Smart Phones Need Smarter Applications

Neal H. Walfield Randal Burns
Johns Hopkins University

Abstract

Smart phones should not just accompany their owners: they should provide them with the data they want whenever and wherever they are. This does not mean that the user should always be able to fetch data on demand: wireless communication requires a significant amount of energy; cellular bandwidth is often limited; and, coverage is not ubiquitous. Instead, scheduling data stream updates, e.g., podcast downloads and photo uploads, should incorporate predictions of where the user will be, when and where data will be needed, and when transfer conditions are good (e.g., WiFi is available). The challenges we have encountered while designing such a framework include scheduling transmissions while respecting multiple optimization goals, application integration, modeling user and data stream behavior, and managing prefetched data. To better understand user behavior and evaluate our framework, we are gathering traces of smart phone use. Initial results show potential energy savings of over 70%.

1 Introduction

Mobile devices promise to keep users connected. Yet, limited energy [12, 2, 10, 11, 1], data-transfer allowances [3, 13], and cellular coverage [11] reveal this assurance to be more a hope than a guarantee. This situation can be improved by increasing battery capacity, providing more generous data-transfer allowances, and expanding cellular coverage. We propose modifying software to be more efficient with the available resources. Many applications exhibit flexibility in when they must transmit data. Such applications include those that read from or write to data streams, e.g., podcast managers and photo sharing services. These applications can prefetch data and delay uploads until good conditions arise.

More efficiently managing the available energy, the user's data-transfer allowance and data availability can improve the user experience. Increasing battery life raises the user's confidence that a charge will last the whole day, even with intense use. Alternatively, a smaller battery can be used. Explicitly managing the data-transfer allowance enables users to choose less expensive data plans without fearing that the allowance will be exceeded, which may result in expensive overage fees and bill shock, a common occurrence in the US [4]. Finally, accounting for availability by, e.g., prefetching data, hides spotty and weak coverage and user-perceived latency is reduced.

We have encountered two main challenges to exploiting scheduling flexibility in data streams: predicting needed data and coordinating resource consumption. First, applications need to predict when and what to prefetch. Consider Alice, who listens to the latest episode of the hourly news on her 5pm commute home. A simple policy prefetches episodes as they are published. As Alice only listens to the 4pm or 5pm episode, downloading episodes as they are published wastes energy and her data-transfer allowance. An alternative policy prefetches when power and WiFi are available, typically overnight. But, Alice wants the latest episode on her commute home, not the one from 6am. The scheduling algorithm needs to learn when and how Alice (the individual, not an aggregate model) uses data streams. The second challenge is coordinating the use of available resources. In particular, the data-transfer allowance and local storage must be partitioned between the applications and the user. This management should not interfere with the user by, e.g., exhausting the transfer allowance so that the user cannot surf the web, or causing an out-of-space error to occur when the user saves files. Further, the allocations should adapt to the user's changing preferences.

Research on scheduling transmissions on smart phones has focused on reducing energy consumption by predicting near-term conditions. Bartendr delays transmissions until the signal strength is likely strong [12]; TailEndr groups transmissions to amortizes energy costs [2]; BreadCrumbs, among others, predicts

WiFi availability to reduce energy spent needlessly scanning [10, 11, 1]. Our work differs from these in that we are concentrating on scheduling transmissions before a *contextual* deadline, as predicted from observed user behavior. We also consider the cellular data-transmission allowance, which is an increasingly common constraint [3, 13]. Further, because we enable aggressive prefetching, we consider how to manage storage. Given multimedia data access patterns in which subscribed to data is used at most once [8, 7], common replacement techniques, such as LRU, perform poorly.

In this paper, we make the case for better scheduling of background data-stream updates to save energy, to make better use of data-transfer allowances, to improve disconnected operation, and to hide data-access latencies, all of which advance our ultimate goal of improving the user experience. Initial results show potential energy savings of over 70%. We present *Woodchuck*, a framework to accomplish this. Applications provide simple descriptions of transmission tasks to a *transmission broker*. The broker uses these and *predictions* of when, where and how data will be used as well as when streams will be updated to schedule the requests so as to minimize battery use, to respect any data-transmission allowance, and to maximize the likelihood that data that the user accesses is available. We also consider how to *manage storage* for holding prefetched data. Finally, we describe a *user study* we are conducting to evaluate Woodchuck.

