A Practical PSM Scheme for Varying Server Delay

Brian Peck, *Student Member, IEEE*, and Daji Qiao, *Member, IEEE*

***Abstract*—Today, many individuals use smartphones or other battery-powered mobile devices equipped with Wi-Fi to access the Internet. Since the network interface cards (NICs) often use a substantial portion of available energy, schemes such as the 802.11 power saving mode (PSM) have been used to limit the amount of time a NIC is awake in an operable state so as to extend battery life. However, since the NIC cannot retrieve packets while sleeping, PSM may negatively impact packet delay. These delays are also unbounded and can significantly increase communication time, particularly in a mobile environment where server delays may change drastically. To mitigate this, we present PSM-AW: a PSM with adaptive wake-up. PSM-AW is a client-side solution that allows the client device to sleep for a maximum time interval while still keeping a tight bound on performance. We present and prove a bound on performance using PSM-AW and demonstrate its effectiveness through extensive simulations and experiments.**

***Index Terms*—IEEE 802.11 WLAN, MadWifi, power saving mode (PSM).**

I. Introduction

TODAY, many individuals use smartphones or other battery-powered mobile devices equipped with Wi-Fi to access the Internet. Because of size and limitations in battery capability, these devices often have strict power constraints. Moreover, we know that the network interface cards (NICs) on these devices use a substantial portion of available energy. Often, a device will be on for long periods with little communication needs. In this case, constantly using a power-intensive Wi-Fi card consumes a large amount of energy.

Further, a wireless connection may be required to perform under various conditions. In particular, a connection on a mobile device may undergo many changes as it moves between environments, drastically changing the total delay experienced by the user in the process. In light of this, we wish to design a system that achieves small delays and low power consumption, while still considering possible changes present in an unstable or mobile environment.

A. Motivation

By default, the wireless NIC operates in constant awake mode (CAM). Here, the wireless card is always in a high-power state and operates with maximum performance but also with minimal energy efficiency. To mitigate the large amount of energy used in CAM, a few different PSMs have been developed and standardized.

1) Static PSM: The first of these is the 802.11 power saving mode (PSM) (termed *Static PSM*) as introduced in [1]. PSM achieves power savings by allowing packets destined for a client to be buffered at the access point (AP) while the device is sleeping. The AP then notifies the client of buffered packets by setting a bit in the traffic indication map, which is included in every beacon. If packets are waiting when the client wakes up to listen to the beacon, it retrieves them by sending a PS-POLL packet to the AP. Additional packets are indicated with a MORE bit in the data frame, and the client continues to retrieve packets until the MORE bit is clear, after which it reenters a sleep state. In this manner, the wireless NIC is only awake when transmitting or receiving data, thus allowing maximum energy efficiency.

However, this scheme suffers from potentially long delays due to the round trip time (RTT) round-up effect. Because a station sleeps immediately after communication, the RTT is effectively rounded up to the nearest beacon interval, even if the actual RTT is much shorter. This effect is further compounded for communications requiring multiple quick responses between the client and a server, which is common in the Transmission Control Protocol (TCP).

2) Dynamic PSM: To partially mitigate these issues with delay, many devices use a variation of PSM called *dynamic PSM* [2]. In this mode, a client will selectively switch from PSM to CAM operation during times of heavy data transfer. Once in CAM mode, the device sends and receives data as normal (without PS-POLL) and remains there until no activity occurs for a time called PSM timeout (specified by the vendor). The device notifies the AP of the switch to CAM (PSM) by sending a NULL data frame with the Power Management bit set to 0 (1).

Generally, Dynamic PSM performs much better than static PSM as it eliminates the overhead of PS-POLL packets and also allows the device to remain awake during short transactions, eliminating any unnecessary delays. However, if staying awake longer, both during an active communication and after data transfer is completed, the device may miss out on sleep opportunities and consume more energy than necessary.

3) PSM-AW: Ideally, a PSM would sleep when no data are being transferred but wake up immediately when incoming data are ready, thereby introducing no additional delay. A client that knows exactly when data will be ready for reception can trivially achieve an ideal PSM. Unfortunately, a client frequently communicates with many remote servers of differing delay, and each server may have greatly varying delay. Thus, we set out to build a practical system that can achieve a performance close to an ideal scheme, while accounting for these varying delays.

