# Informed Mobile Prefetching

Brett D. Higgins* Jason Flinn* T.J. Giuli†
Brian Noble* Christopher Peplin† David Watson†‡
University of Michigan* Ford Motor Company† Arbor Networks‡

## ABSTRACT

*Prefetching is a double-edged sword. It can hide the latency of data transfers over poor and intermittently connected wireless networks, but the costs of prefetching in terms of increased energy and cellular data usage are potentially substantial, particularly for data prefetched incorrectly. Weighing the costs and benefits of prefetching is complex, and consequently most mobile applications employ simple but sub-optimal strategies.*

*Rather than leave the job to applications, we argue that the underlying mobile system should provide explicit prefetching support. Our prototype, IMP, presents a simple interface that hides the complexity of the prefetching decision. IMP uses a cost-benefit analysis to decide when to prefetch data. It employs goal-directed adaptation to try to minimize application response time while meeting budgets for battery lifetime and cellular data usage. IMP opportunistically uses available networks while ensuring that prefetches do not degrade network performance for foreground activity. It tracks hit rates for past prefetches and accounts for network-specific costs in order to dynamically adapt its prefetching strategy to both the network conditions and the accuracy of application prefetch disclosures. Experiments with email and news reader applications show that IMP provides predictable usage of budgeted resources, while lowering application response time compared to the oblivious strategies used by current applications.*

## Categories and Subject Descriptors

D.4.4 [**Operating Systems**]: Communications Management—*network communication*; D.4.8 [**Operating Systems**]: Performance

## Keywords

Mobile prefetching, budgeted resources, adaptation


*MobiSys'12,* June 25–29, 2012, Low Wood Bay, Lake District, UK.


## 1. INTRODUCTION

Prefetching is a fundamental technique for improving application performance. It is employed in numerous domains, including computer architecture [28], databases [22], file systems [13], and distributed systems [12]. In essence, a prefetching system predicts what data a higher system layer, such as an application, will request and speculatively retrieves and caches that data in anticipation of future need. If the prediction is correct, the cached data is provided on request — this improves performance by eliminating the fetching of the data from the critical path of servicing the request. However, if the prediction is incorrect, prefetching consumes resources that could be used for other activities. Most prefetching systems use heuristics to balance these concerns; such heuristics decide when and how much data to prefetch.

In many ways, mobile computing is an ideal domain for prefetching. Applications that run on smart phones and tablets frequently fetch data from the cloud. Yet, mobile devices must often rely on wireless networks, and such networks can exhibit low bandwidth, high latency, and intermittent connectivity. By prefetching and caching data, a mobile application can avoid on-demand use of poor and unreliable wireless networks, leading to a substantial improvement in interactive response time. Thus, the potential rewards of prefetching are high.

Unfortunately, the potential costs of mobile prefetching are also high. Since wireless network bandwidth is limited, care must be taken that prefetching does not interfere with interactive traffic and degrade application response time. Further, prefetching data from a remote server can consume battery energy and cellular data allotments, so profligate prefetching may result in a mobile device's battery expiring too soon or in overage charges on a cellular plan.

Unfortunately, balancing costs and benefits is complex. The prefetching system must consider at least three different concerns (performance, energy usage, and wireless data consumption), no two of which are measured by a common metric. It must understand how the costs of fetching data from a remote server vary as wireless network conditions change, and it must take into account network-specific oddities such as 3G tail time [24]. It must account for differences in hit rates caused by variation in behavior across users, applications, and even among different classes of data within the same application. Finally, it must ensure that prefetching does not substantially degrade foreground network activity.

Given this host of complexities and the limited resources of many mobile application developers, it is understandable that no current mobile application addresses all (or even most) of the above concerns when deciding when to prefetch data. Instead, mobile applications employ simple, yet often suboptimal, heuristics such as prefetch nothing, prefetch everything, or prefetch data subject to an arbitrary constraint (such as only prefetching data items with size

less than 32 KB). Our evaluation shows that use of such heuristics represents a substantial missed opportunity.

Instead, we propose that the mobile computer system provide explicit prefetching support to all applications. Our prototype system, which we call IMP (Informed Mobile Prefetching), is structured as a library to which any mobile application may link. The application interface is unobtrusive; applications must specify the items that could potentially benefit from prefetching, and they must inform IMP when those items are consumed or are no longer needed. The IMP library manages the complex mobile prefetching process, guiding applications to prefetch the right amount of data at the right time in a manner that improves performance while still meeting wireless data and battery lifetime goals. IMP both helps overzealous applications rein in prefetching to meet user budgets and allows tentative applications to make better use of available resources to improve the user's experience.

A key observation behind the design of IMP is that many of the prefetching complexities detailed above have been addressed individually, albeit not always within the domain of mobile computing. We view the major contributions of IMP as unifying these disparate ideas, applying them specifically to the domain of mobile prefetching, and providing prefetching services through a simple, easy-to-use interface that is well-suited for rapid mobile application development.

The design of IMP is inspired by Transparent Informed Prefetching (TIP), which employs a cost-benefit analysis to determine when to fetch data from an array of disks into a file cache [23]. Like TIP, IMP uses a shadowing strategy to predict hit rates for different prefetch depths. IMP uses a substantially different algorithm than TIP's to address unique features of mobile prefetching. IMP considers not just performance but also energy and wireless data usage. IMP also considers how aggregation of multiple prefetches can reduce energy usage. Finally, whereas cache space is the bottleneck resource for disk prefetching, IMP considers wireless network usage and its associated resource costs to be the bottleneck in mobile prefetching.

IMP leverages lower network layers to prioritize foreground network traffic over prefetching activity and to discover, characterize, and effectively match traffic to multiple available wireless networks. While IMP could be built on any layer that provides such services, it currently uses Intentional Networking [10] for these purposes.

Finally, IMP proposes a unified approach for dealing with *budgeted resources*. A budgeted resource has a fixed limit on the amount of the resource that can be consumed within a specified time period. One example is cellular data allotments: many cellular plans allow usage of up to a specified amount of data for a fixed charge. Another example is battery energy: for predictable behavior, it is reasonable to expect the battery of a smart phone to last for an entire day so that it can be recharged at night. While the negative consequences of exceeding a budget have been widely observed, the negative consequences of under-utilizing a budget have received little attention. Yet, we observe that unused cellular data and leftover battery energy represent a substantial wasted opportunity when the mobile system has degraded user experience in order to conserve those resources. In response to this observation, IMP takes a "Price is Right" approach to budgeted resources — it tries to come as close as possible to the budget as possible without exceeding it. IMP continually measures how well it is doing in meeting its budgets and uses a control loop to adjust the cost of battery energy or cellular data usage relative to performance accordingly. This is a variation of Odyssey's goal-directed adaptation [8], which IMP modifies to manage multiple resources and applies to the domain of mobile prefetching.

We have implemented IMP as a Java library and modified two applications, the K9 e-mail client and the OpenIntents news reader, to use IMP. Our experimental results show that IMP is able to meet all specified energy and cellular data usage goals. Additionally, in most cases, whenever simple heuristic-based prefetching strategies also meet the goals, IMP outperforms those strategies on interactive fetch time, often by a factor of two or more.

# 2. DESIGN CONSIDERATIONS

In this section, we examine the tradeoffs involved in mobile prefetching, and discuss why the prefetching decision is more complex in the mobile domain than in many other areas.

## 2.1 Why is mobile prefetching so complex?

Most current prefetching systems have the sole goal of maximizing performance, subject to constraints such as cache space and network bandwidth. Prefetching is beneficial when a data item that was prefetched speculatively is later requested by a higher layer of the system. The latency of servicing the request is reduced by the time that would have been required to fetch the data. However, prefetching consumes resources. For instance, because prefetch requests are interwoven with demand fetches from the higher layer, a prefetch request might degrade the performance of those foreground requests. This interference mitigates the performance benefit of a successful prefetch. Further, when prefetched data is never consumed by the application, the interference with foreground activity degrades overall performance. Thus, prefetching systems employ heuristics to try to maximize performance by only prefetching data when the benefits of doing so exceed the costs.

Similar concerns arise in the domain of mobile prefetching when data is fetched from remote servers. The performance benefit of prefetching can be substantial, especially when wireless networks offer poor or intermittent connectivity. However, because wireless bandwidth is limited, the potential for prefetch requests to degrade other traffic is magnified.

In mobile computing, the prefetch decision must consider resources other than performance. Battery lifetime is often a critical concern. While the performance impact of prefetching can be mitigated by scheduling prefetches for intervals with little or no demand requests for data, the energy cost of prefetching data is difficult to mask. Yet, prefetching can still have substantial energy benefits. For instance, consider that the energy required to fetch data over a WiFi network is often less than that required to fetch data over a cellular network and that the energy required to fetch data over networks varies in proportion to the quality of the wireless connection. Thus, by prefetching data during periods of WiFi or good cellular connectivity, the mobile computer uses less energy than if it were to service demand fetches for the same data during periods of poorer connectivity. Further, prefetching allows the mobile computer to batch multiple requests. This saves energy by amortizing transition costs across multiple requests — for instance, the energy costs of 3G tail time can be ameliorated by sending multiple requests back-to-back.

Since it is increasingly common to cap cellular data consumption, prefetching systems must also consider cellular data usage when deciding whether to prefetch. Like energy usage, prefetching can impact cellular data usage either positively or negatively. Prefetching data over a cellular network will increase data usage if the prefetched item is never requested. On the other hand, the mobile computer can prefetch data over a WiFi network and thereby reduce data usage by avoiding a subsequent demand fetch over

a cellular network. One could imagine setting a monthly budget for all prefetching and switching prefetching off if the budget is exceeded. However, it is unclear how to set such a budget accurately because non-prefetch data consumption can vary widely from month to month and from user to user.