2 Potential Savings

We propose to improve the user experience on smart phones by exploiting particular user behaviors, namely, data stream usage, such as podcasts. Recent user studies suggest that this usage is common among the heaviest data users [6, 9, 14, 8]. And, Ericsson is convinced this behavior will become mainstream [5]. Based on a simple model of power consumption, we show that playing audio podcasts that have been opportunistically prefetched results in 70% less energy use than streaming them over 3G. Moreover, this decreases the use of the cellular data transfer allowance and improves data availability.

Recent user studies show that the heaviest data users spend a lot of time interacting with their smart phones and a large portion of their traffic is not interactive. These findings suggest that Woodchuck has many opportunities to improve the user experience among top users. Hossein et al.'s study of smart phone usage [6] shows users spend between 30 minutes and 8 hours per day interacting with their device, the amount of network traffic is between 1 MB and 1 GB per day, and the percentage of not interactive traffic ranges from 20% to 90%. Trestian et al. [14] suggest that these numbers could be conservative as mobile users avoid bandwidth- and battery-intensive applications when on the go. In their study of mobile device traffic on 20,000 DSL lines, Maier et al. [9] report that web browsing accounted for 15% of the traffic volume generated by mobile devices, which is less than that generated by PCs. The most visited web sites were video-on-demand and streaming sites, and 40% of the volume was multimedia content. They also found that 2-5% of the volume consisted of small XML files, which provides strong evidence that subscription-based content is widely used. Gunawardena et al. studied podcasts on the Zune Social service [8, Fig. 19] and found that subscription content is used regularly: 16% of users listened to three or more podcasts per day. At an average 15 MB per podcast and assuming a 96 Kbit/s encoding, this corresponds to an hour of audio per day.

There is evidence that the always-connected behavior shown by top users will become mainstream, broadening Woodchuck's potential impact. Hossein et al. [6] observe that user behavior is dispersed relatively uniformly between the observed extremes. We interpret this as the adoption of a new technology: as software and services' ease of use improves, more users are willing to invest the time required to use them. This is consistent with Ericsson's observations that consumers' expectations and behaviors are changing with respect to video [5]: so-called digital natives and, increasingly, the general population are no longer satisfied with broadcast TV's fixed schedule; they want the flexibility of streaming and on-demand services, the ability to watch what they want, when they want, where they want. Maier et al.'s 11 month study [9] supports this: they observed a doubling in the number of mobile devices and a six-fold increase in the traffic volume generated by them.

Gunawardena et al.'s data [8, Fig. 29] reveal that users do not just want flexibility, they want fresh content: approximately 10% of podcasts listened to on portable Zune devices were listened to within a day of their *publication*, implying simple prefetching policies, such as fetching when there is power and WiFi, are insufficient. We find this a surprisingly large amount considering that the portable devices could only be synchronized by attaching them to a computer.

To estimate the potential energy savings of prefetching, we measured the energy to stream audio, to play local files, to download over WiFi, and at idle, see Table 1. Listening to an hour's worth of audio requires 3.6 KWs when streamed over 3G but 1 KWs, *just 28% as much*, when the data is available locally. The predicted savings are conservative as our test device was stationary and had a strong signal: Schulman et al. [12] observe that six times as much energy is required when the signal is weak.

Not all episodes can be successfully prefetched when there is free WiFi and power: the hourly news is pub-

Access	Activity	Watts	Ratio
3G	Play 56.Kb/s stream	1.00	12.5
Edge	Play 56.Kb/s stream	0.96	12.0
WiFi	Play 56.Kb/s stream	0.75	9.3
Local Storage	Play 56.Kb/s files	0.28	3.5
Local Storage	Play 128.Kb/s files	0.27	3.4
Local Storage	Play 320.Kb/s files	0.32	4.0
WiFi	Download at 4.7 Mb/s	1.23	15.4
WiFi	Download at 1.0 Mb/s	0.91	11.4
WiFi	Download at 256 Kb/s	0.76	9.5
None	Idle	0.08	1
None	Idle, LCD on	0.27	3.4

Table 1: Energy used on a Nokia N900 to play MP3s, to download via WiFi, and at idle. A full charge has about 18 KWs = 5 Wh.

