ParkingMeter: Balancing Energy Savings and Service Availability

Riccardo Crepaldi
University of Illinois
at Urbana Champaign
rcrepal2@illinois.edu

Ryan Welsh
University of Illinois
at Urbana Champaign
rjwelsh2@illinois.edu

Robin H. Kravets
University of Illinois
at Urbana Champaign
rhk@illinois.edu

ABSTRACT

Vehicular networks offer service coverage for urban environments that would otherwise be too costly for infrastructure-based networks to provide. While many services have already been proposed based on collaboration between moving cars, the inclusion of parked cars in these systems extends their reach, coverage and stability. However, despite the presence of a large car battery, cars still suffer from limited energy availability when they are parked.

In these diverse environments, many services may be offered and existing application-oriented energy management solutions fail to capture the complexity of optimizing energy for each individual service, while current service-oriented solutions fail to capture the interactions between services. In response, we propose ParkingMeter, an energy management framework that handles the provisioning of multiple services concurrently by fairly allocating energy to each service, guaranteeing the most effective energy utilization for the system. We demonstrate the effectiveness of ParkingMeter through simulation of three diverse services and the feasibility of the architecture through the evaluation of our prototype system.

Categories and Subject Descriptors

C.2.1 [**COMMUNICATION NETWORKS**]: Network Architecture and Design; C.2.4 [**COMMUNICATION NETWORKS**]: Distributed Systems

Keywords

VANET, Opportunistic, Power Management, Distributed services

1. INTRODUCTION

The availability of mobile services, including accessing content or connectivity, has exploded, and with it, the demand for users to access those services over limited wireless and cellular infrastructures from where ever they may be, especially in their cars. With the increasing number of devices and the high rates at which data is generated and requested, the combined user bandwidth requirements will quickly overwhelm the current infrastructures, and increasing availability even more would be too expensive. In response, distributed networks of vehicles have been proposed to enhance local services, ultimately providing relief of some of the stress on the infrastructure [1, 2, 3]. Since connectivity between cars is challenged by high mobility and uncoordinated deployment, the inclusion of parked cars has been proposed to improve network stability and so system and service performance [1, 2]. While introducing parked cars seems to be a straightforward extension, despite the large battery in every car, energy management becomes a serious challenge.

To enable the inclusion of parked cars, it is necessary to devise an energy-efficient schedule that drives when each service should be turned on based on the demand for that service over time. One of the biggest challenges in determining such a schedule is the unpredictability of the network and the service demands, which together require flexible and dynamic energy management strategies. Additionally, multiple services may be offered at the same time and the diverse nature of these services may require conflicting schedules. Consider a car-to-car connectivity service using parked cars as relays [1] that runs concurrently with a data collection service for participatory sensing applications [3]. For the former, a good schedule would provide the service when the traffic flow is high, as we have shown in [4]. For the sensing application, it makes more sense to use a duty-cycle approach which evenly alternates periods of activity with periods of sleeping. As hard as it is to measure the utility of either service, the real challenge comes from combining two completely uncorrelated utility metrics to enable an energy efficient schedule. And of course the more services added, the more complex this problem becomes. Finally, all of these decisions must be made with limited input from the service designers, who have no knowledge of which other services are being offered.

In response to these challenges, we propose ParkingMeter, a power management framework that enables effective shared use of the available energy with minimal effort required to the service designers. ParkingMeter decisions are based on estimates of vehicle stop duration, available energy and demand for each service. ParkingMeter autonomously determines a *schedule* that indicates when the system should

VANET'13, June 25, 2013, Taipei, Taiwan

be active based on monitored system metrics (i.e., time or battery level) and service specific metrics (i.e., density of cars or frequency of running the service) that determine the utility of each service. The schedule does not specify exact times at which the system should be active, but rather what conditions should trigger its activation. The main contribution of our research is the design of a framework that enables providing services even in an energy-constrained environment with a minimal effort from the service designers. The main challenge that ParkingMeter must overcome to achieve its goal is that of performing local, independent decisions, in trying to provide a global service. We evaluate the effectiveness of ParkingMeter through simulation of three services and validate our design on a prototype system. Our results show that ParkingMeter can effectively allocate energy across multiple services in dynamic environments.

The rest of this paper is organized as follows. In Section 2, we discuss why current service-oriented techniques cannot be directly applied to support the multiple services expected in vehicular networks. Section 3 presents three services classes and their associated utility functions. In Section 4, we describe the system model and architecture of ParkingMeter. In Section 5, we provide a test case of how ParkingMeter can provide optimal service for three competing services and present the results from our prototype for one specific service, LoadingZones. Finally, in Section 6, we discuss ongoing work and future improvements to ParkingMeter.

2. DISTRIBUTED NETWORK OF SERVICES

Mobile systems were originally designed for *application*-oriented devices that focused on responding to user input and requests. However, as mobile devices become more powerful, they have been used to build *service*-oriented distributed systems, such as wireless mesh, sensor or vehicular networks, where nodes cooperate to provide network-wide services to monitor the environment [**?**, 3], react to specific events, or create a distributed network for users to access the Internet or each other [**?**, **?**, 1, **?**, **?**]. In this context, vehicular networks have been suggested to support participatory sensing [**?**, 3], enhance car-to-car communication to share updates and alerts [**?**, 1, **?**], as well as provide Internet connectivity without overloading the cellular network or incurring its high costs [**?**, **?**].

