A Parking Guidance and Information System for TinyOS

CSE 521S Final Report

Matthew Lindsay Patrick McBryde Michael Schultz

Washington University in Saint Louis

ABSTRACT

"Intelligent Transportation Systems" (ITS) are a mix of systems that gather various data metrics from transportation areas (parking lots, streets, alleyways, etc.), aggregate this data together, and present coherent information to the end-users of the system. Orchestrating such a system presents several challenges at each level. How do you gather the data and at what granularity? Where does the data go once gathered and can it be useful to anyone? If it can be useful, how can it be presented to end-users to provide accurate and simple-to-interpret content? This paper presents and explains our decisions in developing a "Parking Guidance and Information" (PGI) system for TinyOS.

1. INTRODUCTION

Driving to new destinations always brings some level of stress and uncertainty. Where am I going, what do the roads and intersections look like, what side of the road is the place on, where do I park? These are all questions that dart through the brain when beginning to think about going some place new. Luckily, maps bring us the answer or at least a partial answer to some of those questions. In the past decade there has been huge growth of web-based mapping technology to help answer these questions even more fully. For example, Google Maps allows you to get directions from point A to point B, and in the past 5 years has introduced Street View [13] to their interface. Street View allows a user to view the roads they will be travelling on from an in-car perspective. This removes much of the uncertainty when driving, leading to less confusion and a better experience (and potentially fewer accidents).

However, there is still that can be done. Even after you've arrived at your destination you must find where to park. This can also be quite a hassle, as you are unfamiliar with the area and during peak hours there may not be a parking space in near proximity. To help in this matter, Parking Guidance and Information Systems (PGIS) allow drivers to quickly evaluate where best to go to find parking efficiently [12]. Traditional PGI Systems simply count the number of cars that enter and exit a designated area and display the number of spots available in that area to the driver. These systems are imprecise, costly, and can't be integrated with other technologies.

This paper presents our experiences building a parking guidance and information system using the TelosB/TinyOS platform. By using motes, it opens the possibility of deploying a single sensor for each available parking space allowing for much more detailed information about a parking lot as a whole. Moreover, with proper sheltering, these sensors can be mounted on the surface of a parking lot instead of cutting into the lot and installing an inductive loop to detect vehicle presence. We also take advantage of the wireless multi-hop routing abilities of TinyOS-based motes to avoid wiring and enable a heterogeneous mix of sensors to keep track of parking spot availability and usage data. Combining these aspects gives a system that can be more precise, lower cost, and more easily integrated with future technologies.

The remainder of this paper is as follows. Section 2 defines the goals of this project more fully. Section 3 presents and explains the high-level architecture of our PGI System and details the hardware and software used during the course of this project. Section 4 talks about how we convinced ourselves that the system worked correctly and, if we have more resources, could scale up as needed. Sections 5 and 6 give an overview of related works and lessons we learned during this project. Finally, Section 7 concludes this paper with a discussion of potential future work for this project.

2. GOALS

Since we believe other solutions to PGI Systems do not represent what modern technology is capable of, our goals for this project are to build a PGI System that can be easily deployed at a low-cost, aggregates the data at a single location, and integrates with user-facing technology to provide a rich experience. In short, we plan to answer the following questions:

- How do you gather parking data and at what granularity?
- Where does the data go once gathered and can it be useful to anyone?
- How can it be presented to end-users to provide accurate and simple-to-interpret content?

Our goals for the hardware involve prototyping a low cost, wireless system which can function with a het-

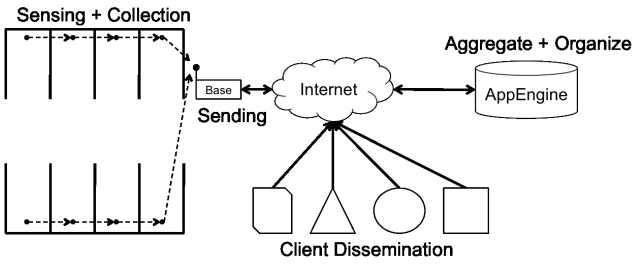

Figure 1: High-level view of our PGI System. A base station at the lot sends data over the Internet to our aggregator, which end-users can query.

erogeneous sensor suite. This will be accomplished by using sensors with generic connection ports, allowing for customizing the system to different sensor methods with ease. Also, using the built in sensors on the motes, we plan to provide drivers with more advanced feedback than is traditional in parking structures, such as headlight detection. Lastly, our project will be adaptable for any current parking structure. Therefore, it must make use of the wireless capabilities of the mote platform, reducing the installation cost by negating the need for expensive wiring.

