PAWP: A Power Aware Web Proxy for Wireless LAN Clients

Marcel C. Roşu, C. Michael Olsen, Chandra Narayanaswami
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598, USA
{rosu, cmolsen, chandras}@us.ibm.com

Lu Luo
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213, USA
luluo@cs.cmu.edu

Abstract

The relative power consumed in the WLAN interface of a mobile device is rising due to significant improvements in the energy efficiency of the other device components. The unpredictability of the incoming WLAN traffic limits the effectiveness of existing power saving techniques.

This paper introduces a Power Aware Web Proxy (PAWP) *architecture designed to schedule incoming web traffic into intervals of high and no communication. This traffic pattern allows WLAN interfaces to switch to a low power state after very short idle intervals.* PAWP *uses a collection of HTTP-level techniques to compensate any negative impact that traffic scheduling may have.* PAWP *does not require any client or web server modifications.*

In this paper, we describe our initial experiences with a PAWP *implementation for 802.11b WLANs. Our experiments show savings of more than 50% in the energy consumed by the WLAN interface. Finally, our experiences give us insights into possible browser improvements when power consumption is taken into account.*

1. Introduction

The relative power consumed in the WLAN interface of a mobile device is rising due to significant improvements in the energy efficiency of the other device components, such as processor, memory, display, and disk. The relative power consumption of the WLAN interface depends on the mobile device and it varies from 5-10% in high-end laptop computers to more than 50% in PDAs [6]. The actual amount of energy consumed for wireless communication depends on the client applications and their usage pattern, and on the actual WLAN technology. In particular, active web browsing and multimedia streaming are characterized by high energy consumption in the WLAN interface.

In this paper we focus on the popular 802.11 WLAN technology. The 802.11 specifications define two power management modes: *active* mode and *power save* mode. The client device is configured to switch the WLAN interface to *power save* mode during idle intervals, when it consumes 5 to 50 times less power than when active. Typically, the timeout value is set to 100 msecs.

The hard-to-predict nature of incoming traffic prevents using a lower timeout value without affecting application performance and interactivity. The comprehensive solution, which minimizes the energy used for wireless communication with no or limited impact on communication performance, requires perfect knowledge of the application traffic patterns, wired and wireless network configurations and conditions, and user preferences.

This paper describes a practical solution to the above problem for web browsing. Namely, we introduce *PAWP*, a *Power Aware Web Proxy* architecture that schedules the traffic directed to the wireless client into alternating intervals of *high* and *no* communication; in this process, the proxy takes into account the configuration of the client WLAN interface. The architecture does not require any modifications of the applications on the client device or remote server.

Our solution reduces the power consumed by the client WLAN interface because the interface can be configured to switch to a low power state after a much shorter delay. The same amount of data is transferred to the client device over approximately the same time interval but using a traffic pattern that allows the client WLAN interface to be powered-off for a larger fraction of the download time. To compensate for any increases in user-perceived latencies that traffic scheduling may induce, *PAWP* takes advantage of the information available at the application level by parsing downloaded documents and optimistically prefetching any embedded objects. The wireless device does not incur any penalty since prefetching is done by the proxy across the wired LAN/WAN.

PAWP is designed for environments where the application-level transfer rates between the client and the proxy are substantially higher than between the client and the origin web server (no-proxy configurations), i.e., where the data rates across the WLAN are higher than across the Internet. Our experiments demonstrate that the

lower the aggregate data rate of a web site, the higher the energy savings are.

In this paper, we discuss the main challenges we faced and some of the lessons we learned. We present experiments with popular new sites, such as *CNN*, *NY Times*, and *BBC*, e-commerce sites, such as *Amazon*, *eBay*, *Chase*, *Citibank*, and *American Express*, and professional organization sites, such as *SIGMOBILE*. The total number of objects in these pages ranges from 25 to 84 and the total size ranges from 42K and 535K bytes. We use one of the most complex web pages we found, *NY Times*, to identify the impact that various *PAWP* features have on client energy consumption and user-perceived latency.

We use IE 6.0 and Mozilla 1.4 to evaluate our solution; we collect HTTP traces using IBM's PageDetailer [16] and also measure of the energy consumed by the WLAN interface. Energy reductions vary between 9% and 61% and depend on the structure of the downloaded page and on the state of the network. Our experiments demonstrate that reductions in the energy consumed by the WLAN interface previously achieved for long transfers, such as multimedia streams [6], are possible for web pages consisting of a large number of embedded objects.

Scheduling incoming traffic into intervals of high and no communication can benefit future system-level techniques that coordinate the power management of multiple subsystems, such as processor and display subsystems. Initially introduced for watch-size devices [13], such techniques have been recently extended to small PDAs [3]. We expect similar techniques to be applied to larger client devices, such as sub-notebooks, and to be extended to managing additional subsystems, such as system memory and the WLAN interface.

Today's processors are capable of processing large amounts of data in very small intervals and can go back to sleep intermittently. As a result, system-level techniques for power management can be used effectively not only during idle periods but also during periods of *light usage*, such as web browsing or text editing. In fact, processors go to sleep in between user key strokes on a fast PC [10] and new technologies, such as *bistable* displays [25], are rapidly evolving to allow displays to be powered off immediately while retaining content. For system-level power management to be effective, mobile devices must be protected from random packet arrivals, i.e., from network interrupts. Therefore, proper scheduling of incoming WLAN traffic increases the applicability of such system-level techniques for power management.

Our solution has applicability beyond web browsing. First, other client applications, such as media players and email clients, and higher-level protocols, such as Web Services, use HTTP. Second, applications not using HTTP, such as the Notes mail client, can use the two main elements of the *PAWP* architecture, i.e., scheduling incoming traffic using an application-level proxy and using application semantics to compensate for any negative effects traffic scheduling may have, to reduce the power consumption of the WLAN interface and to improve application performance. For better performance, the resulting *PAxP* proxies can be integrated with the firewall, thereby avoiding multiple handling of incoming packets.