In summary, mobile prefetching is complex because one must consider at least three metrics (performance, energy usage, and cellular data usage), whereas current prefetching algorithms often consider just performance. Further, it is difficult to compare the three metrics considered in mobile prefetching because no two are expressed in a single currency. For instance, if a potential prefetch would improve performance and save energy, but would also require additional cellular data transmission, should the mobile computer perform that prefetch? The next section describes our solution to this dilemma.

## 2.2 Balancing multiple concerns

Battery energy and cellular data allotments are examples of a resource class that we term *budgeted resources*. For such resources, there exists a fixed amount of the resource that must last for a period of time. For instance, a cellular data plan may provide 4 GB of data for a month. The consequence of exceeding the budget is severe (e.g., additional cost or degradation of quality of service). On the other hand, any resources not consumed during the budgeted period are wasted (unused data allotments do not roll over to the next month).

Battery energy is also best thought of as a budgeted resource. In this case, the budget is the amount of energy in the battery, and the budgeted period is the time until the battery is recharged. Current energy management strategies correctly worry about the negative consequences of exceeding the budget (running out of battery energy). However, they do not focus nearly enough on the negative consequences of undershooting the budget. Since mobile computers increasingly degrade the user experience to preserve battery energy, any time the battery is recharged with a substantial amount of energy remaining represents a substantial wasted opportunity — the user experience was degraded for no purpose.

The inherent challenge is that the importance of a budgeted resource may change dramatically over time. Near the end of the month when there is substantial data left in a cellular plan, or near the end of the day, when the user has a mostly-full battery and plans to recharge soon, conserving a budgeted resource such as a data allotment or energy is relatively unimportant. Instead, performance should be maximized. On the other hand, when the battery is low or most of a data allotment is consumed, then the relative importance of the budgeted resource is very high.

Strategies that assign a fixed conversion rate for budgeted resources (e.g., saving a Joule of battery energy is worth a 10% degradation in application performance) are doomed to be incorrect as the relative importance of the budgeted resource changes. A fixed conversion rate will be too high when the budgeted resource is unimportant and too low when the resource is precious.

Instead, we argue that the management of the budgeted resource should be adaptive. IMP uses a control loop to adjust the conversion rate used to equate budgeted resources with performance. When the system is projected to exceed its budget based on measurements of current supply and demand, the conversion rate is increased to make the budgeted resource more precious. This causes future decisions (prefetch decisions in the case of IMP) to place more weight on conserving the budgeted resource. Thus, demand is reduced to match supply. On the other hand, when the system is projected to use significantly less than the budgeted amount, the conversion rate is decreased and the system becomes more aggressive about using the resource to improve performance (or reduce the use of other budgeted resources).

An additional benefit of the control approach is that the user experience becomes more predictable. For instance, users can come to expect that their phone battery will just last all day in most circumstances, rather than having to constantly monitor the battery level and adjust their own behavior. Further, the budget itself is a simple knob through which the user can adjust system behavior. For instance, if one knows that one will recharge in an hour, one can just change the budget to reflect that decision rather than adjust numerous settings like screen brightness, processor speed, etc.

# 3. DESIGN

We next discuss the design of Informed Mobile Prefetching. We first give some background on Intentional Networking, a multi-network abstraction leveraged by IMP. We then describe how IMP decides whether to prefetch data in response to application hints.

## 3.1 Background: Intentional Networking

Intentional Networking [10] is an abstraction for multiplexing traffic over multiple wireless networks. Intentional Networking discovers and simultaneously connects to multiple networks such as cellular and WiFi. Higher layers of the system (such as IMP) annotate network sends with qualitative labels, which Intentional Networking uses to route traffic over the most appropriate network interface.

IMP uses Intentional Networking to differentiate prefetch traffic from other application network activity. By designating prefetches as background requests, IMP causes Intentional Networking to prioritize other data over prefetches. Further, Intentional Networking throttles low-priority traffic slightly to ensure that queuing delays due to excessive in-flight background data do not adversely affect foreground requests. Experimental results have shown that foreground requests can be serviced with low latency with only a few percent degradation in background throughput. When appropriate, Intentional Networking provides greater throughput for large requests by striping them across multiple networks.

IMP also gathers passive and active measurements of network quality from the Intentional Networking layer. Intentional Networking provides notifications when new networks are available and when existing networks are disconnected.

## 3.2 Prefetch decision algorithm

IMP decides when and how much data to prefetch using a cost/benefit analysis inspired by Transparent Informed Prefetching (TIP) [23]. The TIP algorithm regulates prefetching of file system data from an array of disk drives in a server environment. The core idea behind TIP is that the application should disclose hints that describe opportunities to prefetch data — these hints are predictions of future accesses. TIP dynamically decides when to prefetch data corresponding to the disclosed hints based on a cost/benefit analysis that estimates how prefetching will impact application performance. One of the nice features of the TIP design is that applications do not have to know any details about the environment in which they are operating (e.g., resource constraints), nor do applications have to specifically estimate how effective prefetch hints will be at predicting future accesses. These details are managed by the prefetching system.

Since the use case that TIP targets (prefetching file system data from disk arrays) is considerably different than prefetching data to mobile computers, IMP retains the structure of the TIP cost/benefit model but changes most of the details. For instance, TIP considers cache buffers to be the bottleneck resource, and it attempts to

optimize only performance. On the other hand, in mobile prefetching, the wireless network is the bottleneck resource and one must consider both battery energy and data usage in addition to performance.

We next describe how IMP estimates the potential cost and benefit of prefetching a data item for each of its three metrics: performance, energy use, and data consumption. IMP separately considers the impact of prefetching over each currently available network (e.g., it calculates the impact for cellular and WiFi when both are available).

### 3.2.1 Performance

The performance benefit of prefetching comes from decreasing the time to service a subsequent request for the prefetched data. The precise amount of benefit depends on the size of the data item and the network conditions that will exist at the time the data is requested. If the item is not prefetched, a demand fetch of the data will take time $T_{fetch}$, where

$$T_{fetch} = \frac{S}{BW_{future}} + L_{future}$$

as given by the size of the data item ($S$) and the future network bandwidth and latency at the time the demand fetch occurs ($BW_{future}$ and $L_{future}$, respectively).

Although it may sometimes be feasible to predict future network conditions [19], for many applications it is quite difficult to predict when the user will request data. For this reason, IMP uses the average network conditions observed in the past for the mobile computer as a reasonable approximation of future conditions.

For each type of network, IMP tracks the average availability, latency, and bandwidth of the network. To calculate averages, it obtains active network measurements and supplements those observations with passive measurements of network behavior that occur when data is prefetched or fetched on demand by applications. IMP uses network availability to merge the costs of performing the demand fetch over different network types. It assumes that if the lowest cost network is available, that network will be used. If not, the next lowest cost network will be used if available, etc. In practice, since IMP currently considers just cellular and WiFi transmission and since WiFi almost always dominates 3G when all costs (performance, energy, and data) are considered, this works out to: $T_{fetch-WiFi} \times Availability_{WiFi} + T_{fetch-cellular} \times (1 - Availability_{WiFi})$ or the cost of fetching data over the two network types weighted by the observed average availability of WiFi.

IMP allows applications to specify the size of each data item when a prefetch is requested. For the applications we have modified to use IMP, it has been trivial for the application to retrieve the size of each data item from a remote server at the same instance that it learns of that item's existence. For example, when an email client requests a listing of all new messages, it can ask for the size of each email to be returned in the same request. If an application declines to supply the size of an item to be prefetched, IMP instead uses the average size of items it has fetched for that application in the past.

The calculation as described so far assumes that prefetch hints are always accurate; i.e., that the prefetched data will be requested at some time in the future. However, many prefetched items are never consumed by the application, so IMP must also consider the estimated *accuracy* of a prefetch hint in its calculations. This is calculated on a per-application basis as the number of prefetch hints for which the prefetched data was later consumed divided by the total number of prefetch hints issued. IMP increments the count of total hints when a new hint is issued, and it increments the count of consumed hints when an application first requests prefetched data. It uses shadow caching to determine the number of hints for which it did not prefetch data but for which the application later requested the data. Specifically, IMP remembers all prefetch hints issued, regardless of whether or not it has yet decided to prefetch the hinted data. Applications inform IMP when a hinted request that has not yet been prefetched must be fetched on demand — this information is necessary since prefetching is no longer useful. IMP increments the count of consumed hints when this happens.

IMP allows applications to optionally differentiate among prefetch hints by associating each hint with a *class*. The choice of how to apply classes is left up to the application; in general, applications will benefit most if their prefetch classes separate prefetches by some application-specific or user-specific quality that makes some prefetches more likely to be requested than others. For instance, an email application might use one class for messages in a priority inbox, and another class for the remaining messages. A news reader application might associate a different class with each news feed. IMP maintains a separate estimate of accuracy for each class and uses the appropriate class estimator when new prefetch hints are issued. In this way, the abstraction we provide simplifies the creation of application-specific prefetch strategies; the application need only divide its hints into classes, and IMP takes care of discovering which classes have the most valuable hints.

In summary, IMP calculates the performance benefit of prefetching as the product of the average time to fetch the data on demand and the application-specific or class-specific accuracy estimate. The accuracy estimates allow IMP to help the application focus on prefetching the items that are most likely to be used. Note that since IMP relies on Intentional Networking to ensure that prefetch traffic does not interfere much with foreground application activity, it need not assign a performance cost to each prefetch.

