Context-aware Battery Management for Mobile Phones: A Feasibility Study

Nishkam Ravi, James Scott* and Liviu Iftode
Department of Computer Science, Rutgers University, USA
*Intel Research, Cambridge, UK
{nravi@cs.rutgers.edu, jamesscott@acm.org, iftode@cs.rutgers.edu}

Abstract. In this paper, we propose a system for context-aware battery management that warns the user when it detects that the phone battery can run out before the next charging opportunity is encountered. At the heart of this system, are algorithms that predict: (1) when the next charging opportunity will be available, (2) how much call-time will be required by the user in the interim, and (3) how long the battery will last if the current set of applications continue to execute. We propose algorithms that process user's location traces and call-logs for making some of these predictions. We also propose a technique to predict battery consumption of applications. We present the design of the system and demonstrate its feasibility by experimentally showing that each of the prediction algorithms can perform with fairly high accuracy.

1 Introduction

Mobile devices such as smart phones are riding the wave of Moore's Law, providing increasing functionality due to rapid improvements in processing power, storage capacities, graphics, high-speed connectivity, etc. However, the main problem faced by these devices is *battery management*, since battery capacities are not experiencing the same exponential growth curve as other technologies such as processing power and storage. While there is ongoing research in discovering and exploiting ambient energy sources, it is highly likely that energy will remain the key bottleneck for mobile devices in the near future.

The main implication of battery management is the need for interacting and involving the user of the device in managing this resource. This interaction takes two main forms: (i) Users must be informed of the energy status of a device so they can decide how to prioritise amongst the various tasks that the device can perform (phone calls, gaming, messaging, media creation/playback, etc). (ii) Users must physically plug in a device and surrender its mobility for a period of time in order to charge it when necessary. The current solution for this user interaction takes the form of a battery meter, augmented with the ubiquitous "battery low" audio signals, and, on some devices, with a remaining time estimate at current power consumption.

This de facto user interface standard has remained relatively unchanged for a number of years. For mobile phones, this system is arguably sufficient for users

to get into habits of charging their devices at suitable periods, as they know their call patterns, and the devices have been optimised for low-power standby modes. For laptops, these devices are relatively rarely used in situations where power is unavailable, and, unlike with phones, users are accustomed to bringing the power adaptor with them.

However, a number of factors have conspired to change this comfortable status quo. Firstly, convergence is leading to more multi-functional computing devices with the always-on expectation of phones. Secondly, WLAN interfaces such as Bluetooth and 802.11 have become ubiquitous, and during data transfer (though not during idle) they are relatively hungry consumers of energy. Thirdly, pervasive computing applications have provided reasons for mobile devices to be executing always-on background applications which use sensing, computation and communication and, therefore, break the low standby-mode power profile that users have become accustomed to. The time has come to revisit the issue of battery management for mobile end-user devices.

The key observation that we make is that a simple "battery remaining" or even "time remaining" does not enable the user to make the right decisions as to the spend of the energy budget and the request for recharging. We illustrate this with a few anecdotes. Note that the use of "smart phone" in these anecdotes can be taken to mean "converged mobile device" and they are not necessarily phone-specific.

> BING BING! Alice's smart phone's alarm clock goes off. As she turns off the alarm, she notices that her smart phone is only 15% charged, annoying, since the charger was right by her bed but she forgot to plug it in when she went to sleep. Never mind, she has another charger at work. On the way in, she uses her phone's traffic-aware navigation application, connected to a Bluetooth GPS unit, allowing her to drive the most optimal route based on real-time traffic information. Her battery falls to 10%, a static threshold where her phone automatically turns off Bluetooth, causing her navigation software to silently fail to reroute her off the highway, leading her straight into a traffic jam.

There are two take-away messages here. The first is that the phone should have reminded Alice to charge it when she went to sleep, as the "cost" of losing mobility while charging would have been zero in that period. Sometimes, it does not make sense to maximise perceived battery life by waiting until the battery is low.

Nonetheless, Alice's phone actually still has plenty of energy, since she is just 20 minutes away from another *charging opportunity*. The heuristic that a low absolute battery level means scarce energy, causing Bluetooth to be turned off, is plainly incorrect in this situation. Ironically, this static threshold (common in current devices) may cause her device to run out of battery, as she may be sitting in that jam for some time.

> Bob's phone is at 80% charge, normally enough for 3 days. As he is getting ready for a 5000-mile flight, he decides to try out his phone's music

player and loads some songs onto it. However, by the time he boards the plane, his phone is already down to 50% battery level, due to his music listening and due to background services such as his home automation application, polling unfamiliar Bluetooth devices in the airport. Three hours into the flight, still listening to his music, his phone beeps to say it is at 10% charge and he turns it off, but nonetheless his battery is so low that it dies when he tries to rendezvous with his friend on landing. Consequently, he spends a frustrated half hour searching for a payphone and for a shop where he can get change.