Manuscript received September 10, 2013; revised December 3, 2013 and February 14, 2014; accepted March 23, 2014. Date of publication April 22, 2014; date of current version January 13, 2015. The review of this paper was coordinated by Prof. C.-X. Wang.

The authors are with the Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50011 USA (e-mail: bpeck@iastate.edu; daji@iastate.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVT.2014.2319241

In practice, an end user may accept either high energy usage or long delays, but not both. To capture this tradeoff, we introduce a metric called the *PSM penalty* to fully describe the tradeoff between power efficiency and delay. Further, we present PSM-AW, i.e., a PSM with adaptive wake-up. PSM-AW strategically picks a time to wake up for an incoming packet in an attempt to keep the PSM penalty low. In this way, the device can still sleep with as much efficiency as static PSM, while gaining the delay efficiency of dynamic PSM.

B. Contributions

Our main contributions are as follows.

- We define PSM penalty as a metric to capture the tradeoff between power efficiency and delay in PSM schemes.
- We develop a model to objectively compare the user experience of Wi-Fi power saving options by considering both power consumption and response delay.
- We present PSM-AW, which is a scheme that exploits any available knowledge of server delay to achieve maximum user experience.
- We present and prove a bound on the PSM Penalty for the PSM-AW scheme.
- We perform and present extensive simulation and experimental results, which show that PSM-AW can achieve energy savings while still maintaining a low delay.

The remainder of this paper is organized as follows. We describe a unified model of power saving schemes in Section II and evaluate both static and dynamic PSMs in terms of this model. The PSM-AW scheme is introduced in Section III, along with analysis of the scheme. We present a simulation-based performance evaluation in Section IV. In Section V, we describe an implementation-based performance comparison to existing schemes. An outline of related work is given in Section VI and we conclude in Section VII.

II. Generic Power Saving Model

To objectively compare the existing Wi-Fi power saving schemes to any future work, we first need to establish a model as a basis of comparison. To do so, we first discuss the parameters measured in the model in Section II-A and then offer an evaluation of the schemes in terms of our model in Section II-B.

A. Model Description

Ultimately, any power saving scheme has one major set of decisions that need to be made: when to turn the wireless NIC on or off. In our model then, we are concerned with the times that a wireless NIC is turned on (wakes up) or off (sleeps), which are represented by $t_{\text{wake},i}$ and $t_{\text{sleep},i}$, respectively, where $i \in \{1, \ldots, n\}$ is the index of the current sleep cycle. Between cycles, we define $S_i = t_{\text{wake},i} - t_{\text{sleep},i-1}$ to be the amount of time we sleep.

We also seek to measure the delay experienced by the user. As the communication delay between an AP and a remote server is dependent upon many factors, such as the size of the communication and the RTT to the server, we use a quantity termed $T_{\text{SD},j}$ or server delay, to measure the total communication delay. Because server delay is highly variable, we consider each request/response exchange to have distinct $T_{\text{SD},j}$, where $j \in \{1, \ldots, m\}$ is the index of a particular exchange within a given flow. Each request/response pair j is started at $t_{\text{req},j}$ and completed at time $t_{\text{recv},j}$. We are also interested when a reply arrives at the AP (which may be different from when the reply arrives at the client) and signify this time by $t_{\text{rAP},j}$ such that $T_{\text{SD},j} = t_{\text{rAP},j} - t_{\text{req},j}$. The flow is initiated at $t_{\text{req},1}$, and any subsequent requests are made immediately following the reception of the previous packet, such that $t_{\text{req},j+1} = t_{\text{recv},j} + T_{\text{comm,req},j+1}$, where $T_{\text{comm,req},j+1}$ is the time used to send the $(j+1)$th request. We summarize our notations in Table I.