For vehicular networks, recent proposals have suggested the use of parked cars to improve system performance by introducing more stability into vehicular networks [1, 2]. However, none of these approaches' designs have considered the fact that although services in parked cars do have access to the car battery, that battery is not unlimited and must always provide the power to start the engine when required. Building energy-efficient hardware is a necessity in any battery-powered system, but it might not be enough to provide continuous service when the battery is the only source of energy. When a device does not have enough energy to provide continuous service, it is necessary to plan a schedule and alternate periods of activity with periods of sleeping to save energy. Careful planning can mean the difference between beneficial use of energy to provide services that are actually used or wasting it when requests for the service are low.

Energy management for *application-oriented* systems is a well-researched area. However, such solutions can only be applied to improve the efficiency of the service when it is running and so extend the amount of time a service can be available; they do not address the problem of scheduling service availability to maximize service utilization. While not quite as well fleshed out, energy management for *service-oriented* systems has also been investigated with the main focus on supporting one service, [4]. The presence of multiple, diverse services on the same system makes things even harder to manage.

The main goal of service-oriented systems is to provide a service until a specific deadline. If the devices are active until there is no energy available, the service will work at full potential initially and eventually be completely silent. Devices should instead alternate periods of activity and periods of sleeping to save energy and spread service availability over time.

In many mobile service-oriented systems, such as sensor networks, the main source of energy consumption is the wireless network interface. Synchronization mechanisms such as PSM, or signaling-based protocols such as B-Mac [5], X-Mac [6], or NPM [7] save energy by enabling long periods of sleep during idle times. If the network is dense enough and cooperation can be exploited, more aggressive approaches, such as temporarily shutting down entire nodes [8], can be applied without disrupting the service [9]. When a sleeping node needs to be woken up, a trigger is used to turn the system back on (i.e., Wake on Lan [10], Wake on Wireless [11] or passive [12]), significantly reducing energy consumption of the wireless component.

While existing techniques can be used to reduce network energy consumption, they might not be as effective in vehicular networks since the nodes are large vehicles equipped with powerful and expensive hardware capable of providing multiple, complex services, in which the energy required by the network interface is just a fraction of the overall energy consumption. Additionally, while signaling-based methods reduce idle energy consumption, they assume that the switching time is almost immediate, so that a trigger signal can wake up the radio in a matter of milliseconds. Vehicular networks introduce two complications to applying such solutions. First, systems in a vehicle may be quite complex and so most of the components must be off to effectively save energy. Switching from an energy saving state to an active state implies booting or resuming a complex operating system, establishing network connections and activating one or more services. This transition could take a few seconds. Second, the average contact duration between two cars is also only a few seconds [1]. Essentially, if the system waits until two cars come in contact with each other to take the system out of sleep mode, by the time the system is finally active and ready to provide the services, the contact is almost at its end.

However, as difficult as it is to optimize energy consumption for a single service, the final challenge comes when trying to support multiple services in an energy constrained system. While a solution such as [4], tailored to one specific service, might achieve the highest performance in terms of service offered per unit of energy spent, this approach is limited. First of all, it would add the burden of designing and implementing energy management for every new service. Additionally, while it is generally safe to assume that a wireless sensor network is providing only one service, or a small number of strongly correlated services, a vehicular

network is capable of supporting multiple distinct services at the same time, each with their own specific energy requirements. An uncoordinated implementation of energy management algorithms might result in poor combined performance or, even worse, in conflicts between the different strategies. For this reason, we propose a framework that abstracts energy management so that any service can interface with it and, with a minimal exchange of information, the system can provide fair and efficient resource use across the offered services.

3. SERVICE-DEPENDENT SCHEDULING

The benefits of activating a service can be measured by introducing the concept of the *utility* of a service, which depends on the particular use or performance of a given service. For example, the utility of a routing service can be quantified by the number of messages being routed, while the utility of a sampling service can be quantified by the number of samples per second. Given knowledge of the amount of energy available for a given time period, as well as which services can be made available and their associated utility, the goal of an optimal energy management strategy for multiple services is to determine a active-sleep schedule for the system that satisfies the following two properties: (1) all available energy from the auxiliary battery should be used during a stop (i.e., no available energy is left unused); (2) the chosen active times for each service maximize the utility of the system.

Before we can discuss the optimization of the utility of an entire system, it is necessary to understand the utility of a given service. One of the biggest challenges here is that it is often very difficult to come up with a perfect utility function for a service. However, given the dynamics of our target systems, perfection is not needed. Instead, it is beneficial just to have an approximation of the utility. Therefore, we next present three classes of services and propose associated utility functions. The first two classes are based on measurable properties of the system (i.e., density of cars and frequency of availability), while the third requires a customized utility function provided by the service.

The services discussed below have all been proposed or implemented for vehicular networks. Despite the fact that some have considered the use of parked vehicles, outside of our preliminary work on an energy model for LoadingZones [4], the remaining approaches do not include an energy management policy, but instead either focus on non-energy-constrained environments or propose energy management as a future work. Therefore, for LoadingZones, we briefly describe our energy model and, for the other two services discussed, we describe our proposed energy model for that service, present an associated utility metric, and show how these can be used to optimize energy consumption for the specific service if it is run by itself.