### 3.2.2 Energy usage

IMP calculates the effect of prefetching on battery energy by comparing the energy that would be used to prefetch the data immediately ($E_{prefetch}$) with the expected energy cost of later fetching the item on demand ($E_{fetch}$). The previous section describes how IMP calculates $T_{fetch}$, the expected time to perform a future demand fetch of an item, as well as how it calculates the expected size of the data item (if this is not specified by the application). IMP calculates $T_{prefetch}$, the time to perform a prefetch, the same way it calculates the time for a demand fetch, except that it uses the current estimates of bandwidth and latency for the network over which prefetching is being considered (recall that IMP considers prefetching over WiFi and cellular networks independently). IMP then uses models developed by PowerTutor [30] to calculate the energy impact of both an immediate prefetch and a possible future demand fetch. These power models are specific to the mobile device and cellular carrier. They are generated with a suite of automated tests that isolate and exercise different components of the device in turn. Once a model is derived, it can be re-used for many different purposes (as we are reusing models developed by others in our work).

For prefetching on WiFi, the power model provides a power coefficient $P_{WiFi-xmit}$ that encapsulates the average power usage of the WiFi interface when actively sending and receiving; thus, $E_{prefetch} = P_{WiFi-xmit} \times T_{prefetch}$. When prefetching on a 3G cellular network, IMP also includes the transition costs that may be incurred due to network activation. For instance, tail energy [24] has been shown to be a significant phenomenon in 3G cellular networks. 3G networks use inactivity timers to avoid the delays that can result from frequent channel release and reacquisition. This results in the network interface spending substantial time in a high-power state after the last transmission completes.

If a network transmission is predicted to cause a promotion to a high-power state, IMP includes the transition costs to and from the high-power state (which includes the tail energy cost for 3G networks) in the energy cost of the prefetch. However, if the radio is predicted to already be in a high-power state, only the additional time spent in the high-power state is included. Like PowerTutor, IMP monitors the number of bytes queued on the cellular interface to infer when power state transitions occur. When calculating the energy cost of an immediate prefetch, IMP queries the state transition model to learn whether or not the cellular radio is in a high power state. When calculating the energy cost of a future demand fetch, it uses the average distribution of power states observed for that interface to determine the likelihood of being in each power state. Thus, the energy cost of a 3G prefetch is $(P_{3G-xmit} \times T_{prefetch}) + E_{tail}$, where

$$E_{tail} = \begin{cases} P_{tail} \times T_{tail} & \text{if transition occurs and tail begins} \\ P_{tail} \times T_{inactivity} & \text{if tail time is extended} \end{cases}$$

For simplicity, we here omit details such as the difference between DCH and FACH power states, though we include them in our calculations.

Finally, the net energy cost (or benefit) of a prefetch is $E_{prefetch} - (E_{fetch} \times Accuracy)$, since the energy cost of the future demand fetch is only paid if the user actually requests the item. Note that a prefetch can have an expected energy benefit if current network connectivity is strong (so less energy than average is required to fetch data) or a lower-power network is currently available, and the estimated accuracy of a prefetch is high. Batching (described in Section 3.2.5) can also lead to energy benefits for prefetching by amortizing transition costs across multiple requests.

### 3.2.3 Data consumption

IMP also considers the effect of prefetching on cellular data consumption. The estimated future data cost of a demand fetch is calculated as $D_{fetch} = S \times (1 - Availability_{WiFi})$. The benefit of prefetching over WiFi is $D_{fetch}$, since the cellular data cost on WiFi is zero. The cost of prefetching over a cellular network is simply $S$. As in the case of energy cost, when referring to the data cost of prefetching below, we denote it as $D_{prefetch} - Accuracy \times D_{fetch}$, where $D_{prefetch}$ is zero or $S$ depending on which network is used.

### 3.2.4 Putting it all together

As explained in the last three sections, IMP separately calculates the cost or benefit of prefetching over each type of available network in terms of performance, energy, and data usage. Unfortunately, these three metrics are expressed in different units (seconds, Joules, and bytes), making it difficult to determine what to do when the metrics disagree about whether or not prefetching would be beneficial.

A common solution to this dilemma is to employ a conversion rate or utility function to equate metrics expressed in different units (e.g., one could make 10 Joules of energy expenditure equal to 1 second of interactive delay). However, as argued in Section 2.2, the importance of each metric can change substantially over time, making static conversion rates arbitrary and dooming them to be incorrect at least some of the time.

Instead, IMP dynamically adjusts conversion rates using a feedback loop that takes as input how well the system is meeting its resource budgets. Specifically, IMP maintains two values, $c_{energy}$ and $c_{data}$, that represent the current conversion rate between each respective budgeted resource and performance. For instance, if $c_{energy}$ is 0.2, then 5 Joules is considered equal to 1 second of interactive delay.

IMP uses Odyssey's goal-directed adaptation to adjust these conversion rates. Since we did not modify this algorithm and it is described elsewhere in detail [7], we provide a simple overview of its design here. Note that Odyssey applied goal-directed adaptation only to energy; one of the contributions of this work is the observation that this strategy can also be used to regulate cellular data usage. Additionally, Odyssey used application-specific fidelity adjustments to regulate the amount of resource spending over its budget period, whereas we regulate spending by adjusting the amount of prefetching.

Once a second, IMP measures the remaining supply of a resource. Goal-directed adaptation uses an exponential smoothing algorithm to calculate the rate at which the resource is being consumed from recent observations of supply. It multiplies the estimated rate of consumption by the time remaining in the budgeted period to estimate the remaining demand for the resource.

IMP relies on having an estimate of the future time at which each budgeted resource will be replenished. Many cellular data plans have predictable periods coinciding with a monthly billing cycle, making it simple to estimate the budget period. Additionally, recent work [25] has identified personal patterns in different users' battery recharge behaviors, which suggests it may be possible to predict the budget period for the available battery energy. The accuracy of such predictions is important, however; a too-long or too-short prediction of the budget period will increase the likelihood that IMP underspends or overspends the budget, respectively.

Next, the goal-directed adaptation algorithm subtracts from the latest supply measurement 5% of the current remaining supply plus 1% of the original supply. This provides some insurance against future changes in the resource budgets, which can occur if the user downloads a new application that consumes a lot of data, or if they arrive home later than usual to recharge their phone. Finally, it computes a ratio that expresses how much resource consumption is currently over or under budget:

$$c_{adjustment} = \frac{estimated\_demand}{reduced\_supply}$$

IMP then updates the current conversion rate with this ratio: $c_{new} = c_{old} \times c_{adjustment}$. Thus, if energy consumption is over budget, a Joule becomes more precious compared to other resources. This feedback process leads IMP to bias further against prefetch decisions that cost energy, which will eventually bring the energy consumption within budget.

In summary, IMP calculates the net benefit of prefetching an item over a particular type of network as benefit minus cost, or:

$$T_{fetch} \times Accuracy - (c_{energy} \times (E_{prefetch} - Accuracy \times E_{fetch}) + c_{data} \times (D_{prefetch} - Accuracy \times D_{fetch}))$$

If this value is positive for a single network type, IMP prefetches the item over that network. If the value is positive for multiple network types, IMP tells the Intentional Networking layer that the data item may be prefetched over all such networks. Intentional Networking stripes the item over all such networks if the item is large enough that striping makes sense.

### 3.2.5 Batching

Because some costs (most notably 3G tail time) can be amortized across multiple prefetches, prefetching may be beneficial if several items are fetched in a batch, even though prefetching each item individually is estimated to have a negative impact. IMP checks for possible batching benefits by considering the impact of fetching one item, two items, etc. up to the number of currently outstanding

| Application → IMP | | IMP → Application | |
|---|---|---|---|
| `prefetch(Fetcher)` | → `Future` | `Fetcher.fetch(Labels)` | → `Result` |
| `Future.get()` | → `Result` | | |
| `Future.cancel()` | | | |
| `Fetcher.setSize(int)` | | | |
| `Fetcher.setPrefetchClass(int)` | | | |

This figure shows the IMP API used for hinting, executing, consuming, and canceling prefetches. Applications create a `Fetcher` that implements the `fetch` callback to execute a (pre)fetch. The `fetch` method takes as an argument the set of Intentional Networking *labels* to be used for the invoked fetch. The `prefetch` function takes a `Fetcher` and returns a `Future`, a handle that the application uses to retrieve the result of a fetch. The type of `Result` that a fetch returns is specified by the application. The application may optionally call `setSize` and `setPrefetchClass` to pass additional information to IMP.

**Table 1: Informed Mobile Prefetching API**

application prefetch requests. If it predicts a positive impact for any batch size, it initiates prefetching.

#### *3.2.6 Network disconnection*

If a network becomes unavailable while a prefetch is being performed using that network, IMP recalculates the potential impact of prefetching over the remaining networks as above. If a potential benefit is found, it resumes the prefetch over the selected remaining networks. If not, the prefetch is delayed and resumed when an appropriate network is found. For simplicity, IMP currently relies on application-level support for resuming prefetch requests without retransmitting the data already received, though it could potentially provide this feature transparently with additional buffering.

#### *3.2.7 Discussion*

We have designed IMP to manage resources that have a fixed budget to be spent over a predictable time period. However, there are also cellular data plans where the subscriber pays for the amount of data actually consumed, rather than paying for a monthly budget. Though IMP does not handle such cases in its current form, we could imagine extending it to do so. Rather than targeting a single budget over a single time period, we could treat the pay-as-you-go data plan as a series of small budgets, in the amount of the data purchase increments. Once purchased, a unit of data can be treated in the same way as the budgets IMP currently considers, since there are no refunds for unused data and partially spent units are generally rounded up to the next full unit for billing. However, we would also need to consider the monetary cost of each data increment and also incorporate some notion of how much the user is willing to spend, which would require direct input from the user.

