RC23119 (W0311-072) November 11, 2003
Computer Science

IBM Research Report

HTTP Proxy for Low Power Operation of Wireless Clients

Marcel C. Rosu, C. Michael Olsen, Chandrasekhar Narayanaswami
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Annie Luo
Carnegie Mellon University
5000 Forbes Avenue
Pittsburg, PA 15213

HTTP Proxy for Low-Power Operation of Wireless Clients

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

Annie Luo
Carnegie Mellon University
Pittsburgh, PA 15213-3891
luluo@cs.cmu.edu

Wireless LAN (WLAN) interfaces are responsible for a significant fraction of the total energy consumed by a large class of mobile client devices. WLAN interfaces have power saving features which enable substantial energy reductions during long idle intervals. However, during periods of activity, when data transfers occur, the nature of the network traffic is hard to predict and this prevents the network interface from fully utilizing the power-save features between packet receptions.

This paper introduces a web proxy architecture designed to enable further reductions in the energy consumed by WLAN interfaces. The proxy modulates the traffic directed to the wireless client into alternating intervals of high and no communication. The modulation takes into account the semantics of the web traffic and the configuration of the client's WLAN interface. As a result, the interface spends more time in power-save mode and less time in active mode. Initial experiments show energy reductions of 13% and higher on popular web sites such as CNN *and* NY Times*, and up to 56% when downloading large objects. The architecture does not require any modifications of the client or server applications.*

KEYWORDS: Power Management, WLAN, Proxies, Traffic Shaping

1. Introduction

In 802.11 LANs, the wireless interface of the mobile device is responsible for a significant fraction of its power consumption. The relative power consumed by 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. The actual amount of energy consumed for wireless communication depends on the client applications and their usage pattern. In particular, active web browsing and multimedia streaming are characterized by high energy consumption in the WLAN interface.

The WLAN interface has one or more low-power states and the client device can be configured to take advantage of these features. In these states, the WLAN interface may consume 5 to 50 times less power than when active, according to measurements we have made on several WLAN cards. To reduce power consumption, WLAN interfaces are configured to switch to power-save mode during idle intervals, for example when there has been no network activity for a second or more. In this mode, the interface spends most of its time in one of the low-power states and periodically, it powers-up briefly to receive beacon frames. Upon detecting network activity, i.e., packets to be sent or received, the WLAN interface leaves the power-save mode. The WLAN driver on the client device controls when the device enters and leaves the power-save mode.. Unfortunately, the hard-to-predict nature of incoming traffic prevents using the low-power states on the WLAN interface during active periods without affecting application performance and interactivity.

The optimal solution, which minimizes the energy used for wireless communication, requires perfect knowledge of the application traffic patterns, wired and wireless network conditions and user preferences. Furthermore, such a solution would be application-dependent.

This paper proposes a practical but suboptimal solution to the above problem. Namely, we propose an HTTP proxy architecture, which takes advantage of the information available at the application level, to modulate 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. Furthermore, the proxy attempts to mitigate any negative effects the traffic modulation might have on user-perceived latencies by parsing downloaded documents and optimistically prefetching any embedded objects. Since the prefetching is done by the proxy across the wired LAN, the wireless device does not pay any penalty for this.

The power consumed by the client WLAN interface is reduced because the interface is configured to switch to power-save mode after a much shorter delay to take advantage of the idle intervals inserted by the proxy. When client and proxy communicate, data transfer rates are higher because the latencies and packet losses between the two are lower than in the no-proxy configurations. Overall, 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 power-off for a larger fraction of the download time.

Our solution has applicability beyond web browsing because many client applications, such as media players or email clients, and Web Services use HTTP for communication to bypass firewalls. Finally, the architecture does not require any modifications of the applications on the client device or remote server.

In this paper we describe the architecture of the HTTP proxy and its current implementation. We present experiments with popular web pages, such as *CNN* and *NY Times*, which are composed of tens of embedded objects, and with downloading large objects (upwards of 1MB). We evaluate our solution using client HTTP traces collected using IBM’s PageDetailer and measurements of the power consumed by the WLAN card using an oscilloscope, a Digital Multimeter and a PC application. Energy reductions vary between 13% and 56% and depend heavily on the structure of the downloaded page and on the state of the network.

This paper describes an implementation of the HTTP proxy that is closer to a proof-of-concept than to a product, as we focused our implementation efforts only on the key elements of the architecture. In this paper, we only exercise the proxy with relatively few web sites, albeit we selected some of the most complex ones among the popular web destinations. In spite of all these limitations, our experiments demonstrate clearly that techniques previously applied only to long WLAN transfers, mostly multimedia streams, can be applied to more complex scenarios, such as downloading web pages consisting of 80+ objects, many of which are very small. To achieve these results, a novel architecture had to be defined, which takes advantage of information available only at the application level, i.e., at the HTTP protocol. Furthermore, it is our belief that extending the power-management component of the 802.11 specification could improve the efficacy of a proxy targeting the reduction of the energy consumed by the WLAN interface on client devices. Namely, our architecture can exploit any information on the current power-management status of the WLAN interfaces, provided that such information can be extracted from the WLAN base station in a timely manner.

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 several 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 LANs using *only* the distributed coordination function as access method are described. For a complete description of power management in 802.11 networks, including *ad hoc* configurations and configurations using the optional point coordination function as access method, see [Part11]. In this section and the rest of the paper, we use the same terminology as [Part11].

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, and *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* in the *power save* mode, as described next.

