ACE: An Active, Client-Directed Method for Reducing Energy During Web Browsing[*]

Haijin Yan, David K. Lowenthal, and Kang Li
Dept. of Computer Science
University of Georgia

ABSTRACT

In mobile devices, the wireless network interface card (WNIC) consumes a significant portion of overall system energy. One way to reduce energy consumed by a device is to transition its WNIC to a lower-power *sleep* mode when data is not being received or transmitted.

This paper develops *ACE*, an active, client-directed technique to improve energy efficiency during web browsing. *ACE* actively retrieves buffered packets from an access point based on predictions made through client-side connection tracking. The key novel implementation technique used in *ACE* is connection rescheduling, which results is a better energy/time tradeoff for interactive applications such as web browsing. We demonstrate the effectiveness of *ACE* through actual experiments to real Internet servers.

Categories and Subject Descriptors: C.2.3 [Computer Communication Networks]: Network Operations—wireless communication

General Terms: Measurement, Performance

Keywords: Energy, TCP, HTTP

1. INTRODUCTION

Energy conservation is a key problem in the post-PC era, because the availability of mobile devices depends greatly on battery capacity. One significant source of consumed energy on such devices is the wireless network interface card (WNIC); in fact, it can in some cases be the single largest power drain in a mobile client. For example, the WNIC of an IBM 560X laptop accounts for 15% of overall system energy [1] even when all components are active.

Web browsing is a popular mobile application [2]. A typical web browsing session involves periods of downloading relatively short files followed by user inactivity. As the WNIC is typically designed with multiple power states, the most effective way to reduce the energy consumed by a WNIC is to transition it to a lower-power *sleep* mode when data is not being received or transmitted. This can help mobile devices save energy during user think time as well as during gaps that often occur within an active download.

Generally, energy-techniques trade performance for energy. Mechanisms like IEEE 802.11b power-saving mode (PSM) [3] force data to be buffered at the access point and retrieved periodically by the client. This allows the client to increase the time the WNIC spends in *sleep* mode at a cost of increased delay at the access point—which results in increased download times. As an optimization to PSM, the Bounded Slowdown Protocol (BSD) [4] bounds the RTT increase to a parameterized maximum factor, but typically saves energy only during inactive periods. As an alternative, our prior work [5] proposes a transport layer technique, called CC, that saves energy during web downloads in which concurrent HTTP connections are instantiated. However, the number of concurrent connections must be small in order to achieve reasonable energy savings during downloads, and, more importantly, variation in RTTs could lead to CC transitioning the WNIC to *sleep* mode during packet arrivals—which causes the client to drop these packets.

In this paper, we describe the design and implementation of a novel technique that we call *ACE* (**A**ctive, Client-Directed **E**nergy Savings). Like CC, *ACE* tracks connections in order to predict when packets arrive. *ACE* differs, however, in that it uses these predictions to actively retrieve data from the access point. *ACE* leverages off PSM; in a sense, we are evolving the low-level, passive PSM mechanism into a high-level, "client-directable" mechanism with adaptive buffering periods based on application requirements. By using PSM as part of its implementation, *ACE* will never make an incorrect transition to *sleep* mode.

ACE also leverages knowledge of typical web browsing behavior. In particular, it aligns embedded connections with the main (top-level) web connection. This allows *ACE* to increase the user-perceived latency of the entire page by much less than the increase of any particular embedded connection. This in turn leads to a better energy/time tradeoff in *ACE* than with previous approaches.

ACE requires only a small modification to PSM; the most significant change would be the need for PSM to export a set of APIs to the operating system. This requirement, however, conforms to the ACPI specification, which strongly encourages hardware and software vendors to build operating system directed power management (OSPM) for use by applications.

We have implemented and measured the performance of

---

[*] This research was supported by NSF grants CCF-0429643 and CCF-0234285.

ACE in an emulated environment as well as to real Internet servers. Results show that our solution typically produces a superior energy/time tradeoff as compared to regular TCP, PSM, BSD, and CC. As a particular example, *ACE* saves over 80% energy in the best case—even when a large number of concurrent connections exist. *ACE* can also successfully handle Internet dynamics such as round-trip time variance and packet loss.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 describes the design and implementation of *ACE*, and Section 4 describes our experiments and discusses the results. Finally, Section 5 summarizes this paper.