The paper is organized as follows: Section 2 provides an overview the power-management features available in 802.11 LANs. Sections 3 and 4 describe the architecture and the current implementation of the WLAN proxy. Section 5 discusses several experiments using the WLAN proxy. Section 6 is a brief survey of the related work. The last section describes possible extensions of this work.

2. Power management in 802.11 LANs

This section provides a brief overview of the power-management features of an 802.11 client interface, or *station,* in an infrastructure network. Only features relevant to networks using the distributed coordination function are described. For a complete description of power management in 802.11 networks, see [17].

The power management *mode* of a *station* can be either *active* mode or *power save* mode. The power *state* of a *station* can be either *Awake*, when the *station* is fully powered, or *Doze*, when the *station* consumes very little power but it is not able to receive or transmit frames. In *active* mode, the *station* is in the *Awake* state. In *power save* mode, the *station* is typically in *Doze* state but it transitions to *Awake* state to listen for select beacons, which are broadcasted every 102.4 ms by the wireless *access point*. The *station* selects how often it wakes up to listen to beacons when it associates with the *access point*. The transition between modes is always initiated by the *station* and requires a successful frame exchange with the *access point*. Therefore, the *access point* is always aware of the power management mode and beacon periodicity of each *station* in the LAN. The *access point* uses this information when communicating with *stations* which are in *power save* mode, as described next.

The *access point* buffers all pending traffic for the *stations* known to be in *power save* mode and identifies these *station*s in the appropriate beacons. When a *station* detects that frames are pending in the *access point*, the station may switch to *active* mode. Otherwise, it sends a poll message to the *access point*. If the beacon frame shows that more than one *station* has pending traffic, the poll message is sent after a short random delay; otherwise it is sent immediately. The *station* remains in the *Awake* state until it receives the response to its poll.

The *access point*'s response to the poll is either the next pending frame or an ACK frame, which signals that the *access point* delays the transmission of the pending frame and assumes the responsibility for initiating its delivery. The *station* must ACK every received frame. If the *More Data* field of the frame indicates additional pending frames, the *station* may send another poll frame. Otherwise, the *station* returns to *Doze* power state.

The driver of the client interface can change the power mode of the client *station*. The *station* may switch from *power save* mode to *active* mode after receiving the first data frame from the *access point*, or after sending a data frame to the *access point*. An example of transitioning from *power save* mode to *active* mode and back is shown in Figure 1. The station will switch back to *power save* mode after no frames are received or transmitted for a predetermined interval, shown as $T_{timeout}$. Switching from *active* mode to *power save* mode delays receiving any frames until after the next beacon is received.

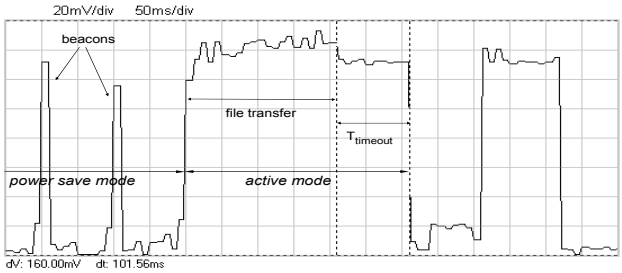

Figure 1. Power consumption of WLAN interface

Switching from *power save* mode to *active* mode to receive frames is very advantageous from a performance standpoint, because in the *active* mode the *access point* will forward data frames to the client as soon as they come in, while in the *power save* mode it must queue them up and wait for the *client* to wake up. Unfortunately, in order to absorb variations in packet delivery, the client must stay *Awake* while waiting for more data, which wastes power. Thus, from an energy standpoint, it is *never* advantageous to transition into the *active* mode except if it is known, or highly expected, that data will be coming in at a high rate.

Client-side only solutions are restricted by the limitations in predicting the next frame arrival time and by the limitations imposed by the 802.11 specifications. This work overcomes these limitations by using a proxy to schedule incoming traffic for the WLAN in a manner that accounts for client-side configuration.

3. Architecture

The *PAWP* architecture is designed to capture WLAN web traffic and to schedule it into alternating bursts of high and no activity. The proxy buffers the downloaded content until there is enough data to justify the overhead of switching the client WLAN interface to *active* mode or until no additional data is expected. Once a data transfer is initiated, all the buffered data is forwarded at the maximum speed allowed by the WLAN conditions. The proxy does not change the forwarded content. In contrast to web caching proxies, this architecture *discards* the forwarded objects immediately. For improved performance, *PAWP* should be integrated with the firewall protecting the WLAN or with the optional caching proxy (see Figure 2).

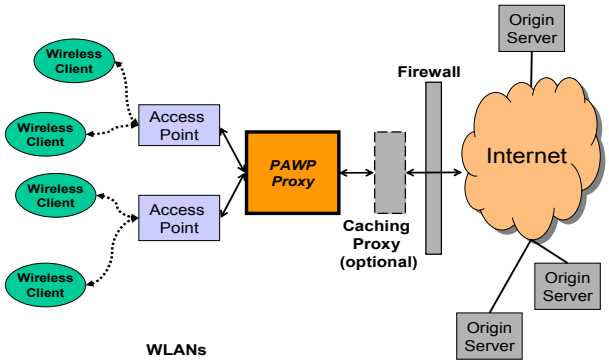

Figure 2. Typical usage setting for *PAWP*

The *PAWP* architecture has four major components, which are shown in Figure 3: the client-side module, the server-side module, the decision module, and the global state module (the blackboard). The client-side module processes HTTP requests from the WLAN clients. If the requested object was already prefetched (with the correct cookie), then the client-side module builds the response immediately and it requests permission to send the object back to the client. Otherwise, the request is added to the global data structures in the blackboard module. The server side-module handles remote servers: establishes and manages TCP connections, constructs HTTP requests, and adds HTTP responses to the blackboard. In addition, this module parses text/html responses that are not compressed, generates prefetch requests for all the embedded objects, and adds them to the blackboard. Every time the client- or server-side modules change the state of the blackboard, the decision module is activated. The decision module determines when a client request is forwarded to the server, when a response can be returned to a client, when to reuse a TCP connection, etc. The decision module acts as the proxy's *oracle* and its behavior is controlled by an extensible set of rules.