The *access point* buffers all traffic pending for the *stations* known to be in *power save* mode and identifies these *station*s in the appropriate beacon frames. When a *station* detects that frames are pending in the *access point,* 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, or a beacon which indicates that there is no pending traffic for the *station*[1].

The *access point*'s response to the poll is either an ACK or the next pending frame. By responding with an ACK frame, the *access point* delays the transmission of the pending frame and assumes the responsibility for initiating the delivery of the pending frame. The *station* must send an ACK for every received frame. If the *More Data* field of the frame indicates additional pending frames, the *station* may send another poll frame. If there are no pending frames, the interface returns to *Doze* power state.

The driver of the client interface controls the power mode of the client *station*. The *station* may switch from *power save mod*e 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 this transition from *power-save mode* to *active mode* is shown in Figure 1. Once in the *active mode*, the station will switch back to *power save* mode after no frames are received or transmitted for a predetermined interval, shown as $T_{timeout}$ in Figure 1. Switching from *active mode* to *power save* mode transitions the *station* to *Doze* state (also shown in Figure 1) but delays receiving any incoming frames until after the next *access point* beacon is received.

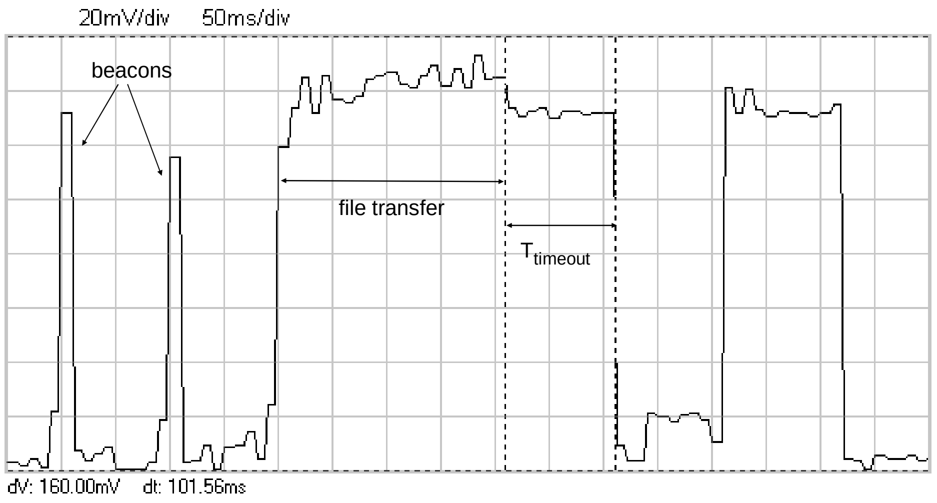


Figure 1. Dynamic Power Consumption in a WLAN Interface

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 energy. 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 very high rate so that the client will only spend a relatively little time waiting for more data.

Client-side solutions are restricted by the limitations in predicting the next frame arrival time and by the limitations imposed by the 802.11 specifications, namely by the fact that both the *More Data* field and the *access point* beacon field indicating pending frames are one bit long. This work overcomes these

[1] The access point deletes frames buffered for excessive periods.

limitations by using a proxy to modulate incoming traffic for the WLAN in a manner that accounts for client-side configuration.

3. Architecture

The proposed proxy architecture is designed to capture all the WLAN HTTP traffic and to shape 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 power modes on the client WLAN interface or until no additional data is expected. Once started, all the buffered data is forwarded at the maximum speed allowed by the WLAN conditions. In many situations, the client device initiates additional requests as a result of receiving data; therefore the shaping of incoming traffic extends to the outgoing traffic as well. The traffic modulation does not change the semantics of the content. In contrast to typical web proxies, which cache frequently accessed objects, this architecture discards the forwarded objects immediately.

The primary objective of this architecture is to introduce a certain degree of predictability in the HTTP traffic from the proxy to the client that the client device can take advantage of. 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 large transmission delays between the client and the servers and their impact on the TCP packet flow. In contrast, the HTTP proxy shapes 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 few tens of milliseconds of no incoming traffic signal a longer interval of network inactivity. Ideally, the HTTP proxy should be integrated with the WLAN Access Point (see Figure 2). As a result the client device can become more energy efficient by switching to power-save mode after much shorter delays than previously possible.