2. RELATED WORK

The IEEE 802.11 specification provides a power saving mode to help mobile devices reduce energy dissipation [3]. In [4], Krashinsky et. al proposed the Bounded Slowdown Protocol (BSD), which dynamically adapts the beacon period. BSD minimizes energy consumption while guaranteeing that RTTs do not increase by more than a given percentage. BSD can save energy relative to PSM during long idle periods. However, as it operates solely at the link layer, with no higher-layer knowledge, it must assume that any data sent by a mobile client (including an acknowledgement) is a request. This causes it to restart the active period; the result is BSD cannot save energy during downloads.

In [5], we proposed a connection tracking and management technique, called CC, that exploits HTTP traffic patterns for energy conservation during Web browsing with concurrent connections. This approach tracked connections to identify intervals where no data is being exchanged, and then the WNIC is transitioned to *sleep* mode during these intervals. This technique depends largely on the ability to predict packet arrival times; when predictions are accurate, CC saves energy without any performance degradation. However, large variations in RTT can cause missed packets, and too many concurrent (overlapped) connections prevents any *sleep* time.

There has been work done that is specific to (different) applications or general, application-independent mechanisms. For example, work in [6] (among others) investigates energy-saving mechanisms for clients that are running multimedia applications. The idea is to use a proxy that is interposed in between servers and mobile clients; the proxy coordinates with the client to schedule burst arrival times.

There have also been application-independent approaches, including employing system level techniques [7], managing energy explicitly [8], and exploiting application information to cooperate with the OS to save energy [9]. It also includes power-awareness at the transport layer [10]. Finally, examples of hardware power management are dynamic voltage-scaling processors [11] and disk spin-down algorithms [12].

ACE differs from all techniques mentioned above as follows. First, it typically obtains a better energy/time tradeoff than TCP, PSM, BSD, or CC. In terms of application-specific approaches, *ACE* is based on web browsing; this means that TCP is used, which differentiates it from proxy-based techniques for multimedia clients. In addition, it exploits both application- and network-level information, which makes application-independent techniques described above complementary.

3. IMPLEMENTATION

This section describes the implementation of *ACE*. We first describe how we handle a single connection. Next, we discuss extending *ACE* to multiple concurrent connections. Then, we provide a few low-level implementation details. We summarize by comparing *ACE* to PSM, BSD, and CC. Note that *ACE* assumes (small) web downloads; our prior work [13] handles large file downloads.

3.1 Single Connection

ACE takes advantage of the bursty nature of Web traffic [14, 15]. Most web pages are small and therefore TCP is in slow start mode for most, if not all, of the download. This allows *ACE* to save energy between two consecutive slow start bursts.

The basic idea for handling a single connection is to identify these intervals using transport layer information and to schedule receipt of data using the link layer. To collect transport layer information, we adopt the connection table and packet prediction techniques from our prior work [5], which was at the transport layer. At the link layer, we employ PSM. We will discuss required PSM extensions below.

Initially, the client web browser will issue a TCP SYN packet to the top-level web site. We insert an entry into the connection table and record all the related information for the connection such as IP address and current status. Upon receiving the SYN-ACK from the web server, we estimate the round-trip time (RTT) for the connection. Note that this is stored in the connection table and re-used.

We then use this RTT to direct fine-grain PSM packet buffering at the access point as follows; the steps below repeat until the entire web page is downloaded.

- The client notifies the access point to begin buffering packets and enters PSM mode. Note that when using *ACE*, the client is *always* in PSM mode; the trick is that it actively, instead of passively, pulls data.

- Whenever there is a HTTP request, we transition the WNIC to high-power mode and send out the request. Simultaneously, we send PS-Polls to pull all buffered packets from the access point. Note that here our PS-Poll packets are sent actively as compared to standard PSM, where PS-Poll packets are sent only in response to a beacon packet that indicates the existence of buffered packets. The PSM specification does not prohibit sending of PS-Poll packets at arbitrary times.

- After emptying the access point, we transition the WNIC to low-power sleep mode while the browser is rendering the downloaded content. As it will take approximately a RTT for the response of the request to arrive at the client, we can safely keep the WNIC in *sleep* mode for one RTT. (The access point will buffer any incoming packets during this period, so even if our prediction is inaccurate, no packets will be lost.)

- After an RTT, we transition the WNIC to high-power mode and actively send PS-Poll packets to retrieve any buffered packets.

