective of sending only 20% of the GPS fixes on average, while allowing the application to build the path taken by the user. Battery life benchmarking tests clearly demonstrate that by reducing the frequency of wireless transmissions to the server by half, the Critical Point algorithm has the ability to nearly double mobile phone battery life. The state machine is also shown to have significant energy savings when the GPS recalculation interval is increased to more than 60 seconds.

Future research will focus on testing the Critical Point algorithm in location-based service applications with different tracking needs, and evaluating the most appropriate angle and speed thresholds that can best represent the user's path. Further testing of the location-aware state machine is also required to determine the ideal number of states (i.e., position recalculation intervals), the ideal interval values for each state, and the ideal rules and patterns for changing states to trigger new position recalculation intervals.

## VII. ACKNOWLEDGMENT

This work was supported in part by the National Science Foundation under grant No. 0453463, the Florida Department of Transportation, and the United States Department of Transportation through the National Center for Transit Research, under grant number BD-549-35. The authors would like to acknowledge the work of Samuel Rivera Gomez in coding and testing the Critical Point algorithm for the Java ME platform.

## REFERENCES

- [1] E. Murakami, D. P. Wagner, and D. M. Neumeister, "Using global positioning systems and personal digital assistants for personal travel surveys in the united states," in *International Conference on Transport Survey Quality and Innovation*, 2004.
- [2] R. Guensler and J. Wolf, "Development of a handheld electronic travel diary for monitoring individual trip-making behavior," in *Transportation Research Board 78th Annual Conference*, 2008. [Online]. Available: <http://www.fhwa.dot.gov/ohim/trb/wolf3.pdf>, accessed June 30, 2008
- [3] P. Winters, N. Georggi, and S. Barbeau, "Traveling smart: Increasing transit ridership through automated collection (trac) of individual travel behavior data and personalized feedback," Tech. Rep., August 2005. [Online]. Available: <http://www.nctr.usf.edu/pdf/576-16.pdf>
- [4] P. Winters, S. Barbeau, and N. Georggi, "Smart phone application to influence travel behavior (trac-it phase 3)," Tech. Rep., February 2008. [Online]. Available: <http://www.nctr.usf.edu/abstracts/abs77709.htm>, accessed June 30, 2008
- [5] D. Aguilar, S. Barbeau, M. Labrador, A. Perez, R. Perez, and P. Winters, "Quantifying the position accuracy of real-time multi-modal transportation behavior data collected using gps-enabled mobile phones," *Transportation Research Record: Journal of the Transportation Research Board*, no. 1992, pp. 54–60, October 2007.
- [6] D. P. Aguilar, "A framework for evaluating the computational aspects of mobile phones," Ph.D. dissertation, 2008, university of South Florida.
- [7] S. Barbeau, M. Labrador, P. Winters, R. Perez, and N. Georggi, "A general architecture in support of interactive, multimedia, location-based mobile applications," *IEEE Communications Magazine*, vol. 44, no. 11, pp. 156–163, 2006.
- [8] A. Kupper, *Locationbased Services - Fundamentals and Operation*. John Wiley & Sons, 2005.
- [9] A. Leonhardi, C. Nicu, and K. Rothermel, "A map-based dead-reckoning protocol for updating location information," in *Proceedings of the 16th IEEE International Parallel and Distributed Processing Symposium*, 2000, pp. 193–200.
- [10] O. Wolfson, A. Sistla, S. Chamberlain, and Y. Yesha, "Updating and querying databases that track mobile units," *Distributed and Parallel Databases*, vol. 7, no. 3, pp. 257–387, 1999.
- [11] A. Leonhardi and K. Rothermel, "Protocols for updating highly accurate location information," *Geographic Location in the Internet*, Kluwer Academic Publishers, pp. 111–141, 2002.
- [12] G. Treu, A. Kupper, and T. Wilder, "Extending the lbs-framework trac: Efficient proximity detection with dead reckoning," *Computer Communications*, vol. 31, pp. 1040–1051, 2008.
- [13] S. Microsystems, "Connected limited device configuration (cldc); jsr 30, jsr139," June 2008. [Online]. Available: <http://java.sun.com/products/cldc/>
- [14] W. Ballantyne, G. Turetzky, G. Slimak, and J. Shewfelt, "Achieving low energy-per-fix in cell phones," *GPS World*, July 2006.
- [15] W. Shi and C. Cheung, "Performance evaluation of line simplification algorithms for vector generalization," *The Cartographic Journal*, vol. 43, no. 1, pp. 27–44, 2006.
- [16] S. Microsystems, "Java specification request (jsr) 179: Location api for j2me," June 2008. [Online]. Available: <http://jcp.org/en/jsr/detail?id=179>
- [17] S. Barbeau, M. Labrador, P. Winters, R. Prez, and N. Georggi, "Location api 2.0 for j2me a new standard in location for java-enabled mobile phones," *Computer Communications*, vol. 31, no. 6, pp. 1091–1103, 2008.
- [18] L. Toole-Holt, S. Polzin, and R. Pendyala, "Two minutes per person per day each year: Exploration of growth in travel time expenditures," *Transportation Research Record*, vol. 1917, pp. 45–53, 2005.