The concerns in this scenario differ from what we consider in IMP's current long-term budget approach, because of the added goal of saving the user money, the smaller units of data for which money is spent, and the fact that less money spent is always favorable. However, this scenario can still benefit from IMP's fixed-budget adaptation strategy for each allotted unit of data. In addition, the same strategy can be applied for each extra gigabyte that the user effectively purchases if they exceed their data cap on a pay-per-month plan.

Caching systems typically have a limited amount of storage available for cached data, along with an eviction policy. We observe that battery energy and cellular data are usually far scarcer than storage on today's mobile devices, so we chose to focus on improving the use of those resources. We could also apply a TIP-like strategy to include the value of cache space in our cost-benefit analysis.

Finally, we note that some users have unlimited data plans, and some users are frequently able to charge their phones throughout the day so that they are rarely in danger of running out of battery. In these situations, IMP will prefetch more aggressively, since it is given larger resource budgets. In the case of unlimited data, the throttling that some carriers perform after the user has crossed a certain data limit can be viewed as a budget in itself, since the reduced bandwidth is undesirable.

# 4. IMPLEMENTATION

This section describes the IMP's Application Programming Interface and the applications we have modified to use IMP.

## 4.1 API

IMP is implemented as a Java library targeted at the Android platform. Its prefetching API is inspired by Java's ExecutorService and Future [9], which simplify programming tasks with arbitrary asynchronous computation. The interface, which is shown in Table 1, is designed so that IMP need not replicate the application-specific details of (pre)fetching data.

#### *4.1.1 Prefetch hinting and execution*

Applications use the `prefetch` method to provide a prefetch hint to IMP. As described in Section 3.2, IMP uses a cost/benefit analysis to decide when and if to act on the hint. The application supplies a `Fetcher` object that both identifies the data item to be fetched and implements the application-specific `fetch` method that retrieves it from the server. This method is used for both prefetches and on-demand fetches. In the former case, IMP initiates the prefetch by calling `fetch`. The application can optionally supply the size of the item to be fetched, as well as a *prefetch class*. Applications can use classes to indicate prefetch priority. IMP will prefetch items in classes with higher observed accuracy in preference to items in other classes. Within a class, IMP prefetches objects in the order that prefetch hints are supplied.

#### *4.1.2 Prefetch consumption*

Like the `submit` method in Java's ExecutorService, IMP's `prefetch` method returns a Future — this is a handle that allows the application to later retrieve the result of the fetch operation by calling its `get` method. If the prefetch has started but not yet completed, the `get` method blocks until the prefetch completes or until an optional timeout expires. If the prefetch has not yet started, calling `get` triggers an immediate demand fetch of the item (which is effected through the `fetch` callback).

#### *4.1.3 Prefetch cancellation*

If the application no longer needs a item for which it has issued a prefetch hint, it cancels the prefetch by either calling the `cancel` method on the Future object or by simply removing the last reference to the Future object, in which case the prefetch is implicitly canceled when the Future object is garbage-collected.

## 4.2 Applications

We have modified the K9 [11] open-source email client and the OpenIntents [20] news reader to use IMP.

### *4.2.1 K9 email*

K9 [11] is is an open-source email application for Android, originally based on the stock Android email client. By default, K9 prefetches messages that are less than 32 KB (the threshold can be adjusted from the application). We added IMP support by using an IMAP proxy to intercept traffic between K9 and our IMAP server.

On application start, the proxy downloads the user's email headers (including the size of each message), decides which emails to prefetch, and issues prefetch hints for those messages (including their attachments), in order from newest to oldest.

### *4.2.2 News reader*

We also added prefetching to OpenIntents, an open-source Android Atom/RSS feed reader. OpenIntents stores the content available in a feed but does not prefetch linked articles. Since the application uses HTTP for its network communication, we modified Apache HttpComponents [3], a Java HTTP library included in the Android SDK, to add support for article prefetching.

Frequently, Atom and RSS feeds only contain a summary and link to a given article, rather than the full contents of the article. Our modified news reader therefore grabs the full-article links from each feed and decides which articles to prefetch. The application-specific fetcher specified via the `Fetcher` object issues an HTTP request for the full content of the article plus any embedded images, and it then stores the fetched data persistently for later retrieval.

Our news reader associates a prefetch class with each feed and uses the `setPrefetchClass` method to specify the class for each hint. To evaluate the impact of using classes, we also created a version of the news reader that does not specify prefetch classes.

## 4.3 Discussion

IMP needs some information from the application in order to make good prefetching decisions, but we must avoid making the developer's task onerous, or else applications will not use our system. Thus, we have only added features when our applications required them (*e.g.* prefetch classes). We could imagine applications that would desire additional features, such as the ability to retrieve a partial prefetch result instead of waiting for all data to arrive. Our API design would make such an addition straightforward.

In general, applications must be modified to work with IMP. We can support unmodified email clients via an email proxy, as features of the IMAP protocol help us discover which items the user might request and when items are requested on-demand or deleted. For other applications such as OpenIntents, it is difficult to gather this information without modifying the application.

# 5. EVALUATION

Our evaluation compares IMP with the prefetch strategies most commonly employed by current mobile applications. We compare results across three metrics: application performance, energy usage, and data consumption. In addition, we examine how well IMP meets budgets for energy and data usage. Finally, we quantify the benefit of using prefetch classes.

## 5.1 Experimental Setup

### *5.1.1 Testbed*

We run K9 and OpenIntents on the AT&T version of the Nexus One phone, running Android version 2.3.4. We modified Android system software to allow the simultaneous use of WiFi and cellular networks and added Intentional Networking support. To generate repeatable experiments, the phone connects to an isolated WiFi access point and a private Cisco MicroCell that is connected to AT&T's network. Since the MicroCell acts as a miniature cellular tower, our evaluation captures the effects of the cellular wireless medium on network bandwidth and latency. We emulate network conditions by passing all traffic through a computer that inserts delays with the netem [16] network emulator and throttles throughput using the Linux Advanced Routing and Traffic Control tools [15]. We run servers for each application on a Dell Optiplex GX270 desktop with a 2.8 GHz Pentium 4 processor and 1GB DRAM.

We measure energy usage by applying the Nexus One power model used in the implementation of PowerTutor [30]. We measure cellular data usage by reading the number of bytes sent and received through the Linux sysfs interface.

When we compare with strategies that do not use IMP, we allow those strategies to use Intentional Networking to isolate prefetching traffic from other foreground traffic and to take advantage of both cellular and WiFi networks. Thus, the results reported in this paper show only the *additional* benefit that IMP provides on top of the benefit already provided by Intentional Networking. All reported results are the mean of five trials — graph error bars are 95% confidence intervals. Where there is a resource budget set for IMP, the budget is indicated by a horizontal line above the bar.

To quantify the benefit of different prefetching strategies, we report the average fetch time over all emails or news articles in a run. We report the average rather than the median because the median does not distinguish between two prefetch strategies that produce more than a 50% cache hit rate, and it is the long fetch delays that dominate the user's experience when not enough data is prefetched. Though the individual fetch times can vary considerably due to several factors – data size, network bandwidth at the time of the fetch – lower average response time generally indicates more cache hits and thus more successful prefetching.

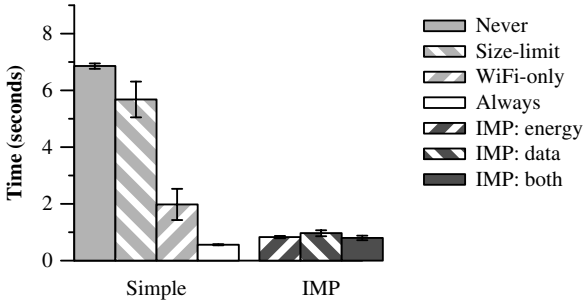
### *5.1.2 Trace-driven evaluation*

To evaluate how our system performs in realistic mobile networking conditions, we use a vehicular network trace gathered in Ypsilanti, MI [10]. Trace-driven evaluation provides experimental repeatability and allows for meaningful comparison between different prefetch strategies.

We gathered this trace by driving a vehicle equipped with WiFi and Sprint 3G network interfaces and continuously measuring the downlink and uplink bandwidth and latency via active probing to a server at the University of Michigan. The median 3G bandwidth is 368 Kb/s downlink and 40 Kb/s uplink, with maximum bandwidth of 1.2 Mb/s downlink and 74 Kb/s uplink. WiFi is available 27% of the time, with a median session length of 7 seconds; the longest WiFi session length is 131 seconds.

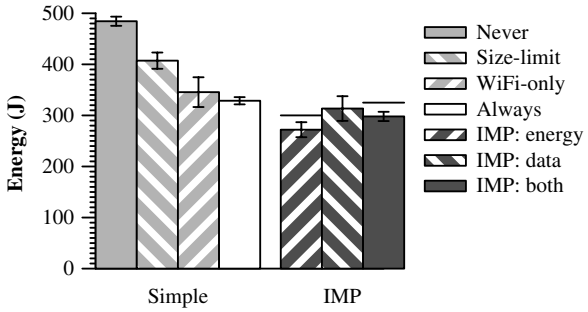
We collected a second trace by walking in downtown Ann Arbor and across the University of Michigan campus, carrying a Nexus One phone and measuring the available open and campus WiFi and the AT&T cellular network. The median 3G bandwidth is 695 Kb/s downlink and 216 Kb/s uplink, with maximum bandwidth of 1.3 Mb/s downlink and 358 Kb/s uplink. WiFi is available 18% of the time, with a median session length of 56 seconds; the longest WiFi session length is 99 seconds.