In this manner, packets always arrive at the client in bursts with approximately an interval of one RTT; the client may sleep in between these bursts. The bursty behavior occurs even if a particular server is in congestion avoidance phase of a TCP stream.
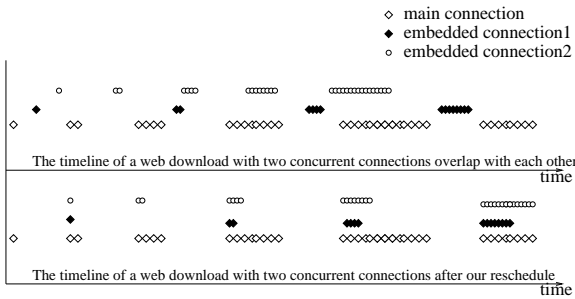
Figure 1: Timeline of a web access with concurrent connections for embedded objects.

For a single connection, ACE adds little, if any, extra delay. The only delay possible occurs if our predicted RTT is larger than the actual RTT. In this case, ACE will pull the data later than it otherwise would arrive (if ACE were not used). Note that these delays are in general small and may be offset by the rendering process.

3.2 Multiple Concurrent Connections

Web pages often consist of many embedded objects such as images, advertisement banners/windows, and audio clips. To reduce latency, popular web browsers often invoke multiple concurrent connections to download the necessary objects in a web page. The typical timeline of downloading and rendering a web page is as follows: the browser first retrieves the main HTML file corresponding to the Web page (through the main connection). Then, it parses the file, identifies the embedded objects that the page contains, and requests some or all of these objects concurrently—without waiting for an intervening response from the server. The concurrent transmissions of the components of a web page and the interleaving of downloading and rendering greatly reduce user-perceived latency.

As illustrated in Figure 1, when concurrent connections overlap, potential sleep time is substantially decreased. To save energy, we need to reschedule connections without significantly increasing overall download time. The key is that we will exploit application-layer information.

We handle concurrent connections as follows. First, we assume that the entire web page is done downloading when all connections have completed. This allows us to add extra delay for embedded objects, potentially without any time increase. To do this, ACE reschedules concurrent connections such that they are aligned to the main connection. This alignment makes web browsing appear to be using only one single main connection. As a result, even when multiple concurrent connections exist, we can maximize the WNIC sleep intervals—basing them on the traffic pattern of the main connection.

We use the RTT of the main connection as our wakeup interval to align the concurrent connections. Each connection occupies an entry in the connection table. We define an outgoing request a SYN packet or an HTTP request. Those two types of packets are active requests from the client as compared to the passive TCP acknowledgements generated in responses to received data packets.

The extended algorithm, for multiple connections, is then as follows. Whenever there is an outgoing request from *any* connection, we transition the WNIC to high-power mode,

send out the request, pull all buffered packets from the access point, and then transition the WNIC to *sleep* mode for one RTT of the *main* connection. (Our prior work has described how to measure RTT accurately using TCP timestamps and/or autocorrelation analysis [16].) Note that the algorithm is almost identical to the single-connection algorithm (Section 3.1)—the only differences are that we wake up for any connection, and the client sleeps based on the main connection. This avoids delaying outgoing requests from any connection at the client and is also important to ensure that ACE does not interfere with rendering done by the web browser.

The advantage of transitioning to high-power mode and pulling all available packets from the access point is that packet buffering delay is reduced. On the other hand, the disadvantage is that it may add extra delay to some packets.

In particular, suppose the client will wakeup at time T_1 to retrieve the (predicted) buffered packets at the access point. Suppose also that at time $T_2 = T_1 - \theta$, the web browser issues an outgoing request. As discussed above, this causes buffered packets to be pulled. However, the WNIC is placed in *sleep* mode until time $T_2 + RTT$. While for the retrieved packets, the buffering delay is reduced by θ, any packets that arrive when originally expected (T_1) will be *delayed* by $T_2 + RTT - T_1$, which is $RTT - \theta$.

The extra delay increases with the number of concurrent connections. This is because the delay described above occurs with a probability that it is in general independent of the particular connection characteristics. (Here we are assuming a reasonably smooth distribution of RTTs between connections.) In the worst case that all the delays are added to the main connection, the extra delay for the entire download will be the product of twice the RTT and the number of embedded objects in a Web page, even if delays occur during slow start. This assumes that each object is downloaded through a separate connection. Because the RTT is generally small and there are a usually at most a moderate number of objects in a Web page, the delay we introduce is small. For example, for the average case of a 60ms RTT and 5 embedded objects, the worst case performance degradation is less than 1s, which corresponds to an overhead of around 20%.