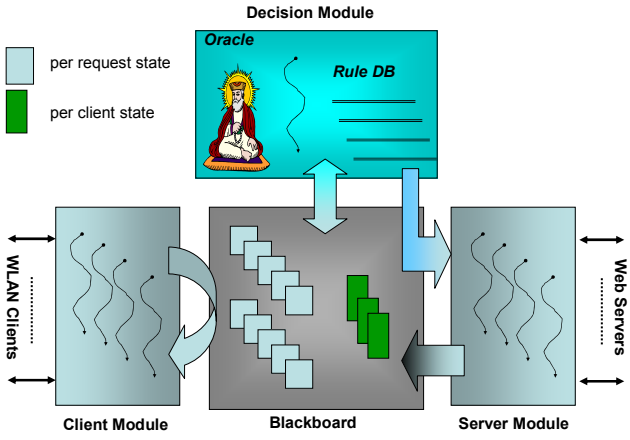

Figure 3. *PAWP* architecture

In network configurations where the WLAN device is connected directly to the Internet, TCP packets arrive at the client in an unpredictable pattern due to the transmission delays between client and web servers and their impact on the TCP packet flow. The primary objective of the *PAWP* architecture is to introduce a certain degree of predictability in the incoming HTTP traffic that the client device can take advantage of. Furthermore, as the client device initiates additional requests upon receiving data, the scheduling of incoming traffic impacts the outgoing traffic as well. *PAWP* schedules the WLAN traffic by turning TCP transfers between client and proxy on and off, using an extensive collection of HTTP-dependent rules. The proxy effectively indicates to the WLAN client that in most situations, a short time interval, such as ten milliseconds, of no incoming traffic signals a longer interval of network inactivity. As a result, the client device can become more energy efficient by switching to *power save* mode after much shorter delays than previously possible.

The set of rules determining when data should be released to the client is complex and expected to evolve. These rules take into account client configuration and the status of the WLAN. They are expected to evolve with the HTTP-related standards and with our understanding of the complex interactions between 802.11, TCP and HTTP. Our current rules are described in the next section.

To create traffic bursts, the proxy has to delay forwarding of downloaded content, which increases user-perceived latency. *PAWP* uses several techniques to compensate for these increases. The proxy parses downloaded HTML documents and aggressively prefetches all the embedded objects in the page. As a result of prefetching, many of the subsequent client requests are served immediately, without incurring the delay of accessing the origin server. In addition, *PAWP* benefits from splitting the TCP connections between the WLAN client and servers.

The *PAWP* architecture has to address several challenges. First, the proxy must correctly handle HTTP cookies. All client cookies are forwarded and 'Set-Cookie' operations are recorded locally, for later use. Objects prefetched without the proper cookie information are discarded. To lower the likelihood of incorrect prefetches, the architecture includes a mechanism for downloading client cookies into the proxy and for sharing cookies between related proxy installations. Second, the proxy has to be efficient in prefetching the embedded objects. For instance, *PAWP* attempts to prefetch objects in the order they are expected to be requested by client. In addition, when the client device caches web objects, a large fraction of the client requests are "conditional GETs" and the proxy uses prefetched objects to handle these requests correctly. Because of client caches, some prefetched objects are never requested and they are discarded after a several tens of seconds. Typical *PAWP* configurations aim at avoiding any increases in user-perceived latencies while scheduling traffic for the maximum power savings in the WLAN interface. The next section describes the current proxy prototype.

4. Implementation

In this section, we describe the current *PAWP* implementation. Our prototype runs on a separate server, i.e., the implementation is not integrated with any of the network elements shown in Figure 2. The implementation is heavily multithreaded and it uses some open source code, mainly from the GNU wget project [9].

The client-side module consists of one thread for each client connection. Upon receiving a valid request, a client thread searches the blackboard for the requested object. If the object is found, the client thread constructs the response and attempts to send it back to the client. No data is sent to the client without permission from the decision module. If the object is not found but there is a pending prefetch for it with the same cookies as in the client request, if any, the thread blocks waiting for the prefetch to complete. If neither the object nor a pending prefetch request for the object is found, the client request is added to the blackboard and the thread blocks. Note that only objects prefetched with the same cookies as in the client request, if any, are considered valid. The proxy keeps client connections open unless HTTP semantics require their closing.

The server-side module consists of one thread for each active request on the blackboard. A request becomes active after it is associated a server connection by the decision module's *oracle*. First, the *oracle* attempts to reuse an existing TCP connection, if one is available, or create a new one if allowed. The proxy is configured to open no more connections than the client browser. There can be more than one pending request for each server connection when *request pipelining* is enabled.

Cookie-related information found in the HTTP response headers is stored locally. Cookies are added to a prefetch request before it is sent to the server, if present in the local store. The proxy uses the cookies in the client request for forwarded requests.

When *pipeline response* is enabled, the decision module is informed about a new response as the object is downloaded, to pipeline it to the client. Otherwise, the decision module is signaled after the entire object is downloaded. For prefetched objects without a pending client request, the decision module is always signaled after the download completes.

When *prefetching embedded objects* is enabled, responses containing uncompressed text/html documents are parsed and for each embedded object, a prefetch request is added to the blackboard. Parsing is performed as the document is downloaded, since main pages are typically large (several tens of KBytes) and they are received slowly from remote web servers.

The decision module consists of a single thread, called *oracle*, which controls the actions of the client and server modules. For instance, the *oracle* controls the maximum number of TCP connections that the proxy can open to each server and to all web servers. The *oracle*'s decisions are based on the information stored on the blackboard by the two modules. When *traffic scheduling* is enabled, the *oracle* uses request and response descriptors, the timestamp of the last request received from, and the timestamp of the last response sent to each client to decide when to release data to the client. Otherwise, data is forwarded to the client immediately.