The opposite happens here. 100% charge does not necessarily mean plentiful energy; the next charging opportunity may be many hours or days away. The background application, and even the foreground but *non-crucial* music application, have been allowed to drain the phone's battery to the point where the *crucial* application of telephony is unavailable at the time it is required. No warning that this would happen has been given until it was too late. The spectre of this scenario may cause many people to NOT make full use of the broad capabilities of smart phones, since they do not trust that applications they may download and run will not drain the battery in this way.

In this paper we propose a new context-aware battery management architecture for mobile devices (henceforth CABMAN), based on three principles:

- The availability of crucial applications to users should not be compromised by non-crucial applications.
- The opportunities for charging should be predicted to allow devices to determine if they have scarce or plentiful energy, instead of using absolute battery level as the guide
- Context, such as location information, can be used to predict charging opportunities

To achieve these goals, we propose three "prediction" algorithms that predict: the next charging opportunity, the call time requirements of the user over a period of time (assuming that telephony is the most critical application), and the "discharge speedup factor" of the set of non-crucial applications running.

We present CABMAN's system design (Section 3) followed by an evaluation of the system using real devices and real traces of human mobile phone activity (Section 4), followed by related work (Section 5) and conclusions (Section 6).

2 Problem Definition

The key role of our battery manager is to be able to answer the query: *Will the phone battery last until the next charging opportunity is encountered?* In order to answer this query without involving the user, the battery manager should be able to answer the following three questions (1) when the next opportunity for recharging the battery will be available and hence what is the total battery lifetime available to the user? (2) what fraction of this battery lifetime will be

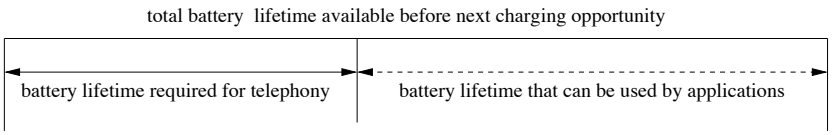

Fig. 1. Battery lifetime model

consumed by critical applications such as telephony? and (3) what fraction of this battery lifetime can be left for use by non-critical applications? This battery lifetime model is captured in Figure 1. For the purpose of this paper, we assume that telephony (i.e making and receiving calls) is the critical application and the daemon applications running on the phone are non-critical.

We identify three subproblems: (1) prediction of the next charging-opportunity, (2) prediction of the calltime that might be required by the user in the interim, and (3) prediction of how long the battery will last if the current set of applications continue to execute on the device. With this knowledge, the battery manager will be able determine if the user will run out of battery sooner than they should, and ask them to terminate one or more applications or look for a charging opportunity. In order to solve these three subproblems, we need a system that can monitor user context and sense battery level of the device; a set of algorithms for making predictions and a central component for assimilating the information together and warning the user appropriately. In the following section, we present the design of such a system along with the set of prediction algorithms.

3 CABMAN System Design

CABMAN consists of eight components (Figure 2) divided into three categories: system-specific monitors, predictors, and the viceroy/UI, which we now discuss in turn, paying particular focus to the three prediction algorithms (for charging opportunities, call time needs, and battery lifetime), and how they are accomplished. These algorithms are evaluated in Section 4.

3.1 System-specific Components

In order to perform context-aware battery management, we need to detect various data from the device's operating system, including battery status, list of processes running on the device, the calls made on the device (under the assumption that this will be regarded as a "crucial" application), and finally, some context information to allow us to predict the next charging opportunity. The *Process monitor* is responsible for keeping track of the processes running on the device and informing the viceroy whenever a new process is detected. *Battery monitor* probes the battery periodically and enquires about remaining charge

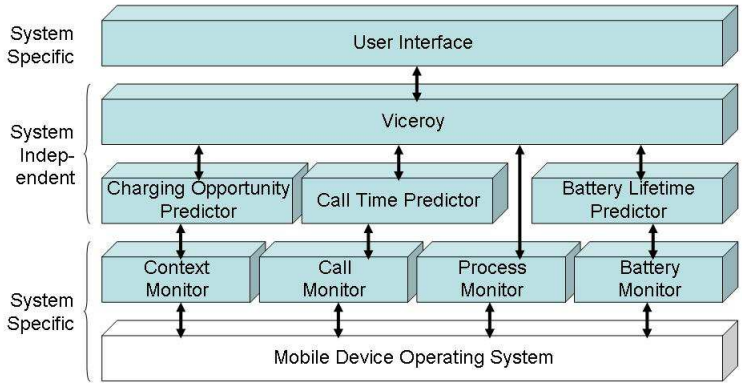

Fig. 2. CABMAN system architecture

and voltage level. *Context Monitor* is responsible for sensing and storing context information (such as location) on the phone. *Call Monitor* logs communication (incoming/outgoing calls, incoming/outgoing SMSs).