3.1 System and Energy Model for Services

Each vehicle supports the desired services using an embedded device that can be active, requiring power P_a, or idle, requiring a much lower power, P_i.[1] The vehicles are either moving, in which case no energy management is needed, or parked for time δ_s and have available energy E_{av}. In this section, we assume that the duration of the stop is known to the system. Since the stop duration in a deployed system must be estimated, in the rest of this paper, we consider the duration of the stop to be a random variable with normal distribution, $\mathcal{N}(\mu_s, \sigma_s)$, where μ_s is the average, and σ_s the variance observed for the specific location and time. Although it might seem difficult to accurately estimate the duration of a stop for a parked vehicle, it has been shown that it is strongly correlated to the parking location and time of day. For example, if a car is parked in a restaurant area at dinner time, the stop is most likely going to last between one and two hours, while if it is parked at its home address at night, it is more than likely that it will be parked until the next morning. An even more accurate estimate can be made if historical data stored on a specific vehicle can be queried to locate previous stops in the same area at similar times. The assumption of a normal distribution for parking duration needs further investigation to be verified. However, an analysis of how statistical data can improve the accuracy of this estimate is not within the scope of this paper, and we leave it for future work. Although the absolute performance might change if the normal distribution does not hold, our system architecture will still be valid.

To determine an optimal schedule for an individual service, it is first necessary to determine what fraction of the stop the service can be active for, and then select a schedule of active and idle times. The total time the system can spend in the active state is a function of E_{av}, P_a, P_i and δ_s. The total energy spent during a stop, E, is given by: $E = \delta_a \cdot P_a + \delta_i \cdot P_i$, where δ_a and δ_i are the times spent in the *active* and *idle* states, and $\delta_s = \delta_a + \delta_i$. The maximum length of δ_a can be calculated using the maximum amount of energy that can be spent, $E = E_{av}$:

$$\delta_a = \frac{E_{av} - P_i \cdot \delta_I}{P_a} = \frac{E_{av} - P_i \cdot \delta_s}{(P_a - P_i)}. \quad (1)$$

This time accounts for a fraction $f = \frac{\delta_a}{\delta_s}$ of the total estimated duration of the stop.

3.2 LoadingZones

The first service we consider is LoadingZones [1], a communication system that uses parked cars as relay agents between moving vehicles and APs. LoadingZones divides what would be a single connection between a moving car and an access point into a two hop link: the moving vehicle communicates with a parked car, which relays the packets to and from an indoor AP.

The utility of running LoadingZones is determined by the number of passing cars that need connectivity and the amount of data they need to transfer. One way to approximate this is to use a utility function that is proportional to the density of vehicles. For example, if rush hour is known to be between 5 and 6 PM, a car parked between 12pm and 8pm could save energy from 12pm - 5pm to make sure that LoadingZones can be provided during the most demanding time.

Our preliminary work on energy management for parked cars focused on supporting only one service, LoadingZones in [4]. In that work, we presented an energy governor that, given an estimate of the traffic density, identifies a threshold Th such that when the density is above this value, the

[1]This base service energy model is identical to the one presented in [4]. We present it here so we can build off of it to show that it is generic enough to schedule multiple types of services.

system is activated and that the total active time during the stop is equal to δ_a, thus optimizing the utility of the service and using all of the available energy.

3.3 MobEyes

MobEyes [3] is a participatory sensing system that leverages the pervasiveness of cars on the road in an effort to extend the scope of sensing to an urban environment without the necessity of deploying a new infrastructure. However, the networking required to upload the samples, which can include video, sound recordings or similar complex samples, requires a high bandwidth, energy-expensive connection. Additionally, complex sensors, such as those for air quality, use energy demanding hardware and it would be prohibitive to keep them active when new samples are not required.

For sensing applications, the utility of specifically sensed samples is application-specific since knowing what quantity is being sensed and its characteristics is necessary to understand precisely what defines the utility. Mobeyes did not include any energy manager, however it is straightforward to approximate the utility of a sensing service based on the frequency at which the samples are taken (i.e., more frequent samples represent higher utility). Based on how long it takes to take one sample, δ_{sample}, our proposed energy management policy schedules regular sampling intervals, i, so that the sum of the active time stays within the constraints

$$i = \frac{\delta_s}{\frac{\delta_a}{\delta_{sample}}}. \quad (2)$$

The more energy available, the more frequently samples can be generated. For this service, δ_s could be computed as $\mu_s + \sigma_s$ to guarantee that the estimate is longer than 80% of the stops, so that energy can last until the end of each stop.

3.4 MapShare

Some services focus on content delivery and sharing. For example, TomTom's MapShare [13] is a service for disseminating more frequent map updates using a sharing technology among users, which can update their maps and share these updates with other users in their "network". Currently, MapShare works only off-line, connecting the device with a USB cable to a computer. With a WiFi-radio embedded in the navigation unit and a proper energy manager, not only could updates be downloaded several times a day, but each vehicle could also cooperate in pushing new information for other vehicles to use.

Utility for this type of service is a little more complex and will likely need a customized utility function from the service provider. In our generic implementation, we define the utility of uploading data for other vehicles as decreasing with the age of the data, while that of downloading data that improves the navigation decision for the upcoming trip increases as the departure time is approaching. Although it is true that the user will start the car, providing enough energy, moments before leaving, this time might not be enough to download the updates, so it is still a good idea to start the download before the user returns.