Figure 4 describes the main rules used for shaping the traffic. First, data is released to the client if it is available before the WLAN NIC switches to *power save* mode, which is computed as the moment the last client request was received plus $T_{timeout}$. Second, no object is delayed for more than a maximum amount of time, called *MaxDelay* in Figure 4. Third, whenever more than *MinObjects* are ready to be sent, they are forwarded to the client; *MinObjects* is always smaller or equal to the maximum number of outstanding requests from the client device. Finally, if enough data is buffered to justify the overhead of switching the WLAN to *active* mode, data is forwarded even when the other conditions are not satisfied.

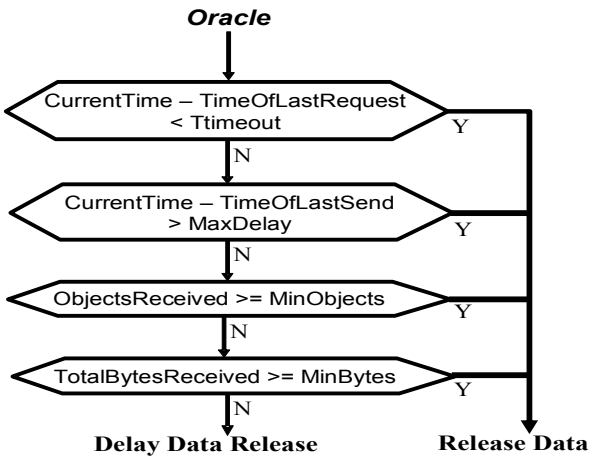

Figure 4. Rules for releasing data to the client

When threads are assigned different *priorities*, all client threads have the same priority, which is higher than the priority of the *oracle* thread; all server threads have the same priority as well but their priority is lower than the *oracle*'s priority. The priority assignment was designed to support the traffic shaping rules.

5. Experimental results

This section presents the results of our experiments with using *PAWP*. First, we describe the experimental testbed and the tools used for measurements. Second, we describe several experiments that exercise different features of the proxy. Third, we present the results of using the proxy to access several popular web sites. Last, we compare results from experiments with the same sites using different browsers.

5.1. Experimental testbed

In all experiments, the client device is an IBM ThinkPad T20, with a 700 MHz Pentium III CPU and 512 MB of memory, running Windows XP Professional. We selected this client device because its capabilities are in between those of a PDA featuring a 400 MHz Intel XScale processor and those of an ultralight notebook featuring 1000+ MHz Intel or Transmeta processor.

The client browsers used in these experiments are IE 6.0.2600 and Mozilla 1.4. Both are configured to use HTTP 1.1 in both proxy and no-proxy configurations. In addition, we enabled request pipelining for Mozilla. For better repeatability, the browser is started with an empty cache in all the experiments presented in this section. This is similar to the methodology used in [12].

The proxy is hosted by a dual-processor 933 MHz Pentium III with 512 MB of memory running RedHat Enterprise Linux AS rel. 3. The proxy connects directly to the Internet through the corporate firewall, i.e., the caching proxy in Figure 2 does not exist in our testbed.

Client and proxy hosts use the same two DNS servers. In contrast to Windows XP, the default RedHat Linux configuration does not cache DNS entries. To provide the same advantage to *PAWP*, we added a caching-only DNS server to the proxy machine.

The WLAN NIC is an Intersil PRISM3 PCMCIA card and it was selected because of the versatility and programmability of its 'power'-related capabilities. The PRISM 3 interface consumes 848 mW in the *Awake* state and 25 mW in the *Doze* state. This interface switches to *active* mode when it detects pending frames in the *access point*. The WLAN access point is an Intel PRO/Wireless 2011B and it is on the same FastEthernet LAN as the proxy. The latency between proxy and the *access point* is less than 1 ms.

The client device uses two configurations for the WLAN interface. In both configurations, the interface listens to every beacon sent by the *access point*, i.e., every 102.4 msecs. In the first configuration, typical for this interface, the driver switches the interface to *power save* mode after 100 ms of inactivity. This configuration is used only in the experiments when the device is connected directly to the Internet. In the second configuration, the driver switches the interface to *power save* mode after only 10 ms of inactivity ($T_{timeout}$). This timeout value is used in experiments with and without proxy.

The proxy releases data immediately in the first 10 ms ($T_{timeout}$) after receiving a request and does not delay any object for more than 500 ms (*MaxDelay*). As typical retrieval latencies are higher than 10 ms, the proxy releases data in the first 10 ms after receiving a request only if the requested object or a fraction of it has already been received as a result of an earlier prefetch; in addition,

immediately after receiving a request, the proxy can release (part of) the response to a previous request from the same client. The proxy handles object fragments only when configured to pipeline responses. In addition, the proxy releases data if two (*MinObjects*) or more objects are waiting to be sent to the client, or if the cumulative size of the waiting objects or object fragments exceeds 4KB (*MinBytes*).

For each experiment, we collect HTTP protocol traces on the client device using PageDetailer [16] and power measurements using the experimental testbed shown in Figure 6. To verify measurement accuracy, we correlate the two sets of measurements.

PageDetailer displays information on each web page that has been opened since it was started. This information includes the amount of time it took to open the page, the total size of the page, the number of items comprising the page, and detailed information on each of these items. For each of them, PageDetailer lists the type (e.g., text, picture, java script, etc.), the amount of time it took to retrieve and display the item, the size of the item and the HTTP headers of the request and response message. Most important for our work, PageDetailer displays the download time of each item as a horizontal bar, scaled and proportional to the time it has taken to load the complete page, working from left to right. The horizontal bar is divided into separate activities, which are displayed in different colors: yellow for the connection setup time, blue for the response time, i.e., the time between when the HTTP request is sent until the first segment of the response is received, and green for the time needed to receive the additional data needed to fulfill the request. In a B&W image, yellow, blue and green translate into light gray, black and dark gray, respectively.