The purpose of separating out system-specific functionality is to make clear the requirements that CABMAN has from the underlying operating system, and thus facilitate porting of CABMAN to the multiple platforms used for mobile phones (e.g. Symbian, Linux, Windows Mobile).

3.2 Charging Opportunity Predictor

The crux of CABMAN is that a phone should determine if a charging opportunity is near enough for the battery to "last" until then (by which we mean maintaining the ability to execute the *crucial application*). If this is true, then CABMAN should not inconvenience the user with unnecessary warnings or actions such as going into low-power modes with reduced functionality. If this is false, then even if the phone battery is relatively full, CABMAN should warn the user that they risk a dead battery.

Since a growing number of mobile phones can sense their location, either through GPS or beacon-based techniques such as Place Lab [10], we have chosen to focus on location sensing as a way of inferring charging opportunity in this preliminary feasibility study. The disadvantage of using *only* location information in inferring charging opportunity, is that it does not accomodate for mobile chargers (such as those in cars). Additional context information, such as time-of-

day, speed, presence of other wireless devices, and charge-logs can help alleviate this problem. In this preliminary study, we only evaluate with respect to static charging opportunities.

Cell-based charging opportunity prediction algorithm Many phones are capable of detecting the id of the current cell that the phone is connected to. We therefore propose an algorithm that makes use of this data. If richer location information were available, either based on more cells, detection of other beacon types (WiFi APs, etc), or direct positioning information (GPS, A-GPS, etc), that could be used instead, however fewer phones currently have such capabilities. The basis of this algorithm is that certain cells (e.g. those at home or perhaps the workplace) are marked as being charging opportunities, and the time until the user is expected to reach those cells is used as the prediction. The choice of which cells are marked is easily accomplished using the battery monitor to determine when charging occurs and marking the cells in which this normally occurs. User feedback can also be implicitly integrated — when the phone asks for more charge, if the user often "refuses" by not charging the phone, then the cell can be unmarked.

The prediction of when marked cells will be reached is accomplished by pattern-matching the current pattern of cell movements against a larger historical set of cell movement patterns. We represent the current pattern by using a number of *samples* being the current and most recent cell ids to which the phone has been associated. For the historical set we use a rolling *history* of a number of days of cell movement patterns. The appropriate choice of sample and history sizes are the subject of experimentation in Section 4.

Intuitively, the algorithm proceeds as follows. If the current samples are, say, ABC, then a search is made through the history of all traces that contain the sequence ABC (say $DEABCFG$), and for each of the resulting traces, the time will be determined between the time of entry of the current cell and the time of entry of the next charging-capable cell. These times are then averaged to provide a prediction of the current time-until-charging-opportunity.

Formally, let a location trace tr be denoted as a sequence of tuples: *(l,t,c)*, where l denotes location, t denotes the time when the user entered that location and c is either 1 or 0 depending on whether or not the user can charge the phone at that location. Let the j^{th} location trace tr_j be denoted by (l^j, t^j, c^j) let the i^{th} tuple in this trace be denoted by (l^j_i, t^j_i, c^j_i) and let the last tuple in this trace be denoted by $(l^j_{last}, t^j_{last}, c^j_{last})$. For a given location trace (l^j, t^j, c^j), let t^j_{charge} denote the time when the user enters the next charging-location. A location trace tr_j is said to be *contained* in another location trace tr_k if $\exists i(l^j_0 = l^k_i, l^j_1 = l^k_{i+1}, .., l^j_{last} = l^k_{i+last})$. Given a location trace $(l^{today}, t^{today}, c^{today})$, the time interval before the next charging opportunity becomes available is estimated as $(\sum_j (t^j_{charge} - t^j_k))/N | (contained(today, j), l^j_k = l^{today}_{last})$, where $contained(today, j)$ implies that tr_{today} is *contained* in tr_j, and N is the total number of traces for which $contained(today, j)$ is true.

This statement of the algorithm hides a complicating issue: A hardly moving or stationary phone still changes cell ids, causing the sample patterns to be repetitive strings. We handle this using an automated detection algorithm which finds pairs of cell ids for which repetitive and relatively high-frequency (minutes not many-hours) back-and-forth changes are observed. These cell ids are then coalesced into a virtual cell id representing both cells. This procedure is iteratively run so that multiple cells can be coalesced into a single virtual id. This procedure is similar to those employed by beacon-based place detection [5] where many cells can be marked as corresponding to a single "place."