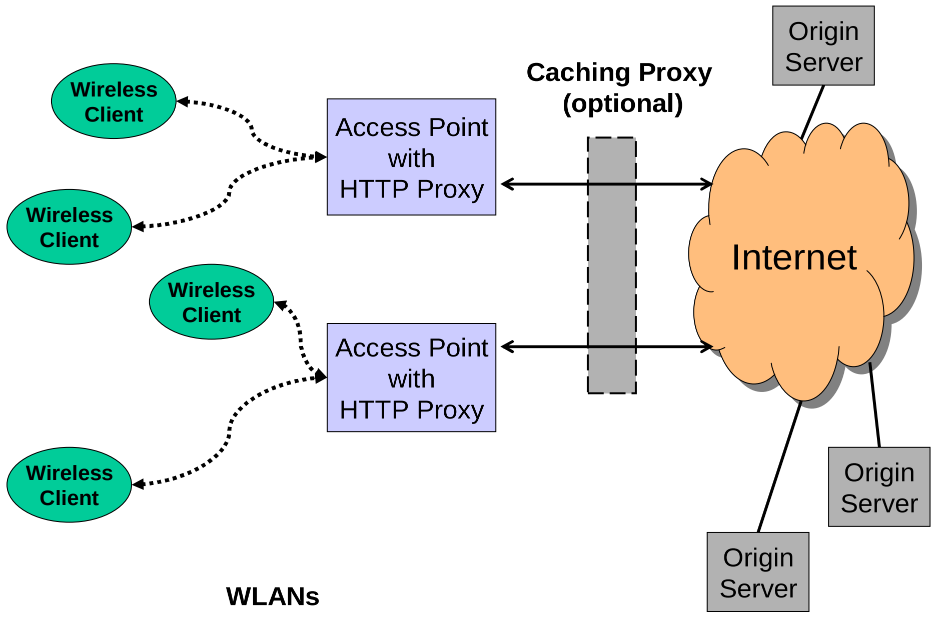


Figure 2. Typical Usage Setting for the HTTP Proxy

The HTTP proxy uses several techniques to offset the negative impact that traffic modulation can have on user-perceived latencies. Namely, in order to create traffic bursts, the HTTP proxy has to delay forwarding of downloaded content, which increases user-perceived latency. To partially offset this effect, the proxy parses downloaded HTML documents and aggressively prefetches all the embedded objects. As a result, many of the subsequent client requests are served immediately, without incurring the delay of accessing the origin server. In addition to prefetching, this proxy architecture benefits from splitting the TCP connections between the WLAN client and servers. Object download times, measured as the time interval between the arrivals of the first and last data packets, are shorter as TCP transfer rates on WLAN are higher than in the original WLAN+WAN configuration. TCP rates are higher because the latencies and the loss rates between client and proxy are lower than between client and the remote web servers.

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. As a result, we carefully isolated the decision component from the rest of the proxy in both the general architecture and the current implementation (see Figure 3).

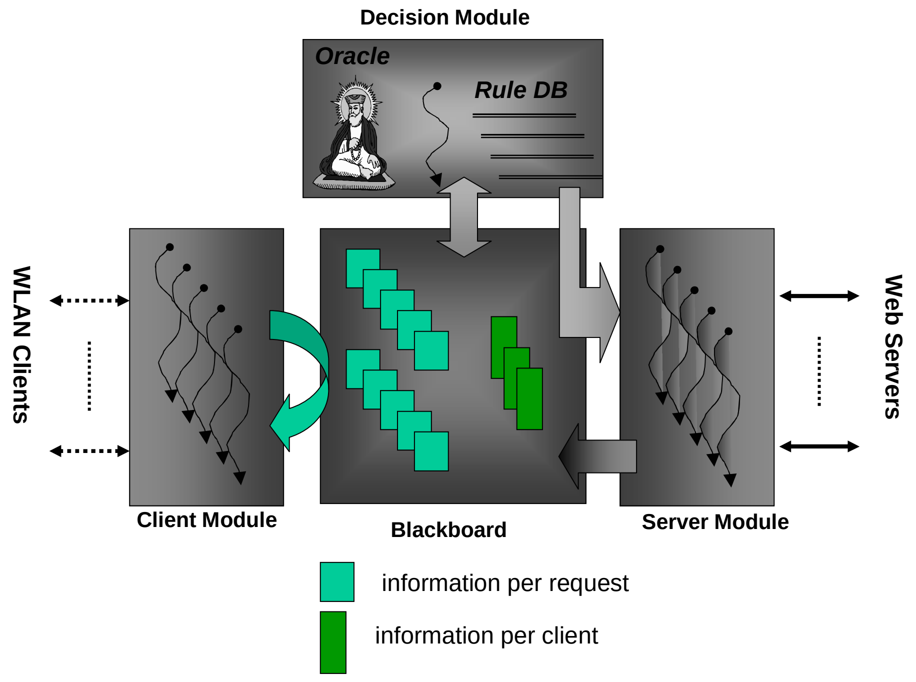


Figure 3. HTTP Proxy Architecture

The resulting architecture has four major components: the client-side module, the server-side module, the decision module (the oracle and its rule database) 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), 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 the TCP connections, constructs HTTP requests, and adds HTTP responses to the blackboard. In addition, this module parses the responses that are html documents, 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. To perform its tasks, the decision module uses an extensible collection of rules. Effectively, this module controls the shaping of the WLAN traffic by determining the conditions user which the client module can return a response to the client device.

The HTTP proxy has to address several problems. 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 should include a mechanism for downloading client cookies into the proxy and for sharing cookies between related proxy installations. Second, the proxy must be efficient in prefetching the embedded objects. For instance, it should attempt 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 should use prefetched objects to handle them correctly. Because of client caches, some prefetched objects are never requested and they should be discarded after the page download completes. Third, the proxy must shape SSL traffic and multimedia streams. Fourth, the proxy must balance traffic shaping granularity and CPU overheads, by invoking the decision module upon receiving new data from remote servers, upon completion of an object download, periodically, or a combination of these. Typical configurations parameters aim at avoiding any increases in user-perceived latencies while shaping traffic for the maximum power savings in the WLAN interface. The next section describes the current implementation of the HTTP proxy.

4. Implementation

In this section, we describe the details of the current implementation of the proposed HTTP proxy. We focused our implementation effort on the traffic shaping capabilities of the proxy and on methods for lowering download latencies. For simplicity, this prototype requires the proxy to work in non-transparent mode, thus the client browser has to be appropriately configured. Also, the prototype does not support the downloading of user cookies into the proxy. Instead, the proxy builds a copy of the user’s cookie collection from scratch using the information in the HTTP header fields. In addition, we made extensive use of freely-available code, mainly from the GNU *wget* project [GNU-wget].