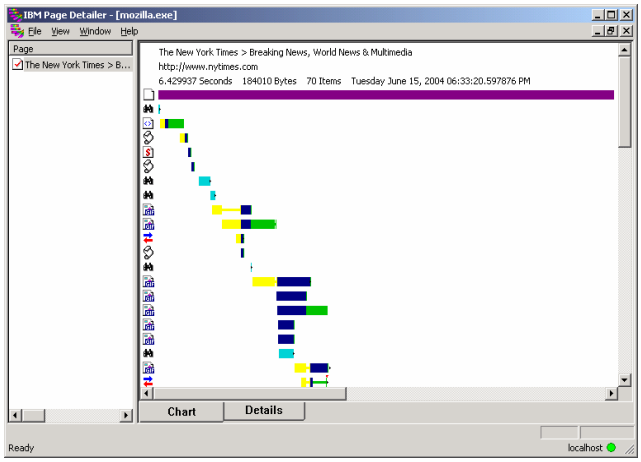

Figure 5. PageDetailer screenshot

Figure 5 shows a PageDetailer screenshot after the download of the NY Times main page. In this experiment, the client device connects directly to the Internet. The connection setup times (yellow) and the object download times (green) represent a large fraction of the total download time. In contrast, in the *PAWP* experiments, connection setup times are negligible. Similarly, object download times are small due to the high bandwidth transfers between client and proxy. When using the proxy, the download time is dominated by the response times (blue). The proxy performs traffic scheduling by controlling response and download times.

Figure 6 shows the power measurement testbed. The oscilloscope is used to sample the instantaneous power consumption of the WLAN interface. The sampled data is then sent to the data collection PC, which runs an oscilloscope application, thus enabling us to analyze the dynamic power consumption of the WLAN interface. The PC also collects data from the programmable Digital Multimeter to compute the average power consumption.

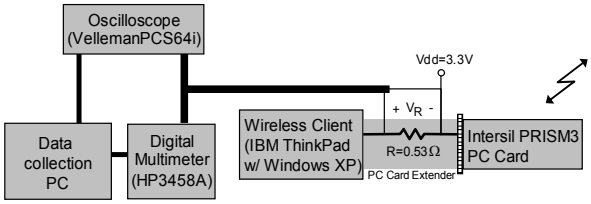

Figure 6. Testbed for power measurements

5.2. Proxy configuration

PAWP has several features that can be enabled or disabled independently: traffic shaping, prefetching of embedded objects, response pipelining, and request pipelining, and assigning different thread priorities. In these experiments, when response pipelining is enabled, the oracle is signaled for every 4KB of data received. For request pipelining, the maximum number of pending requests per connection is set to four.

Table 1. Costs and benefits of proxy features

NY Times (www.nytimes.com) 240kB/77	Download Energy [s]	Download Time [s]
Direct (no proxy)	2.70	8.75
Proxy: all features disabled	2.46	8.95
Proxy: scheduling, prefetching	2.38	8.05
Proxy: scheduling, prefetching, request & response pipelining	2.15	7.54
Proxy: all features on	1.94	6.99

The experiments presented in this section attempt to quantify the impact of some of these features on the proxy performance. The first set of experiments uses no proxy. The second set uses PAWP with all the features disabled. In the third set of experiments, traffic scheduling and prefetching of embedded objects are enabled. In the fourth set of experiments, request and response pipelining are enabled as well. In the fifth set of experiments, all the proxy features are enabled, i.e., in addition to the features

enabled for the previous set of experiments, the client, server, and *oracle* threads are assigned different priorities. For these experiments, we use the main page of the NY Times, which varies between 190kB and 270 kB and uses between 45 and 80 embedded objects. The results of the experiments are summarized in Table 1. The experiments show that each set of features contributes to the reduction of download latency and energy. However, these results are preliminary since they are based on only one page.

5.3. Proxy performance

This section describes experiments with downloading the main page of several popular sites. We selected the main pages because, in most cases, they are larger and more complex than pages for individual articles, products, or subdomains; the difference in size is due mainly to the additional embedded objects. A typical NYTimes article is between 168kB and 210 kB and uses between 28 and 56 objects. In addition, interactions with news and e-commerce sites always start with the main page. Table 2 summarizes the results of experiments with the IE browser. For each site, the total size of the page and the total number of objects, which include main page, embedded objects and pop-up ads, are given, as listed by PageDetailer. In these experiments, all the *PAWP* features analyzed in the previous section are enabled. Although our selection is biased towards more complex web pages, we believe that the wide range of page sizes and embedded objects used in these experiments make the results in Table 2 representative for a large range of web pages.

Table 2. Proxy performance with IE

Website size[kB]/ objetcs	Connection Type	Download Energy [J]	Download Time [s]	Throughput [kBytes/s]
cnn 281kB/84	Direct Proxy	2.47 2.25 (-9%)	8.13 7.33 (-10%)	34.6
nytimes 253kB/76	Direct Proxy	2.36 1.89 (-22%)	8.17 5.78 (-29%)	30.1
washingtonpost 535kB/73	Direct Proxy	6.14 2.83 (-54%)	9.08 8.58 (-6%)	56.0
americanexpress 42kB/25	Direct Proxy	2.11 0.80 (-61%)	3.42 3.05 (-11%)	12.3
chase 125kB/31	Direct Proxy	1.38 1.10 (-21%)	5.12 3.34 (-35%)	24.4
ebay 112kB/74	Direct Proxy	5.77 2.62 (-55%)	10.80 10.12 (-6%)	10.4
citibank 135kB/51	Direct Proxy	4.19 2.51 (-40%)	15.18 7.53 (-50%)	8.9
amazon 91kB/51	Direct Proxy	2.69 1.35 (-50%)	5.40 5.38 (0%)	16.9
bbc 61kB/31	Direct Proxy	2.10 1.05 (-50%)	3.56 3.37 (-5%)	17.1
sigmobile 74kB/34	Direct Proxy	0.98 0.78 (-21%)	3.24 2.33 (-28%)	22.8

In the 'Proxy' experiments, the WLAN interface and *PAWP* are configured as previously described. In the 'Direct' experiments, the WLAN interface is configured to received beacons from the access point every 102.4 ms (as in the 'Proxy' experiments) but the timeout parameter ($T_{timeout}$) is increased to 100 ms, which is the typical value seen in commercial WLAN cards.