3.3 Call Time Predictor

We regard telephony as the "crucial application" for mobile phones, in that users always want to be able to use this application (e.g. for emergency calls or rendezvous with friends). The "non-crucial applications" should not be allowed to drain the battery to the stage where the user is deprived of telephony service. Other applications may be deemed "crucial" (e.g. the navigation application used in our first motivating example), however for this paper we focus on telephony as the most compelling example.

To protect the availability of telephony, we need to predict the call time needs of the user. There are a number of options in order to achieve this. A simple method would be to ask the user could set a minimum call time level which they always like to maintain. However, this is not dynamic, so a user must continually ensure this setting is up-to-date. A more complex but dynamic method, which does not require user input, is to use past calling behaviour to find the average number of minutes of call time that the user needs during each hour of the day, and to use this to get an upper bound on the total call time required within a given time interval. This can be enhanced by viewing weekdays and weekend days separately since call behaviour is likely to differ in that time.

Formally, let a call-trace *calltr* be denoted as a sequence of tuples: *(h,t)* where h denotes the hour of theday and t denotes the calltime used in that hour. Let the j^{th} calltrace $calltr_j$ be denoted by (h^j, t^j), let the i^{th} tuple in this trace be denoted by (h_i^j, t_i^j). The calltime to be used in a given hour h_k is estimated as $(\sum_j t_k^j)/N$, where $calltr_j$ is a past call-trace, and N is the total number of past call-traces selected (e.g over the last three months). This algorithm is tested in the evaluation section and shown to work well.

A third call time predictor is to use a hybrid of the two listed above (or others) to achieve a conservative prediction. For instance, we could use the policy "keep twice my average call time available, and a minimum of 10 minutes for emergencies in addition to the predicted call time". The correct choice of policy depends on user preferance such as the user's perceived annoyance at being unable to make a call versus their perceived annoyance at having to charge their phone more often (which is the tradeoff being contemplated).

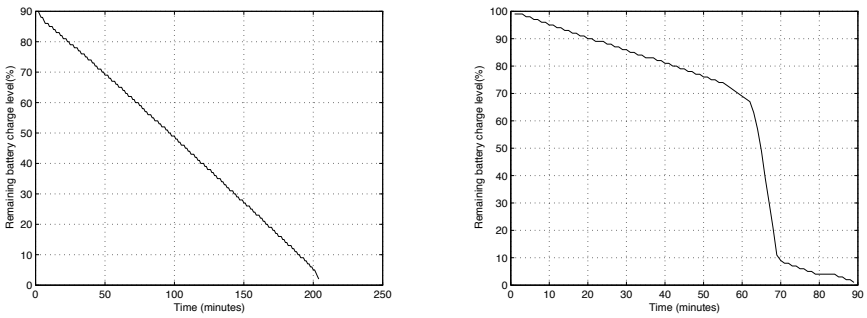

Fig. 3. Base curve for a new HP laptop (left) and old Dell laptop (right)

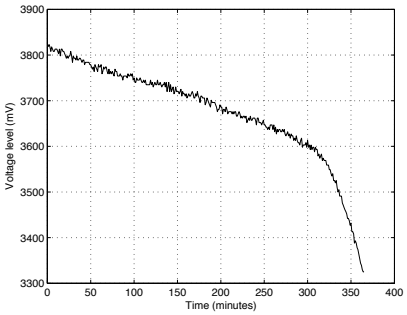

Fig. 4. Base curve for an HP iPAQ

3.4 Battery Lifetime Predictor

Battery management software and hardware has developed recently and may provide accurate estimates of the charge level left in the battery. However, this by itself is insufficient to predict lifetime accurately, since applications may vary their battery demands over time. Also, batteries have different chemistries with different reactions to types of load (constant, spiky, etc) and they age. In this section, we propose a battery lifetime metric that is independent of battery age and takes into account applications' battery usage.

One obvious and oft-used method of predicting battery lifetime is to monitor the rate of drain of the battery and extrapolate this linearly to exhaustion. However, due to the factors mentioned above, this is not reliable. To illustrate this, we can examine the battery discharge curves of a number of devices in idle mode (i.e. not running any applications, but with no power-down power management software active). We call this diagram the *base curve*. Figures 3 and 4 show base curves for a new laptop, an old laptop and an HP iPAQ. Since

people do not replace old batteries, either by choice or due to affordability, we found it important to include an old battery in our experimental setup.

As we can see, while the base curve of new laptop looks linear, that of an old laptop is highly non-linear (consistently so over many iterations). At the same time, the base-curve of the PDA (which reported just the voltage level) was also non-linear and spiky. We were not yet able to obtain traces involving a mobile phone, but the battery technologies used are similar (Li-Ion).