When running benchmarks, we replay the traces on the emulation computer, which throttles bandwidth and delays packets for each network according to the conditions observed. When no WiFi or cellular coverage is observed in a trace, the throttling computer causes the connection to drop — the Android OS typically discov-



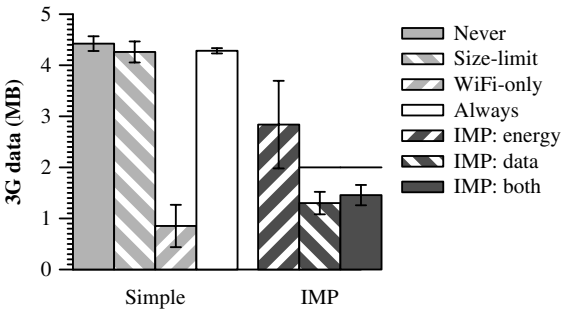

Average fetch time for 28 emails fetched over 20 minutes. The left set of bars shows results for simple heuristic-based prefetching strategies; the right set of bars shows results for IMP when energy, cellular data, or both are constrained. Each bar is the mean of 5 trials. The error bars are 95% confidence intervals.

**Figure 1: Average email fetch times, driving trace**




Total energy usage for the email benchmark. The left set of bars shows results for simple heuristic-based prefetching strategies; the right set of bars shows results for IMP when energy, cellular data, or both are constrained. Where there is an energy budget set for IMP, it appears as a solid line above the bar. Each bar is the mean of 5 trials. The error bars are 95% confidence intervals.

**Figure 2: Email energy usage, driving trace**




Total cellular data usage for the email benchmark. The left set of bars shows results for simple heuristic-based prefetching strategies; the right set of bars shows results for IMP when energy, cellular data, or both are constrained. Where there is a data budget set for IMP, it appears as a solid line above the bar. Each bar is the mean of 5 trials. The error bars are 95% confidence intervals.

**Figure 3: Email data usage, driving trace**

ers the network disconnection after several seconds. Since the collected traces are longer than our experiments, we use only the first portion of each trace.

### *5.1.3 Comparison strategies*

For each application, we compare IMP's performance to four prefetch strategies commonly used by mobile applications today. Unlike IMP, none of these strategies explicitly considers the cost and benefit of prefetching. The strategies are: never prefetch anything, prefetch items with size less than an application-specific threshold, prefetch over WiFi when it is available but never over the cellular network, and always prefetch everything.

### *5.1.4 IMP scenarios*

IMP attempts to maximize performance subject to one or more resource constraints. For each application, we consider three scenarios: one where energy usage is constrained, one where cellular data usage is constrained, and one where both resources are constrained. Although we do not show the results, we did verify that when neither budgeted resource is constrained, IMP (correctly) emulates the always-prefetch strategy because that strategy maximizes performance.

To run a large number of experiments, we limit execution time to 20 minutes. The goals for energy and data usage are scaled proportionately. Note that even though longer experiments would likely show a wider variety of network conditions and usage behaviors, shorter experiments like these are much more challenging for a feedback strategy like the one employed by IMP because there is little time to react to under-usage or over-usage of a budgeted resource. For instance, if IMP is over-using the cellular data budget after half the budgeted time, it has 15 days to react during a month-long experiment, but only 10 minutes to bring the usage back under budget in these experiments. Similarly, IMP would have more time to react to unexpected changes in a user's mobility patterns over the course of a long experiment – for example, taking a different route to the office and encountering far fewer open WiFi APs. Thus, we expect that if IMP can meet budgets in these shorter experiments, it should do quite well in longer ones.

## 5.2 Results

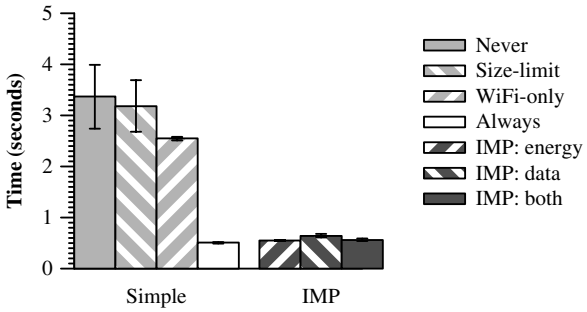
### *5.2.1 Email*

We constructed an email workload from day-long traces of email activity within our department collected during prior projects. We use a trace of an email user fetching messages from a server [2] to derive the timing of email fetches and the interleaving user think time. The size of each email is randomly sampled from a distribution of email sizes in another trace [27].

At the start of the benchmark, the proxy fetches the list of 35 emails from the server along with their sizes. It issues prefetch hints for all emails. The reported accuracy of Gmail's Priority Inbox is 80% [1]. Since this seems a reasonable mechanism for deciding which email should be hinted, we assume that 28 of the emails are read during the experiment while the other 7 are deleted before they are read. This models a scenario in which a user flips through the inbox and decides, based on the subject line and message preview text, whether or not to read the email. Thus, the accuracy of prefetch hints is 80%. Of course, IMP is not told which hinted emails will be read and which will be discarded.

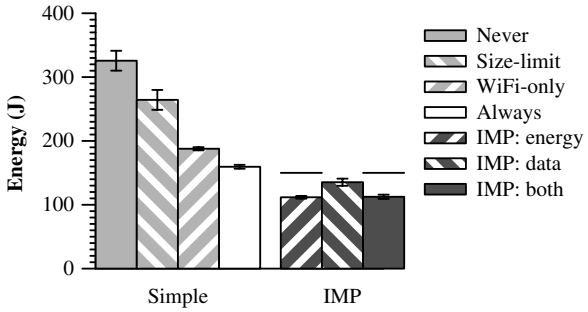
K9's default settings specify that email messages less than 32 KB will be fetched automatically upon inbox update. We use that value for the size-limit strategy.

Figure 1 shows the average email fetch time for each strategy. As expected, the always-prefetch strategy produces the lowest fetch time, since every fetch is a cache hit. However, as shown in Figures 2 and 3, this strategy uses more energy and data than any IMP strategy.



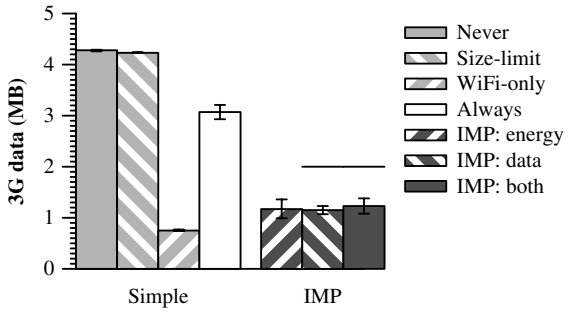

Average fetch time for 28 emails fetched over 20 minutes. The left set of bars shows results for simple heuristic-based prefetching strategies; the right set of bars shows results for IMP when energy, cellular data, or both are constrained. Each bar is the mean of 5 trials. The error bars are 95% confidence intervals.

**Figure 4: Average email fetch times, walking trace**




Total energy usage for the email benchmark. The left set of bars shows results for simple heuristic-based prefetching strategies; the right set of bars shows results for IMP when energy, cellular data, or both are constrained. Where there is an energy budget set for IMP, it appears as a solid line above the bar. Each bar is the mean of 5 trials. The error bars are 95% confidence intervals.

**Figure 5: Email energy usage, walking trace**




Total cellular data usage for the email benchmark. The left set of bars shows results for simple heuristic-based prefetching strategies; the right set of bars shows results for IMP when energy, cellular data, or both are constrained. Where there is a data budget set for IMP, it appears as a solid line above the bar. Each bar is the mean of 5 trials. The error bars are 95% confidence intervals.

**Figure 6: Email data usage, walking trace**

Interestingly, the never-prefetch strategy uses more energy than any other strategy. This is due to a combination of the tail energy incurred with each demand fetch and poorer 3G bandwidth in later portions of the trace causing the 3G interface to remain in the high-power state longer.

When resources are scarce, IMP is able to meet resource budgets that are set lower than the amounts used by most simple strategies. In the constrained-energy scenario, we set the energy budget to use no more than 300 Joules over 20 minutes (this is indicated by the solid line over the `IMP: energy` bar in Figure 2). Despite this goal being lower than the energy usage of any simple strategy, IMP comes in under budget in every trial. Further, IMP provides average fetch time within 300 ms of the (performance-optimal) always-prefetch strategy. Compared to the other strategies, IMP improves average fetch time by 2–8x and reduces energy usage by 21–43%. The fact that IMP produces a lower average fetch time than the common WiFi-only prefetching strategy lends credence to our assertion that prefetching on 3G is often a wise way to spend resources when the budget allows for it, since doing so provides a clear benefit to the user.

It is perhaps surprising that IMP uses less energy than the WiFi-only strategy in this scenario, especially since WiFi transmission is usually more energy-efficient than cellular transmission. Our analysis shows that because the WiFi-only prefetching strategy does not prefetch over the cellular network, it cannot prefetch some items before the user requests them. This leads to demand fetches later in the experiment that must be serviced over cellular because WiFi is unavailable at the time. Further, the cellular network quality at the time is poor, causing even more energy usage than normal. IMP avoids these demand fetches through more aggressive prefetching. Additionally, although IMP sends more data over cellular than the WiFi-only strategy, it reduces energy usage due to 3G tail time by batching multiple requests together. In contrast, the demand fetches cannot be batched due to user think time.