The results are computed by taking the average of seven experiments. The experiments were run in the evening, and all the experiments using the same site were run in a batch, alternating 'Direct' and 'Proxy' experiments to minimize the effect of changing loads on web servers or in the Internet. Each batch of experiments takes approximately 15 minutes to complete.

In these experiments we report the total energy consumed by the card to perform the download, as download times are different. In all the experiments, using *PAWP* reduces download energy by up to 55%. Note that in contrast to IE, *PAWP* implements request pipelining.

In addition to the two sets of downloads summarized in Table 2, we run experiments with the WLAN interface configured as in the 'Proxy' experiments, i.e., using a 10ms timeout, but with the client device configured for direct access to the Internet. In these experiments, download times were between 25% and 60% higher than in the 'Proxy' experiments while the energy reductions were negligible. This demonstrates that reducing the timeout of the WLAN interface alone, without scheduling the traffic, does not yield any practical power benefits.

Table 3. Proxy performance with Mozilla

Website size[kB]/ objects	Connection Type	Download Energy [J]	Download Time [s]	Throughput [kBytes/s]
cnn 252kB/84	Direct Proxy	3.30 1.37 (-59%)	4.63 3.88 (-16%)	54.3
nytimes 190kB/45	Direct Proxy	3.29 1.11 (-66%)	6.85 3.20 (-53%)	23.3
washingtonpost 504kB/67	Direct Proxy	4.99 2.20 (-56%)	7.34 7.01 (-5%)	44.4

To measure the benefit that request pipelining in the proxy can provide to browsers that have this feature, we run experiments for the first three sites (*CNN*, *NYTimes*, and *WashingtonPost*) using the Mozilla browser. Mozilla was configured to use request pipelining in both 'Direct' and 'Proxy' experiments. The proxy was configured to limit the number of pending requests on a connection to four, while for Mozilla we use its default configuration parameters. The results are presented in Table 3.

Experiments with both Mozilla and IE show energy savings and download time reductions. However, the results are site dependent. This is partly because the IE and Mozilla experiments were run on different dates, which explains the differences in page size and complexity.

One other important difference between IE and Mozilla is the number of connections between browser and proxy. IE opens at most two connections while

Mozilla opens up to four. This makes Mozilla more resilient to downloads of pages with embedded objects hosted on slow sites. IE performance degrades dramatically in these situations, as one or both connections exhibit the head-of-line blocking.

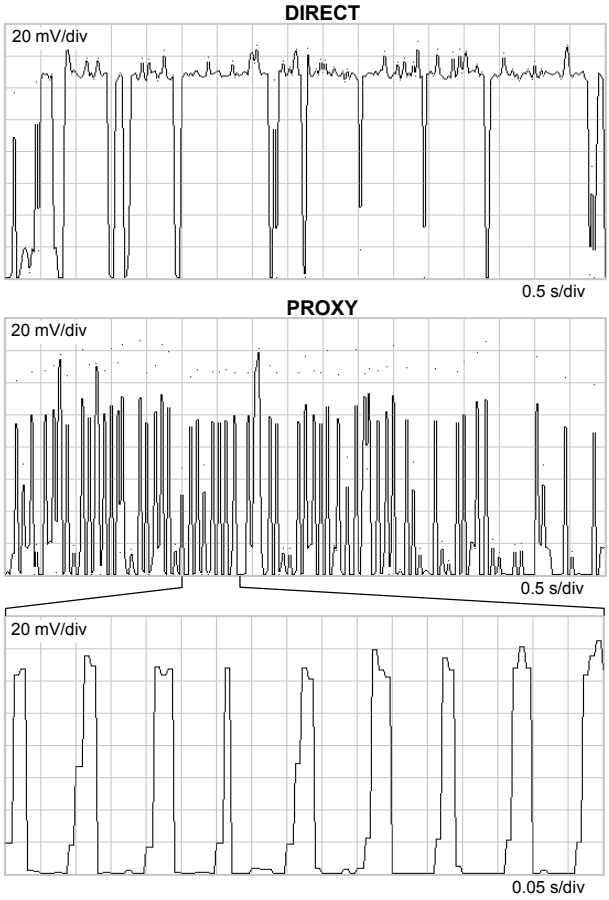

Figure 7. Dynamic power traces (eBay.com)

To explain the significant energy savings in the 'Proxy' experiments, we show in Figure 7 two dynamic power traces collected with the oscilloscope application. The traces were collected while downloading eBay.com in 'Direct' and 'Proxy' experiments, respectively. The traces represent the entire download process which lasted 8.0 secs and 6.6 secs in the 'Direct' and 'Proxy' case, respectively. The contrast in power dynamics is striking. In the Direct case the WLAN stays in the *Awake* state almost continuously due to the long timeout value of 100 msecs. This allows packet inter-arrival times of less than 100 msecs to cause the WLAN to reset its timeout timer and effectively keep the WLAN in the *Awake* state for an unnecessarily long time while waiting for more data. In the 'Proxy' case the timeout value is only 10 msecs, thus allowing the WLAN to drop out of the *Awake* state much faster while the proxy server continues to prefetch more data. Then the next time the WLAN wakes up, it rapidly depletes the buffered data in Access Point and quickly returns back to the *Doze* state. This behavior may be seen in the last figure, which shows an expanded view of a 0.85s slice of the power trace.