Our approach is to compare the actual discharge when applications are running against the measured base curves in order to predict for battery life. This requires a one-time offline measurement of the base curve, which a device can do for itself during a period where the user does not need it (similar to battery reconditioning that users perform today). This can be repeated periodically (e.g. on the order of months) to make sure that the changing performance of the battery over its lifetime is compensated for. Our algorithm proceeds as follows: with a given set of applications running, we measure the *discharge speedup factor* over a particular drop in battery energy (or voltage). This is calculated by comparing the time it would take for the battery to reduce by that amount during idle (from the base curve), divided by the actual time that it took for the battery to drop by that amount. In other words, with applications running, we measure the battery capacity c_1 and c_2 at two time instances t_1 and t_2 respectively; we find time instances t_3 and t_4 that correspond to battery capacities c_1 and c_2 on the base curve. The *discharge speedup factor* is calculated as $(t_4 - t_3)/(t_2 - t_1)$.

We then divide the remaining lifetime of the battery if the device were idle (from the base curve) by the discharge speedup factor, to obtain the predicted remaining time for the battery. As we shall see in the evaluation section, this has proven to be a very accurate measure of battery lifetime remaining.

Although this method allows for the on-the-fly measurement of a set of applications, one question that might be asked is: can we predict the discharge speedup factor for a new set of applications that have not previously been observed together? This would allow us to warn the user immediately as they attempted to start a new application, rather than having to wait until that application consumed some power. However, we have discovered that this is not easily possible, because the discharge speedup factors of applications cannot simply be added. Intuitively, for applications that do not heavily share system resources (e.g movie player and web server), energy usage can be added, while for applications that commonly share system resources (e.g movie player and audio player), it is not possible to add energy usage profiles. However, once the node has an opportunity to measure the discharge speedup factor, it can be cached and later used to flag immediate warnings to the user when they attempt to start a new application.

3.5 Viceroy and User Interface

The viceroy is CABMAN's central component. It uses input from the predictors and directly from the process monitor, in order to decide when action must be taken using some form of user interface (UI). The main job of the viceroy is to

continually monitor whether the battery lifetime prediction, combined with the battery requirement of the estimated call time requirement from the call time predictor, means that the battery will expire before or after the next charging opportunity. If the energy level is not sufficient to last until the next charging opportunity, then the viceroy must use the UI (audible or visual signals) to notify the user. Formally speaking, the user should be warned if $t > r - f(m)$, where t is an estimate of the time interval before the next charging opportunity surfaces, r is an estimate of the remaining battery lifetime, m is an estimate of the required calltime and $f(m)$ is the map from call time to battery lifetime.

When warned, the user may be able to correct matters by (i) killing some battery-hungry applications (up to and including powering down the device as a whole), (ii) change their behaviour so as to make less power demands of the device, (iii) plan to charge the device according to the timescale that the viceroy predicts the device will last, or (iv) accept and understand that they may lose the ability to make calls (or, in general, execute critical applications).

When the user is *at* a place with a charging opportunity (as may often be the case, e.g. for users with a home charge and office charger), the viceroy's job is to decide whether to use the UI to ask the user to charge the device, or whether the battery level is sufficient to reach the next charging opportunity.

4 Evaluation

We have implemented a CABMAN prototype for Linux using Java, Perl, shell-scripts and C++. The purpose of this prototype was to carry out a feasibility study by evaluating the performance of the prediction algorithms. The ability to make predictions with acceptable errors is a key indicator of the real-life performance of CABMAN.

4.1 Charging Opportunity Predictor

To evaluate the performance of the charging opportunity predictor and call time predictor we used a large trace of measurements of real users captured in MIT's Reality Mining project [1] which in turn used context logging functionality from the University of Helsinki. This excellent data set was gathered by deploying Nokia 6600 phones to more than 80 subjects for around nine months.

For charging opportunity, we used the cell id logs of the phones, and first used the previously-described clustering algorithm to create a smaller set of "virtual cells" for each device, where cells in the same physical location were clustered. Then, by hand, we identified out virtual cells which corresponded to charging opportunities ("charging stations"). For half the subjects, we chose a single charging station where the device spent most nights. For the other half, we chose two charging stations where the device spent most nights, and where the device spent most time during the day. In real life deployment, charging stations could be identified by monitoring the cells where charging actually took place.

