Minimizing Energy for Wireless Web Access with Bounded Slowdown

Ronny Krashinsky and Hari Balakrishnan
MIT Laboratory for Computer Science
200 Technology Square, Cambridge, MA 02139
{ronny, hari}@lcs.mit.edu

Abstract

On many battery-powered mobile computing devices, the wireless network is a significant contributor to the total energy consumption. In this paper, we investigate the interaction between energy-saving protocols and TCP performance for Web-like transfers. We show that the popular IEEE 802.11 power-saving mode (PSM), a "static" protocol, can harm performance by increasing fast round trip times (RTTs) to 100 ms; and that under typical Web browsing workloads, current implementations will unnecessarily spend energy waking up during long idle periods.

To overcome these problems, we present the *Bounded-Slowdown (BSD) protocol*, a PSM that dynamically adapts to network activity. BSD is an optimal solution to the problem of minimizing energy consumption while guaranteeing that a connection's RTT does not increase by more than a factor p over its base RTT, where p is a protocol parameter that exposes the trade-off between minimizing energy and reducing latency. We present several trace-driven simulation results that show that, compared to a static PSM, the Bounded-Slowdown protocol reduces average Web page retrieval times by 5–64%, while *simultaneously* reducing energy consumption by 1–14% (and by 13× compared to no power management).

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*; C.2.5 [**Computer-Communication Networks**]: Local and Wide-Area Networks—*Internet*

General Terms

Algorithms, Design, Measurement, Performance.

Keywords

Wireless, protocols, energy saving, power saving, bounded slowdown, IEEE 802.11, TCP, HTTP, Web.

MOBICOM'02, September 23–28, 2002, Atlanta, Georgia, USA.

1. Introduction

The capabilities of mobile computing devices are often limited by the size and lifetime of the batteries that power them. As a result, minimizing the energy usage of every component in a mobile system is an important design goal. Wireless network access is a fundamental enabling feature for many portable computers, but if not optimized for power consumption, the wireless network interface can quickly drain a device's batteries. This paper seeks to minimize the energy consumed by the wireless network interface for a mobile device generating request/response traffic (e.g., while browsing the Web) over a reliable transport protocol such as TCP. We investigate several interactions between energy-saving mechanisms and network performance, and show that understanding these interactions enables better energy-saving protocols to be designed that have provable performance-energy trade-offs.

Many wireless network interfaces, especially wireless LAN cards, consume a significant amount of energy not only while sending and receiving data, but also when they are idle with their radios powered up and able to communicate [3, 5, 7, 11, 15]. However, because wireless applications typically use the network in bursts, wireless interfaces are designed so they can be disabled when not in use to save energy. In this *sleep mode*, the radio is turned off, and the device has no way to determine when data is being sent to it over the wireless network link. This breaks the "always on" abstraction convenient for transport protocols such as TCP, where a node should be able to receive data from the network at any time. Therefore, any energy-saving protocol that puts the network interface in sleep mode must have a mechanism to handle the resulting communication outage.

As an example, consider the popular IEEE 802.11 wireless LAN specification, which describes a power-saving mode (PSM) that periodically turns the network interface off to save energy, and on to communicate [9]. In the so-called infrastructure mode (as opposed to the ad hoc network mode), a mobile device communicates with a wired access point (AP). When 802.11 PSM is enabled, the AP buffers data destined for the device. Once every *BeaconPeriod*, typically 100 ms, the AP sends a beacon containing a traffic indication map (TIM) that indicates whether or not the mobile device has any data waiting for it. The mobile device wakes up to listen to beacons at a fixed frequency and polls the AP to receive any buffered data. Typically, it listens to every beacon, but the mobile device can also be configured to skip *ListenInterval* beacons between listen times. Whenever the AP sends data to the mobile device, it indicates whether or not there is more data outstanding, and the mobile device goes to sleep only when it has retrieved all pending data from the AP. When the mobile device itself has data to send, it can wake up to send the data without waiting for a beacon.

The 802.11 PSM is an example of a *static* power-saving algorithm, since it does not adapt the sleep and awake durations to the degree of network activity; we will refer to it as *PSM-static* in this paper. We find that while PSM-static does quite well in saving energy, it does so at significant performance cost. In Section 2, we demonstrate that the round trip time (RTT) of a TCP connection can increase substantially with PSM-static, since the effect is to *round up* the RTT to the nearest 100 ms. This has an especially adverse impact on short TCP connections, whose performance is dominated by the connection RTT. We also find that an interesting *inversion effect* can occur, where under some conditions, the time to transfer a file over a wireless network running PSM-static *increases* when the bandwidth of the wireless link increases! Furthermore, with PSM-static, the power consumed while sleeping and listening for beacons dominates the total energy consumption if the network is accessed only sporadically. Section 3 shows that for Web workloads, the long (but randomly distributed) idle periods ("think time") end up being most important in terms of energy usage, and that PSM-static does not handle this situation well.

A PSM protocol addresses the following fundamental question: *When should a wireless interface go to sleep, and when should it be awake?* Based on our observations of the adverse and unexpected interactions that occur when a TCP connection is superimposed on PSM-static, we consider the problem of optimizing energy consumption under the constraint that interactive request/response performance does not degrade by more than a known amount. Specifically, we address the following problem: *Find an algorithm that minimizes energy consumption using the sleep and wake-up primitives such that any RTT does not exceed $(1 + p) \cdot R$, where R is the original RTT and p is a tunable parameter that controls the maximum percentage slowdown.*

Our solution to this problem results in the *Bounded-Slowdown (BSD) protocol*, described in Section 4. The idea is to adapt the sleep durations depending on past activity, so that no RTT is lengthened by more than a factor p, which exposes the performance-energy trade-off in a provable manner. This also allows the network interface to sleep for longer periods of time when there is no activity, thereby reducing the energy consumed while listening to beacons. In fact, for future network cards, this method could allow the network interface to go into a deeper sleep mode and save more energy.

Section 5 presents trace-driven simulations, using power parameters from a commercially available 802.11b card, to evaluate the effectiveness of the BSD protocol as a function of p and compare it to PSM-static for real-world Web traffic. We find that BSD tightly bounds the performance slowdown of Web retrieval times in all cases, and also often beats PSM-static in terms of energy consumption. For example, PSM-static reduces energy by 11× compared to no PSM, but does this at the cost of increasing average Web page retrieval times by 16–232% for network round trip times of 80 ms down to 10 ms. When $p = 1.0$, BSD increases average Web page retrieval times by only 11–19% over the base performance with no PSM, and simultaneously uses 1–14% less energy than PSM-static (and up to 13× less than no PSM)—both its performance and energy usage are always better than the 802.11 PSM. When $p = 0.2$, the BSD protocol essentially eliminates the slowdown in Web page retrieval times while only using up to 13% more energy than PSM-static.

The performance benefits of BSD over PSM-static are most significant when the TCP connection RTT is much smaller than 100 ms (the beacon period). We note that with the increasing deployment of Web content distribution networks (CDNs), server replication systems, Web proxies, and caches, the RTT for Web TCP connections is often small, especially for popular sites where CDNs and replica abound. For example, from both MIT on the east coast of the U.S., and Berkeley on the west coast, RTTs to several popular sites such as Google, Yahoo, CNN, etc. are less than 30 ms most of the time. In another common wireless network scenario, users often transfer or synchronize files (e.g., email) between a mobile device and a local server, and the base connection RTT is a few milliseconds. Although the performance of PSM-static could be improved by reducing the 100 ms beacon period, this would lead to significantly higher energy consumption (we discuss this in Section 5.3). We therefore believe that because of the trend toward smaller connection RTTs, the BSD protocol will be especially useful in bounding performance slowdown while saving considerable energy.

We discuss related work in Section 6 and conclude with a summary of our results in Section 7.

2. TCP Performance over PSM-static

TCP is the transport layer of choice for the majority of Internet applications. Its performance is therefore a critical consideration in the design of any network component. This section evaluates the impact of PSM-static on TCP performance.