Once our system has been installed, we plan to aggregate the data at a base station, located within the parking structure, which handles interactions between the WSN and the cloud. Communication between the base station and the network will rely upon Collection Tree Protocol (CTP), a robust method of point-to-sink wireless communication. Our base station then communicates with a computer via serial port, which forwards the data to the cloud.

3. DESIGN

Figure 1 shows a high-level view of the infrastructure of our PGI System. Our system begins with a parking lot (left side of Figure 1), each space in this lot is equipped with a sensor built on the TelosB/TinyOS platform. Data is collected at a per-lot base station, which then sends the data over the Internet to our backend. When our backend receives the data it updates and logs the changes to the data store and organizes the lot information. Once the data is stored, clients are able to query the backend and get up-to-date information about a specific lot and lots in the vicinity.

Building this system requires a fair amount of hardware and software knowledge and is described below.

3.1 Hardware

This should cover the physical hardware we used in building this. Including (but not limited to):

- TelosB motes (listing any relevant specifications, why we used TelosB and not something else) [5].

- External sensor we used, specifications for that, more detail than TelosB since the reader will be less familiar with it, why we used that.

At least one pictures of the telosb, the sensors, or the telosb with sensor attached.

We would like 6 TelosB/Tmote Sky sensors (2 for sensor testing and 4 for network development to build a non-trivial routing topology). It would also be interesting to use a Mica family board with the MTS310CA (magnetometer) sensor board, to test vehicle detection with a magnetometer, though not strictly required as we can find other sensors to use. Alternatively, if we can interface a magnetometer with the TelosB/Tmote Sky mote board that could be used instead of a Mica family board.

3.2 Collection Software

Yup we had software.

A paragraph or two about TinyOS and building whatever we used there (interfaces, interaction with other sensors and base station, CTP) [9].

A paragraph about the base station software and what it does.

The major components of this project are building and developing the sensing devices, writing and testing the networking and communications software to handle data delivery, and creating a friendly front end to present and track the information as needed. If we discover any of these components are significantly easier than others, it is easy to combine forces to develop new/better methods for any of them. We also have an interest in discovering more specific information of existing systems and getting up-to-date in the area of intelligent transportation systems.

3.3 Aggregation Software

Data leaving the base station of a parking lot is directed over then Internet to our aggregation software hosted on Google's AppEngine [2]. For this project, we decided the AppEngine environment was an appropriate choice for several reasons:

- The service is free for light usage (testing and development).
- It provides a Python programming environment.
- Most importantly, it is designed to scale up with ease transparently with large datasets.

The nature of our project is to have one sensor per parking space in a parking lot for every parking lot. This implies that over time our backend must be able to accept, store, and recall large amounts of current and historical data. We could have sunk our resources into using a traditional server model for the purposes of a prototype, however if the project were to start scaling up great

```
[
  {
    "space_id": 8,
    "is_empty": false,
    "magnet": 218,
    "sonar": 114
  },
  ...
]
```

Figure 2: Client to server JSON data.

amounts we would start running into resource barriers. Google's AppEngine is designed to scale with demand, this is largely due to their use of a datastore built on top of Bigtable [8]. Though transparent to users of our system, this non-SQL based datastore forced a few notable differences from a SQL based datastore.

First, while queries appear to be SQL, they are actually "GQL," a SQL-like language. In general this does not bring any problems, but because of the organization of the datastore imprecise queries are not acceptable. You must know the data you are looking for. You cannot simple SELECT one or two columns from the datastore, rather you must get all the columns or a unique key identifier that matches the query.