The HTTP proxy can be either integrated with the WLAN access point, as shown in Figure 2, or running on a separate server, preferably on the same wired LAN with the WLAN access point(s). Integrating the proxy with the WLAN access point eliminates communication latencies between them but it can be expensive, as it requires upgrading the access point hardware with faster CPU(s) and more memory. The prototype described in this paper runs on a separate server, which is on the same wired LAN with the access point.

The implementation is heavily multithreaded, and it is built on top of LinuxThreads, which is the standard POSIX thread library in RedHat Linux 9.0. As the implementation uses only a few of the most common *pthread* calls, it can be easily ported to other operating systems and POSIX-compliant thread libraries. We carefully avoided performing any blocking I/O operations, such as connect, read or write, while in one of the critical regions. As a result, exceptional network events with large timeouts affect only the performance of the related client request and have little or no impact on other requests.

The client and server modules (see Figure 3) are multi-threaded while the decision module is single-threaded; the blackboard is a global data structure optimized for concurrent access.

The client module handles the connections to WLAN clients, processing requests and constructing responses. Each client connection is assigned a separate thread and connections are assigned to clients using on the remote IP address. The client thread exits when the connection is closed by the client or by the proxy. The proxy closes client connections when the HTTP semantics requires it or upon receiving a malformed client request.

Upon receiving a valid request, a client thread searches the blackboard for the requested objects. If the object is not found and a prefetch request was already issued for the object, the client thread blocks waiting for the prefetch to complete. If neither the object nor a prefetch request for the object is found, the client request is added to the blackboard and the thread blocks. If the object is found or after a request for the object is completed, the client thread constructs the response and attempts to send it back to the client. No data is sent back to the client and no client connections are closed without permission from the decision module.

The server module handles connections to remote web and proxy servers. For each request placed on the blackboard a new fetching thread is created. First, the thread attempts to reuse an existing TCP connection, if any available, or create a new one, if allowed. For prefetch requests, the proxy uses HTTP 1.1; therefore, the proxy opens at most two connections to each server on behalf of each client. For forwarded requests, the proxy uses the HTTP version specified by the mobile client; there the HTTP version is checked before establishing any new TCP connections. Just before the request is sent to the server, the local copy of the user cookies is searched and the request is changed to incorporate the new cookie, if necessary. Responses are added to the blackboard and the decision module is signaled. Cookie-related information found in the HTTP response headers is used to update or extend the local copy of the user cookies. The decision module is informed about the new response only after the object download is completed. In future implementations, we plan to experiment with policies that allow pipelining of responses, which will require invoking the decision module before the download is completed.

Responses containing html documents are parsed and pre-fetch requests for all the identified embedded objects are added to the blackboard. Parsing is performed as the document is downloaded, as main pages are typically large (several tens of Kbytes) and hosted on busy servers. In contrast, many of the embedded objects have much lower download times because they are small or hosted by CDN providers, such as Akamai. As a result, some embedded objects are retrieved before the download of the main page finishes. In order to efficiently parse on-the-fly, we made extensive changes to the *wget* parser, which expects the entire html document stored in a linear byte array.

The decision module consists of a single thread, called *oracle*, which controls the actions of the client and server modules. The *oracle* decisions are based on the information stored in the blackboard by the two modules. The *oracle* uses request and response descriptors, and the timestamp of the last request received from and of the last response sent to each client; both timestamps are collected by the client module. Figure 4 shows a simplified diagram of the *oracle*: only the component controlling the client module is shown. The only restrictions imposed on the operations of the server module are related to the maximum number of TCP connections that the proxy can open to a remote server.

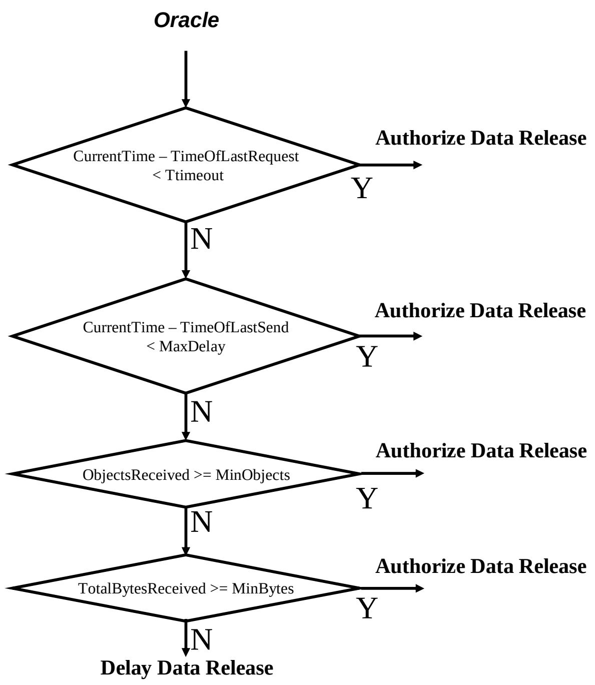

Oracle
CurrentTime – TimeOfLastRequest < Ttimeout
Authorize Data Release
Y
N
CurrentTime – TimeOfLastSend < MaxDelay
Authorize Data Release
Y
N
ObjectsReceived >= MinObjects
Authorize Data Release
Y
N
TotalBytesReceived >= MinBytes
Authorize Data Release
Y
N
Delay Data Release

Figure 4. Releasing Data to Client

As presented in Figure 4, data is released to the client if it is available before the WLAN NIC switches to power-save mode, which is computed by the proxy as the moment the last client request was received plus $T_{timeout}$. Next, no object is delayed for more than a maximum amount of time, called *MaxDelay* in Figure 4. Whenever more than *MinObjects* that were previously requested are ready to be sent, they are forwarded to the client. *MinObjects* is always smaller or equal to the maximum number of outstanding requests of the client device. Finally, if enough data is buffered to justify the overhead of switching between modes, it will be forwarded even when the other conditions are not satisfied.