3.3 Other Implementation Details

There are several other aspects to our implementation. Due to space limitations, we mention them only briefly here. First, determining when a connection is idle is difficult because of the widespread use of persistent HTTP connections. We mark a connection idle when (1) there are no pending requests, and (2) no packets arrive on the connection for two consecutive RTTs. If there are packet exchanges on a marked idle connection, we resume tracking this connection. Second, when we are in a user think period, we adaptively increase the beacon period in a similar way to [4].

Updating existing PSM to support ACE should be fairly straightforward. It only requires the PSM interface to be exported to the operating system. Through this interface, all WNIC link-layer power management functionality can be exported to an OSPM module and integrated with power management in other components in the system.

A potential concern is potential missed beacon packets, as the mobile clients might not be awaken when access points are broadcasting beacons. One solution could be for access
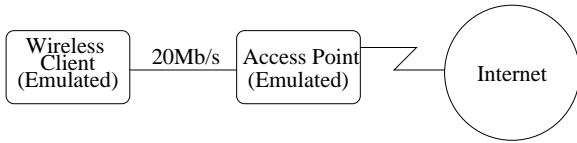
Figure 2: Our experimental setup for the tests to actual Internet servers.

points to send an unicast beacon whenever there is a PS-Poll request. Moreover, as the content of a beacon provides primarily time and traffic synchronization information, once a mobile client is associated with an AP and operates actively to predict and send poll packets, receiving only a subset of beacons is acceptable.

Table 1 summarizes the properties of our algorithm as well as the three competing algorithms. *ACE* improves upon prior work in that it is superior when considering all desired dimensions of a solution. In particular, while *ACE* is not best in any category, it performs well in each dimension.

4. PERFORMANCE

This section describes our experiments and presents our results. It is important to note that we are running actual experiments in this paper. Although simulation can be a feasible approach, it does not reflect real situations where concurrent connections are established as a result of web page rendering and the effect of network dynamics such as round-trip time variations.

Our experimental setup is shown in Figure 2. The wireless client was emulated by a 1GHz Pentium desktop machine running Linux 2.4-18. The access point is emulated using another 1GHz desktop running FreeBSD 5.1stable; we used tunneling so that *DummyNet* [17] could be used to provide a 20Mb/s bandwidth between access point and client. This value was selected by experimenting with 54 Mb/s access points with an actual wireless client and measuring the peak bandwidth attained. We use a 100Mb/s connection from the access point to the Internet.

On the access point, we use a customized snoop agent [18] to emulate PSM packet buffering. When notified by the client, the snoop agent begins to buffer packets for the client. Upon receiving PS-Poll packets, the snoop agent sends the buffered packets to the client, following the standard 802.11 PSM specification. In our implementation, we use specific ICMP packets to emulate PS-Poll packets and beacons.

On the wireless client side, we emulate the operation of *ACE* through connection tracking and prediction at the transport layer. At the predicted wakeup time, we transition the WNIC to idle mode and send PS-Poll packets to retrieve buffered packets from the access point. We transition the WNIC back to sleep mode after retrieving all buffered packets. Any packets that arrive when the WNIC is in *sleep* mode will be buffered at the access point.