Assuming knowledge about the parking duration, δ_s, the best policy is to immediately upload any local updates and then keep the device in an *idle* state, activating the service right before departure, so that the energy is used to download the most recent information. Therefore, the system should be active half of δ_a to upload information at the beginning of a stop, and half of it to download the latest update in an interval centered around $\mu_s - \sigma_s$. The even distribution of active time between uploading and downloading information is just an example, and different strategies could easily allocate more energy to one or the other activity.

3.5 Supporting Multiple Services

In a resourceful device like a car, many of these services can be provided simultaneously, which introduces an interesting problem of resource sharing. Essentially, it is important to balance the use of the available energy fairly across the multiple services. However, given the dynamics of the systems and the environment it is deployed in, this fair share must be based on the utility of offering the service at a given time.

Without knowledge of the other services and their associated utilities, the combined schedule determined by an independent set of services could end up being suboptimal, leaving unused energy that could have improved the performance of the system, or overoptimistic, trying to use more energy than that available to the system.

In the next section, we present the basics of our system, Parking Meter, which abstracts the energy management policies and enables a seamless integration of many services with minimum effort for the service designers. ParkingMeter is designed to manage any type of service that can be associated to an utility function, with a minimal effort required to the system designers, and without requiring any explicit interaction between services.

4. PARKING METER

Given the number and diversity of the expected services offered in a vehicular network, a generalized energy management strategy is key to extending the network to parked cars. However, it is critical to provide a system interface that simplifies the load on the service designers and so allows them the means to integrate energy management. To support these services, we designed an energy management framework, ParkingMeter, that provides a common place for all services to register and obtain a fair share of the energy resources. ParkingMeter integrates our previous Energy Governor for LoadingZones, which only managed the energy for that one specific service, by managing multiple simultaneous and diverse services. Services that want to be scheduled must provide some information about themselves to allow ParkingMeter to determine an efficient activity schedule. However, ParkingMeter keeps this energy awareness of the service to a minimum.

Given a vehicle's device that can support multiple services with different characteristics, the basic idea of ParkingMeter is to control the energy-saving modes of the device (i.e., when it is active or sleeping). With information about the services' costs and utility functions, ParkingMeter determines a schedule for when to activate the system. Additionally, by supporting a simple callback function for more complex services, ParkingMeter is flexible enough to fully integrate services whose utility definition requires customized behavior.

In this section, we first describe our system model and then provide an overview of the ParkingMeter architecture, with a particular focus on the three scheduling strategies.

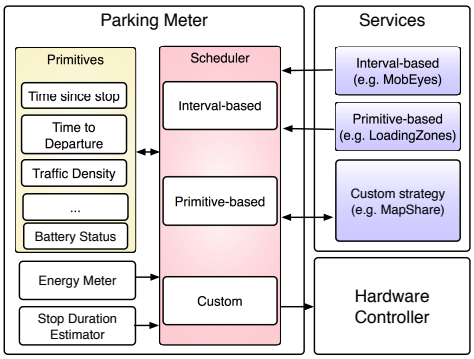

Figure 1: ParkingMeter Architecture

Finally, we describe how the central framework facilitates the fair distribution of energy among multiple services.

4.1 System Model

The base system model for ParkingMeter is similar to our energy Governor and includes three states, Active, Idle and Deep Sleep. However, the design of ParkingMeter is not limited to these states and can be expanded as needed. While these states are certainly differentiated by their energy consumption, it is also important to understand what functionality is available in each state.

In **Active** mode, all hardware components are active and all services are enabled and available. In this state, the system power is P_a.

For **Idle** mode, only low-power hardware is active, with a power footprint of $P_i << P_a$ but limited functionality. In this mode, the system can monitor a number of environmental characteristics or *primitives* that are only dependent on the low-power hardware. Essentially, primitives can be based on simple system parameters, enabling the system to track elapsed time since the beginning of the stop or estimated time to departure, or battery levels, enabling the system to determine available energy for a given stop. Additionally, other simple metrics can be added using low-power hardware, such as acoustic sensors that expose environmental noise levels. In our prototype, we add a low-power ZigBee radio to monitor traffic density.

The final state is the **Deep Sleep** mode, which is the most energy saving state that runs at $P_{ds} \approx 0$. In this state, no functionality is provided.

4.2 System Architecture

ParkingMeter sits between the system controller, which controls the system state transitions, and the services. ParkingMeter includes an energy meter to monitor the level of the battery and a stop duration estimator, as well as access to a number of primitives, all of which are accessible when the system is active or idle. In our current implementation, ParkingMeter's main primitives are traffic density based on monitoring the number of passing cars and timing information about the time of the stop.

One of the most important components for ParkingMeter is the *stop duration estimator*, which uses local history, GPS position, and statistics collected by a central server, to predict an estimate for the duration of a new stop, δ_s. While fine grained statistics are hard to obtain, although some promising effort is being pursued, the complexity estimation and the verification of its accuracy are out of the scope of this paper. Instead, we consider the output of the stop duration estimator to be a random Gaussian variable $\mathcal{N}(\mu_s, \sigma_s)$. The *stop duration estimator* uses a conservative approach by computing the estimated length of a stop as:

$$\delta_s = \mu_s + \sigma_s. \tag{3}$$

which guarantees that, for 80% of stops, parking duration is not underestimated.