The blackboard module consists of a global data structure and code to access it; both are optimized for shared access. To lower lock contention, a versioning system is used. The blackboard is organized into information per request and information per client. The request information is collected from the request and response headers. The client information consists of static information, such as IP address, and of information that changes permanently, such as number of requests outstanding, number of connections open to remote servers, cookie set, etc.

The current implementation lacks several features. Most importantly, the proxy does not pipeline requests. We expect request pipelining to be important in experiments with a large number of clients and plan to add it to future versions together with a more elaborate resource management to ensure fair allocation of proxy CPU and memory among clients. The goal of the current prototype is to demonstrate the feasibility of shaping the incoming HTTP traffic for a wireless device at a finer granularity than previously done and for a different workload and to evaluate the savings in the energy consumed by the client WLAN interface. The next section describes several experiments and the results of our measurements.

5. Experimental Results

This section presents the results of some of our experiments with the HTTP proxy. First, we describe the experimental testbed and the tools used for measurements. Next, we describe several experiments with a few popular web sites and one simple experiment downloading a large object.

The client device is an IBM ThinkPad A30, with a Pentium III CPU running at 729MHz and 256 MB of memory, running Windows XP. The WLAN NIC is an Intersil PRISM3 PCMCIA card. The HTTP proxy runs on a IBM NetVista desktop, with a Pentium 4 running at 1.8 GHz and 512 MB of memory, running RedHat Linux 9. The WLAN access point is Intel PRO/Wireless 2011B connected to the same wired LAN as the desktop running the HTTP proxy. The latency between the desktop and the *access point* is under 100 microseconds. The proxy connects directly to the Internet, i.e., the optional caching proxy in Figure 2 does not exist in our test bed.

The client device is configured to switch the WLAN interface to *power save mode* after 10 ms of inactivity ($T_{timeout}$) and to listen to every beacon sent by the *access point*, every 102.4 ms. The PRISM 3 interface consumes 848 mW in the *Awake* state and 66 mW in the *Doze* state. The HTTP proxy releases data immediately in the first 10 ms after receiving a request and does not delay any object for more than 100 ms (MaxDelay). In addition, the proxy releases data if two or more objects are waiting to be sent to the client or if the cumulative size of the waiting objects exceeds 1KB.

We collect two types of data. First, we collect HTTP protocol traces on the client device using IBM's PageDetailer [PageDetailer]. Second, we collect power measurements using the experimental setup shown in Figure 5.

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.

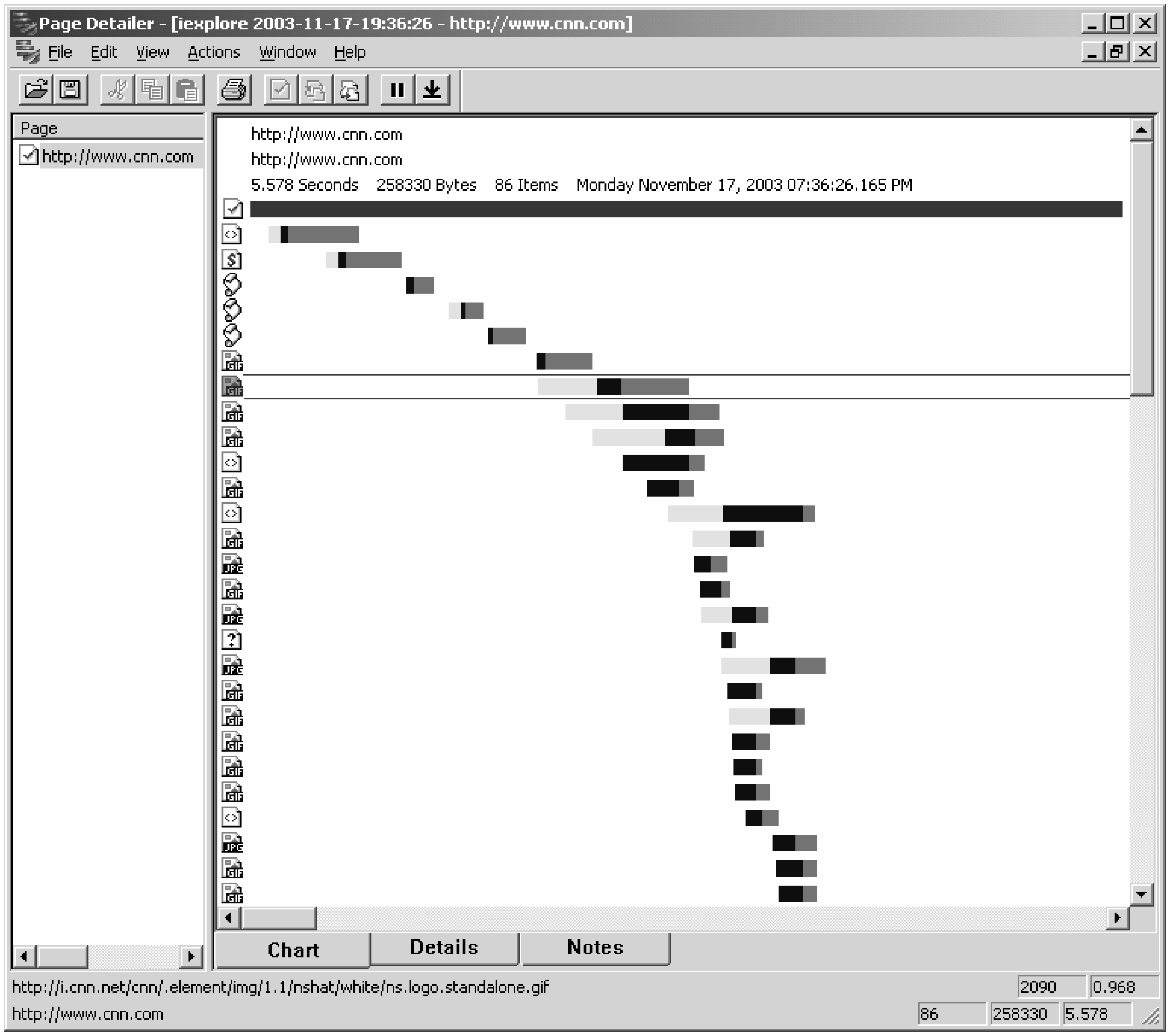
Most important for our work, PageDetailer displays the download time of each item as a horizontal bar, scaled and proportional to the time it has taken to load the complete page, working from left to right. Furthermore, the horizontal bar is divided into separate activities, which are displayed in different colors: yellow for the connection setup time, blue for the response time, i.e., the time between the HTTP request is sent until the first segment of the response is received, and green for the time needed to receive all the additional data needed to fulfill the request[2].