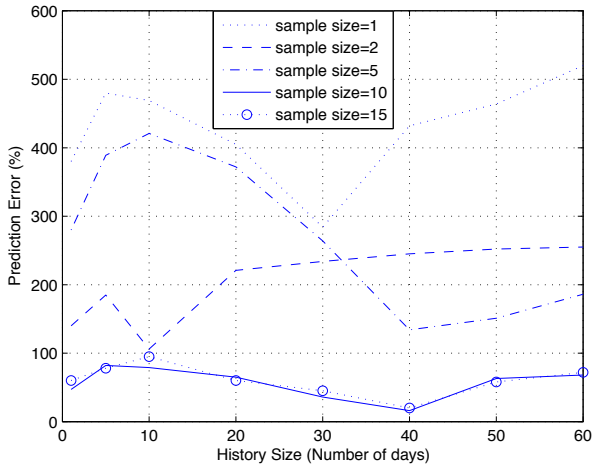

Fig. 5. Charging opportunity prediction error for various sample sizes and history sizes

We report the percentage prediction error averaged over the traces of all subjects, varying our algorithm's parameters of *sample size* and *history size*, in Figure 5. As we expect, an increasing sample size generally increases accuracy (the curve is lower) and reliably (the curve wavers less) as more closely-fitting historical data is used in prediction, with a sample size of 1 being only the current cell id used, and a sample size of 15 being the previous 15 (possibly "virtual") cell ids used. The effect of increasing sample size appears to bottom out at around 10. From a tested range of 1–60 days, history size appears to be optimal at 40 days; intuitively more history provides a longer averaging period, but longer-term changes in user behaviour (e.g. the change of home or workplace) mean that use of too much history has a negative effect.

Using the parameters of 10 samples and a 40 day history, the average prediction error across the 80 user, 9 month trace is 16%, which corresponded to an absolute error of 12 minutes on average. This indicates that our charging opportunity prediction algorithm is highly likely to give useful results in practice, and is a key result for the feasibility of CABMAN.

We also evaluated the algorithm by conditioning it on the day type (weekend or weekday), that is, if the current location trace is that of a weekday then we compare it against the past location traces of only weekdays. We did not notice any significant difference in performance.

4.2 Call Time Predictor

The Reality Mining traces are also used to evaluate the various possible algorithms for prediction of the call time needs of users. Using the call logs of these

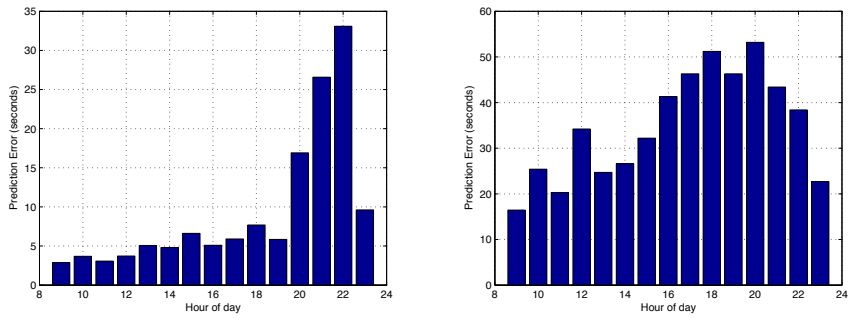

Fig. 6. Absolute call time prediction error for weekdays (left) and weekends (right)

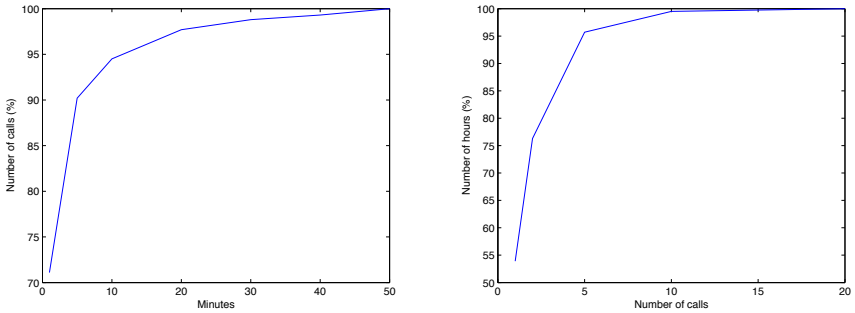

Fig. 7. CDF of the length of phone calls (left) and the number of calls made during each hour (right)

traces, we can execute the prediction algorithm described previously, where the number of minutes of call time used in a given hour of the day is predicted by the average number of minutes used in previous days. The average prediction error of this algorithm is shown in Figure 6. We can see that, as one might expect, it is easier to predict call time requirements for the middle of the day than it is for evenings (where users typically make many calls), and easier for weekdays than weekends. However, even in the worst case the average prediction error is under a minute out of the hour.