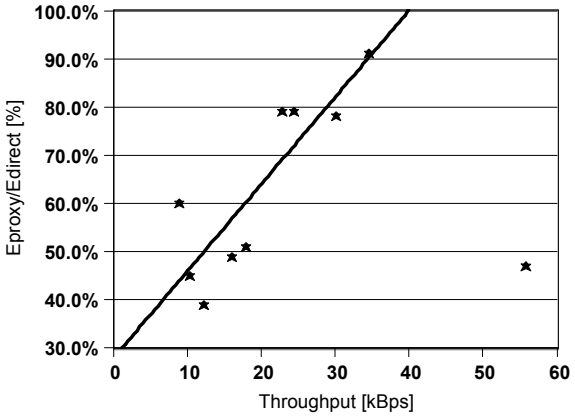

Figure 8. Energy consumption vs. throughput

We observed that for most sites, there is a correlation between energy savings and the download throughput (5th column in Table 2.) Figure 8 shows that the lower the application-level throughput is, the larger the energy savings are. We believe the correlation stems from the fact that the lower the client-server bandwidth is, the longer is the packet inter-arrival times and thus the more time the WLAN idles in the *Awake* state in the 'Direct' case, thus wasting more energy. This longer inter-arrival time doesn't affect the 'Proxy' case where the client WLAN doesn't "see" these inter-arrival times due to the *PAWP* traffic scheduling. The straight line in the figure is a linear fit to all but one of the IE data points. The exception, which is the *WashingtonPost* main page, has an average object size (7.5kB) much larger than the other sites, which vary between 3.7kB for *Chase* and 1.5kB for *eBay*. We suspect that the energy savings for pages with larger objects are less sensitive to throughput because larger objects require multiple roundtrips to download, which naturally generate longer packet inter-arrival times. We plan to investigate this hypothesis in our future work.

To summarize, our results show that a simple prototype can reduce the energy consumption of the WLAN interface by more than 50%. Over time we can expect that improvements in processor power management and display technologies will modestly increase the total fraction of power consumed by the wireless subsystem. Application aware software can leverage processor frequency scaling. Other hardware technology improvements will also reduce the power consumed by the processor. Displays based on Organic LEDS (OLEDs) are becoming available in larger sizes and are expected to be used in mobile computers including laptops. OLED displays are expected to consume less power; moreover their power consumption depends on the number of lit pixels. This property allows the designer of the user interface more flexibility in reducing the power consumed by the display [11]. The advent of longer range wireless technologies such as 802.16 may increase the power

consumed by the wireless subsystem as well. Therefore we believe that the technology described in this paper will be even more applicable in the future.

6. Related work

We believe that our work is the first to take advantage of the application level knowledge at the proxy server to reduce the energy consumption of the WLAN interface by scheduling the traffic directed to the wireless client. Two categories of work are closely related to ours: research on using proxy servers to reduce web latency, and research on reducing the energy consumption of WLAN interfaces.

Proxy servers have been developed for many purposes. Most commonly, proxies are used for web caching and as firewall components. Proxies are also used for transcoding content to better suit the capabilities of mobile client devices. The idea of pre-fetching web pages to reduce web latency was previously explored. The authors of [15] found that local proxy pre-fetching could significantly reduce web latency and that pre-fetch lead-time is an important factor in the performance of pre-fetching. A survey of 14 related studies on web pre-fetching can be found in [7]. More recently, [4, 21] propose session-level techniques for using transparent proxies to reduce browsing latencies over 3G wireless networks. Our work focuses on reducing the power consumption of WLAN interfaces by using explicit proxies to prefetch embedded objects.

Chandra et al. [5, 6] investigate an application-specific protocol for reducing the network interface power consumption for streaming media applications. Their approach is limited to streaming media applications and requires proxies at both ends of the wireless LAN, while our approach can be applied to any application that uses HTTP traffic without any client side modifications.

Many techniques that reduce the energy consumed by the WLAN interface can be found in literature. The power saving mode of IEEE 802.11 is based on the work of Stemm and Katz [24], which shows that leaving the WLAN card in sleep mode whenever possible can dramatically reduce the power consumption of the device. At the transport level, the "Bounded Slowdown Protocol" [14] introduces a power saving mode that dynamically adapts to network activity and guarantees that a connection's round trip time does not increase by more than a preset factor. At the MAC level, Qiao et al. [20] propose to combine Transmit Power Control and PHY rate adaptation to pre-compute an optimal rate-power combination table for a wireless station. Gundlach et al. [8] describe a transport-level scheduling policy designed to burst packets to clients. This approach is similar to ours to the extent that it also enables periodical releasing of data. However, as our approach employs HTTP-level information, it is better able to optimize data delivery to the client. Our approach is capable of handling more complex situations, such as web pages with a large number of embedded objects while theirs cannot.

At the system level, Shih et al. [23] introduce a technique to reduce *idle power*, which is the power that a wireless LAN-enabled PDA phone consumes in "standby" mode. Their approach is to shutdown the device and its wireless card when the device is not being used. A secondary, lower-power wakeup mechanism is used to wakeup the device only when an incoming call is received. Simunic et al. [22] describe system-level power management strategies that turn the network interface off completely during idle periods to reduce its power consumption. At the application level, Barr and Asanovic [2] explore the energy efficiency of different compression and decompression algorithms and show overall energy reductions when an energy-aware data compression strategy is applied. The STPM algorithm proposed in [1] adaptively manages the power consumption of the WLAN card using knowledge from application, network interface, and mobile platform. For real-time applications, Poellabauer and Schwan [19] integrate the power management of the WLAN interface and processor with application-level knowledge to increase idle periods and decrease the number of switches between power modes.

The work presented in [18] employs an idea similar to ours to manage hard disk power consumption by suggesting the use of aggressive pre-fetching and the postponement of non-urgent requests in order to increase the average length of disk idle phases.

7. Conclusions and future work

In this paper we first described why existing approaches and work on 802.11 power management do not sufficiently address power management when network activity is present. We then presented *PAWP*, a web proxy that schedules network traffic so that the wireless interface can be turned off for longer periods of time while the proxy is prefetching and buffering data on behalf of the wireless client. The proxy was implemented and a test bench for making power measurements was created. Our implementation and experiments validate the concept of a web proxy for power management. Results on popular web pages showed reductions of more than 50% in energy consumed by the WLAN interface.

Our experiments show that a simple approach, which switches the WLAN interface to *power save* mode after shorter timeouts, without proxy support, does not yield any practical benefits. Implementing a power management proxy at the HTTP level rather than the TCP level allowed us to exploit traffic information that is available only in the application layer. Given the trend of growing number of applications and middleware based on HTTP, we believe that application-dependent HTTP proxies for power management will be an attractive option for lowering the energy consumed by the WLAN interface.