Figure 5 shows a PageDetailer screenshot documenting the download of the *CNN* main page, starting with an empty browser cache[3]. In this experiment, the client device connects directly to the Internet, i.e., it does

[2] PageDetailer display more information than it is described here. See the PageDetailer User Guide for additional information.
[3] In a B&W image, yellow, blue and green translate into light gray, black and dark gray.

not use the HTTP proxy. The connection setup times (yellow) and the object download times (green) represent a large fraction of the total download time.

In contrast, in the experiments with the HTTP proxy, connection setup times are negligible because of the small latency between client and proxy. Similarly, object download times are small due to the high bandwidth transfers between client and proxy. As a result, the total download time is dominated by the response times (blue). In contrast to the other two components, the response time is under the control of the proxy and increasing it is the main mechanism for shaping the WLAN traffic.


Page Detailer - [iexplore 2003-11-17-19:36:26 - http://www.cnn.com]
File
Edit
View
Actions
Window
Help
Page
Chart
Details
Notes

http://www.cnn.com

http://www.cnn.com

http://www.cnn.com

5.578 Seconds 258330 Bytes 86 Items Monday November 17, 2003 07:36:26.165 PM

http://i.cnn.net/cnn/.element/img/1.1/nshat/white/ns.logo.standalone.gif 2090 0.968

http://www.cnn.com 86 258330 5.578

Figure 5. PageDetailer Screenshot

Figure 6 shows the power measurement testbed. The oscilloscope is used to sample the instantaneous power consumption of the WLAN interface. The sampled data are then sent to the 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 for calculation of the average power consumption.

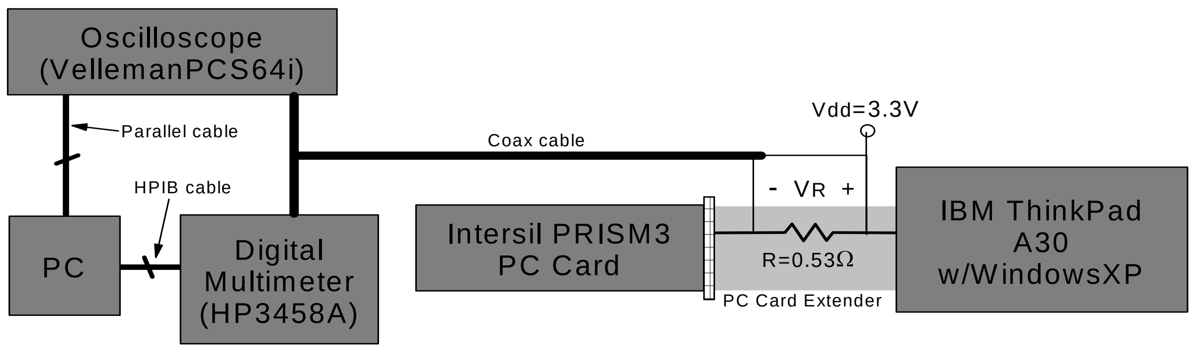


Figure 6. Testbed for Dynamic Power Measurements

Table 1 summarizes the results of experimenting with several popular web sites. For each site, the total size of the page and the total number of objects, which include main page, embedded objects and pop-up adds, are given, as listed by PageDetailer.

Website		Download Energy [J]	Download Time [s]	Application-level Throughput [kBps]
cnn.com • 234 kB • 83 objects	Direct Proxy	2.95 2.58 (-13%)	5.16 7.49	45.3 31.2
nytimes.com • 238 kB • 62 objects	Direct Proxy	1.64 1.43 (-13%)	2.23 3.24	107 73.4
atomfilms.com • 330 kB • 66 objects	Direct Proxy	4.09 2.95 (-28%)	6.04 10.3	54.6 32.0
ACPIspec_2.0b.pdf • 1.6 MB • 1 object	Direct Proxy	10.4 4.57 (-56%)	14.1 14.9	113 107

Table 1. Summary of Experimental Results

In the 'Proxy' experiments, the WLAN interface and HTTP proxy 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 10 times to 100 ms, which is the typical value seen in commercial WLAN cards. Obviously, the HTTP proxy is not used in the 'Direct' experiments.

