A Parking Guidance and Information System for TinyOS

CSE 521S Final Report

Matthew Lindsay Patrick McBryde Michael Schultz

Washington University in Saint Louis

ABSTRACT

The Internet, in its current form, provides up to the minute updates on traffic conditions and can provide motorists with detailed mapping and directions for nearly any destination. However, once the driver arrives at their destination, they must wander and search for a parking space, wasting fuel and time, as well as causing traffic buildups and unnecessary expulsion of greenhouse gasses. By introducing Wireless Sensor Networks (WSN) into the process, drivers can be provided with up to the minute updates about parking at a frequency and cost lower than current implementations. Currently, expensive induction loops or other, more permanent systems must be installed, typically only at the entrance and exit to a parking structure.

"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 [15] 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 [14]. 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

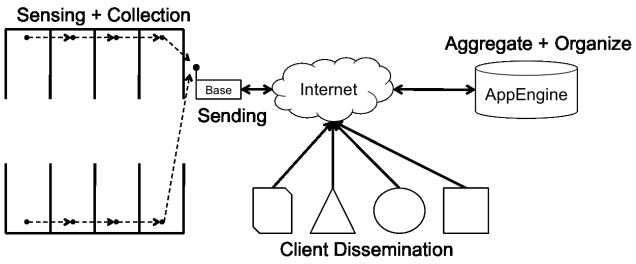

Figure 1: High-level view of our PGI System. A base station at the lot sends data over the Internet to our aggregator, which end-users can query.

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 heterogeneous 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

When choosing the hardware for our PGI system we had specific goals in mind. The system would need to be low cost, reliable, power efficient, and easily customized to meet the needs of the specific structure/lot. We believe most of our potential business would come from upgrading existing parking structures versus that of new construction. Depending on there environment, some sensors may work better than others but the best sensor for a particularity environment still may not be the best choice for the system. Take for example an induction loop sensor, these are typically the best sensor for detecting the presence of a vehicle but in an pre-existing structure it would prove to be very expensive to cut into the floor of every parking space to add them. This is why it is necessary for the system to be easily customized and it is likely the most important requirement for the successful adoption of our system by the parking industry. When adding a system such as this, it is typical for the installation cost to be more prohibitive than that of the PGI system. By including wireless technology and supporting many different sensor types and configurations we believe we can successfully implement this system in both pre-existing and new parking structures.

3.1.1 Parking Space Monitors

The Parking Space Monitors need to be able to do more than just determine if a vehicle is present in a parking space. They must be able to run on a set of batteries for an extended period of time, support multiple sensor types and interfaces, and wirelessly transmit required data to a base-station for further processing. For all these reasons we chose the Telos Revision B (TelosB) wireless sensor module as the base for our Parking Space Monitors.

Shown in Figure 2, the TelosB provides a multitude of features and sensors, but still manages to use little power and support fast wireless communications. We chose the TelosB because it includes a 250 kbps IEEE 802.15.4 wireless transceiver; on-board temperature and light sensors; low power consumption; an on-board antenna; simple programming/collection interface; and a 10+6-pin expansion slot allowing analog, digital, or serial connections [6].

The TelosB includes many different interfaces we could use TODO **Finish describing the TelosB interface we could connect sensors to. Such as analog, UART, I2C, Digital**

There are many different types of sensors that could have been used to monitor if a parking space is occupied or vacant. These include infrared range finders,

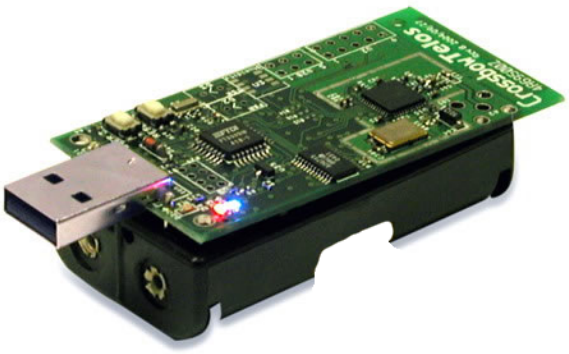

Figure 2: Telos Revision B.