We compare *ACE* with regular TCP, PSM, BSD [4], and CC [5]. PSM was emulated by a transparent proxy. As most BSD variants have identical behavior during active transmission time, we chose BSD-10%. In each experiment, a client executes Mozilla, version 1.2 with HTTP 1.1, and requests a specific web page. We select the 100 most popular web sites (denoted *top-level sites*) as determined by the Alexa Top Sites web pages [19]. We carried out our experiments
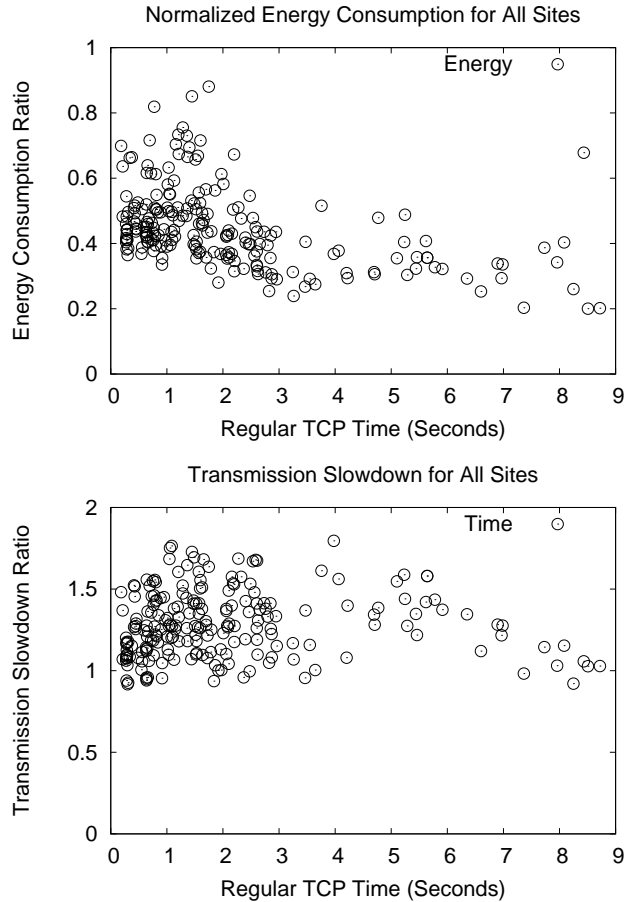

Figure 3: Overall normalized energy consumption and transmission time results for *ACE* to the 100 most popular Internet sites.

by running a script that accesses those sites in sequence. We present the overall mean download slowdown and energy savings of all techniques as compared to regular TCP.

4.1 Overall Experiments

Figure 3 shows the overall transmission slowdown and energy consumption for *ACE* when accessing the main page from the top 100 popular web sites. Here we only investigate *ACE* during active web page downloads, i.e., without user think time involved. As the selection of user think time could affect performance results considerably, we discuss here only active download time. Note that [4] argues that while think time is the majority of the time in a typical web session, active downloading periods are still significant. During think times, *ACE* can be set to perform identically to BSD, and BSD is the best out of the (safe) prior techniques during think times. We run our experiments multiple times and select the mean result for each site. On the x-axis is the download time for regular TCP, and on the y-axis is the transmission slowdown or the normalized energy consumption compared to regular TCP. Each site is represented by a point in the figure.

We see clearly that for most sites, *ACE* incurs only a

Alg	Time Increase	Energy (Active/Think)	Handle RTT Variation?	Handle Concurrent Connections?
TCP	None	Poor/Poor	Yes	Yes
PSM	Based on beacon period	Near-opt/Good	Yes	RTT increase per connection
BSD	User preference	None/Near-opt	Yes	Yes
CC	Depends on network variance	Good/Near-opt	No	Moderate
ACE	**Small**	**Good/Near-opt**	**Yes**	**Yes**

<div align="center">

Table 1: Summary of algorithms.

</div>

modest transmission slowdown—on average, about 32%. In general, the slowdown tends to decrease as download time increases, as indicated by the right portion of the graph. However, for fast responding sites, i.e., sites with a download time within 2 seconds, there is no obvious relationship between the transmission slowdown and the download time.

ACE slows the transmission because its connection scheduling (or RTT variations) can cause extra packet delay at the the access point. For delays caused by connection scheduling, as long as the delayed packet does not belong to the last connection to finish and does not contain any external links, the delay does not affect transmission time. This is generally the case for packets carrying embedded object data. On the other hand, when packets arrive later than expected due to network congestion, i.e., after a client poll, they are delayed until the next poll. In either the case, the maximum a packet can be delayed at the access point in our technique is bounded by one RTT (see Section 4.3).

The extra delay incurred by *ACE* affects performance most during slow start. When there are extra delays at the access point (see Section 3) during slow start, the sender does not increase its congestion window as fast as regular TCP. This is apparent by looking at short download times (especially two seconds or less). For web sites with a large download time, the delays incurred during slow start are amortized over the lifetime of the download. Also, a larger number of concurrent connections *decreases* delay because the client polls the access point (and pulls packets) more frequently.