During the initial slow-start phase of a TCP connection, the RTT dominates the overall transfer time for data. Since most TCP connections on the Internet are smaller than a few tens of kilobytes [8, 13], RTTs are a critical determinant of Web browsing performance. In this section, we analyze the impact that PSM-static has on the first RTT of a connection, then investigate the impact of PSM-static on RTTs for subsequent packets in a TCP transfer, then present experimental measurements of TCP transfers, and finally discuss an emergent performance inversion effect caused by PSM-static.

2.1 PSM-static Impact on RTT

With PSM-static enabled, the network interface enters a sleep state whenever it is not sending or receiving data. When the mobile device has data to send (e.g., a TCP `SYN` or `ACK` packet, a TCP data packet containing a Web request, etc.), it can wake the network interface up at any time. However, the network interface will go to sleep as soon as this data has been transmitted to the AP. When the response data arrives from the server after some delay, it must be buffered at the AP until the next beacon occurs. This delay increases the observed RTT for the connection.

If the mobile device initiates a request/response transaction, the observed RTT depends on when it sends the request data relative to the beacon period. For example, with an actual RTT of 20 ms and a beacon period of 100 ms, if the mobile device sends the request immediately after a beacon, the response will be buffered at the AP and received after the next beacon; thus the observed RTT will be 100 ms. If the mobile device sends the request 79 ms after a beacon, the AP will receive the response just before the next beacon and the observed RTT will be just over 20 ms. However, if the mobile

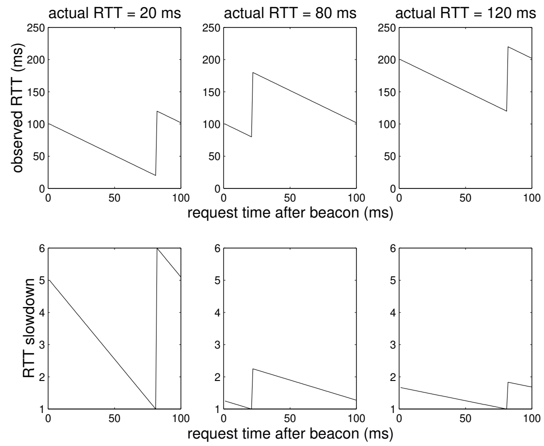

Figure 1: Slowdown due to PSM-static (the 802.11 PSM) for the first RTT of a TCP connection. The upper graphs show how the observed RTTs vary with how long after a beacon the request is sent. The lower graphs show the RTT slowdown (the observed RTT divided by the actual RTT).

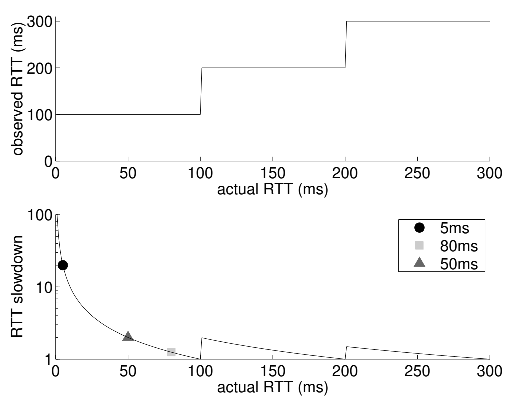

Figure 2: Slowdown due to PSM-static for initial RTTs of a TCP connection (excluding the first RTT). The upper graph shows how the observed RTTs vary with the actual RTT. The lower graph shows the RTT slowdown.

device sends the request 81 ms after a beacon, the AP will receive the response just after the next beacon and will have to buffer the data until the subsequent beacon; the observed RTT will be 120 ms, a factor of 6 slowdown. Figure 1 shows the impact of PSM-static on three example RTTs. The figure shows the observed RTTs and the slowdown compared to the actual RTT (the observed RTT divided by the actual RTT). The PSM-static slowdown is greatest for smaller RTTs.

PSM-static similarly affects RTTs when the mobile device responds to a request. In this case, the observed RTT depends on when the request arrives at the AP relative to the beacon period.

2.2 PSM-static Impact on TCP

When TCP is run over PSM-static, the initial RTT for a connection depends on when the request is sent in relation to the beacon period as shown in Figure 1. However, since the TCP data packets destined for the mobile device are delayed until the beginning of a beacon period, the mobile device always responds with TCP `ACK` packets immediately after the beacon and the TCP connection becomes *synchronized* with the PSM-static beacon period. Thus, the observed RTTs are *rounded up* to the nearest 100 ms. Figure 2 shows the estimated observed RTTs and slowdowns with PSM enabled. The slowdown is greatest when the actual RTT is significantly less than 100 ms.

As a TCP connection over a PSM-static link evolves, each window of data takes 100 ms to transmit until enough data is in transit to prevent the network interface from going into sleep mode. At the beginning of a beacon period the amount of data buffered at the AP is equal to the TCP window size (assuming sufficient bandwidth between the server and the AP). As soon as the mobile device wakes up and receives the first TCP data packet, it sends an acknowledgment prompting the server to send more data. The new data arrives from the server approximately one RTT (the actual server RTT, not 100 ms) after the start of the beacon period, during which time the wireless link continually transmits data at the link rate. If the AP finishes sending the buffered window of data to the mobile device before the new data arrives from the server, the mobile device will enter sleep mode until the next beacon time. However, if the buffered window of data keeps the wireless link busy until the new window of data begins to arrive from the server, the network interface will stay awake continuously and the power-saving mode will no longer degrade performance.

Thus, assuming sufficient bandwidth between the server and the AP, the transmission of each TCP window takes 100 ms until the window size grows to the bandwidth-delay product (the wireless link bandwidth multiplied by the actual server RTT) and one window can keep the wireless link busy for an entire round trip time. For this to occur, the mobile device's advertised TCP window must be sufficiently large. Additionally, enough buffering must be available at the AP to store the window of data; otherwise, TCP packets will be dropped and the connection will never be able to fully saturate the wireless link.

2.3 Measured TCP Performance

Figure 3 shows the measured evolution of a TCP connection with and without PSM enabled. For this test, the mobile client[1] opened a TCP connection with a server and sent a request for 40 kBytes of data; the server responded with the data. The network interface card (NIC) was rated at 11 Mbps, although the maximum possible TCP data throughput was less than this as shown in the results below. The server was in the same building as the 802.11 access point and three network hops away; the RTT was about 5 ms, and the bandwidth between the server and AP was at least 10 Mbps. The times at which data packets were sent from the server are shown, where time zero is the time that the server saw the initial `SYN` packet.

With PSM-static off, the connection quickly saturates the available bandwidth of the network—the maximum is around 6.4 Mbps, limited by the 802.11 wireless link. However, with PSM-static on, the initial RTTs are increased to 100 ms. With an actual RTT of 5 ms, only about 4000 bytes of buffered data should be required to keep the 6.4 Mbps link

[1]The mobile client used for all tests in this paper was a Compaq iPAQ H3600 series hand-held computer running Familiar Linux version 0.4 with an Enterasys Networks RoamAbout 802.11 DS High Rate network interface card.

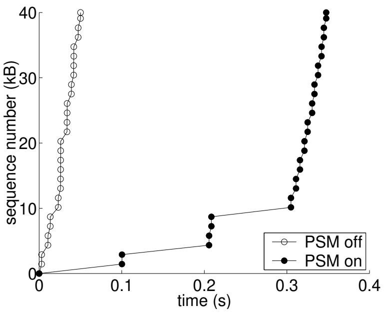

Figure 3: Measured evolution of a TCP connection with and without PSM enabled.

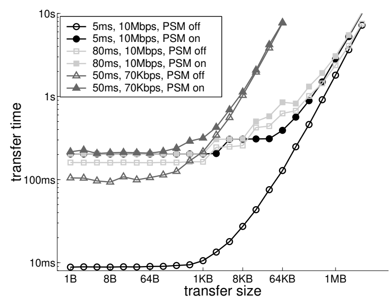

Figure 4: Measured data transfer size vs. transfer time for request/response transactions over TCP with various servers and PSM on and off. Both axes use log scale.