Figure 3: LV-MaxSonar®-EZ1™ by MaxBotix® (our "sonar sensor").

pressure sensors, and inductive sensors. We chose to use the LV-MaxSonar®-EZ1™ by MaxBotix® (hereafter "sonar sensor") because of it's price and feature set [3]. These include:

- TODO **Clean up this List**
- Supports supply voltage from 2.5V to 5.5V
- Detects objects from 6 inches out to 254 inches with 1 inch resolution (0-6 inches range as 6 inches)
- Output formats include pulse width, analog voltage, and serial digital

3.1.2 Base-Station

For ease of development a MacBook Pro was used as the base-station in this project. However, keeping with our theme of low cost, we tried to keep the requirements of the base-station to a minimum by only using features found in any low end unix/linux supported computer. The only requirements of the base-station are that it be capable of connecting to the internet and that it provides a USB connection. Because video is not a requirement, we would recommend a small form factor Plug computer. These are typically very small, power efficient and cost effective.

The base-station must be able to support IEEE 802.15.4 communications with the Parking Space Monitors. This is the reason the base-station is required to support a USB connection. Rather than implementing a costly IEEE 802.15.4 solution directly into the base-station, we chose to connect a TeloB via USB and use a simple serial connection to allow the two devices to communicate.

3.2 Collection Software

3.2.1 Parking Space Monitors

The Parking Space Monitors are powered by TinyOS. We chose TinyOS because it fully supports the TelosB and provides many features that rapidly speed-up development and TODO ** Finish describing the Parking Space Monitor software functionality**

One of out key design goals was wireless reliability. We need to ensure that packets containing sensor details always make it back to the base-station for processing. This can be very difficult in dynamic environment such as a parking structure, especially a multi-floored garage where line of site is impossible, and transmissions will not carry through the industrial walls present in the structure. Therefore, alternative methods of packet forwarding must be introduced in order for the system to be able to operate with a single, centrally located base station. In this project, we attempted to use Collection Tree Protocol (CTP).

CTP is a dynamic point-to-sink routing protocol, which creates a tree topology for the network through which all packets travel to the base station, known as the root node or sink. By creating this tree based topology,

motes dynamically create an efficient routing method for collecting all data at a centralized point. CTP does not handle point-to-point communication, though it does provide support for broadcasting to all nodes. In our final implementation, due to a miscommunication about integrating the various parts of the project, CTP was left out of the demonstration in the interest of a robust demo. However, initial tests of a non-trivial CTP network gave indications that it would function well within our project [11].

3.2.2 Base-Station

The base-station software has multiple functions. It was written in C so as to be compatible with most current versions of unix/linux. It is responsible for configuring the Parking Space Monitors, monitoring their status, collecting data from them, and sending the collected data to the aggregation software over the internet.

Since the base-station needs to communicate with the motes over IEEE 802.15.4 a TelosB is connected via USB as part of the base-station. The TelosB is running the default BaseStation app (included with the TinyOS install) with only minor tweaks to support CTP.

The base-station has the ability to send configuration packets to the Parking Space Monitors. TODO **Finish describing the config process**

The base station collects sensor readings from the parking space monitors and then determines what data is required by the aggregation software and then JSON encodes the data. TODO **Make sure the JSON data is moved ahead of this section and finish describing this process**

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 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

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

Figure 4: Client to server JSON data.