For each primitive, ParkingMeter computes a cumulative distribution function (CDF) over the estimated duration of the stop. For some metrics, such as time passed since the beginning of the stop or until the departure, this metric is easily computed and solely based on the duration of the stop. Other primitives, such as traffic density, have more complex behavior and their distribution is a function of location, time of day and duration of the stop. For such primitives, ParkingMeter implements the necessary procedures to compute or obtain the CDF. For traffic distribution, when the vehicle is stopped, ParkingMeter connects to a central server that stores traffic logs, such as those found in [14]. This information is enough to compute an estimated CDF. The level of detail of this information can be enriched over time, especially for the most popular locations, by updating the data on the server as well as using a cache of the vehicle history on the system itself.

Finally, the central component of ParkingMeter, and its main differentiating component from our original Energy Governor, is the scheduler, which collects the stop duration estimation, the value of the primitives and the properties of the services that need to be be scheduled. The full system architecture is illustrated in Figure 1. To better accommodate the needs of the largest possible number of services, ParkingMeter offers three scheduling alternatives to any registered service: *primitive-based*, *interval-based* and *custom*. We first describe each of these schedulers and then discuss how ParkingMeter addresses the complexity of running multiple services with different scheduling requirements at the same time.

Primitive-based scheduler

This scheduler follows the algorithm described in [4] for the LoadingZones governor, and it is based on the utility of the service, $\mathcal{U}_s$. If no specific utility function is provided, ParkingMeter assumes a linear utility function.

To maximize at the same time the amount of time the service is active and its utility, the scheduler defines a threshold Th for the utility and activating the service every time $\mathcal{U}_s(t) > Th$. The identification of the right threshold that satisfies the properties of an optimal schedule is easier if we leverage the characteristics of the CDF of the metric. The CDF is a monotonic function and is much less susceptible to the noise that perturbs metrics such as traffic density. After determining what fraction f of the stop the service can be active, using (1), the threshold is computed as:

$$Th = CDF^{-1}(f), \tag{4}$$

where CDF^{-1} is the inverse of the CDF, or quantile function. Algorithm 1 describes the steps followed by the primitive-based scheduler.

Algorithm 1: Primitive scheduler for service s

```
E_av : available energy();
δ_s : stopEstimator→get Stop Duration();
int_min : s →get min active time();
p: s →get primitive();
CDF: p →get CDF();
/* compute f using (1)                    */
f: get active frac(E_av,δ_s);
/* compute threshold using (4)            */
Th: get threshold(CDF,f);
while parked do
    u: p → get_primitive_value();
    if u > Th then
        SetActive();
    else
        SetIdle();
    end
    /* repeat test after int_min          */
    wait(int_min);
end
```

Algorithm 2: Interval scheduler for service s

```
int_min : s →get minimum active time();
E_av : available energy();
δ_s : stopEstimator→get Stop Duration();
i : compute interval();
while parked do
    SetActive();
    /* stay active for int_min            */
    wait int_min;
    SetIdle();
    /* complete the period                */
    wait i − int_min;
end
```

Interval-based scheduler

Some services define utility based on the frequency of operation. For example, a participatory sensing service such as MobEyes must collect as many samples as possible, according to the available energy E_{av}, equally distributed over the total duration of the stop, at intervals i. The interval-based scheduler defines the optimal (i.e., smallest) interval at which the service should be activated autonomously. The registered service only provides the minimum continuous active time that should be guaranteed. For example, a sensing service should specify how long it takes to boot the sensors, collect the samples and process them (e.g. δ_{sample} for MobEyes).

With this information, the scheduler computes an optimal interval using (2), and then activates the system at intervals i, for a time δ_{sample}, until the car moves or all the available energy is consumed. The behavior of the interval-based scheduler is described in Algorithm 2.

Custom scheduler

For some services the utility is a non-monotonic function of one primitive or an aggregate of multiple primitives. For example, the utility for MapShare depends on the utility of uploading data collected during the last driving period, which decreases with the age of that data, and the utility of downloading new data, which is a function of how "fresh" the data will be when the car leaves the parking space. For MapShare, we can quantify the benefits of keeping the system alive at a given time combining two primitives: "time since stop", t_s, and "time to departure", Δ_s:

$$\mathcal{U}_{MS} = \max(-t_s, -\Delta_s) \quad (5)$$

For such services, ParkingMeter provides the *custom scheduler*. The procedure for these services is similar to that described for the primitive-based scheduler. However, in this case ParkingMeter requires the service itself to provide a function that returns the value of $\mathcal{U}$, for the service and the respective CDF. The procedure is described in Algorithm 3.

Algorithm 3: Custom scheduler for service s

```
E_av : available energy();
δ_s : stopEstimator→get Stop Duration();
int_min : s →get min active time();
p: s →get primitive();
CDF: s →get CDF();
/* compute f using (1)                    */
f: get active frac(E_av,δ_s);
/* compute threshold using (4)            */
Th: get threshold(CDF,f);
while parked do
    u: s → get U(p);
    if u > Th then
        SetActive();
    else
        SetIdle();
    end
    /* repeat test after int_min          */
    wait(int_min);
end
```

While this does require knowledge of energy conservation from the service designer to provide the metric $\mathcal{U}$ and its CDF, the overhead is lower than that required to implement an independent power management system. Additionally, the benefit of centralizing the scheduling of each service in ParkingMeter includes a more efficient distribution of the limited energy across multiple services with different requirements.