The low prediction errors seem suspiciously good, until one examines the actual calling pattern of users, as shown in Figure 7. These CDFs show that the typical call is short (71% calls are less than a minute, 90% calls are less than 5 minutes), and in a typical hour very few calls are made (75% with 2 calls or fewer), although both of these curves have a "long tail", which account for the occasional long call. While we obviously cannot account for the latter case (how can we predict an incoming call from an occasionally-in-contact friend

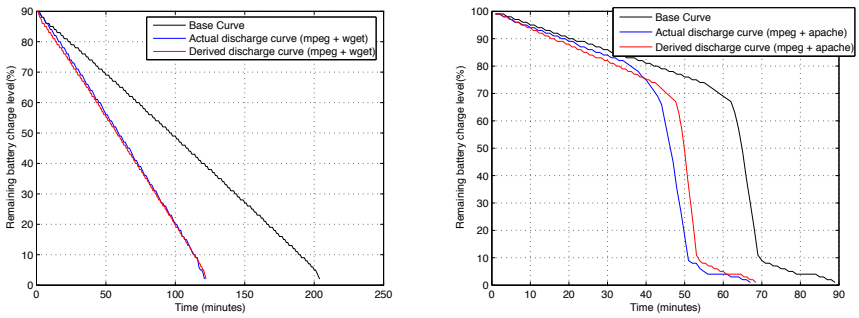

Fig. 8. Base curve together with discharge curves (actual and derived) for the new HP laptop (left) and old Dell laptop (right)

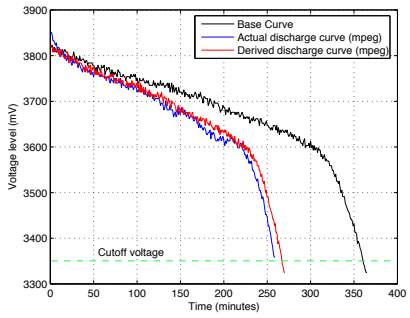

Fig. 9. Base curve together with discharge curves (actual and derived) for HP iPAQ

when humans cannot?), the type of functionality we are looking to preserve in CABMAN is the crucial application of telephony for rendezvous, emergencies, or suchlike. The user does have a choice to specify the additional amount of call time they would like to reserve over and above the predicted call time.

Therefore, the short calls forming the majority of usage are those that interest us, and the algorithm presented is shown to provide a feasible way of dynamically estimating the call requirements of the users in this large trace. Particularly when combined with user-specified minimum thresholds for call-time-remaining (as discussed previously), we believe that CABMAN can make useful estimates for user needs for the crucial application of telephony.

4.3 Battery-lifetime Predictor

In order to evaluate the performance of the battery-lifetime predictor, we experimented with three different machines: an HP laptop with a new battery,

a Dell laptop with a very old battery and a regular HP iPAQ PDA. We first obtained the *base curves* for all the three machines as shown in Figures 3 and 4. Next, we obtained the actual discharge curves for a set of applications (web, music and video) and all combinations of these applications running together (7 combinations) on all three machines. We simultaneously ran our own prediction algorithm for battery lifetime (using a 2 minute observation window to obtain a prediction), and monitored the device's own battery lifetime prediction via the ACPI interface.

As previously described, our prediction algorithm essentially calculates the *discharge speedup factor* based on observation of the running set of applications, and uses it to predict the remaining battery lifetime. In the process, the algorithm implicitly derives a predicted discharge curve for the given set of applications, which is useful in order to illustrate the performance of the algorithm.

Figures 8 and 9 show the base curve for the three devices together with the actual and derived discharge curves for a web application and movie player executing together. In all cases the actual discharge curve and the derived discharge curve track each other closely, showing that our algorithm performs well. Note that in the case of iPAQ, the discharge curve is a measure of the instantaneous voltage level as opposed to the battery charge level as in the case of laptops. The iPAQ fails when the voltage level falls below a certain threshold. We have not yet been able to test this algorithm on a mobile phone platform, but we expect similar performance due to similar battery chemistry (Li-Ion) and similarly multi-functional operating systems on newer phones.

Table 1. Comparing accuracy of our algorithm with ACPI's

Machine	Average Prediction Error	
	Our algorithm	ACPI
New HP laptop	1.2%	23.5%
Old Dell laptop	6.1%	120.2%
HP iPAQ	3%	52.2%

Table 1 summarises the performance of our battery prediction algorithm against the devices' own battery lifetime predictors (obtained via ACPI), across experiments using all 7 combinations of applications. Our algorithm clearly wins, with average errors of 1%, 3% and 6% while ACPI's estimates are (on average) over 100% off (i.e. predicting more than double the actual lifetime) in the case of an older battery with a non-linear discharge curve. In absolute terms, we are able to predict battery lifetime with an average accuracy between 4 minutes for the new laptop and 12 minutes for the iPAQ. This experiment shows that our battery prediction algorithm is capable of providing accurate enough information to support the feasibility of CABMAN.