TABLE I
Description of Model Parameters: tRefers to an Absolute Time, TRefers to Duration

Term	Definition
$t_{wake,i}$	i^{th} time that a NIC wakes up
$t_{sleep,i}$	i^{th} time that a NIC is put to sleep
S_i	amount of time spent sleeping before i^{th} cycle
$T_{comm,i}$	amount of time spent communicating packets during i^{th} cycle
$t_{req,j}$	time of j^{th} request
$t_{recv,j}$	time of j^{th} response at client ($t_{req_j+1} = t_{recv,j} + T_{comm,req,j+1}$)
$t_{rAP,j}$	time of j^{th} response at AP
$T_{SD,j}$	j^{th} server delay in a given flow
$T_{comm,recv,k}$	time spent receiving k^{th} response
$T_{comm,req,k}$	time spent requesting k^{th} request

With model parameters defined, we consider how to quantify the performance of a given scheme. We note that the end user of any mobile Wi-Fi device is primarily concerned with two items: power consumption and response delay.

1) Power Consumption: The user is concerned with how much power is consumed by the wireless device. For our purposes, we are concerned primarily with the power consumed by the wireless NIC. Now, as power is consumed quickest when the NIC is in an awake state, our goal is naturally to minimize the amount of time the NIC is awake. Further, since the same amount of time (and energy) is consumed to transmit and receive packets regardless of the PSM scheme used, we don not consider the time the device spends transmitting and receiving packets. Thus, we define the metric A, which is the extra amount of time the NIC is awake for a particular flow. The awake time during a particular wake cycle is $A_i = t_{\text{sleep},i} - t_{\text{wake},i} - T_{\text{comm},i}$, where $T_{\text{comm},i}$ is the time spent transmitting and receiving packets during this cycle. For a given flow then with n sleep cycles, we have

$$A = \sum_{i=1}^{n} A_i = \sum_{i=1}^{n} (t_{\text{sleep},i} - t_{\text{wake},i} - T_{\text{comm},i}). \quad (1)$$

Simply being awake and idle is not the only form of power consumption in Wi-Fi networks. The following also consume energy but are not included in our model for various reasons:

- *Wakeup Costs*: A nontrivial amount of energy is required to wake up the NIC. However, our scheme wakes up at most once per packet request and remains awake until after

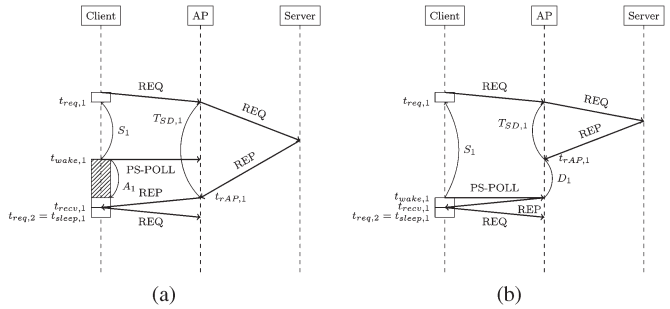


Fig. 1. Sample communication when waking up early and late. The rectangles represent the time when the devices are awake with idle portions shaded. The unshaded parts of the rectangle thus represent any communication delays while receiving and requesting information or $T_{\mathrm{comm,recv},j}$ and $T_{\mathrm{comm,req},j}$. (a) Waking up early with $t_{\mathrm{wake},1} < t_{\mathrm{rAP},1}$. We show A_1 as calculated with (1). $D_1 = 0$. (b) Waking up late with $t_{\mathrm{wake},1} > t_{\mathrm{rAP},1}$. We show D_1 as calculated by (2). $A_1 = 0$.

the reply is received, and as such, it does not introduce any additional wake-ups compared with existing schemes.

- *Computation Costs*: As will be shown in Section III, the proposed PSM-AW scheme requires more computation overhead than other existing schemes. However, the power consumption of these computations is trivial compared with the power consumption of active communication; thus, they are not included in our model.