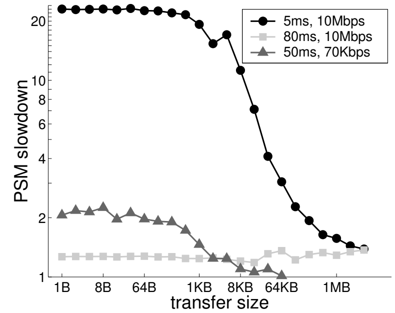

Figure 5: PSM slowdown from Figure 4. Both axes use log scale.

busy for long enough to allow the next window of data to begin arriving from the server and prevent the network interface from going to sleep. As shown in Figure 3, this happens after the third 100 ms RTT when the TCP window grows to 4 packets (about 6000 bytes). Since this connection has a short server RTT and small bandwidth-delay product, it is close to the best-case scenario for PSM-static in terms of saturating the link in the fewest number of 100 ms RTTs.

In another test of PSM-static, the mobile client opens a TCP connection to a server and sends a request for some number of bytes; the server responds by sending the requested block of data. By doing this for power-of-two data transfer sizes between 1 Byte and 4 MBytes, we determined the relationship between data transfer size and transfer time. The client used was the same iPAQ device. The server was run on various machines to evaluate different network characteristics. The first server was in the same building and three network hops away from the AP; the RTT was 5 ms, and the bandwidth was at least 10 Mbps. The second server was located around 3000 miles and 20 network hops away and had a high bandwidth network path to the AP; the RTT was 80 ms and the bandwidth was at least 10 Mbps. The third server was located around 3 miles and 8 network hops away and behind a DSL network connection; it had a 50 ms RTT and outgoing bandwidth of 70 Kbps. Each performance test was run ten times alternating between PSM on and PSM off (five tests each). The results showed no significant variations between runs, and the mean values are presented.

Figure 4 shows the total transfer time (including the request and response) as a function of data transfer size for each server with both PSM on and PSM off. Figure 5 presents another view of the same data; it shows the slowdown incurred using PSM. For small data transfer sizes the entire response fits in one or two TCP data packets, and the total time for the transaction is equal to two RTTs—during the first RTT the client sends a `SYN` packet to the server, and the server responds with a `SYN+ACK` packet; during the second RTT the client sends the request to the server and it responds with up to two data packets. With PSM off, the transfer time is determined by the RTT to each server; however, with PSM on, the transfer times are 200 ms regardless of the server. The observed slowdowns match those predicted by Figure 2.

The transfer times for the low-bandwidth (70 Kbps) server become bandwidth-limited even before the transfer requires more than one RTT. For the high-bandwidth servers, the transfer times begin to take multiple RTTs as the data transfer size increases and eventually become bandwidth-limited; the maximum bandwidth achieved was about 4.9 Mbps. With PSM on, the maximum bandwidth achieved was about 3.4 Mbps. Apparently, the maximum bandwidth is limited by the overhead incurred by the PSM signaling; a close look at Figure 3 reveals that the data packet spacing in steady state is slightly higher with PSM on.

In some cases, the mobile device sends data to a remote machine rather than vice-versa; for example, this occurs if a mobile user is uploading a file, serving data in a peer-to-peer application, running a Web server, etc. In this case, PSM-static causes the TCP `ACK`s to be delayed instead of the data packets. We ran the same performance test with the mobile device configured as the server, and a machine on the 5 ms, 10 Mbps network configured as the client. The results were essentially identical to those obtained when the mobile device was the client.

The main finding from these measurements is that the 100 ms sleep interval used in PSM-static is *too coarse-grained* to maintain good performance, especially for short TCP data transfers that are dominated by RTTs.

2.4 Performance Inversion

Somewhat paradoxically, TCP may achieve *higher throughput* over a *lower bandwidth* PSM-static link, resulting in performance inversion! As discussed in Section 2.2 and shown in Figure 3, PSM-static causes the transmission of each TCP window to take 100 ms until the window size grows to the product of the wireless link bandwidth and the network RTT delay between the mobile device and the server. Therefore, a lower bandwidth PSM-static link will become saturated sooner and prevent the network interface from entering sleep mode. Figure 6 shows simulation data that demonstrates this behavior.[2] The figure shows the transfer times versus the wireless link bandwidth for various server RTTs, both with and without PSM and for data transfers of 10 KBytes and 1 MByte. The AP to server bandwidth was set to 100 Mbps.

Figure 6 shows that a 1 Mbps wireless link is faster than

[2]The simulation methodology is described in Section 5.1.

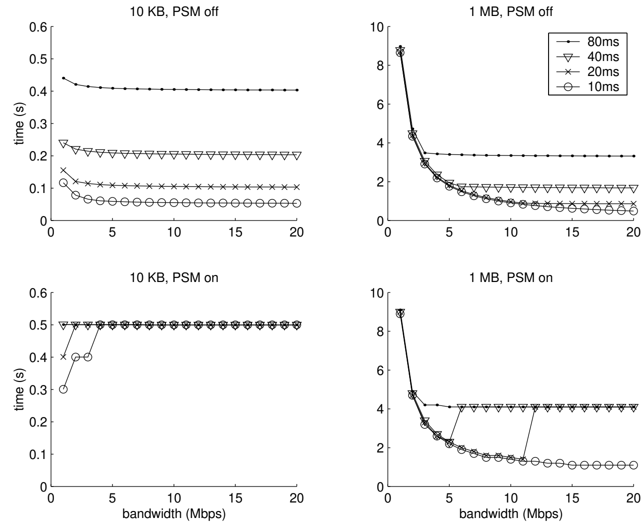

Figure 6: TCP request/response transfer times versus wireless link bandwidth for various server RTTs. The left graphs represent a 10 KB data transfer, and the right graphs represent a 1 MB data transfer. The upper graphs show transfers with PSM off, and the lower graphs show transfers with PSM on. Paradoxically, in some cases the transfer times are shorter with a lower bandwidth PSM-static link.

higher bandwidth links for a 10 KB data transfer. With a 10 ms server RTT, the connection has a bandwidth-delay product of 1,250 bytes (10,000 bits). Therefore, it becomes saturated during the initial TCP round trips, and the PSM stops putting the network interface into sleep mode; the request/response transaction takes just over 3 round trip times (300 ms). For wireless link bandwidths greater than 3 Mbps, or for server RTTs greater than 20 ms, the TCP window never grows to the bandwidth-delay product (or does so only on the last round-trip), and the request/response transaction always takes about 5 round trip times (500 ms).

The 1 MByte transfer size demonstrates an interesting interaction between TCP and PSM-static. Whenever the receiver's advertised maximum TCP window size is not large enough to keep the wireless link busy for an entire RTT, the throughput is limited to one maximum TCP window per beacon period. For the simulations, the mobile device's advertised window is 20 TCP packets (1,500 bytes each), or about 240 kbits, and one maximum window per 100 ms beacon period is equivalent to 2.4 Mbps. The bandwidth-delay product for a 20 ms server RTT crosses the 240 kbits threshold when the wireless link bandwidth increases from 11 Mbps to 12 Mbps, and for a 40 ms server RTT when the wireless link bandwidth increases from 5 Mbps to 6 Mbps. Once they cross this threshold, the transfer times increase sharply to 4.1 s, an average throughput of 2.05 Mbps. This shows an unexpected, emergent interaction between TCP and PSM-static. With long server RTTs, the receiver's advertised TCP window limits performance even with PSM off (e.g., the throughput saturates at about 2.5 Mbps for an 80 ms server RTT), but it does not lead to the performance inversion.

These results also demonstrate that, if PSM-static is used, absolute performance may degrade if the wireless link bandwidth increases (e.g., with 802.11a).

3. Client Network Usage Characteristics

In optimizing a network access protocol to minimize power consumption, it is important to consider how clients use the network. Since there is a trade-off between the extent to which power consumption is minimized in sleep mode and how long it takes to wake up (and also how much energy the transition takes) [1, 14], the sleep duration determines how low the power consumption can be. In addition, waking up to listen to beacons consumes energy; the listen interval determines the significance of this overhead.

To evaluate the characteristics of client network usage, we analyzed client Web traffic from the UC Berkeley Home IP dialup service traces [8]. The network activity for these traces is dominated by long transfer times over the slow modem links, but certain aspects are relevant to general client usage patterns. In particular, the time that clients spend idle (presumably due to user "think time") or waiting for responses from servers present opportunities for the network interface to enter a sleep mode, and these times are probably not critically dependent on the bandwidth of the client's network link.