To our surprise, we also achieve noticeable reductions in download latency with *PAWP*. In addition to improving the user experience, this may enable further energy savings in other subsystems, such as processor and display. One can expect the reductions in download latency to diminish as the internet backbone becomes faster and browser technology improves. However, we expect the benefits of improved system predictability to continue to enable better energy management.

We were also surprised by the impact of browser technology and web site design on *PAWP* efficiency. We believe that our work on *PAWP* has identified areas for improvement in browser and web page design. For instance, pages should be designed to facilitate object prefetching and web servers should support request pipelining efficiently, especially for sites targeted towards mobile clients. Furthermore, the more connections the browser opens to the proxy, the more resilient the transfers between the two are.

We plan to interleave disparate http applications and study the impact of simultaneous applications on the client. In addition, we plan to experiment with multiple WLAN clients. As described in Section 2, a station waits a short random delay before it sends its first poll message to the base station when frames for multiple stations are buffered in the base station during the previous beacon interval. As a result, multiple stations could be idling in the *Awake* state. We plan to determine how often this situation arises and search for a solution, if necessary. Over time we will study how the requirements on the *PAWP* architecture scale to typical enterprises, where multiple stations connect to a single access point.

PAWP was designed for low latency 802.11-based WLANs. Although some of its elements may provide benefits when used with other wireless technologies, such as CDMA2000, others may not. Another aspect to pursue is the benefit of the *PAWP* architecture for faster wireless networks such as 802.11g, 802.16, and ultrawide band.

8. References

1. M. Anand, E. B. Nightingale, and J. Flinn, "Self-Tuning Wireless Network Power Management," In *Proceedings of ACM MOBICOM 2002.*
2. K. Barr and K. Asanovic, "Energy Aware Lossless Data Compression," In *Proceedings MobiSys 2003*.
3. L.S. Brakmo, D.A. Wallach, M.A. Viredaz, "µSleep: A Technique for Reducing Energy Consumption in Handheld Devices," In *Proceedings of MobiSys 2004*.
4. R. Chakravorty, S. Banerjee, P. Rodriguez, J. Chesterfield, and I. Pratt, "Performance Optimizations for Wireless Wide-area Networks: Comparative Study and Experimental Evaluation," in *Proceedings of ACM Mobicom 2004.*
5. S. Chandra, "Wireless Network Interface Energy Consumption Implications of Popular Streaming Formats," In *Proceedings of MMCN* 2002.
6. S. Chandra and A. Vahdat, "Application-Specific Network Management for Energy-Aware Streaming of Popular Multimedia Formats," In *Proceedings of The 2002 USENIX Annual Technical Conference.*
7. D. Duchamp. "Prefetching Hyperlinks," In *Proceedings of USITS 1999*.
8. M. Gundlach, S. Doster, D. K. Lowenthal, S. A. Watterson, and S. Chandra, "Dynamic, Power-Aware Scheduling for Mobile Clients Using a Transparent Proxy," In *Proceedings of ICPP 2004*.
9. GNU wget, http://wget.sunsite.dk/.
10. Intel Low-Power Technologies: Bringing Longer Battery Life - and Higher Productivity - to Mobile Computing, intel.com/ebusiness/pdf/prod/related_mobile/wp021601.pdf
11. S. Iyer, L. Luo, R. Mayo and P. Ranganathan, "Energy-Adaptive Display System Designs for Future Mobile Environments", in *Proceedings of MobiSys 2003*.
12. H. Jamjoon and K. Shin, "Persistent Dropping: An Efficient Control of Traffic Aggregates", In *Proceedings of ACM SIGCOMM 2003*.
13. N. Kamijoh, T. Inoue, C. M. Olsen, M. Raghunath, C. Narayanaswami, "*Energy Trade-offs in the IBM Wristwatch Computer*," In *Proceedings of The 5th IEEE International Symposium on Wearable Computers (ISWC'01)*.
14. R. Krashinsky and H. Balakrishnan, "Minimizing Energy for Wireless Web Access with Bounded Slowdown," In *Proceedings of ACM MOBICOM 2002*.
15. T. M. Kroeger, D. D. E. Long and J. C. Mogul, "Exploring the Bounds of Web Latency Reduction from Caching and Prefetching," In *Proceedings of USITS 1997*.
16. Page Detailer, www.alphaworks.ibm.com/tech/pagedetailer
17. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, ANSI/IEEE Std 802.11, 1999.
18. A. E. Papathanasiou and M. L. Scott, "Energy Efficiency through Burstiness," In *Proceedings of IEEE WMSCA 2003*.
19. C. Poellabauer and K. Schwan, "Energy-Aware Traffic Shaping for Wireless Real-Time Applications," In *Proceedings of IEEE RTAS 2004*.
20. D. Qiao, S. Choi, A. Jain, and K. G. Shin, "MiSer: An Optimal Low-Energy Transmission Strategy for IEEE 802.11a/h," In *Proceedings of ACM MOBICOM 2003*.
21. P. Rodriguez, S. Mukherjee, and S. Rangarajan, "Session Level Techniques for Improving Web Browsing Performance on Wireless Links," In *Proceedings of WWW 2004*.
22. T. Simunic, L.Benini, P. Glynn, and G. De Micheli, "Dynamic Power Management for Portable Systems," In *Proceedings of ACM MOBICOM 2000*.
23. E. Shih, P. Bahl, and M. J. Sinclair, "Wake on Wireless: An Event Driven Energy Saving Strategy for Battery Operated Devices," In *Proceedings of ACM MOBICOM 2002*.
24. M. Stemm and R. Katz. Measuring & Reducing Energy Consumption of Network Interfaces in Handheld Devices. In *IEICE Transactions on Fundamentals of Electronics, Communications, and Computer Science, August 1997.*
25. Z. L. Xie, Z. G. Meng, K. W. Ng, B. Z. Tang, Man Wong and H. S. Kwok, "Bistable Twisted Nematic Liquid Crystals Display with Permanent Grayscales and Fast Switching", In *Society for Information Display Digest 2001*.