of Bigtable [9]. 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 [13]. 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 [10] format. A `PUT` request attempts to store updated status information to the lot. This request contains the 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

```
{
  "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 5: Server to client JSON data.

be seen in Figure 4. 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 5. 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 6 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 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

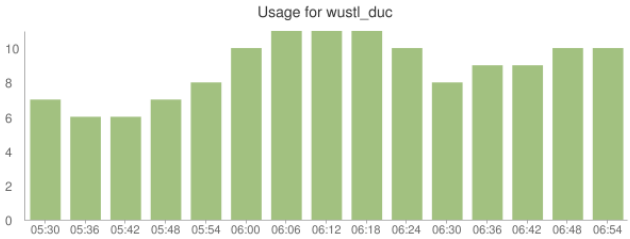

Figure 6: Example of a usage chart for the 5:30am–7:00am time slot.

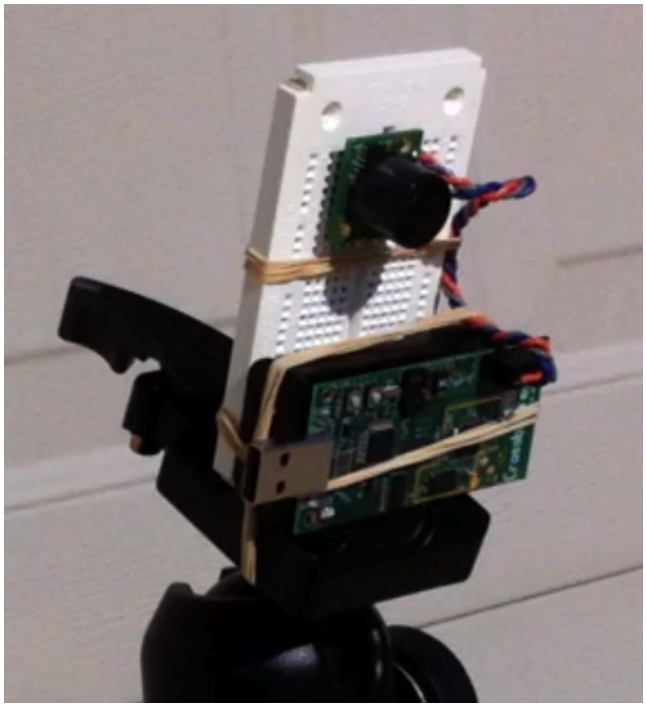

Figure 7: Parking Space Monitor on tripod for testing.

earlier they would have a much greater chance of easily finding a parking space.

4. EXPERIMENT

Since our system was split into two major components, we were able to test both somewhat independently. The first component included physically sensing the vehicle and relaying the data to the backend. The second component involved testing the backend works for extended periods of time and correctly logs/recalls historical data.

4.1 Sensing and Sending

A large amount of testing was performed on the sensing and sending portion of this project. Figure 7 shows a Parking Space Monitor mounted on a tripod. This set-up was used to test the system with an actual vehicle. It was through this testing that we found an issues

with the sensing range of our Parking Space Monitors. Our sonar sensor claims to support a distance up to 254-inches, however we were only able to detect a vehicle up to approximately 120-inches or around ten feet. This did not seem to be an issues with detecting a vehicle because they will be much closer than ten feet from the front of the parking space, but could be an issue with detecting smaller objects such as motorcycles. More testing an analysis would need to be performed to determine if the ten foot range of the sonar sensor would really be an issue. It was determined while testing that the issue was with the supply voltage to the sonar sensor. The sonar sensor range finder will support a supply voltage of 2.5V - 5.5V. We chose to power the sonar sensor using the TelosB analog supply which can only supply about 3.1V. When we used an external supply to test just the sonar sensor at around 5V it had a much greater range compared to 3.1V. If after more testing it is determined to be an issue, there are solutions such as using an external power supply for the sonar sensor or possibly mounting the sensor above the space to cover a greater area of the space.

4.2 Backend and Frontend

Since it was not feasible to test a large scale deployment due to cost, time, and logistical constraints a base station simulator was developed. Instead of interacting with a sensor network for gathering data, the simulator was designed to generate a number of changes to a lot every one to two minutes. The changes include modifying the status of a random number of parking spaces with some weight attached to the likelihood of a space becoming empty for a time interval. For example, during the overnight hours there only a 16% chance of a parking space becoming full causing the overnight hours to present less lot usage.

This simulator helped test the ability of the backend to continue to be stable over a number of hours while periodically inserting new log information and maintaining the current status of a parking lot. It also helped generate large amounts of data for testing the analytics portion of our frontend (charting for time periods).

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 [4]. 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 [5]. The Streetline system has recently (Summer 2010) begun to roll out sensors and upgraded meters for select areas in the San Francisco area [8, 7]. 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 [12]. 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