In terms of energy, *ACE* achieves more than a 40% reduction compared to regular TCP. For complicated web sites such as www.espn.com, even though there are a large number (over 10) of concurrent connections, *ACE* still reduces energy consumption more than 50%. This points towards *ACE* being able to conserve energy by scheduling concurrent connections to create profitable sleep intervals. *ACE* does best (in terms of energy-time tradeoff) with a simple main connection and a large round-trip time (e.g., www.bbc.co.uk).

4.2   Comparison To Other Techniques

In this section, we compare *ACE* to existing techniques (PSM, BSD-10, and our prior work, denoted CC). We selected the top 10 sites from our Alexa list, which comprised a variety of content and network conditions.

Figure 4 gives the comparison results. Of all the techniques, *ACE* achieves the best performance/energy tradeoff. Previous work showed that CC saved more energy than BSD because the latter is almost always identical to regular TCP during active transmission time [5]. *ACE* improves significantly upon CC and is comparable to PSM in energy conservation. This is especially true for sites with a large number of concurrent connections or a short RTT. For example, for www.msn.com and www.ebay.com, there are more than 10 (of-
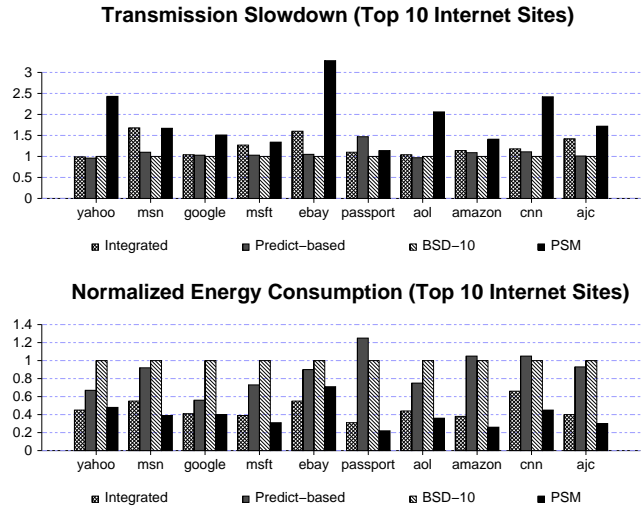

Figure 4: Energy and transmission time for *ACE*, CC, BSD, and PSM. All results are normalized to regular TCP (so smaller bars are better)

ten overlapping) concurrent connections; for these, CC saves almost no energy. On the other hand, *ACE* schedules the connections and so saves significant energy, despite the small amount of overhead in the scheduling itself.

The other improvement is for sites with short RTTs like www.cnn.com and www.ajc.com. *ACE* saves energy while CC does not, because it leverages buffering at the access point and never misses a packet. For example, CC cannot save energy on www.passport.net and www.amazon.com because a packet loss occurred in each, while *ACE* can. Of course, *ACE* has the disadvantage that it has to rely on PSM, whereas CC is independent of the network infrastructure.

Compared to PSM, *ACE* transitions the WNIC more frequently based on connection tracking. This enables energy conservation while still maintaining good performance. *ACE* is actually better than PSM in terms of energy in cases where PSM significantly increases the transmission time.

In terms of transmission slowdown, PSM is the worst of all four techniques—often more than a factor of 2. This is expected because actual RTTs during transmission are rigidly aligned to the nearest $100ms$. For sites with RTTs less than $100ms$, all BSD variants are equivalent in that they save almost no energy during active downloading. As a result, they do not have any transmission slowdown. BSD can only save energy during inactive user think times by skipping some beacons. CC incurs slowdown only when packets
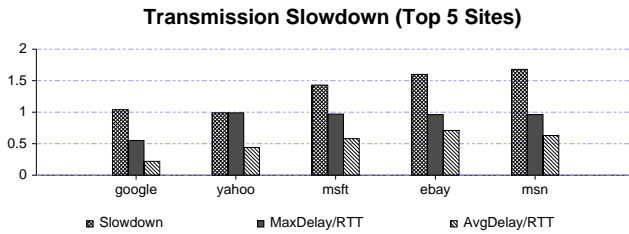
Transmission Slowdown (Top 5 Sites)

Figure 5: Transmission slowdown and max/avg delay for selected sites. Delays are normalized to the RTT.

are missed because it is passive and so does not affect web traffic. With *ACE*, packets may experience extra delay in the access point and thus incurs transmission slowdown, as discussed in Section 4.1.