4.3 Combining multiple services

The benefits of a framework like ParkingMeter are most important when multiple services are provided on the same system and so must share the limited available energy. In this case, ParkingMeter focuses on fairness among services, and optimality for each service given its share of energy to use. Optimality within a service is defined by the service's utility function, as described in the previous section. When multiple services are active, it is likely that two or more services are scheduled at the same time. In this fortunate situation, their energy cost is shared, and the saved energy can be used to extend the schedule of the services.

Since an optimal solution requires perfect knowledge of stop duration, ParkingMeter applies effective heuristics to define the activity schedule. ParkingMeter initially takes a

Algorithm 4: Multi service schedule

```
E_a v : available energy();
δ_s : stopEstimator→get Stop Duration();
S: set of services that must be scheduled;
active_cnt = 0;
interval : max(s →get min active time(), ∀s ∈ (S));
foreach s in S do
    /* compute e_{av,s} using (6)                    */
    E_{av,s}: get_available_energy();
    run schedule (s,E_{av,s});
end
while parked do
    activeCount = 0;
    foreach s in S do
        if s.isActive() then activeCount++;
    end
    /* multiple services using the same energy
       */
    if active_cnt > 1 then
        reschedule(S);
    end
    wait(interval);
end
```

pessimistic approach and evenly divides the available energy among the services. This distribution could also take priorities into consideration by assigning each service a priority p_i. The available energy for each service, $E_{av,i}$ is computed as follows:

$$E_{av,i} = E_{av} \cdot \frac{p_i}{\sum_{\forall j \in S} p_j} \quad (6)$$

Each service then determines its optimal schedule for the given amount of energy for the stop.

Since this is a pessimistic solution that does not consider the frequent overlaps among services, ParkingMeter reassesses the available energy when two or more services overlap, sharing the same amount of energy and recalculates the schedule for each service. The behavior of ParkingMeter in managing multiple services is described by Algorithm 4.

To understand the impact of ParkingMeter's scheduling decisions, we measured the energy required by the schedule defined by ParkingMeter for LoadingZones, MapShare and MobEyes, individually, and compared it to a schedule that assumes no knowledge of other services and tries to use all of the energy for each service (see Table 1). In this case the system would require too much energy, and would cease working before the end of the vehicle stop. Given the preference of MapShare to wait until the end of the stop for downloading updates, this service would most likely be the most heavily affected, in fact, with all likeliness it will never be activated due to lack of energy. MobEyes would also produce too frequent samples initially, and leave a significant hole towards the end of the stop.

5. EVALUATION

Our evaluation of ParkingMeter is based on two factors: its capability to schedule efficiently individual services using either of its schedulers, and its ability of providing a fair and efficient schedule for multiple concurrent services. We compare it to two non-adaptive strategies. The first and most simple one, FrontLoad, activates the system at the beginning of the parking period and runs until the battery is completely discharged. The DutyCycle approach alternates active and sleep periods at regular intervals to guarantee an even distribution of the active time over the estimated duration of the stop. We evaluated different period lengths for DutyCycle, but only show the results for a period of 20 minutes. This guarantees that in each period, MobEyes can be active for the minimum sensing time, 2 minutes. For space constraints we do not display the results for other settings of these parameters. However, the adaptivity of ParkingMeter consistently achieved near-optimal schedules for every configuration we tested, while the performance of DutyCycle was extremely dependent on the choice of the two parameters.

We implemented the model, the ParkingMeter framework and the two other strategies in a MATLAB simulator. The flexibility of the simulated environment allowed us to generate a large number of scenarios and evaluate performance in ideal conditions. Furthermore, using our simulator, we were able to control the level of precision of our stop duration estimation, as well as the noise that affects the traffic density with respect to its average value provided by online databases.

Service	% energy used
LoadingZones	99.7%
MapShare	93.2%
MobEyes	84.7%
Combined	148%

Table 1: Energy used when combining services that are scheduled independently. Despite overlapping, the total energy used is well above the total available energy.

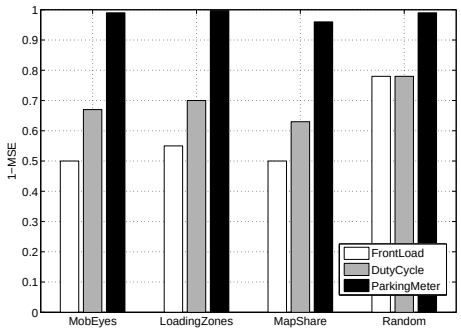

Figure 2: Proximity of the schedule produced by each strategy to the optimal schedule. 1 =**Perfect matching,** 0 =**no overlap.**

5.1 System Performance

To measure the performance of each schedule, we compute the similarity with the optimal schedule, computed *a posteriori*, as the MSE of the difference between two binary

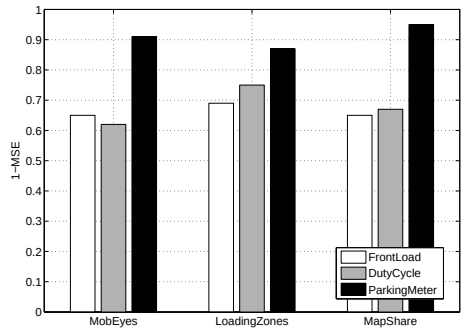

Figure 3: Proximity of the schedule produced by each strategy to the optimal schedule when the stop duration is a random variable.