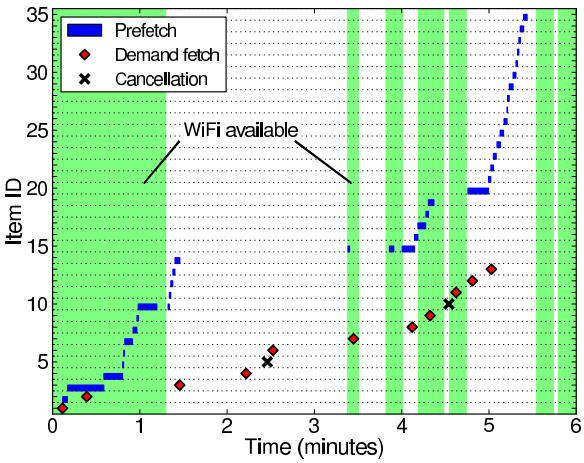
In the constrained-data scenario, we set the budget to use no more than 2 MB of data. Figure 3 shows the 3G data usage for each prefetching strategy. IMP meets this goal in all trials, whereas WiFi-only is the only simple strategy to use as little data. However, the WiFi-only strategy more than doubles average fetch time compared to IMP. Meanwhile, IMP is still able to provide average fetch time within 410 ms of that of the always-prefetch strategy and 2–7x less than that of the never-prefetch and size-limit strategies.

To constrain both resources, we set an energy budget of 325 Joules and a data budget of 2 MB. The results are similar to the single-budget scenarios. IMP provides average fetch time within 240 ms of the always-prefetch strategy and 2–8x lower than the other strategies, while reducing energy usage by 9–38% compared to the simple strategies. Compared to the always-prefetch, never-prefetch, and size-limit strategies, IMP reduces cellular data consumption by 3x.

IMP sometimes undershoots its budgets because it is hedging against future spikes in demand. In this experiment, however, undershoot occurs simply because IMP runs out of items to prefetch.

Figures 4, 5, and 6 show the results for the email application on the walking trace. In the walking scenario, the best strategy is to always prefetch, because the best WiFi is available at the beginning of the trace, and the little available WiFi that comes later is poor. In all scenarios, IMP emulates always-prefetch closely, achieving average fetch time within 40–150 ms of that of the always-prefetch strategy and 4–6x lower than the other strategies.

For the energy-constrained experiments using the walking trace, we set an energy budget of 150 Joules. IMP meets the goal in all trials, reducing energy usage by 30–65% compared to the simple strategies. For the data-constrained experiments, we set a 3G data budget of 2 MB. IMP meets this goal as well in all trials, reducing




Email prefetch and demand fetch activity for one run of IMP, energy-constrained, on the driving trace. The horizontal bars show issued prefetches. The diamonds show demand fetches. The shaded regions show periods when WiFi is available. Each canceled prefetch hint is marked with an X. Since prefetching finishes early in the experiment, only the first six minutes are shown.

**Figure 7: Email + IMP prefetch activity**

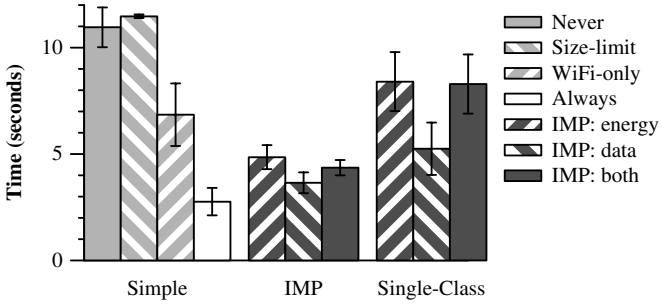
3G data usage by 2–4x compared to the never-prefetch, size-limit, and always-prefetch strategies. Only the WiFi-only strategy meets the data goal, but it increases average fetch time by over 4x compared to IMP.

To provide some insight into the decisions that IMP makes over the course of an experiment, we show in Figure 7 the activity of the email application in one run of the driving trace, with IMP running with the energy budget only. IMP starts prefetching on WiFi, until the WiFi fades shortly after the 1-minute mark. At this point, the 3G interface is already active due to an Intentional Networking control message, so IMP starts prefetching on 3G. It completes a few prefetches before deciding that the next prefetch is too large to send on 3G, so it subsides. A few minutes later, it begins encountering some spotty WiFi, which it uses to attempt prefetching the next message. Soon after it starts making forward progress on WiFi, the WiFi network fails again. IMP then decides that it has enough budget to send a large batch of prefetches over 3G (taking the tail time savings into account), so it finishes prefetching the rest of the messages.

### *5.2.2 News reader*

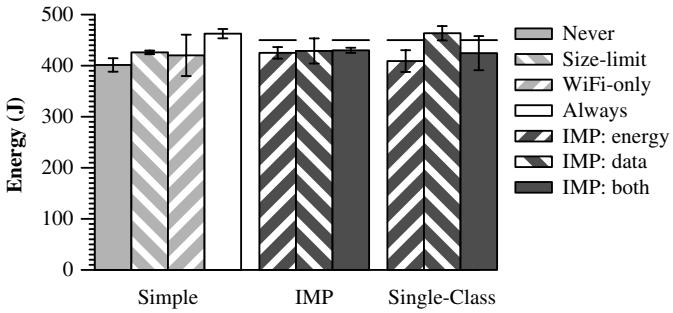
We constructed a news reader workload based on one of the authors' Google Reader subscriptions. Google Reader provides statistics for a user's feeds for the past 30 days, including information such as the number of articles published and the number of articles read, along with the calculated read percentage. We choose five feeds with articles widely varying in size (primarily due to attached images). We used the relative feed volume over the past 30 days to choose a subset of each feed's articles for the benchmark, starting with the most recent articles. There are 25 articles across the five feeds.

Our simulated user selects a feed and begins iterating through the articles in that feed, opening and reading some of them and marking others as read without actually opening them. The fraction of articles that the user actually reads is determined by the read rate



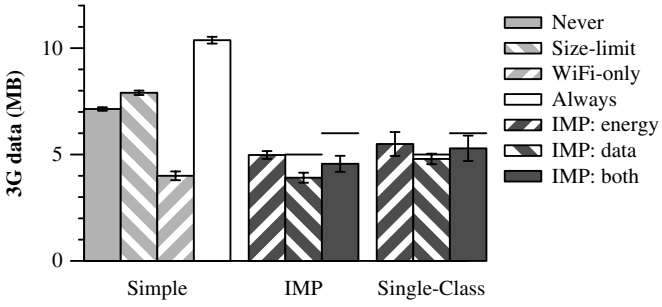

Average news article fetch time for 16 articles fetched over 20 minutes. The left set of bars shows results for simple heuristic-based prefetching strategies. The middle set of bars shows results for IMP when energy, cellular data, or both are constrained. The right set of bars shows results for IMP with the same constraints when IMP does not separate prefetch hints by class. Each bar is the mean of 5 trials. The error bars are 95% confidence intervals.

**Figure 8: Average news article fetch times, driving trace**



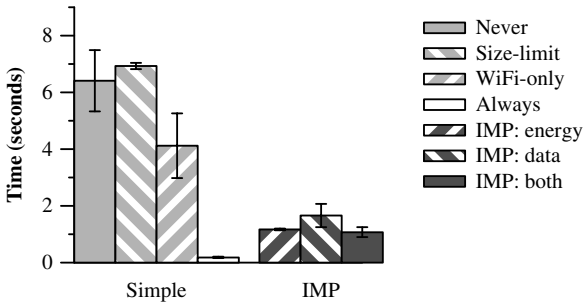

Total energy usage for the news reader benchmark. The left set of bars shows results for simple heuristic-based prefetching strategies. The middle set of bars shows results for IMP when energy, cellular data, or both are constrained. The right set of bars shows results for IMP with the same constraints when IMP does not separate prefetch hints by class. Where there is an energy budget set for IMP, it appears as a solid line above the bar. Each bar is the mean of 5 trials. The error bars are 95% confidence intervals.

**Figure 9: News reader energy usage, driving trace**



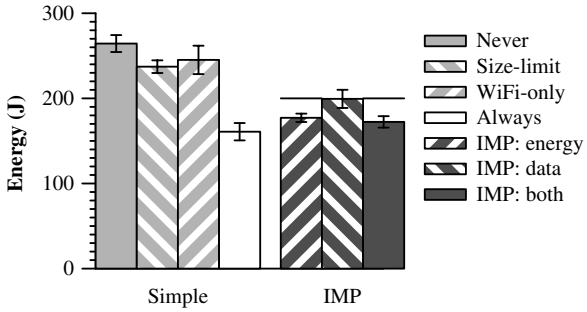

Total cellular data usage for the news reader benchmark. The left set of bars shows results for simple heuristic-based prefetching strategies. The middle set of bars shows results for IMP when energy, cellular data, or both are constrained. The right set of bars shows results for IMP with the same constraints when IMP does not separate prefetch hints by class. Where there is a data budget set for IMP, it appears as a solid line above the bar. Each bar is the mean of 5 trials. The error bars are 95% confidence intervals.

**Figure 10: News reader data usage, driving trace**



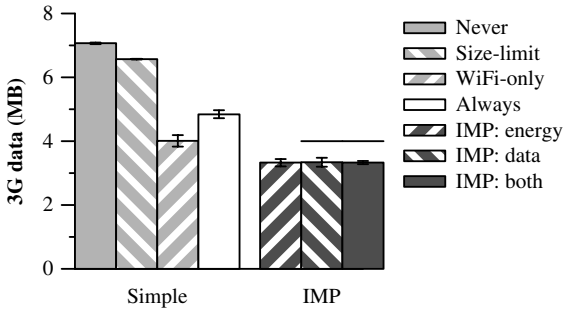

Average news article fetch time for 16 articles fetched over 20 minutes. The left set of bars shows results for simple heuristic-based prefetching strategies. The right set of bars shows results for IMP when energy, cellular data, or both are constrained. Each bar is the mean of 5 trials. The error bars are 95% confidence intervals.