5 Related Work

Prior research on dealing with the limited battery lifetime problem has concentrated on optimizing energy at different levels of the stack, starting with hardware all the way up to the application layer [6, 3, 4], including compiler-based energy optimizations [8, 7]. There has been limited research on managing battery and treating energy as a first-class operating system resource. To the best of our knowledge, *ECOSystem* [15] is the only piece of work that takes this approach. On mobile devices, the interfaces that inform the user of the battery levels (such as ACPI [2]) have not kept up with the evolution of the capabilities of these devices for context-sensing, and the always-on multi-functional roles they promise to play in the near future.

There is some literature on predicting location of the user based on mobility traces [5, 9]. It is worth noting, however, that predicting the location of the user is not always the end goal. In our case, location traces are utilized to predict when the next charging opportunity will be available, and hence the algorithm had to be tailored to make this prediction accurately. We are not aware of any research on processing user call logs to predict future calltime requirements.

Our battery-lifetime prediction algorithm outperforms existing algorithms such as ACPI's [2]. Prior research on predicting battery lifetime has focussed on the use of analytical methods in studying battery characteristics and deriving models for battery-lifetime prediction offline [11–13]. There has also been some research on making online predictions based on the execution history of applications [6] under dynamically changing workloads. The idea of profiling the battery in idle mode offline to make predictions online has also been proposed [14]. Our approach is simpler, more accurate and uses the idea of a *discharge speedup factor* to predict the remaining battery lifetime for constant workloads online and in an application independent manner.

6 Conclusions

We have described the motivation behind our context-aware battery management system CABMAN, in that battery management using only the current battery level as an indicator of scarcity is both too conservative in some situations and too optimistic in others. We describe three key components of CABMAN: (1). the use of context information such as location to predict the next *charging opportunity*, (2). more accurate battery life prediction based on a *discharge speedup factor* and (3). the notion of *crucial applications* such as telephony. For each of these components, we have proposed a new prediction algorithm, and we have evaluated these algorithms using traces of real users and against experiments on real devices. Our test results are very positive, with charging opportunity prediction exhibiting an average error of 12 minutes, battery life prediction having average errors of between 4 and 12 minutes depending on the device used (significantly outperforming the standard prediction algorithms on the devices tested). For call time prediction, we show that users call patterns

are typically quite sparse, so our prediction algorithm has average errors measured in seconds, though in actual deployment a "minimum call time remaining" setting by the user is likely to be useful.

The next and obvious step for CABMAN is to integrate these algorithms onto a mobile phone platform, and then deploy it and study its use; until then, we have not quantified the value of CABMAN to users, which is the ultimate measure of the system.

References

1. Reality Mining, http://reality.media.mit.edu/.
2. Advanced Configuration and Power Interface (ACPI), http://www.acpi.info/.
3. M. Anand, E. B. Nightingale, and J. Flinn. Ghosts in the machine: interfaces for better power management. In *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, 2004.
4. R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H.-I. Yang. The case for cyber foraging. In *Proceedings of the 10th workshop on ACM SIGOPS European workshop*, 2002.
5. K. L. et al. Adaptive on-device location recognition. In *Proceedings of Pervasive*, 2004.
6. J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *SOSP '99: Proceedings of the seventeenth ACM symposium on Operating systems principles*, 1999.
7. T. Heath, E. Pinheiro, J. Hom, U. Kremer, and R. Bianchini. Code transformations for energy-efficient device management. In *IEEE Transactions on Computers*, 2004.
8. U. Kremer, J. Hicks, and J. Rehg. A compilation framework for power and energy management on mobile computers. In *Proceedings of the 14th International Workshop on Parallel Computing (LCPC'01)*, 2001.
9. J. Krumm and E. Horvitz. Predestination: Inferring destinations from partial trajectories. In *Proceedings of Ubicomp*, 2006.
10. A. LaMarca, Y. Chawathe, and S. C. et al. Place lab: Device positioning using radio beacons in the wild. In *Proceedings of Pervasive*, 2005.
11. D. Panigrahi, S. Dey, R. Rao, K. Lahiri, C. Chiasserini, and A. Raghunathan. Battery life estimation of mobile embedded systems. In *VLSID '01: Proceedings of the The 14th International Conference on VLSI Design (VLSID '01)*, 2001.
12. D. Rakhmatov, S. Vrudhula, and D. A. Wallach. Battery lifetime prediction for energy-aware computing. In *ISLPED '02: Proceedings of the 2002 international symposium on Low power electronics and design*, 2002.
13. P. Rong and M. Pedram. Remaining battery capacity prediction for lithium-ion batteries. In *Conference of Design Automation and Test*, 2003.
14. Y. Wan, R. Wolski, and C. Krintz. Online prediction of battery lifetime for embedded and mobile devices. In *Proceedings of PACS*, 2003.
15. H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. Ecosystem: managing energy as a first class operating system resource. In *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, 2002.