lished hourly, but WiFi and power are often only available at night. Significant energy savings are possible in this case by opportunistically fetching data when there is good WiFi, e.g., before the user leaves work. Consider the case where 8 hours of audio, about 200 MB of data, are prefetched over WiFi. At 4.7 MB/s, this requires 420 Ws (2.5% of the capacity). If the user listens to 30 minutes of audio (25 MB) on the commute home, only an additional 480 Ws (2.7% of the capacity) are required. Streaming 30 minutes of audio over 3G requires 1800 Ws, twice the amount of energy to prefetch 8 times the data and listen to the same audio. If these energy savings are realized, users may use more fresh content, increasing Woodchuck's impact: currently, users avoid draining the battery by, e.g., not listening to music [14].

In terms of managing the data transfer allowance, if the current top 16% of users who listen to 3 or more podcasts per day streamed them, they would transfer 1.5 GB per month. This is three times the new monthly limit on T-Mobile UK's network [13]. Verizon users with a 200 MB allowance would exhaust their plan in just 4 days and those with the 2 GB plan would have little for anything else [3]. If prefetching resulted in a 90% hit rate, just 150 MB would need to be streamed.

Finally, prefetching can improve data availability. Rahmati and Zhong report that although users in their study were connected to a cell tower 99% of the time, they were only able to transfer data 80% of the time [11].

3 Woodchuck Architecture

Woodchuck consists of three main components. First, Woodchuck implements a *transmission broker* whose main job is to schedule applications' transmission requests and gather information about them. Unlike individual agents in a decentralized scheme, a broker has a global view of the system and runs continuously, which enables agile adaptations. Second, to predict what data is likely to be used, Woodchuck *collects contextual information*, such as location tracks, WiFi and power availability, and cellular signal strength. Finally, Woodchuck *manages storage*. Woodchuck adjusts the size of applications' prefetch areas according to the user's preferences, which it learns from the user's behavior. The storage manager can adapt even if the application is not running, thereby avoiding annoying out-of-space errors.

```
enum type { META_DATA, CONTENT };

Application → Broker:
  task (stream_id, request_id, recurrent_p,
        timeout, size, type, callback)
  cancel (stream_id, request_id)
Broker → Application:
  transmit (stream_id, request_id, seq)
Application → Broker:
  success (request_id, seq, actual_size,
           struct { filename, discardable } data[])
  failure (request_id, seq, error, retry_p)
```

Figure 1: Transmission Broker API

3.1 Transmission Broker

The transmission broker consists of a transmission request API (see Figure 1), a data miner, and a scheduler.

To make integration of the transmission broker in existing applications as easy as possible, we examined a few applications that automatically synchronize data streams. We found that a common pattern is to set up a timer to invoke a callback when the next synchronization should occur, e.g., if using `g_timeout_add` for GTK. When the timer fires, the program optionally checks a few conditions, e.g., if WiFi is available, and then initiates the transfer. This process is repeated (only) as long as the program is running.

Using Woodchuck, instead of a timer-based callback, an application registers each data stream (e.g., each podcast subscription) using **task**. When the broker decides that a transmission should occur, it, using, e.g., D-Bus, starts the application if it is not already running and invokes the **transmit** upcall. As the broker is always running, streams can be synchronized at any time. When the transmission completes, the application calls either **success** or **failure** as appropriate. Recurrent requests are automatically reinserted into the scheduling queue. If the application reports an error, the transmission is retried.

The API encourages, but does not require, programmers to separate transmissions of meta-data from object content (e.g., updating a mailbox's index vs. downloading new e-mails). This better enables the broker to manage the data transmission allowance: if transmitting data is expensive, the broker can decide to synchronize the meta-data, but delay fetching the objects. Thus, the user

is still informed about the arrival of new data, e.g., new e-mails, and can choose to explicitly authorize the potential overage charge to download the content.

To separately manage meta-data and content, the application registers the stream as a meta-data stream and, for each new object, the application creates a new non-recurrent request with an immediate timeout. This enables the broker to decide whether to delay the download or whether to bother downloading the data at all. This does not preclude the application from downloading the data, if the user explicitly requests it. In such cases, the application calls **success** as usual to tell the broker that the request was fulfilled. The broker uses this to adjust its model of the user's preferences.