**Figure 11: Average news article fetch times, walking trace**




Total energy usage for the news reader benchmark. The left set of bars shows results for simple heuristic-based prefetching strategies. The right set of bars shows results for IMP when energy, cellular data, or both are constrained. Where there is an energy budget set for IMP, it appears as a solid line above the bar. Each bar is the mean of 5 trials. The error bars are 95% confidence intervals.

**Figure 12: News reader energy usage, walking trace**




Total cellular data usage for the news reader benchmark. The left set of bars shows results for simple heuristic-based prefetching strategies. The right set of bars shows results for IMP when energy, cellular data, or both are constrained. Where there is a data budget set for IMP, it appears as a solid line above the bar. Each bar is the mean of 5 trials. The error bars are 95% confidence intervals.

**Figure 13: News reader data usage, walking trace**

for that particular feed. For the chosen feeds, the read rate ranges from 25% to 100%, and the total read rate across all feeds is 64%.

Lacking a trace for newsreader user behavior, we chose to model a user's think time between article fetches as a random variable. Recent work in modeling web user dwell times observes that the majority of page visits last one minute or less [17]. Thus, after the user reads an article, the benchmark pauses for a random think time uniformly distributed between 30 and 60 seconds, then moves on to the next article in the feed, continuing until the feed is exhausted. The user does not pause after choosing to skip an article. As with the email application, the benchmark lasts 20 minutes.

Since the news reader does not prefetch article contents by default, we set the value used in the threshold strategy to 128 KB, which is just above the median article size of the 25 articles.

Our news reader associates a different class with each feed. The results of this evaluation are shown by the middle set of bars in Figures 8, 9, and 10. In the next section, we evaluate the benefit of prefetch classes by using only a single class. For ease of comparison, these results are shown by the rightmost set of bars in each figure.

As with the email benchmark, the always-prefetch strategy produces the best average fetch time, though the greater inaccuracy of the prefetch hints in this experiment reduces the benefit of prefetching compared to the email experiment.

In contrast to the email benchmark, aggressive prefetching has an adverse affect on energy usage. This is primarily due to the lower prefetch hint accuracy and the large size of some articles. Nevertheless, when given an energy budget of 450 Joules, IMP performs enough prefetching to reduce average fetch time by 29–58% compared to the never-prefetch, size-limit, and WiFi-only strategies, while also meeting the energy goal. As we discuss in the next section, IMP benefits from separating prefetch hints into classes and first prefetching articles most likely to be read. While the always-prefetch strategy has better performance than IMP, it does not meet the energy goal.

When we constrain data usage to 5 MB, IMP has more freedom in scheduling prefetches than in the energy-constrained scenario and reduces average fetch time by 47–68% compared to the never-prefetch, WiFi-only, and size-limit strategies. IMP meets the cellular data goal in all runs, reducing 3G data usage by 45–62%.

When constraining both resources, we kept the 450 Joule energy budget and set a 6 MB data budget. IMP's behavior is similar to its behavior in the energy-constrained scenario: it meets each goal in all runs and reduces average fetch time by 36–62% compared to the never-prefetch, size-limit, and WiFi-only strategies.

IMP uses less than the given cellular data budget, for the same reasons as above: IMP finishes prefetching all the hinted articles, after which no more articles are sent on the cellular network.

Figures 11, 12, and 13 show the results for the newsreader application on the walking trace. As in the case of the email application, the best strategy for this trace is to always prefetch. In this application, IMP comes closer to the average fetch time of always-prefetch than any of the other simple strategies, achieving average fetch time 2–6x lower than those strategies.

For the energy-constrained experiments using the walking trace, we set an energy budget of 200 Joules. IMP meets the goal in all trials, reducing energy usage by 25–35% compared to the never-prefetch, size-limit, and WiFi-only strategies.

IMP uses 7–10% more energy than the always-prefetch strategy on average. The reason for this is that IMP decides to pause prefetching when it encounters a particularly large prefetch early in the benchmark. At this time in the experiment, IMP has sent a few prefetches on 3G to amortize the tail time, but it decides the potential savings of the tail time is not worth the energy that will be spent to fetch the large article, especially since the article is less likely to be read than the articles that IMP has already prefetched.

This results in two demand fetches on 3G later in the benchmark, since IMP does not resume prefetching until it has ceased spending energy for a while to catch up to its target rate. Thus, IMP incurs additional tail time compared to the always-prefetch strategy.

For the data-constrained experiments, we set a 3G data budget of 4 MB. IMP meets this goal as well in all trials, reducing 3G data usage by 17–53% compared to the simple strategies. Interestingly, IMP reduces 3G data usage even compared to the WiFi-only strategy. The reason for this is that IMP uses prefetch classes, whereas WiFi-only does not, which causes IMP to prioritize articles most likely to be read. Since WiFi is limited in this trace, the WiFi-only strategy fails to prefetch some particularly large articles that later must be demand-fetched over 3G.

#### *5.2.3 Benefit of prefetch classes*

To evaluate the benefit of prefetch classes, we executed the same benchmark, using the driving trace, without having the news reader use classes to differentiate prefetch hints of articles in different feeds.

The benefit of prefetch classes is greatest in the scenarios in which energy is constrained; in these two scenarios, the multi-class news reader reduces average fetch time by 42–47% compared to the single-class news reader scenarios. Though the energy constraint causes both versions of the news reader to prefetch cautiously, the multi-class news reader benefits greatly by first prefetching the articles with the highest likelihood of being read.

When the application uses prefetch classes, IMP meets all goals in every trial. However, without prefetch classes, IMP meets only 87% of the resource goals, failing to meet the data goal in 1 of 5 trials of the data-constrained scenario and the energy goal in 1 of 5 trials of the dual-constraint scenario. In the two trials in which IMP missed the goal, it overshot by less than 5.3% due to a late demand fetch in the benchmark. While no adaptive system will ever be perfect, we expect that a longer experiment with larger goals (typical of mobile phone usage today) would give IMP substantially more room for error and probably eliminate these overshoots.

Nevertheless, prefetch classes provide an added margin of safety by allowing IMP to make more informed decisions. IMP is able to target its prefetch effort to articles that are more likely to be read. Further, IMP's per-item predictions are more accurate since they are based on per-feed statistics rather than overall hit rates.

#### *5.2.4 Discussion*

From the above results, we conclude that IMP does an excellent job of meeting budgets for constrained resources. IMP meets all budgets in every trial for both the email and news reader applications (as long as the latter uses prefetch classes to differentiate feeds).

In the newsreader experiment on the walking trace, when energy is the only constraint, the always-prefetch strategy is best for both fetch time and energy usage, as discussed above. However, in all other cases, when one or more budgetary constraints are specified, IMP outperforms all heuristic-based prefetching strategies we examined that also meet the budgetary constraints. Often, the performance improvement is a factor of two or more.

The benefit of prefetch classes is apparent not only in improving the percentage of budgets met for the news reader from 87% to 100%, but also in substantially improved response time. There is clearly a tradeoff between complexity and accuracy in the application interface. IMP can make better decisions if the application provides more information, but each additional specification places more burden on the application developer. In the case of prefetch classes, our results indicate that the small effort of differentiating hints by class (in the case of the news reader) can substantially improve results. This indicates that other, more contextual methods for differentiating prefetch accuracy could be beneficial.

## 6. RELATED WORK

One of the primary contributions of IMP is the unification of prior solutions to a seemingly disparate collection of problems to meet the unique challenges of mobile prefetching. In this section, we describe prior work in prefetching and in managing limited resources in mobile computing, and we discuss how IMP unifies and builds upon these ideas.

### 6.1 Prefetching

Prefetching is a long-studied technique used in a wide variety of computing domains. Processors predict which instructions or data will be needed and populate cache lines in advance [28]. File systems and databases attempt to infer application access patterns and fetch data items before they are needed [13, 22]. Distributed file systems fetch files from servers in advance of their use, both to improve performance and to maintain availability in the face of unreliable network connections [12].

The design of IMP is inspired by Transparent Informed Prefetching (TIP), which uses cost/benefit analysis to manage allocation of disk cache buffers between competing consumers: prefetched data and the LRU cache [23]. Such work recognizes that prefetching must be done carefully, lest it harm more than it helps. We observe the same high-risk/high-reward nature of prefetching in mobile computing and construct a cost/benefit analysis based on user-perceived performance improvement and the cost of prefetching in terms of battery energy and cellular data.

In the domain of mobile computing, prefetching has long been recommended as a primary technique for improving application performance [29]. Prior theoretical work in prefetch algorithm modeling and analysis observes, as we do, the tension between improving data access performance and spending limited bandwidth and energy resources to achieve that end. Persone et al. develop a prefetch cost model by which to numerically evaluate prefetch strategies based on mobility patterns [6]. However, their work considers different costs of prefetching in isolation from each other and from the benefit of prefetching. In contrast, IMP explicitly considers how different costs, such as energy usage and 3G data consumption, may be meaningfully combined and weighed against the benefits that prefetching can provide.

Lee et al. seek to improve the efficacy of prefetching and counteract its inherent uncertainty by improving the accuracy of predictions, using location-specific or context-specific information [14]. Web prefetching has long used spatial locality to predict what data users will request next [21]. Such efforts are beneficial and complementary to IMP. IMP allows the application to decide which data should be prefetched and instead addresses the decision of when to prefetch given limited resources and changing network conditions.

As we have demonstrated, having more accurate prefetch hints from the application allows IMP to deliver better results. Thus, IMP might benefit from prediction of network availability and quality, as done systems such as BreadCrumbs [19]. By knowing with greater accuracy what future bandwidth and WiFi coverage to expect, IMP can make better-informed decisions about whether prefetching now or waiting until later would be more cost-effective.