If we were to pursue this project into the future we would probably modify the design to allow a single TelosB to monitor multiple sensors. The ultrasonic range finder we used has multiple output types, one of them being a two wire digital serial connection. This connection allows for up to twelve sensor connections (which can include our sonar sensors). This would allow us to either use two sensors per space to improve accuracy and monitor six spaces per TelosB or continue with a single range finder per space and monitor twelve parking spaces per TelosB. This could greatly decrease the cost of the overall system.

Another interesting lesson was using Google's AppEngine for the backend. The cost structure of AppEngine is set up to be free as long as resource usage stays below a daily threshold. After modifying the frontend to automatically update the lot usage, we easily surpassed the daily threshold giving us two options: back the AppEngine instance with some money to increase our limits or refactor the code to be less resource intensive. Luckily, we were able to find a few code optimizations that greatly reduced the resources for the automatic update code putting us under the daily allotment again. AppEngine also recently introduced a new feature ("channels") that would have also allowed us to provide true asynchronous updates instead of query oriented updates.

7. CONCLUSIONS AND FUTURE WORK

This project created an extensible wireless sensor that is able to detect the presence of a vehicle in a parking space. That data is transferred from the sensor, to the base station, and over the Internet to our backend aggregator. Once at our backend, the data is logged and inserted into a scalable web infrastructure. At any point an end-user is able to query our web service to determine the current fullness of a lot at their destination and, if desired, find the fullness of lots that are near-by to their destination. The end-user is also able to discover when the lot is likely to be most empty or most full. Moreover, through the use of a web-based API using the standard JSON format, application developers can interact with our system hassle-free allowing rich web apps, smart-phone apps, or data mining opportunities.

As with many projects, there are both many features that can still be added and much research into the viability of deploying such a sensor network. Feature-wise, the web API could support richer queries. For example, the ability to select multiple date ranges for charting allowing a user to view average aggregate data for the weekdays over the past month to decide if a parking lot should expand. Another option is to integrate our system with a mapping service and combine up to the minute traffic reports to allow a user to plan their trip while being mindful of the route, traffic conditions, and available parking. On the hardware side, a careful evaluation of battery life and how to increase battery life would be helpful. Stripping away unused components from the TelosB could help greatly as well as integrating solar panels for power support and creating an external casing to protect the sensitive electronics from the environment.

8. REFERENCES

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

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

[3] LV-MaxSonar-EZ1 High Performance Sonar Range Finder. `http://www.maxbotix.com/uploads/LV-MaxSonar-EZ1-Datasheet.pdf`.

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

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

[6] Telos (Rev B): PRELIMINARY Datasheet, December 2004. `www2.ece.ohio-state.edu/~bibyk/ee582/telosMote.pdf`.

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

[8] 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/`.

[9] 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.

[10] D. Crockford. JavaScript Object Notation. `http://www.json.org/`.

[11] R. Fonseca, O. Gnawali, K. Jamieson, S. Kim, P. Levis, and A. Woo. The collection tree protocol (ctp). TEP 123 (Documentary). TinyOS-2.x, URL `http://www.tinyos.net/tinyos-2.x/doc/html/tep123.html`.

[12] 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.

[13] 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.

[14] 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.

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