The results are computed as the average of five experiments. The experiments were run in the evening, and all the experiments using the same site were run in batches shorter than 15 mins to minimize the effect of changing loads on web servers or in the Internet.

In addition to the two sets of downloads summarized in Table 1, we run experiments with the WLAN interface configured as in the 'Proxy' experiments but with the client device configured for direct access to the Internet. In these experiments, download times were significantly higher than in the 'Proxy' experiments while the energy reductions were negligible. This demonstrates that reducing the timeout of the WLAN interface alone, without shaping the traffic, does not yield any practical energy benefits.

Figure 7 shows two dynamic power traces collected with the oscilloscope application. The traces were collected while downloading the ACPI specification in 'Direct' and 'Proxy' experiments, respectively. We selected to include the power traces for the ACPI experiments because the contrast between the traffic shapes in the two experiments is much higher. In the 'Direct' experiment, the WLAN interface remains in the *Awake* state for a long period, as packets inter-arrival times are less than 100 ms. In the 'Proxy' experiment, the WLAN interface switches to power save mode until the proxy downloads the entire object; after the download is completed, the object is forwarded to the client across the WLAN in a very short period of time. Traces for the other three web sites show different traffic patterns but the differences are less obvious.

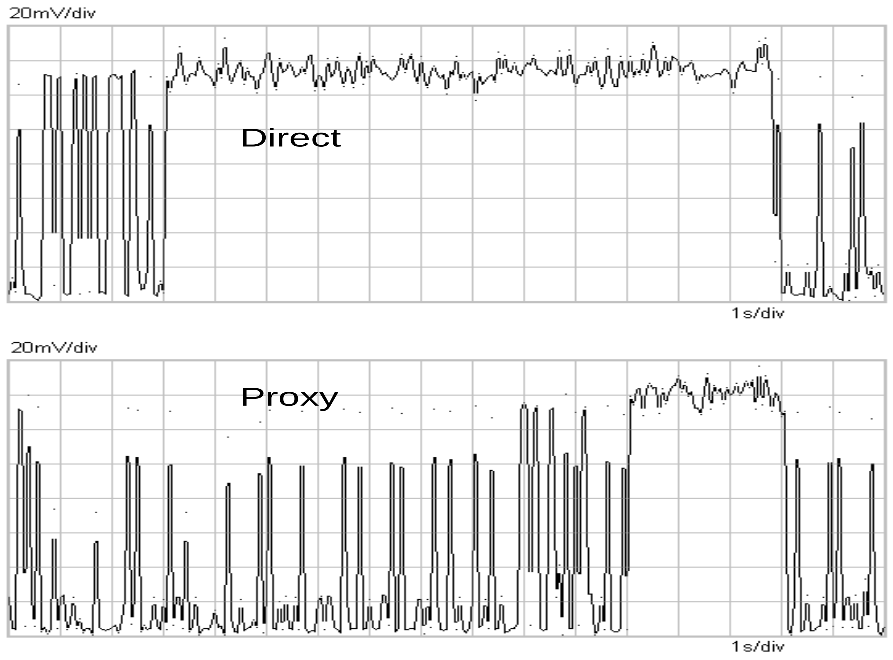


Figure 7. Dynamic Power Traces

Our results show that a simple prototype can reduce the energy consumption of the WLAN interface by 13% and higher. We are working on improving the HTTP proxy implementation to reduce its negative impact on download latencies.

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 modulating the traffic directed to the wireless client. Two categories of existing work are related to ours: research on proxy servers to reduce web latency; and research on reducing the energy consumption of WLAN interfaces.

Proxy servers have been developed for many purposes. A good survey of the types of proxy servers and how they function can be found in [O'Reilly]. The most well known types of proxy servers are the web cache proxy server and the security firewall proxy server. Proxies are also used for transcoding content to better suit the capabilities of a certain device (such as a Personal Digital Assistant). The idea of pre-fetching web pages is commonly used to reduce web latency. The authors of [KLM97] found that local proxy pre-fetching could reduce web latency by up to 41% based on locally available information, and up to 57% if server-hints are added. They also stated that pre-fetch lead-time is an important factor in the performance of pre-fetching. In [Duchamp99], it is recognized that accurate web pre-fetching is possible based on following HREFs in recently fetched pages, based on that the HREFs within a page are strongly skewed to "hot" and "cold". By having clients pass record of their references up to the relevant server, which then distributes them to all clients, one client can learn from the usage patterns of others. A survey of 14 related studies on web pre-fetching can be found in [Duchamp99]. There are, however, no proxy servers that optimize the power consumption of wireless clients.

Many techniques that reduce the energy consumed by the WLAN interface can be found in literature. The power saving mode of IEEE 802.11b is based on the work of Stemm and Katz, which shows that leaving the WLAN card in sleep mode whenever possible can dramatically reduce the power consumption of the device [SK1997]. The authors of [KB2002] present the "Bounded Slowdown protocol", a power saving mode that dynamically adapts to network activity and guarantees that a connection's round trip time (RTT) does not increase by more than a factor *p*. At the MAC level, Qiao, et. al. propose to combine IEEE 802.11 Transmit Power Control and physical layer rate adaptation and pre-establish an optimal rate-power combination table for a wireless station to determine at run time [QCJS2003]. In [GLDWC2003] (not published yet), a scheduling policy at the transport-level (UDP/TCP) is implemented in a transparent proxy between the server and the wireless access point to burst packets to clients. This approach is similar to ours to the extent that it also enables periodical releasing of data. However, our approach employs HTTP level information thus is enable to better optimize the data delivery to the client. Our approach is capable of handling more complex situations, such as web pages with a lot of embedded objects while theirs cannot.