In analyzing the traces, we tracked each client's state as one of:

wait-for-server: the client has one or more outstanding requests, but is not receiving any responses

receive-response: the client is receiving one or more responses

inactive: the client has no outstanding requests and is not receiving any responses (this includes both user "think time" and browser processing time)

Figure 7 shows the cumulative distribution function (CDF) for the client wait-for-server times. The solid line shows the percentage of wait-for-server events that last for less than a given elapsed time. The dashed line shows the percentage of the total wait-for-server time that is spent in these events. For example, 88% of all wait-for-server events take less than 1 s, and these events account for 19% of the total time spent in all wait-for-server events.

Figure 8 shows the CDF for the client inactive times. The solid and dashed lines are as in Figure 7. However, in the traces many clients have no activity over a period of several days; if this data is included these inactive times completely dominate the total inactive time (as shown by the dotted line which is barely visible above the x-axis). Therefore, inactive events longer than 1000 s (around 2% of all inactive events) were excluded from the total inactive time represented by the dashed line. The figure indicates that 26% of all inactive events take less than 1 s, and these events account for 0.5% of the total inactive time. If only inactive events less than 100 s are included (versus 1000 s as shown in Figure 8), the inactive events less than 1 s account for 1.1% of the total; and if only inactive events less than 10 s are included, the inactive events less than 1 s account for 6.8% of the total.

For completeness, Figure 9 shows the CDF for the client receive-response times. Since the clients use slow modem links, long transfer times are the norm. The prevalence of receive-response times less than 0.1 s is presumably due to responses that fit in one TCP packet.

The important point about these results is that although most wait-for-server and inactive events are of short dura-

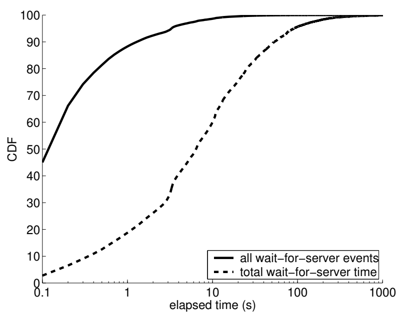

Figure 7: CDF for client wait-for-server events and total wait-for-server time.

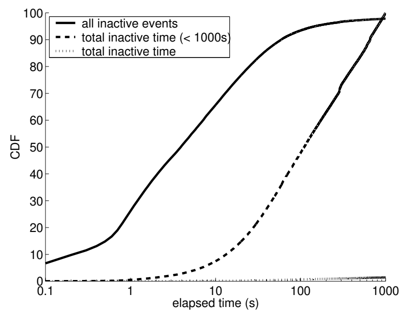

Figure 8: CDF for client inactive events and total inactive time.

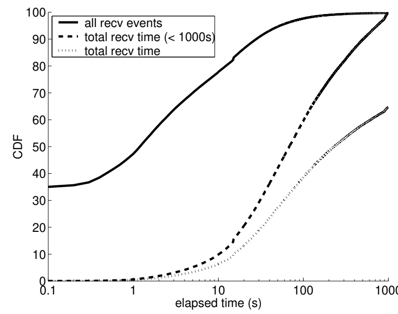

Figure 9: CDF for client receive-response events and total receive-response time.

tion, long latency events account for most of the total wait-for-server and inactive times. For example, over 80% of the total wait-for-server time and virtually all of the total inactive time is spent in events longer than 1 s. This holds true even when inactive times longer than 100 s or even 10 s are excluded. Since the energy spent by a network interface in its power-saving sleep mode is directly proportional to the sleep duration, this finding indicates that wait-for-server and inactive periods longer than 1 s account for most of the network interface sleep energy. Thus, considering that it requires the network interface to perform the energy-consuming operation of waking up to receive a beacon every 100 ms, the 802.11 PSM seems *too fine-grained* to minimize energy consumption effectively.

4. Bounded-Slowdown (BSD) Protocol

Section 2 demonstrated that a static PSM protocol such as that used by 802.11 can be too coarse-grained to give good Web performance, while Section 3 demonstrated that the same protocol can be too fine-grained to minimize energy. This section presents the BSD protocol that employs an adaptive algorithm to maintain performance while minimizing the energy consumed by a wireless network interface.

In the context of request/response network traffic, a static PSM protocol guarantees that RTTs are not delayed by more than one beacon period. We claim that this guarantee is inadequate both in terms of performance and energy consumption. Our basic assumption is that, for request/response network traffic, the percentage increase in round trip times is more important than the absolute increase from the perspective of higher-layer protocols and human users. For example, PSM-static might increase a 40 ms RTT to 100 ms and a 9,940 ms RTT to 10,000 ms; the first situation is far worse than the second. Although, one might think that keeping round trip times under 100 ms is good enough for human perception, the important point is that request/response transactions involve multiple *additive* RTTs. For example, in Section 5.2 we show that by increasing a 40 ms RTT to 100 ms, PSM-static *more than doubles* the time to retrieve many Web pages that originally had download times as long as one second. In terms of energy, PSM-static will wake up to listen to 100 beacons during a 10 s idle period, in the end ensuring that a 10 s RTT is not increased by more than 1%. If a 10% slowdown is acceptable, the energy spent listening to beacons can be reduced by an order of magnitude.

Motivated by these observations, we seek a protocol that consumes the minimum possible energy while guaranteeing that round trip times do not increase by more than a given percentage. In contrast to a static protocol, our algorithm must dynamically adapt to network activity. To avoid delaying very fast RTTs, the network interface can *stay awake* for a short period of time after the link is active. Then, to consume less energy listening to beacons, it can *back off* and listen to fewer beacons when there is no network activity.

A constraint on our power-saving protocol is that it must operate completely at the link layer with no higher-layer knowledge. Since it does not know whether particular blocks of data actually comprise a request or response, it should conservatively assume that any data sent from the mobile device is a request and it should not assume a correspondence between any particular blocks of send and receive data. A result of designing a low-level protocol is that its guarantees are valid even when different connections share the same network interface; e.g., RTT slowdowns will be bounded even when the mobile device has multiple TCP connections to different servers with different network delays.

Formally, if the base RTT in the absence of PSM is R, then the goal is to minimize energy while limiting the observed RTT to $(1+p) \cdot R$; for a specified parameter $p > 0$, this limits the RTT increase to $100 \cdot p$ percent. We present an optimal algorithm that meets this goal. We start with an observation about sleep durations:

LEMMA 1. *If, after sending a request at time $t_{request}$, the mobile device has received no response at time $t_{current}$, then the network interface can go to sleep for a duration up to $(t_{current} - t_{request}) \cdot p$ while bounding the RTT slowdown to $(1+p)$.*

This is true because for the greatest slowdown, the actual RTT, $R_{actual} = (t_{current} - t_{request})$, and the observed RTT, $R_{observed} = R_{actual} + R_{actual} \cdot p$; therefore $R_{observed} \leq (1+p) \cdot R_{actual}$.

To minimize energy, an optimal algorithm must clearly always put the network interface into the sleep state as soon as possible and for as long as possible. However, to bound the slowdown, the mobile device must periodically check with the AP for buffered data as governed by Lemma 1. Therefore, if (for the moment) we neglect synchronization constraints between the wireless network interface and the AP, we can state the following theorem:

THEOREM 1. *To minimize energy while bounding RTT slowdown to a factor $(1+p)$, a network interface should go to sleep*