Second, its harder to perform proximity searches with the datastore because there is not concept of "similar" or "like" queries, you either know the data or don't know the data. However, with the help of the geomodel API [1] we are able to use geohashing to perform proximity searches on the datastore. This is useful to locate nearby parking lots, allowing the end-user to identify other potential parking areas.

With an understanding of the underlying technology, we are able to present an easy to use API.

3.3.1 A RESTful API

REpresentational State Transfer (REST) is an abstract model for building large-scale web services [11]. The principles of a RESTful architecture are an identification of resources through a URI, a "pure" HTTP interface, and self-contained hyperlinks. In this section we describe how we achieved these principles.

Our URI scheme is straight-forward. Each lot is given a unique identification consisting of alphanumerics and the underscore (_) character. When attempting to access a specific lot, that identifier is part of the URI (i.e., `http://<host>/lot/wustl_millbrook/` identifies the Milbrook lot on Washington University's campus).

Any request concerning this lot must go through its URI. Requests use one of the HTTP verbs of `GET`, `PUT`, or `POST`. A `GET` request simply returns the information for the requested URI in either the HTML or JSON format. A `PUT` request attempts to store updated status information to the lot. This request contains the

```
{
  "lot_id": "wustl_snowway",
  "geo_pt": "38.650233,-90.313529",
  "timestamp": 1291946695.0
  "spaces":
  [
    {
      "space_id": "4",
      "is_empty": false,
      "timestamp": 1291946695.0,
      "extra_info": "sonar:55;magnet:201;"
    },
    ...
  ]
}
```

Figure 3: Server to client JSON data.

list of parking spaces to update, their identification and status, as well as any associated meta-information that should be put in the datastore, an example of this can be seen in Figure 2. Finally, a `POST` request handles the creation of new parking lots.

With the API in place, we are able to create rich applications with ease for expansion and integration into other software. As a demonstration, we build up a web-based frontend in the next section that takes advantage of our API.

3.3.2 Frontend

As mentioned above, a `GET` request on the parking lot identifier will result in either a HTML or JSON response. An example of the JSON response can be seen in Figure 3. The response provides enough information to a client application to identify the name of the lot, its location, the time of the most recent activity, and the list of spaces. Each space has a unique identifier, the status of that spot (full or empty), the time it was last updated, and any additional information about that spot (likely to be sensor readings at the last report). A client can combine this information with a virtual representation of the lot's topography, then keep that representation up-to-date by periodically querying the server for new information.

For our project, we've also built an HTML interface for viewing the information on a lot. Once a lot is selected by the user, our interface provides a map of the area surrounding the lot with markers indicating other lots in a two mile radius. It also displays a "health" indicator giving an approximate empty-to-full ratio, as well as a list of spaces that are filled and how long they've been filled.

Additionally, by keeping a log of changes to a lot on the backend, we are able to provide a chart to indicate the average fullness of a lot over a time frame. For example, Figure 4 shows the average fullness of the "DUC" lot from 5:30am until 7:00am. At the 6:06am time slot there are 11 spaces being used, however only 20 minutes

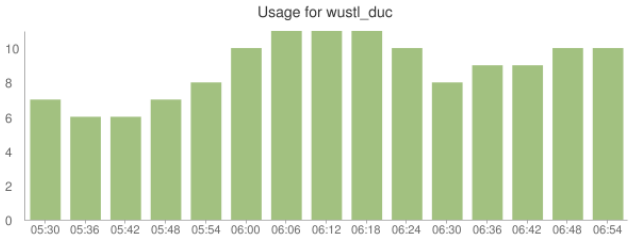

Figure 4: Example of a usage chart for the 5:30am–7:00am time slot.

earlier only 6 spaces are being used. Having access to this information would allow a daily commuter, or one time visitor, discover that by arriving only 20 minutes earlier they would have a much greater chance of easily finding a parking space.