vectors, one of which represents the status (active or not) of the service over time of the optimal schedule while the other represents the schedule produced by the evaluated strategy. The schedule computed by FrontLoad and DutyCycle are fixed and do not depend on the service properties. This has a negative impact on the effectiveness of their schedules even in ideal conditions and when only one service must be scheduled (see Figure 2, where a value equal to 1 identifies a perfect match between a schedule and the optimal solution). We individually scheduled MobEyes, MapShare and LoadingZones simulating many stops of different lengths, keeping the battery level constant. To limit the dependency of the specific traffic distribution that we used for the performance of LoadingZones, we also simulated a service based on a metric whose values are generated from a random process.

In all cases, the schedule produced by the adaptive approach of ParkingMeter is almost perfectly overlapping with the optimal schedule. FrontLoad and DutyCycle perform equally poorly in both configurations. Notably, DutyCycle has a slight advantage over FrontLoad in scheduling MobEyes, for which an optimal schedule is indeed a regular interleaving of active and sleeping times. However, as opposed to ParkingMeter, DutyCycle does not adapt the length of an active interval based on the information provided by the service.

For LoadingZones, not only the duration of the stop is important, but also the time of the day, which influences the traffic density values during the stop. The value shown in Figure 2 for LoadingZones, taken from [4], are an average over multiple simulations of the same duration, starting at different times of the day. We computed the standard deviation of the results for each run, which show an almost null value for the adaptive solution proposed by ParkingMeter, while the proximity between the DutyCycle and FrontLoad schedule and the optimal one show a very large dependency on the time at which the schedule starts.

5.2 Effect of Approximation

The duration of the stop, δ_s and in some case the values of the metrics, such as traffic density, are approximations based on statistical analysis of historical data. To investigate the negative impact of approximations on the performance of the various scheduling strategies, we ran our simulations generating δ_s as a random Gaussian variable $\mathcal{N}(\mu_s, \sigma_s)$.

Strategy	Energy Used	Average Similarity
FL	100%	0.52
DC	100%	0.65
PM-Individual	148%	0.98
PM-Conservative	78%	0.40
PM-Combine	100%	0.85

Table 2: Total energy used and average similarity to the optimal schedule for different strategies in multi-service systems.

As described in our system architecture, in an effort to guarantee service availability throughout the whole stop, we used a conservative estimate for duration, $\delta_{s,est} = \mu_s + \sigma_s$. We used the same heuristic for DutyCycle, while FrontLoad keeps using all energy at the beginning of the stop. The average results for several simulations with random values of δ_s, shown in Figure 3 for $\sigma = 0.1 \cdot \delta_s$, confirm the benefits of the adaptive approach used by ParkingMeter, which consistently achieves a schedule that is closer to optimal with respect to the non-adaptive approaches. FrontLoad has some advantage with respect to the non-random simulations because it always uses all energy, while the conservative approach taken by ParkingMeter and DutyCycle in many cases causes an under-utilization of the available energy. This is still preferable to using all energy before the stop ends, since this latter approach would cause an unfair distribution of the resources, penalizing those services such as MapShare that must be active towards the end of the stop. We evaluated the effect of errors between the primitives and their expected distribution (the CDF that ParkingMeter uses for its primitive-based strategy), and we observed a similar behavior, with ParkingMeter outperforming the other approaches and finding a schedule that overlaps with the optimal one for the largest part.

5.3 Combining Services

ParkingMeter provides a quasi optimal schedule for single-service systems. We also evaluated its ability to concurrently schedule multiple services that are registered on the same system. We measure the similarity of the produced schedule to the optimal one that each service would obtain if scheduled independently. The results are summarized in Table 2, where the average similarity for the three services is shown for each scheduling strategy.

Since in our model all services can be provided in the same *active* status, FrontLoad and DutyCycle are not affected at all by the presence of multiple services, and provide the same schedule as the single case. In a real deployment, the slightly higher hardware utilization due to the presence of multiple services would actually negatively affect the system energy consumption, but we consider this factor negligible in our evaluation. PM-Individual refers to a scheduler that assumes no knowledge of other services, and tries to use all of the energy for each service. The system would require too much energy too satisfy this schedule, and would cease working prematurely. Given the preference of MapShare to wait until the end of the stop for downloading updates, this service would most likely be the most heavily affected, in fact, with all likeliness it will never be activated due to lack of energy. MobEyes would initially produce too frequent

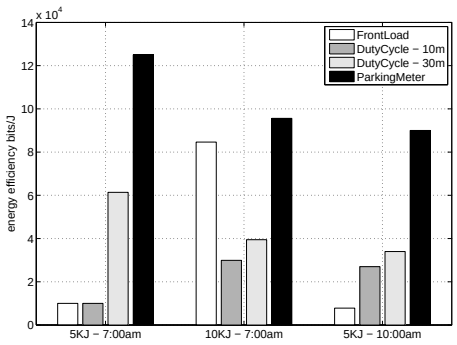

Figure 4: Energy efficiency in our prototype implementation.

samples, and leave a significant hole towards the end of the stop.