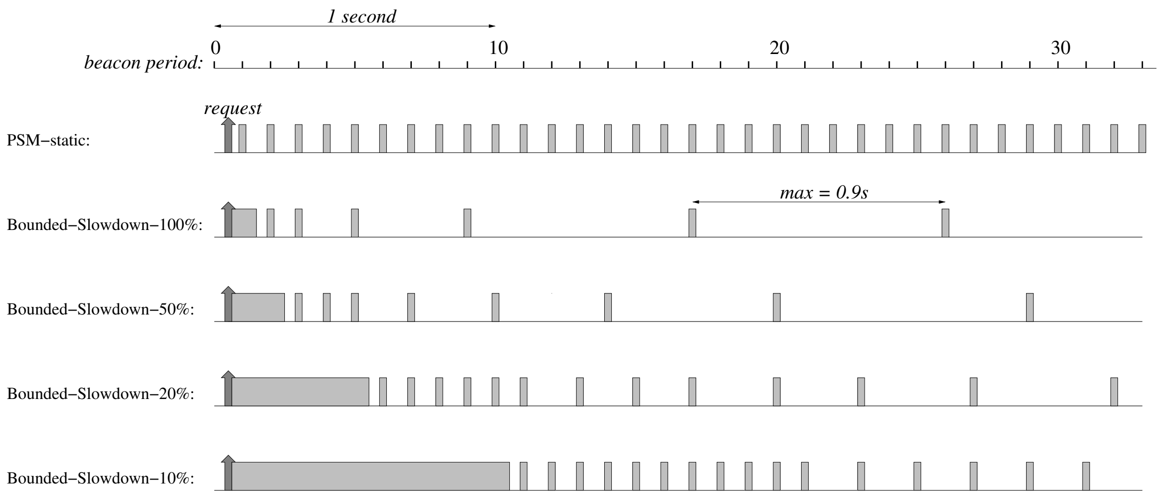

Figure 11: Schematic representation of PSM-static (the 802.11 PSM) and various Bounded-Slowdown alternatives (labeled as $100 \cdot p$ percent). The arrow indicates a request sent by the mobile device, the initial shaded area indicates when BSD stays awake for a set time T_{awake} after the request, and the shaded bars indicate when the network interface wakes up to listen to beacons.

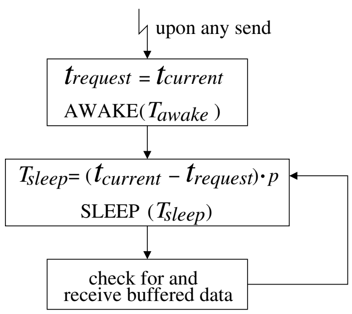

Figure 10: The BSD Algorithm. After the mobile device sends any data, the network interface initially stays awake for T_{awake}. Then it sleeps for $T_{awake} \cdot p$ before waking up to check for and receive any buffered data. It repeats this pattern, each time sleeping for the duration since the request was sent multiplied by p. The algorithm is not affected by data being received, but it restarts whenever the mobile device sends any data (e.g., a TCP ack for data received). Note that $t_{current}$ continuously changes to reflect the current time.

an infinitesimally short period of time after it sends any request data, and only wake up to check for response data as governed by Lemma 1.

The Bounded-Slowdown algorithm is summarized in Figure 10. For the ideal case with no synchronization constraints, T_{awake} is an infinitesimally small positive value. The algorithm restarts whenever the mobile device sends new data; this can never cause Lemma 1 to be violated for a previous request because BSD will check for data *more* frequently than if it did not restart.

Although the ideal algorithm minimizes energy, it results in sleep and wake intervals that are of arbitrary length and infinitesimally small. To use the protocol in a realistic implementation, we assume the mobile device and AP are synchronized with a fixed beacon period T_{bp}, as in 802.11 PSM. Then, T_{sleep} must always be rounded down to a multiple of T_{bp}. Under these constraints, the mobile device might delay an RTT by up to T_{bp} the first time it goes to sleep. Therefore, $T_{awake} = T_{bp}/p$, so that the mobile device initially stays awake for $1/p$ beacon periods; if the response arrives during this time it will be delivered without delay.

Figure 11 shows the behavior of PSM-static and the BSD protocol for various values of p (these are labeled as $100 \cdot p$ percent). To allow direct comparisons with the 802.11 PSM, we set T_{bp} to 100 ms. Additionally, in our implementation the BSD protocol sets the maximum sleep duration to 0.9 s to avoid TCP timeouts.[3] Considering one example in Figure 11, when $p = 0.2$ (20%), $T_{awake} = T_{bp}/p = 500ms$, so the network interface stays awake for half a second after the mobile device sends a request. Then, it begins sleeping and waking up to listen to every beacon while T_{sleep} is rounded down to 100 ms. After a second has elapsed since the request, T_{sleep} is 200 ms, so it sleeps for two beacon periods, and so on.

We note that this algorithm is slightly conservative in its assumption that an RTT might be delayed by T_{bp} the first time the mobile device goes to sleep. The maximum delay will actually be less than this if the mobile device goes to sleep in the middle of a beacon period, but it would be difficult for a practical implementation to calculate the exact relationship between $t_{request}$ and the beacon period. Therefore, our algorithm conservatively assumes that $t_{request}$ occurs just after a beacon and always stays awake for a set period of time. This assumption does not alter the correctness (in terms of the RTT bound) of the algorithm since it never violates Lemma 1.

Updating the existing 802.11 MAC to support the BSD protocol should be fairly straightforward. One difference is that instead of going to sleep immediately after sending data to

[3]Increasing the maximum sleep duration would only serve to further reduce the energy spent listening to beacons. However, as shown in Section 5.1, this would not lead to worthwhile overall energy reduction with current 802.11 network cards since the sleep energy dominates the listen energy for a 0.9 s sleep interval.

the AP, the mobile device stays awake for a set period of time. To ensure that the AP forwards data to the mobile device without delay, it could be informed of this time period. Or, it could always notify the device as soon as data arrives from the network instead of waiting for the next beacon; in this case the device could retrieve the data if awake. The other difference under the BSD protocol is that the mobile device does not listen to beacons at a set interval. The 802.11 specification already allows for a ListenInterval which is different than the BeaconPeriod; the only enhancement is to enable the ListenInterval to change more dynamically. A potential concern is that the amount of buffering required at the AP is now larger, since the mobile device listens to fewer beacons; however, since the mobile device stays awake after sending data, it will usually receive responses immediately and thereby *reduce* the AP's overall buffering load. The reduced frequency of listening to beacons typically occurs when there is little networks activity to the mobile device.

In summary, with the BSD protocol, fast response times are not delayed, while longer ones are increased by up to a parametrized maximum factor, $1 + p$. Compared to PSM-static, the active energy is increased since the transition to sleep mode is delayed, but the energy spent listening to beacons is decreased due to the longer sleep intervals. We note that the BSD protocol is designed for a mobile device that initiates request/response network traffic. As such, it is not appropriate for real-time communication, or for a mobile device that acts as a server and responds to external requests.

5. Evaluation

To analyze the performance and energy impact of PSM-static, and to evaluate the proposed PSM enhancements, we simulated a mobile client browsing the Web over a wireless network link.

5.1 Simulation Methodology

Using the network simulator `ns-2` [17], we modeled a mobile client communicating with an access point over a wireless link with PSM. Since we were not concerned with many details that 802.11 accounts for—such as signal strength, channel contention, node movement, and multicast—we chose not to model the detailed MAC protocol, but instead made some simple modifications to the basic link object in `ns-2`. Sleep mode is simulated by deactivating the queue elements of a link so that they do not forward any packets, and waking up simply entails activating the queues. The mobile device wakes up whenever it has data to send to the AP. After sending the data it stays awake for the duration determined by the BSD algorithm (this time is restarted if the mobile device sends more data in the interim), and then goes to sleep. The beaconing is implemented using a timer that expires every 100 ms. We determine whether the mobile device wakes up to listen to each beacon based on the BSD protocol. If it does, it receives any data buffered at the AP and then goes back to sleep. Based on the experiments described in Section 2, we modeled the AP-to-mobile-device and mobile-device-to-AP links as 5 Mbps with a latency of 100 μs.

To model a client browsing the Web, we used the trace-based HTTP traffic generator in the `ns-2` distribution (in `ns-2.1b8a/tcl/http/`). In the model, the retrieval of a Web page begins with a client opening a TCP connection with the server and sending a request. The original model uses one-way TCP connections, but we updated it to use FullTcp connections. After some delay, the server sends a response, and then the client retrieves some number of embedded images. To get these, the client opens up to four parallel TCP connections with the server. Then, the client waits for some amount of think time between Web page retrievals. The various parameters in this model are randomly chosen based on empirical data [13]. As in Section 3, we limited the user think time to 1000 seconds because otherwise think times as long as an entire day would completely dominate the total think time. We also added a server response time that delays the start of an HTTP response; it does not affect the subsequent RTTs for the TCP connection. We based this on the wait-for-server data in Figure 7, but subtracted 100 ms from these times since they include the network delay (so, 45% of the time the server responds with no delay).