At the system level, Shih, *et al.* introduced a technique in [SBS2002] to reduce the *idle power*, the power that a wireless LAN-enabled PDA phone consumes in a "standby" mode. Their approach is to shutdown the device and its wireless network card when the device is not being used. A secondary low-power wakeup mechanism is used to power the device only when an incoming call is received. Simunic *et al.* also describe system-level power management strategies that turn the network interface off completely during idle periods to reduce its power consumption [SBGM2000]. At the application level, Barr and Asanovic explore in [BA2003] the energy efficiency of different compression and decompression algorithms and show overall energy reduction to send compressive web data over wireless networks when energy-aware data compression strategy is applied. The STPM algorithm proposed in [ANF2003] adaptively manages wireless NIC power consumption based on knowledge from application, network interface, and mobile platform.

The work presented in [PS2003] employs similar idea as 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. Chandra, et al [Chandra2002, CV2002] investigate an application-specific protocol for reducing the network interface power consumption for streaming media applications, such as Microsoft media. They propose a server-side or local proxy and a client-side proxy are combined to transmit network packets at predictable intervals and to schedule the WNIC to enter sleep state in between. This approach is limited to streaming media applications and requires efforts at both the server/local proxy side and the client side, while our approach can be applied to any application that uses HTTP traffic without any modification at the client side.

7. Future Work and Conclusions

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 an http proxy that can modulate 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 an http proxy for power management. Results on popular web pages showed reduction in power consumption higher than 13% for the WLAN interface. Our experiments show that simply switching to power save more 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 in the application layer. Given the trend around growing number of applications and middleware based on HTTP we believe that application-dependent

HTTP proxies for power management will be attractive. Over time, such proxies may even be incorporated into the wireless access points.

At present our reduction in power consumption comes with a penalty in perceived user latency. Some things we plan to pursue in the future to mitigate this effect by including the request pipelining. One can expect the internet backbone to become faster and that latencies will reduce in general. When that happens, an increase in latency from say 500 to 1000 ms, may be traded off for increased energy efficiency even more readily.

We also plan to interleave disparate http applications and study the impact of simultaneous applications on the client. Our proxy is designed to handle multiple clients. Over time we will study the how the requirements on the power saving proxy scale to typical enterprises where several stations connect to a single access point. Another aspect to pursue will be the efficiency of the technique for faster wireless networks such as 802.11g and ultrawide band.

References

[ANF2003] M. Anand, E. B. Nightingale, and J. Flinn, "Self-Tuning Wireless Network Power Management," In *Proceedings of ACM MOBICOM 2002,* pp. 176-179

[BA2003] K. Barr and K. Asanovic, "Energy Aware Lossless Data Compression," In *Proceedings of the First International Conference on Mobile Systems, Applications, and Services (MobiSys'03)*, May 2003

[Chandra2002] S. Chandra, "Wireless Network Interface Energy Consumption Implications of Popular Streaming Formats," In *Proceedings of Multimedia Computing and Networking* (*MMCN*), January 2002, pp 85-99.

[CV2002] 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,* pp 329-342.

[Duchamp99] D. Duchamp. "Prefetching Hyperlinks," In *Proceedings of The 2nd USENIX Symposium on Internet Technologies & Systems, 1999,* pp. 127-138.

[GLDWC2003] M. Gundlach, S. Doster, D. K. Lowenthal, S. A. Watterson, and S. Chandra, "Dynamic, Power-Aware Scheduling for Mobile Clients Using a Transparent Proxy," Submitted to *Multimedia Computing and Networking (MMCN),* June 2003 (http://www.cs.uga.edu/~dkl/research/publications.html)

[KB2002] R. Krashinsky and H. Balakrishnan, "Minimizing Energy for Wireless Web Access with Bounded Slowdown," In *Proceedings of ACM MOBICOM 2002,* pp 119-130.

[KLM97] 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 the USENIX Symposium on Internet Technologies and Systems, 1997,* pp.13-22.

[SBGM2000] T. Simunic, L.Benini, P. Glynn, and G. De Micheli, "Dynamic Power Management for Portable Systems," In *Proceedings of ACM MOBICOM 2000,* pp 11-19.

[SBS2002] 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,* pp 160-171.

[SK1997] M. Stemm and R. H. Katz. Measuring and Reducing Energy Consumption of Network Interfaces in Handheld Devices. In *IEICE Transactions on Fundamentals of Electronics, Communications, and Computer Science,* August 1997

[O'Reilly] D. Gourley and B.Totty. HTTP: The Definitive Guide. *O'Reilly, 2002,* ISBN: 1-56592-509-2

[PageDetailer] IBM Page Detailer, http://www.research.ibm.com/pagedetailer

[Part11] Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, ANSI/IEEE Std 802.11, 1999

[GNU-wget] http://wget.sunsite.dk/

[PS2003] A. E. Papathanasiou and M. L. Scott, "Energy Efficiency through Burstiness," In *Proceedings of IEEE WMSCA 2003*, pp. 44-53.

[QCJS2003] 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*, pp. 161-175.