PM-Conservative is the overly conservative alternative to solve this issue, which assigns a third of the available energy to each service, and computes the optimal schedule for each based on that value. This approach is successful in meeting the goal of extending the lifetime of the system for the whole duration of the stop, but since this approach ignores the overlap of the schedules for each service, it ends up saving a lot of energy, with a negative impact on the benefits for each service, as shown by the very low similarity with the optimal schedule.

PM-Combine addresses both these issues by computing a new schedule every time two services overlap adapting the schedules and achieving a 100% energy utilization.

5.4 Prototype

While simulations can provide extensive exploration of the system space, it is important to understand how ParkingMeter works in a real system. Therefore, we implemented a prototype of ParkingMeter and tested it as part of the Illinois Vehicular Project using our implementation of LoadingZones as a test service. An evaluation of the efficiency of ParkingMeter in a real deployment is challenging due to the long time that each experiment would require and the large number of devices that would be required to test the scheduler for services like LoadingZones. Consequently, we ran the experiments in our lab by generating traffic according to traces retrieved from databases such as [14]. We tested different values for the available energy, and different stop times and duration, confirming how ParkingMeter can always adapt to these parameters and maximize its energy efficiency, while the performance of its competitors are strongly dependent on them. For example FrontLoad, which spends all the energy in the first part of the stop, works well if the stop begins during rush hour, but poorly if the car is parked overnight when traffic is low (see Figure 4).

We are continuing our effort for the integration of ParkingMeter in the IVP hardware and its deployment on a larger scale.

6. CONCLUSION

The successful deployment of distributed networks of services is a promising opportunity that vehicular networks, and in particular parked vehicles, can contribute to. However, a number of challenges must be addressed to make this vision a reality. One of the most important is to manage the challenging task of maximizing service availability even when limited energy and long stop duration make it necessary to enforce a sleep-active schedule. The co-existence of several diverse services sharing the same resources make this task even harder.

In this paper, we presented ParkingMeter, a framework that is capable of providing near-optimal scheduling for multiple services with a minimal effort for the service designers. ParkingMeter offers three scheduling strategies, which are capable of automatically scheduling services that rely on a number of primitives that the framework can measure. At the same time, it allows the service to provide more specific details about its requirements through the *custom scheduler*.

We demonstrate that the adaptive strategy of ParkingMeter achieves near-optimal solutions in an ideal scenario with perfect estimates of the stop duration. We also verified that the system is able to react to random errors with less performance loss with respect to its non-adaptive competitors. Finally, we provide the results of our experiments on a custom-designed hardware which not only confirms the performance of ParkingMeter, but also its ability to schedule our prototype service without requiring any change to its code.

We are working on improving the robustness of the implementation of ParkingMeter and adding more primitives to our hardware prototype to extend the range of services. We are also working on how to integrate collaboration between devices into the system to enable even more energy savings when multiple devices are offering the same service in the same location.

7. REFERENCES

[1] R. Crepaldi, R. Beavers, B. Ehrat, M. Jaeger, S. Biersteker, and R. Kravets, "LoadingZones: Leveraging Street Parking to Enable Vehicular Internet Access," in *Proceedings of CHANTS '12*.

[2] N. Liu, M. Liu, W. Lou, G. Chen, and J. Cao, "PVA in VANETs: Stopped cars are not silent," in *Proceedings IEEE INFOCOM'11*.

[3] U. Lee, B. Zhou, M. Gerla, E. Magistretti, P. Bellavista, and A. Corradi, "Mobeyes: smart mobs for urban monitoring with a vehicular sensor network," *IEEE Wireless Communications*, vol. 13, 2006.

[4] R. Crepaldi, R. Welsh, and R. Kravets, "Governing Energy for Parked Cars," in *Proceedings of WONS '13*.

[5] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *Proceedings of ACM SenSys '04*.

[6] M. Buettner, G. V. Yee, E. Anderson, and R. Han, "X-MAC: A Short Preamble MAC Protocol for Duty-Cycled Wireless Sensor Networks," in *Proceedings of ACM SenSys '06*, 2006.

[7] F. Ashraf, R. Crepaldi, and R. H. Kravets, "Know your neighborhood: A strategy for energy-efficient communication," in *Proceedings of IEEE MASS '04*.

[8] C. Sengul and R. Kravets, "Heuristic approaches to energy-efficient network design problem," in *Proceedings of IEEE ICDCS '07*.

[9] S. Das, "Avoiding Energy Holes in Wireless Sensor

Networks with Nonuniform Node Distribution," *IEEE Transactions on Parallel and Distributed Systems*, 2008.

[10] R. A. Williams and J. R. Dwark, "Apparatus and method in a network interface for enabling power up of a host computer using magic packet and on-now power up management schemes," 1999.

[11] E. Shih, P. Bahl, and M. J. Sinclair, "Wake on wireless: An Event Driven Energy Saving Strategy for Battery Operated Devices," in *Proceedings of ACM MobiCom '02*.

[12] L. Gu and J. Stankovic, "Radio-Triggered Wake-Up for Wireless Sensor Networks," in *Real-Time Systems*, 2005.

[13] (2013) TomTom Mapshare. [Online]. Available: http://www.tomtom.com/en_gb/maps/map-share/

[14] (2012) Massachusetts Department of Transportation. [Online]. Available: http://mhd.ms2soft.com/tcds/