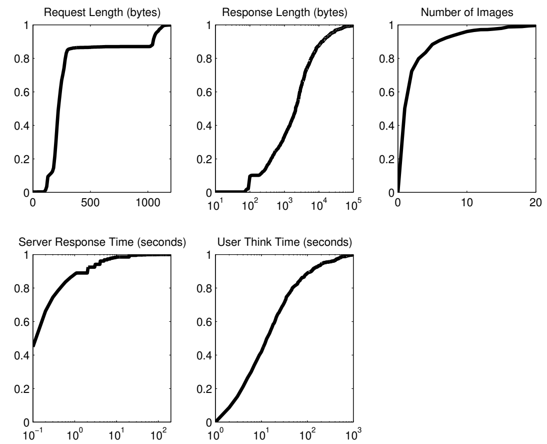

Figure 12: CDFs for randomized parameters used in HTTP traffic simulations.

To evaluate PSM-static and BSD, we modeled a network consisting of a mobile client, an access point, and a server. For each of various PSM settings, we simulated a client retrieving 10,000 Web pages; these comprised a total of 38,428 HTTP request/response transactions, and around 541,000 seconds of client Web browsing time. Figure 12 shows the CDFs for the actual randomized parameters used in the simulations. Admittedly, using a single server with a set bandwidth and RTT is a simplification and significantly affects the performance impact of the PSM. We always set the AP to server bandwidth to 10 Mbps, but we show results for various server RTTs to show the variation.

To simulate the power consumption of the 802.11 network interface card, we used a simple model inspired by various reported measurements [3, 5, 7, 15] and our own power measurements of the Enterasys Networks RoamAbout NIC shown in Figure 13 (the methodology used to make the measurements was similar to that in [3]). We modeled the power usage as 750 mW while awake (sending data, receiving data, or idle), and 50 mW while asleep. In reality, 802.11 cards consume somewhat more power while sending and receiving data than while idle; however, as demonstrated below, the additional energy used for actually transmitting data while Web browsing tends to be insignificant. After analyzing Figure 13(c), we modeled the energy consumed in waking up and listening to a beacon as 1.5 mJ, based on an approximation of 750 mW being consumed for 2 ms.

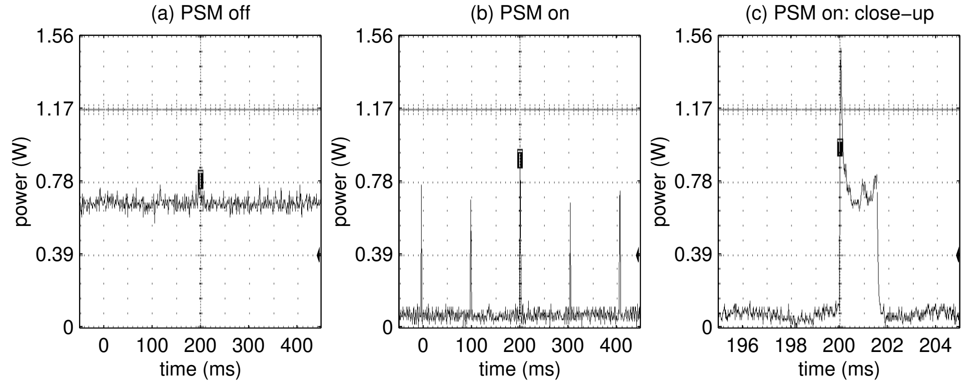

Figure 13: Power measurements of an Enterasys Networks RoamAbout 802.11 DS High Rate NIC. The first graph shows power consumption with PSM off, the second shows power consumption with PSM on, and the third shows a close-up of the power consumed when the NIC wakes up to listen to a beacon.

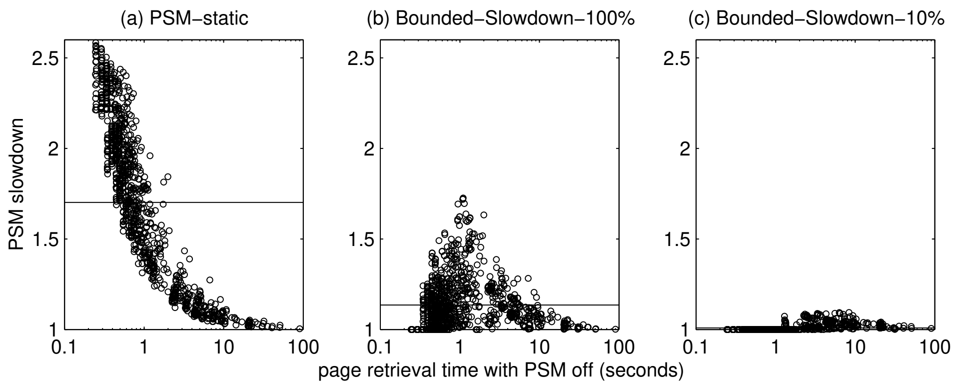

Figure 14: Per-page slowdown for three PSM alternatives based on a server RTT of 40 ms. Each marker represents a single Web page; the x-coordinate is the retrieval time with PSM off, and the y-coordinate is the slowdown when PSM is on. The horizontal line in each graph shows the mean slowdown (1.69 for PSM-static, 1.14 for BSD-100%, and 1.01 for BSD-10%).

5.2 Web Page Retrieval Times and Energy

Figure 14 compares the performance of PSM-static with two BSD alternatives based on a server RTT of 40 ms.[4] Each marker on Figure 14 represents the retrieval of a single Web page; the x-coordinate is the retrieval time with PSM off, and the y-coordinate is the slowdown when PSM is on (that is, the retrieval time with PSM on divided by the retrieval time with PSM off). The figure shows that PSM-static has the greatest negative impact on pages with fast retrieval times. These are slowed down by up to about $2.5\times$ which is the penalty for extending a 40 ms RTT to 100 ms. BSD-100% shows a large improvement, and does bound the worst-case slowdown to be smaller than $2\times$. In fact, all of the slowdowns are far less than this bound because the protocol keeps the network interface awake for 100 ms after the mobile device sends data, so fast RTTs are not slowed down at all. BSD-10% further improves performance and shows almost no slowdown.

Figure 15 compares the mean per-page energy and retrieval time of various PSM alternatives. To show the sensitivity to the server RTT, each has results for RTTs of 10 ms, 20 ms, 40 ms, and 80 ms. The energy values were obtained by dividing the total energy by the number of pages, while the slowdown values were obtained by taking the mean of the slowdown for each individual page (rather than dividing the total retrieval time by the number of pages). In this way, each page is given equal weight independent of its retrieval time; this is shown graphically by the horizontal lines in Figure 14. The first set of bars in Figure 15 shows a link with PSM off, and the second set shows PSM-static. The next four sets show instances of the BSD protocol with the maximum slowdown p set to 100%, 50%, 20%, and 10%.

Enabling PSM-static reduces energy by about a factor of 11, but suffers from a slowdown of 16–232% depending on the server RTT. Based on the estimates, the energy spent while awake is negligible since the network interface is in sleep mode for around 1000 times longer than it is awake. Waking to listen to beacons accounts for 23% of the total power consumption; a direct result of the energy cost of listening to a beacon (which takes 2 ms) compared to the energy cost of sleeping for 98 ms: $\frac{1.5mJ}{1.5mJ+98ms\cdot 50mW} = 23\%$.

In all cases, the BSD protocol results in faster average page retrieval times than PSM-static; even BSD-100% never increases the average retrieval time by more than 19% compared to a link with no PSM. BSD successively improves re-

[4]Throughout this section, note that even with a fixed server RTT, the actual RTT for request/response transactions may be longer due to the server response time shown in Figure 12.