4. EXPERIMENT

Paragraph about experiments that show our system works.

5. RELATED WORK

There are other proposals and existing systems for parking guidance and information, here we'll discuss and differentiate ourselves from these systems.

Signal-Park, uses sonar sensors above each parking spot to detect the presence of a vehicle [3]. Approximately three times per second, the sensor is queried to determine vehicle presences and updates a central computer. This system uses serial lines to connect each sensor with the central computer, leading to more costly deploy. It also appears to be limited to the parking lot that it is deployed in and lacks the ability to aggregate the information for others to benefit. It would certainly be possible to improve the connectivity of this system to integrate with our back-end aggregation system to take advantage of existing installations.

Streetline, is a startup in the San Francisco area that similarly uses a wireless mesh network to link all the sensors together [4]. The Streetline system has recently (Summer 2010) begun to roll out sensors and upgraded meters for select areas in the San Francisco area [7, 6]. This system uses surface (and sub-surface) sensor mounts that contain a wireless transmitter and a magnetometer for vehicle detection. Status is sent to a central aggregation server where users can see the status of street parking via their web browser, text message, or smart phone; it also allows municipal workers to identify vehicles that have not paid fully. Presuming the Streetline trial is successful and the system expands, our system and their system should be able to integrate with little effort and could be seen as competitors. We view this a positive, as it demonstrates that our project is a useful idea with potential buyers available.

There has also been a proposal to use existing security cameras with a view of a parking lot to detect vehicular presence [10]. The results of their experiment, while decent for their scenario, seem unlikely to generalize to other systems. While it may be true that some lots have existing surveillance cameras, the paper also observes that cameras should be mounted higher for a more consistent and complete view of the lot. This goes against their principal of using existing cameras for the information, since additional cameras would have to be mounted to perform correctly (moving security cameras further from the observation area is probably not acceptable to the security officers).

6. LESSONS LEARNED

Anything we would have done differently if we were to pursue this project in more detail again.

7. CONCLUSIONS AND FUTURE WORK

Conclusions from this project.

8. REFERENCES

[1] geomodel. `http://code.google.com/p/geomodel/`.

[2] Google appengine. `http://code.google.com/appengine/`.

[3] Signal-park parking management solutions. `http://www.signalpark.com/`.

[4] Streetline parking management system. `http://www.streetlinenetworks.com/`.

[5] TelosB Datasheet. `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/TelosB_Datasheet.pdf`.

[6] SF*park*, 2010. `http://sfpark.org/`.

[7] K. Barry. City parking smartens up with Streetline. *Wired Magazine*, November 2010. `http://www.wired.com/autopia/2010/11/city-parking-smartens-up-with-streetline/`.

[8] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. In *In Proceedings of the 7th Conference on USENIX SymposiuM on Operating Systems Design and Implementation*, pages 205–218, 2006.

[9] R. Fonseca, O. Gnawali, K. Jamieson, and P. Levis. Collection. TEP 119 (Documentary). TinyOS-2.x, URL `http://www.tinyos.net/dist-2.0.0/tinyos-2.x/doc/html/tep119.html`.

[10] S.-F. Lin, Y.-Y. Chen, and S.-C. Liu. A vision-based parking lot management system. In *International Conference on Systems, Man and Cybernetics, 2006*, volume 4, pages 2897–2902, Oct. 2006.

[11] C. Pautasso, O. Zimmermann, and F. Leymann. Restful web services vs. "big"' web services: making the right architectural decision. In *Proceeding of the 17th international conference on World Wide Web*, WWW '08, pages 805–814, New York, NY, USA, 2008. ACM.

[12] A. Sakai, K. Mizuno, T. Sugimoto, and T. Okuda. Parking guidance and information systems. In *Vehicle Navigation and Information Systems*

Conference, 1995. Proceedings. In conjunction with the Pacific Rim TransTech Conference. 6th International VNIS. 'A Ride into the Future', pages 478 –485, July 1995.

[13] L. Vincent. Taking online maps down to street level. *Computer*, 40(12):118 –120, 2007.