### 6.2 Limited resources in mobile networking

Several recent projects have proposed techniques for dealing with the different strengths and weaknesses of WiFi and cellular networks. Wiffler [4] explored the use of intermittently-

available WiFi to reduce 3G data usage and ease pressure on cellular networks, subject to application-specific delay tolerance. Bartendr [26] observed that the energy cost of sending data on a cellular network increases significantly as signal strength drops and that energy savings can be realized by predicting periods of good signal strength and, when possible, delaying transmissions to target those periods.

Additionally, much recent work targets the poor interaction between naive applications' cellular data usage patterns and the energy cost of 3G tail time. Balasubramanian et al. measured the significant amount of energy wasted in 3G tail time and developed TailEnder to amortize tail energy cost with batching and prefetching [5]. The Tail Optimization Protocol [24] predicts long idle periods and direct the cellular radio to release radio resources and enter the low-power idle mode without waiting for the tail time. TailTheft [18] "steals" parts of the tail time for small transmissions without extending the tail time.

We share these systems' goal of reducing energy and cellular data costs, and indeed, many of the techniques they describe are applicable to IMP as well. However, we also observe that resource conservation, while a worthy goal, is not always the most important goal for a mobile user, and that often, energy and cellular data can and should be spent more freely in order to improve the user's experience. We also observe that prefetching is not always beneficial, and that aggressively prefetching data that will not be used is unnecessarily wasteful. Hence, IMP explicitly considers the changing relative importance of these goals to tune its decisions over time, and it also considers the observed accuracy of prefetch hints to determine the value of prefetching.

IMP uses Intentional Networking [10] to simplify the use of multiple wireless networks and to prevent prefetch traffic from penalizing the performance of interactive traffic. IMP also benefits from the power modeling work of the PowerTutor project [30], which enabled automatic derivation of power models through a series of tests that exercise different components of a mobile device in turn and isolate the power consumption of each. Finally, IMP adopts goal-directed adaptation techniques developed in Odyssey for energy savings [8], which we additionally apply to the cellular data resource. IMP applies these ideas to the new domain of mobile prefetching.

# 7. CONCLUSION

Prefetching is an invaluable tool for improving the performance of mobile applications. However, prefetching can be costly in terms of the energy and cellular data consumed, and these costs are exacerbated by the possibility of prefetching data that the user never requests. As a result, many mobile applications employ simple but suboptimal heuristics as a means of grasping some benefit while avoiding undue complexity.

We propose that mobile computing systems provide explicit prefetching support to applications. IMP shifts the burden of complex prefetching decisions from the application to the system. It assumes responsibility for estimating the cost and benefit of prefetching, tracking the importance of energy and cellular data over time, and scheduling prefetches to improve interactive performance while meeting specified resource usage goals.

Our evaluation shows that IMP is able to meet specified energy and cellular data usage goals. Additionally, we show that in most cases, whenever simple heuristic-based prefetching strategies also meet the goals, IMP outperforms those strategies on interactive fetch time, often by a factor of two or more.

# Acknowledgments

We thank the anonymous reviewers and our shepherd, Arun Venkataramani, for comments that improved this paper. The work reported in this paper was supported in part by DARPA under grant HR0011-10-1-0081. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA, the University of Michigan, Ford, Arbor Networks, or the U.S. government.

# 8. REFERENCES

[1] ABERDEEN, D., PACOVSKY, O., AND SLATER, A. The learning behind Gmail priority inbox. In *NIPS 2010 Workshop on Learning on Cores, Clusters and Clouds* (Mount Currie, British Columbia, Canada, December 2010).

[2] ANAND, M., NIGHTINGALE, E. B., AND FLINN, J. Ghosts in the machine: Interfaces for better power management. In *Proceedings of the 2nd International Conference on Mobile Systems, Applications and Services* (Boston, MA, June 2004), pp. 23–35.

[3] APACHE HTTPCOMPONENTS. `http://hc.apache.org/`.

[4] BALASUBRAMANIAN, A., MAHAJAN, R., AND VENKATARAMANI, A. Augmenting mobile 3G using WiFi. In *Proceedings of the 8th International Conference on Mobile Systems, Applications and Services* (San Francisco, CA, June 2010), pp. 209–221.

[5] BALASUBRAMANIAN, N., BALASUBRAMANIAN, A., AND VENKATARAMANI, A. Energy consumption in mobile phones: A measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement* (Chicago, IL, USA, November 2009), pp. 280–293.

[6] DE NITTO PERSONE, V., GRASSI, V., AND MORLUPI, A. Modeling and evaluation of prefetching policies for context-aware information services. In *Proceedings of the 4th International Conference on Mobile Computing and Networking* (Dallas, TX, USA, October 1998), pp. 55–65.

[7] FLINN, J. *Extending Mobile Computer Battery Life through Energy-Aware Adaptation*. PhD thesis, Department of Computer Science, Carnegie Mellon University, December 2001.

[8] FLINN, J., AND SATYANARAYANAN, M. Managing battery lifetime with energy-aware adaptation. *ACM Transactions on Computer Systems (TOCS) 22*, 2 (May 2004), 137–179.

[9] FUTURE (JAVA 2 PLATFORM SE 5.0). `http://docs.oracle.com/javase/1.5.0/docs/api/java/util/concurrent/Future.html`.

[10] HIGGINS, B. D., REDA, A., ALPEROVICH, T., FLINN, J., GIULI, T. J., NOBLE, B., AND WATSON, D. Intentional networking: Opportunistic exploitation of mobile network diversity. In *Proceedings of the 16th International Conference on Mobile Computing and Networking* (Chicago, IL, September 2010), pp. 73–84.

[11] K-9 MAIL. `http://code.google.com/p/k9mail/`.

[12] KISTLER, J. J., AND SATYANARAYANAN, M. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems 10*, 1 (February 1992), 3–25.

[13] KOTZ, D., AND ELLIS, C. S. Practical prefetching techniques for parallel file systems. In *Proceedings of the First International Conference on Parallel and Distributed Information Systems* (Miami Beach, FL, USA, December 1991), pp. 182–189.

[14] LEE, D. L., XU, J., ZHENG, B., AND LEE, W.-C. Data management in location-dependent information services. *IEEE Pervasive Computing 1*, 3 (2002), 65–72.

[15] LINUX ADVANCED ROUTING AND TRAFFIC CONTROL. `http://lartc.org/`.

[16] THE LINUX FOUNDATION. *netem*. `http://www.linuxfoundation.org/collaborate/workgroups/networking/netem`.

[17] LIU, C., WHITE, R. W., AND DUMAIS, S. Understanding web browsing behaviors through weibull analysis of dwell time. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (Geneva, Switzerland, July 2010), pp. 379–386.

[18] LIU, H., ZHANG, Y., AND ZHOU, Y. TailTheft: Leveraging the wasted time for saving energy in cellular communications. In *Proceedings of the 6th ACM International Workshop on Mobility in the Evolving Internet Architecture* (Washington, D.C., USA, June 2011), pp. 31–36.

[19] NICHOLSON, A. J., AND NOBLE, B. D. BreadCrumbs: Forecasting mobile connectivity. In *Proceedings of the 14th International Conference on Mobile Computing and Networking* (San Francisco, CA, September 2008), pp. 46–57.

[20] OPENINTENTS NEWS READER. `http://www.openintents.org/en/newsreader`.

[21] PADMANABHAN, V. N., AND MOGUL, J. C. Using predictive prefetching to improve world wide web latency. *Computer Communication Review 26*, 3 (1996), 22–36.

[22] PALMER, M., AND ZDONIK, S. B. Fido: A cache that learns to fetch. In *Proceedings of the 17th International Conference on Very Large Data Bases* (Barcelona, Spain, September 1991), pp. 255–264.

[23] PATTERSON, R. H., GIBSON, G. A., GINTING, E., STODOLSKY, D., AND ZELENKA, J. Informed prefetching and caching. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles* (Copper Mountain, CO, December 1995), pp. 79–95.

[24] QIAN, F., WANG, Z., GERBER, A., MAO, Z. M., SEN, S., AND SPATSCHECK, O. TOP: Tail optimization protocol for cellular radio resource allocation. In *Proceedings of the 18th IEEE International Conference on Network Protocols* (Kyoto, Japan, October 2010), pp. 285–294.

[25] RAHMATI, A., QIAN, A., AND ZHONG, L. Understanding human-battery interaction on mobile phones. In *Proceedings of the 9th ACM International Conference on Human Computer Interaction with Mobile Devices and Services* (Singapore, September 2007), pp. 265–272.

[26] SCHULMAN, A., NAVDA, V., RAMJEE, R., SPRING, N., DESHPANDE, P., GRUNEWALD, C., JAIN, K., AND PADMANABHAN, V. N. Bartendr: A practical approach to energy-aware cellular data scheduling. In *Proceedings of the 16th International Conference on Mobile Computing and Networking* (Chicago, IL, USA, September 2010), pp. 85–96.

[27] SHAH, S., AND NOBLE, B. D. A study of e-mail patterns. *Software: Practice and Experience 37*, 14 (2007), 1515–1538.

[28] SMITH, A. J. Cache memories. *ACM Computing Surveys 14*, 3 (September 1982), 473–530.

[29] WATSON, T. Application design for wireless computing. In *Proceedings of the First Workshop on Mobile Computing Systems and Applications (HotMobile)* (Santa Cruz, CA, USA, December 1994), pp. 91–94.

[30] ZHANG, L., TIWANA, B., QIAN, Z., WANG, Z., DICK, R. P., MAO, Z. M., AND YANG, L. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis* (Scottsdale, AZ, USA, October 2010), pp. 105–114.