When a transmission is successful, the application includes as a parameter to **success** the files it updated. The broker monitors these files for accesses, which often implies use, using, e.g., Linux's *inotify*. The broker knows which stream a file came from and when the data was downloaded thanks to the `stream_id` and a `request_id` parameters to **task**. Using this knowledge, it can learn a stream's object arrival pattern, which streams are used, which objects are used, object access patterns (e.g., in order, only new objects, random), how frequently objects are used, and a stream's delay tolerance.

3.2 Predicting the When, Where and How

In addition to learning how streams are updated and used, Woodchuck predicts the context in which objects are used, how the user moves between contexts, and what resources (e.g., power, WiFi, and signal strength) are available in different locations. We plan to evaluate different data mining and machine learning algorithms for these tasks. With this information, Woodchuck predicts which objects have the highest expected utility and prefetches them when conditions are good.

A context is a set of circumstances, a combination of temporal and spatial properties. A user's morning routine, which includes exercise and listening to a podcast, describes a context. She may do this even when travelling, in which case location plays only a minor role. A context may include movement, e.g., the morning commute to work is a context. To identify contexts, we are considering clustering algorithms over locations, location tracks, time of day and day of week. As Trestian et al. found that most users spend most of their time in approximately three places [14], if quantized carefully, the state space appears manageable.

To determine the objects used in a particular context, whenever an object is used, Woodchuck records it and the current context. Woodchuck will use this information to compute the probability of a stream conditioned on the context. Using this knowledge, Woodchuck will predict what streams a user will most likely access given a context and will prefetch accordingly. A complication is that users' behaviors change over time. This can be accounted for by aging the data. If this makes computing probabilities too expensive, Woodchuck can compute the probabilities when there is power and cache the results.

We also plan to use a Bayesian analysis to predict succeeding contexts and the resources available in a context. Again, Woodchuck records historical data to construct a prior probability, which it will use to make predictions.

```
Application → Broker:
  cleanup (callback)
  query (stream) ↦ total_bytes
Broker → Application:
  free (struct { stream_id, request_id } requests[])
```

Figure 2: Storage Management API

3.3 Managing Storage

When applications aggressively prefetch data, storage becomes a shared resource and it must be partitioned among the applications as well as the user. Woodchuck allows applications to specify all of their data requirements ahead of time (using **task**). Woodchuck chooses which data to prefetch based on maximizing data utility under space availability, connectivity and cost constraints. Because applications can specify any data and Woodchuck chooses, they can be arbitrarily aggressive.

Woodchuck keeps the maximum utility objects based on observed access patterns, as recorded by the transmission broker; it implicitly computes the cache size for each stream. Woodchuck models streams that have not been used for a long time with a smaller utility than streams that have been used more recently. Woodchuck also recognizes and reacts to other access patterns. Returning to the example of the hourly news cast, only the recent episode has any value. Woodchuck accordingly assigns only the most recent episodes any utility. Woodchuck also detects streams that exhibit a use-once property, e.g., multimedia [7]. In these cases, Woodchuck assigns used files no utility.

Woodchuck provides two mechanisms for reclaiming space: discardable files and an application upcall. When an application registers downloaded data (using **success**), it indicates whether Woodchuck can discard the corresponding files (see Figure 1). We expect this to be common as many applications can already deal gracefully with files that disappear, which happens, for example, when users use the file manager to reclaim space rather than the application. As the application keeps the meta-data separately, e.g., the list of podcast episodes, it can redownload the data should the user subsequently need

it. Some data does not lend itself to being stored as files. For instance, the application might keep all data in a single database. In this case, Woodchuck relies on the applications to disentangle the data and free space. If an application has registered a callback using **cleanup**, Woodchuck invokes the **free** upcall with each object the application should delete (Figure 2). Woodchuck starts the application if it is not running. If the application frees an insufficient amount of space, Woodchuck must accept this; Woodchuck must not delete precious user data. In this case, the user must intervene. To help understand the situation, Woodchuck provides a tool to visualize applications' storage use. To avoid multiple managers conflicting, the storage manager and API can be extended to also manage other applications' storage caches, e.g., the browser's cache or the file manager's file previews.

4 Evaluation

We made four claims: Woodchuck can save energy, manage a data allowance, predict what files a user is likely to need, and effectively manage storage. To evaluate these claims, we have obtained institutional review board approval to collect traces of user behavior. We have written monitoring software for Nokia's N900 that logs file accesses, when programs start and exit, when the user interacts with the device, the battery status (available energy and when it is charged), network connection statistics (AP or provider's network name, signal strength and bytes transferred), and available networks (visible cell towers and WiFi APs and their signal strength).