In summary, *ACE* achieves a smaller, yet comparable, energy reduction than PSM while allowing downloads to be *much* faster. *ACE* also is superior to BSD in energy consumption. *ACE* is also better than CC in three common situations: for sites with (1) many concurrent connections, (2) short RTTs, and (3) high RTT variance.

4.3 Overheads

In this section, we give a detailed analysis of *ACE*. In particular, we investigate the kinds of overhead incurred by *ACE*. Space limitations prevent presentation of full results.

Our technique incurs two types of overhead: (1) packets experience extra delays in the access point, and (2) there is extra cost for transitioning the WNIC from *sleep* to *idle* mode. We know that the first delay is bounded to one RTT. To give a quantitative analysis, we monitor the delay of each packet at the access point and calculate the maximum and average packet delays over several sites (see Figure 5).

Clearly, we see that the maximum delay is bounded to RTT for all the sites. Generally, the higher the average delay, the larger the slowdown. The mean average delay for all the tested sites is about one-half of one RTT. As we discussed in Section 4.1, delays in slow start degrade the performance the most, while delays for downloading embedded objects do not affect the performance nearly as much. Overall, the delay can be controlled by the wireless client, which implies that better QoS energy-delay tradeoffs can be supported by *ACE* in the future.

5. CONCLUSION

In this paper, we present *ACE*, a novel active, client-directed approach to reducing energy consumption during web browsing. Our approach reschedules concurrent connections to align concurrent connections, which increases the possible time the WNIC can spend in *sleep* mode to save energy. *ACE* tracks connections and predicts when packets will arrive. It uses this information to guide the behavior of PSM. We demonstrated that *ACE* can save significant energy with comparably little overhead, compared to PSM, BSD, and CC.

6. REFERENCES

[1] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *SOSP*, pages 48–63, December 1999.

[2] David Kotz and Kobby Essien. Analysis of a campus-wide wireless network. In *Mobicom*, pages 107–118, September 2002.

[3] IEEE Computer Society LAN/MAN Standards Committee. IEEE Std 802.11: Wireless LAN medium access control and physical layer specification. Technical report, August 1999.

[4] Ronny Krashinsky and Hari Balakrishnan. Minimizing energy for wireless web access with bounded slowdown. In *Mobicom*, September 2002.

[5] H. Yan, R. Krishnan, S. A. Watterson, and D. K. Lowenthal. Client-centered energy savings for concurrent HTTP connections. In *NOSSDAV*, 2004.

[6] S. Chandra and A. Vahdat. Application-specific network management for energy-aware streaming of popular multimedia formats. In *USENIX*, 2002.

[7] Tajana Simunic, Luca Benini, Peter Glynn, and Giovanni De Micheli. Dynamic power management for portable systems. In *Mobicom*, pages 11–19, 2000.

[8] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. ECOSystem: Managing energy as a first class operating system resource. In *ASPLOS*, October 2002.

[9] Manish Anand, Edmund Nightingale, and Jason Flinn. Self-tuning wireless network power management. In *Mobicom*, September 2003.

[10] S. Singh, M. Woo, and C. S. Raghavendra. Power-aware routing in mobile ad hoc networks. In *Mobicom*, pages 181–190, 1998.

[11] T. Pering, T. Burd, and R. Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *ISPLED*, August 1998.

[12] D. P. Helmbold, D. D. E. Long, and B. Sherrod. A dynamic disk spin-down technique for mobile computing. In *Mobicom*, pages 130–142, 1996.

[13] H. Yan, R. Krishnan, S. A. Watterson, David K. Lowenthal, Kang Li, and Larry L. Peterson. Client-centered energy and delay analysis for TCP downloads. In *IWQOS*, June 2004.

[14] Long Le, Jay Aikat, Kevin Jeffay, and F. Donelson Smith. The effects of active queue management on web performance. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 265–276, 2003.

[15] Mark Crovella and Azer Bestavros. Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes. In *SIGMETRICS*, May 1996.

[16] B. Veal, K. Li, and D. K. Lowenthal. New methods for improving passive estimation of round-trip times using tcp timestamps. In *PAM*, March 2005.

[17] Luigi Rizzo. Dummynet: A simple approach to the evaluation of network protocols. *ACM Computer Communications Review*, 27(1), January 1997.

[18] Hari Balakrishnan et al. Improving TCP/IP performance over wireless networks. In *Mobicom*, November 1995.

[19] Alexa Top Sites. http://www.alexa.com/site/ds/top_500.