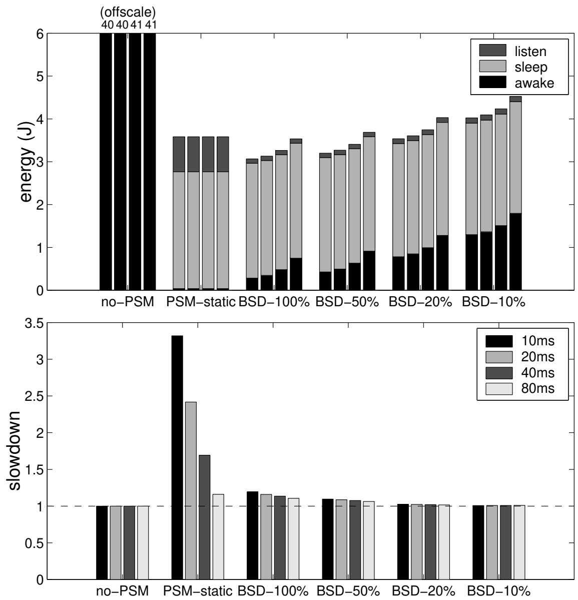

Figure 15: Mean per-page energy and slowdown comparisons for various PSM alternatives and various server RTTs. In both graphs, each set of bars show results for server RTTs of 10 ms, 20 ms, 40 ms, and 80 ms.

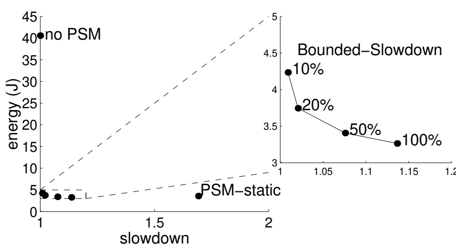

Figure 16: Mean per-page energy versus slowdown for various PSM alternatives (for 40 ms server RTT).

trieval times as the slowdown parameter is decreased, and eventually it almost completely eliminates the slowdown.

To improve performance as the slowdown parameter is decreased, BSD successively increases the awake energy since it stays awake for longer after the mobile device sends data. The awake energy also increases with slower server RTTs since BSD typically remains awake for entire TCP data transfers, and these become longer. However, BSD also reduces the energy spent listening to beacons since it adaptively increases the listen interval when there is no activity. The listen energy is reduced by $6.8\times$ with BSD-10% and $8.2\times$ with BSD-100%, close to the maximum reduction of $9\times$ that would be achieved by listening every 900 ms (the maximum listen interval we allow) instead of every 100 ms. Combining these two energy effects, BSD uses even less energy overall than PSM-static in many cases, and even in the worst case it only increases the energy by 26%. Figure 16 shows the trade-off between page retrieval time and energy consumption based on a 40 ms server RTT. Clearly the parameterized BSD protocol reduces communication latency at the cost of increased energy consumption, and vice-versa.

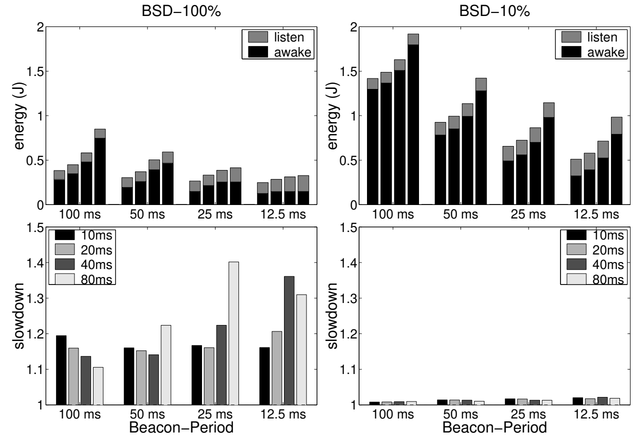

Figure 17: Mean per-page energy and slowdown comparisons for various beacon periods. The graphs on the left show results for BSD-100%, and those on the right for BSD-10%. In all graphs, each set of bars show results for server RTTs of 10 ms, 20 ms, 40 ms, and 80 ms. The "sleep" energy has been eliminated for clarity.

5.3 Further Energy Reduction

The results show that the energy remaining after PSM is enabled is mostly dominated by the power consumed while the network interface is *sleeping*. There is no fundamental limit that prevents this power from being reduced further, but doing this can result in additional energy and delay overhead when the network interface awakens. For example, Simunic *et al.* report that the NIC can be turned off so that it consumes no power, but the transition to the off state takes around 62 ms and the transition back takes around 34 ms [15]. Clearly this mode cannot be used when the network interface must wake up every 100 ms to listen to a beacon, but as Simunic finds, turning off the network interface during long idle periods can save considerable energy. Thus, by extending the sleep intervals BSD has the potential to use deeper sleep modes to significantly reduce the sleep energy.

Assuming that hardware advances can reduce the sleep energy toward zero, the overall energy consumption for BSD will become dominated by the awake energy. Most of this energy is consumed after the link is active and the network interface stays awake for a short period of time; this behavior is demonstrated in Figure 11.

To minimize the awake energy while still preserving the bounded slowdown guarantee, we can decrease the beacon period. Doing so allows the network interface to go to sleep sooner after the link is active because, $T_{awake} = T_{bp}/p$; this brings the protocol closer to the ideal algorithm described by Theorem 1. Figure 17 shows the effect of reducing the beacon period on BSD-100% and BSD-10%. As expected, the awake energy is reduced with shorter beacon periods. For example reducing the beacon period by a factor of 8 reduces the awake energy by 56–80% for BSD-100% and 44–75% for BSD-10%. However, at the same time, the energy spent listening to beacons increases since the sleep/listen cycle starts sooner. Additionally, performance degrades if the link does not stay awake for as long as the minimum round trip times. For example, with BSD-100%, the network interface stays awake for one beacon period before going to sleep; hence, with an 80 ms RTT, the average slowdown increases substantially when the beacon period is reduced from 100 ms to 50 ms, while with a 40 ms RTT, the average slowdown increases substantially when the beacon period is reduced from 50 ms to 25 ms. BSD-10% stays awake for 10 beacon periods initially, so even reducing the beacon period to 12.5 ms barely harms its performance.

Even if we do not assume zero sleep energy, reducing the beacon period can be an important energy saving technique for the BSD protocol. For BSD-10%, reducing the beacon period to 12.5 ms results in it using between 12% less to 2% more energy than PSM-static depending on the server RTT. Thus, it essentially eliminates the page retrieval slowdown while simultaneously reducing energy in almost all cases. Although the page retrieval times with PSM-static could also be improved by reducing the beacon period to 12.5 ms, this would increase the energy spent listening to beacons by a factor of 8, and increase the overall energy by a factor of 2.6.

6. Related Work

This paper has two main contributions: First, it presents a detailed analysis of the effect that fine-grained intermittent connectivity, such as that of the 802.11 power-saving mode, has on TCP throughput and latencies. We believe that this is the first work to explore this issue in depth. Second, it presents the BSD protocol to minimize energy consumption while providing a guaranteed bound on RTT slowdown for request/response network traffic.

A survey of energy efficient network protocols for wireless networks is provided in [10]. Although many researchers have explored power management strategies, we believe that ours is the first work that also focuses on provably bounding the performance slowdown.

6.1 Reducing Idle Energy in Infrastructure Networks

The BSD protocol operates solely at the MAC layer and does not require any higher-layer information. Other proposals have advocated power management at the system-level [15], the application-level [2], or a hybrid of the two [11]. Our view is that, in many cases, this unnecessarily increases overall system complexity. Higher-layer protocols may have difficulties when multiple applications share the same network interface, but BSD has no problems with such a situation since it is agnostic to applications. We have shown that a low-level protocol can provide performance guarantees for request/response network traffic, thus covering an interesting and broad (although not comprehensive) class of network applications, while flexibly and dynamically adapting to network activity in order to eliminate energy during long idle periods. An appropriate system-level or application-level decision could be to choose the BSD protocol for request/response network access (along with an appropriate value for p, the slowdown parameter), and to choose a different PSM mechanism (e.g., PSM-static) for other kinds of network applications.

Simunic *et al.* describe system-level power management strategies that turn the network interface off completely during idle periods to reduce its power consumption by about $3\times$ compared to 802.11 without power management [15].[5] However, the policy cannot guarantee a bound on the performance slowdown. BSD is a simpler algorithm than the proposed time-indexed semi-Markov decision process model (TISMDP); and, even while providing guaranteed bounds on the slowdown by taking network performance (RTTs) into account, it can achieve additional energy savings on top of the $11\times$ energy reduction of the 802.11 PSM-static protocol. Although not evaluated in our paper, BSD could save even more energy by (like TISMDP) transitioning from sleep to off when the listen interval becomes sufficiently long. Furthermore, BSD is designed to operate at a finer granularity than TISMDP and they could potentially be used in conjunction.