We can replay the traces to measure how Woodchuck might perform. We say might as different prefetching decisions can result in different user behavior. Consider a user on an airplane: the data that the user can access is just that which is on the device; different prefetching schemes could have resulted in different cache contents.

We also want to measure Woodchuck relative to other policies. In particular, we are interested in simple policies such as always fetch on-demand and only prefetch when there is WiFi and power. These measurements will allow us to evaluate whether the complexity of the approach is justified. We are also interested in understand what the limits of this approach are, which can be measured using an oracle policy.

Having found policies with reasonable trade-offs between complexity and effectiveness, we will conduct another user study in which users actually test Woodchuck.

5 Conclusions

The smart phone user experience must be improved: battery life is too short, data allowances are unnecessarily constricting, and connectivity is not universal. A promising approach is to improve software by enabling aggressive prefetching. We found that listening to prefetched audio needs less than 30% of the energy required to stream it over 3G while stationary and with a strong signal, i.e., in ideal conditions. Because users want fresh content, it is not sufficient to prefetch when there is WiFi and power. Based on this, we propose Woodchuck, a framework which learns how users use data and in which contexts, predicts what data a user will need, and prefetches it when conditions are good. To evaluate Woodchuck, we are gathering user traces.

References

[1] ANANTHANARAYANAN, G. AND STOICA, I. Blue-Fi: Enhancing Wi-Fi Performance Using Bluetooth Signals. In *MobiSys* (June 2009).

[2] BALASUBRAMANIAN, N., BALASUBRAMANIAN, A., AND VENKATARAMANI, A. Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications. In *IMC* (Nov. 2009).

[3] CHENG, J. Verizon Confirms the Future of 3G Data is Tiered. `http://arstechnica.com/gadgets/news/2010/09/verizon-confirms-the-future-of-3g-data-is-tiered.ars` (Sept. 2010).

[4] CONSUMER REPORTS. Bill Shock is Common. `http://www.consumerreports.org/cro/magazine-archive/2011/january/electronics/best-cell-plans-and-providers/cell-phone-bills/index.htm` (Jan. 2011). Accessed 12 Jan. 2011.

[5] ERICSSON AB. What Consumers Want from TV/Video Solutions. `http://www.ericsson.com/news/090626_what_consumers_wants_254740099_c` (June 2009). Accessed 1 Jan. 2011.

[6] FALAKI, H., MAHAJAN, R., KANDULA, S., LYMBEROPOULOS, D., GOVINDAN, R., AND ESTRIN, D. Diversity in Smartphone Usage. In *MobiSys* (June 2010).

[7] GUMMADI, K. P., DUNN, R. J., SAROIU, S., GRIBBLE, S. D., LEVY, H. M., AND ZAHORJAN, J. Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload. In *SOSP* (Oct. 2003).

[8] GUNAWARDENA, D., KARAGIANNIS, T., PROUTIERE, A., AND VOJNOVIC, M. Characterizing Podcast Services: Publishing, Usage, and Dissemination. In *IMC* (Nov. 2009).

[9] MAIER, G., SCHNEIDER, F., AND FELDMANN, A. A First Look at Mobile Hand-Held Device Traffic. In *PAM* (Apr. 2010).

[10] NICHOLSON, A. J. AND NOBLE, B. D. BreadCrumbs: Forecasting Mobile Connectivity. In *MobiCom* (Sept. 2008).

[11] RAHMATI, A. AND ZHONG, L. Context-Based Network Estimation for Energy-Efficient Ubiquitous Wireless Connectivity. *TOMC 10* (2011).

[12] 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 *MobiCom* (Sept. 2010).

[13] T-MOBILE UK. Changes to Mobile Internet Fair Use Policies. `http://support.t-mobile.co.uk/help-and-support/index?page=home&cat=DATA_CHANGES` (Jan. 2011). Accessed 11 Jan. 2011.

[14] TRESTIAN, I., RANJAN, S., KUZMANOVIC, A., AND NUCCI, A. Measuring Serendipity: Connecting People, Locations and Interests in a Mobile 3G Network. In *IMC* (Nov. 2009).