Chandra investigates an application-specific protocol for reducing the network interface power consumption for streaming media applications [2]. The proposal uses a history-based strategy to set the sleep interval (analogous to the listen interval). The protocol is only applicable for regular access patterns, in contrast to BSD which dynamically responds to network activity.

Kravets and Krishnan investigate the energy and delay impact of a power-aware transport protocol [11]. Ideally, applications inform the protocol when the network interface should be turned off; the paper also discusses using a timeout period (analogous to the initial stay awake time in BSD), but does not evaluate this parameter's energy and delay impact. The paper investigates the sleep duration (analogous to the listen interval), and finds that a 500 ms sleep duration achieves most of the possible energy savings; it reduces Web browsing energy by 30–80% compared to no power management, at the cost of increasing delay by 300–700 ms. Kravets and Krishnan also present an adaptive implementation of their algorithm in which they set the sleep duration to 250 ms when there is network activity, and adaptively back off to 5 minutes by doubling when there is no activity; they find that this can improve both energy and delay. Ultimately, the power-saving mechanisms are similar to those used by BSD, but the proposed protocol does not use them to guarantee a bound on the performance degradation.

6.2 Other Energy Reduction Techniques

Chen *et al.* evaluate the energy consumption of various access protocols for wireless infrastructure networks [4]. In contrast to our work, their study focuses on the active energy consumption and the impact of contention for the wireless channel. These factors are certainly important for some environments, but with sporadic network activity the idle energy consumption dominates the active energy.

There have been many studies on the performance and energy consumption of ad hoc wireless networks (e.g. [3, 7, 16, 18]). Infrastructure networks have fundamentally different requirements than ad hoc networks because the access point is a centralized controller and is not constrained by power consumption. However, the basic concepts behind the BSD protocol could still potentially improve the performance and energy consumption of ad hoc networks.

Another area of related work is power management of hard disks [6, 12]. Like the network interface, hard disks can be disabled to save energy. Adaptively varying the disk spin-down threshold [6] shares similarities with adaptively

[5] The wireless card used in [15] consumes 1.65 W while sending, 1.4 W while receiving, and 0.045 W while sleeping, but the average power consumed during a Web browsing trace is 1.41 W; therefore, we conclude that the card is either constantly receiving data or not using its sleep mode while idle.

increasing the network interface listen interval. However, hard disks use mechanical components and require orders of magnitude more time and energy to transition into sleep modes. Another fundamental difference with the network interface is that the information for determining when to reactivate the component may not be local to the mobile device; a packet can arrive from the network (external to the device), and the device must wake up to receive it.

7. Conclusion

We investigated the performance effects of superimposing a TCP connection on a static PSM protocol (PSM-static) modeled after the popular IEEE 802.11 wireless LAN power-saving mode. Using a combination of analysis, measurement, and simulation, we found that while this protocol reduces the energy consumed during Web access by 11× compared to no PSM, the RTTs of a TCP connection get rounded up to the nearest 100 ms until the TCP window size grows to the network bandwidth-delay product. This has an especially adverse impact on the short TCP connections typical for Web workloads, whose performance is dominated by the RTT; for a client-side Web trace, we found that the average Web page retrieval time increases by 16–232%. We also discussed an emergent interaction between TCP and PSM-static that leads to performance inversion in which TCP achieves higher throughput over a lower bandwidth PSM-static link. Furthermore, for Web workloads characterized by bursts of activity interspersed with long idle periods, PSM-static consumes most of its energy sleeping and unnecessarily waking up to listen to beacons during the idle periods.

To overcome these problems, we presented the BSD protocol that dynamically adapts to network activity. We proved that BSD uses the minimum possible energy necessary to guarantee that connection round-trip times do not increase by more than a given factor p compared to the RTT in the absence of any PSM, where p is a protocol parameter that exposes the trade-off between minimizing energy and reducing delay. To accomplish this, BSD stays awake for a short period of time after a request is sent, and adaptively listens to fewer beacons when the link remains idle. Staying awake reduces communication delay but increases energy consumption, while listening to fewer beacons saves energy but can increase delay. We evaluated BSD using trace-driven Web traffic simulations and parameters from a commercially available 802.11b wireless LAN card, and found that, compared to PSM-static, BSD reduces average Web page retrieval times by 5–64%, while *simultaneously* reducing energy consumption by 1–14% (and by up to 13× compared to no power management).

Acknowledgments

We gratefully acknowledge Kyle Jamieson for his help and comments, especially with measuring the network interface card power consumption. We also thank Magdalena Balazinska, Ken Barr, Chris Batten, Allen Miu, and the Mobicom reviewers for helpful feedback. This work was funded in part by DARPA PAC/C award F30602-00-2-0562.

8. References

[1] C. Andren, T. Bozych, B. Rood, and D. Schultz. Prism power management modes. Technical Report AN9665, Intersil Corporation, February 1997.

[2] S. Chandra. Wireless network interface energy consumption implications of popular streaming formats. In *Proc. Multimedia Computing and Networking (MMCN)*, pages 85–99, San Jose, CA, January 2002.

[3] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *ACM Wireless Networks*, 8(5), September 2002.

[4] J.-C. Chen, K. M. Sivalingam, and P. Agrawal. Performance comparison of battery power consumption in wireless multiple access protocols. *ACM Wireless Networks*, 5(6):445–460, 1999.

[5] Cisco Systems, Inc. Quick reference guide Cisco Aironet 340 series products, January 2001.

[6] F. Douglis and P. Krishnan. Adaptive disk spin-down policies for mobile computers. *Computing Systems*, 8(4):381–413, 1995.

[7] L. Feeney and M. Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *Proc. IEEE INFOCOM*, Anchorage, AK, April 2001.

[8] S. D. Gribble and E. A. Brewer. System design issues for Internet middleware services: Deductions from a large client trace. In *Proc. USENIX Symposium on Internet Technologies and Systems*, Monterey, CA, December 1997.

[9] IEEE Computer Society LAN MAN Standards Committee. *IEEE Std 802.11: Wireless LAN Medium Access Control and Physical Layer Specifications*, August 1999.

[10] C. E. Jones, K. M. Sivalingam, P. Agrawal, and J. C. Chen. A survey of energy efficient network protocols for wireless networks. *ACM Wireless Networks*, 7(4):343–358, July 2001.

[11] R. Kravets and P. Krishnan. Application-driven power management for mobile communication. *ACM Wireless Networks*, 6(4):263–277, 2000.

[12] K. Li, R. Kumpf, P. Horton, and T. E. Anderson. A quantitative analysis of disk drive power management in portable computers. In *Proc. Winter USENIX*, pages 279–291, San Francisco, CA, January 1994.

[13] B. A. Mah. An empirical model of http network traffic. In *Proc. IEEE INFOCOM*, Kobe, Japan, April 1997.

[14] E. Shih, S.-H. Cho, N. Ickes, R. Min, A. Sinha, A. Wang, and A. Chandrakasan. Physical layer driven algorithm and protocol design for energy-efficient wireless sensor networks. In *Proc. ACM MOBICOM*, pages 272–286, Rome, Italy, July 2001.

[15] T. Simunic, L. Benini, P. W. Glynn, and G. D. Micheli. Dynamic power management for portable systems. In *Proc. ACM MOBICOM*, pages 11–19, Boston, MA, 2000.

[16] S. Singh and C. Raghavendra. PAMAS: Power aware multi-access protocol with signaling for ad hoc networks. *ACM SIGCOMM Computer Communication Review*, 28(3):5–26, July 1998.

[17] The VINT Project. *The ns manual*, November 2001.

[18] H. Woesner, J.-P. Ebert, M. Schlaeger, and A. Wolisz. Power-saving mechanisms in emerging standards for wireless LAN's: The MAC-level perspective. *IEEE Personal Communication Systems*, 5(3):40–48, June 1998.