2) Response Delay: The user is also concerned with the delay in retrieving data once it is requested. For instance, a user viewing a web site will notice the time between following a link on the page to the time the page loads. To this end, we desire to decrease the amount of time that the user waits for the flow to complete. Now, since the server delay is variable, we only seek to measure the amount of time that a user waits for a flow to complete *after* the data are waiting at the AP. Thus, we define the metric D to be the extra amount of delay time imposed on a user by the scheme. The extra delay for a single response D_j is the amount of time between the reception of the response at the AP and the client without considering the processing time, such that $D_j = t_{\mathrm{recv},j} - t_{\mathrm{req},j} - T_{\mathrm{SD},j} - T_{\mathrm{comm,recv},j} = t_{\mathrm{recv},j} - t_{\mathrm{rAP},j} - T_{\mathrm{comm,recv},j}$. For a flow with m requests, we have

$$D = \sum_{j=1}^{m} D_j = \sum_{j=1}^{m} (t_{\mathrm{recv},j} - t_{\mathrm{rAP}_j} - T_{\mathrm{comm,recv},j}). \quad (2)$$

Together, we have two metrics that measure different aspects of user experience: power consumption and response delay. We depict the tradeoff between these two metrics in Fig. 1, such that only one of A or D will be large. In other words, if the device spends less time sleeping, then the difference $t_{\mathrm{sleep},i} - t_{\mathrm{wake},i}$ will often be large, leading to a large A. In this case, there will rarely be large delays between packet reception at the AP and the device such that $t_{\mathrm{recv},j} - t_{\mathrm{rAP},j}$; hence, D will often be small. Conversely, if a device spends more time sleeping, such that A is small, then we may often experience large delays, and $t_{\mathrm{recv},j} - t_{\mathrm{rAP},j}$ and D will remain large. We examine this tradeoff in more detail when we study the potential wake-up intervals in Section III-B.

To capture the tradeoff mathematically, we define a parameter $0 \leq \gamma \leq 1$ that allows us to give one metric priority over the other. Since these two metrics effectively show the penalty imposed on a client for utilizing a PSM scheme, we define the PSM penalty X for a flow as follows:

$$X = \gamma D + (1 - \gamma) A. \quad (3)$$

This formula clearly captures the essence of PSMs: an attempt to minimize the amount of time a device is awake while also minimizing the delay experienced by the user. The user can then select a value of γ that corresponds to their situational need—a high value when performance is desired, and a low value when power efficiency is desired.

B. Model Evaluation

In Fig. 2, we illustrate the given parameters across multiple possible schemes by considering a simple TCP transaction consisting of a SYN/ACK exchange followed by a request for data that contains two data packets. We neglect T_{comm} in the figure for clarity.

1) CAM Client: We first consider a client that does not utilize any PSM scheme, but rather remains in CAM at all times, as shown in Fig. 2(a). We see that, since the client is always awake, there are no wake-up or sleep times. Moreover, packets received at the AP are able to be immediately delivered to the client.

2) Static PSM Client: We next consider a device using static PSM as shown in Fig. 2(b). Here, the device goes to sleep immediately after sending a packet and does not wake up until the next beacon. At a glance, we see that static PSM spends very little time awake, but instead, it spends most of the time asleep. However, since the client only wakes up at beacons, there are significant extra delays.

3) Dynamic PSM Client: We also consider the same communication but with a dynamic PSM client, as shown in Fig. 2(c). In this case, the server responds before the timeout value expires; thus, the device never sleeps until well after the entire communication is completed. This leads to delay times that are identical to the CAM case but with much less power efficiency than static PSM.

4) Ideal PSM Client: Finally, we consider an ideal client as shown in Fig. 2(d), which sleeps only when there are no data to be transmitted but is awake as soon as data are ready to not introduce any delay.

III. Proposed Power Saving Mode With Adaptive Wake-Up

A. Overview and Problem Statement

From the preceding schemes, we see that there is a distinct tradeoff between power savings and delay in PSM schemes. We also observed that we can maximize the total user experience if the client wakes up precisely when a packet is ready at the AP. Thus, our scheme has a simple primary goal. Given the history of previous server delays, we seek to pick the next wake-up time that minimizes PSM penalty. More specifically, we consider previous server delays $T_{\mathrm{SD},1}, \ldots, T_{\mathrm{SD},k}$ along with previous

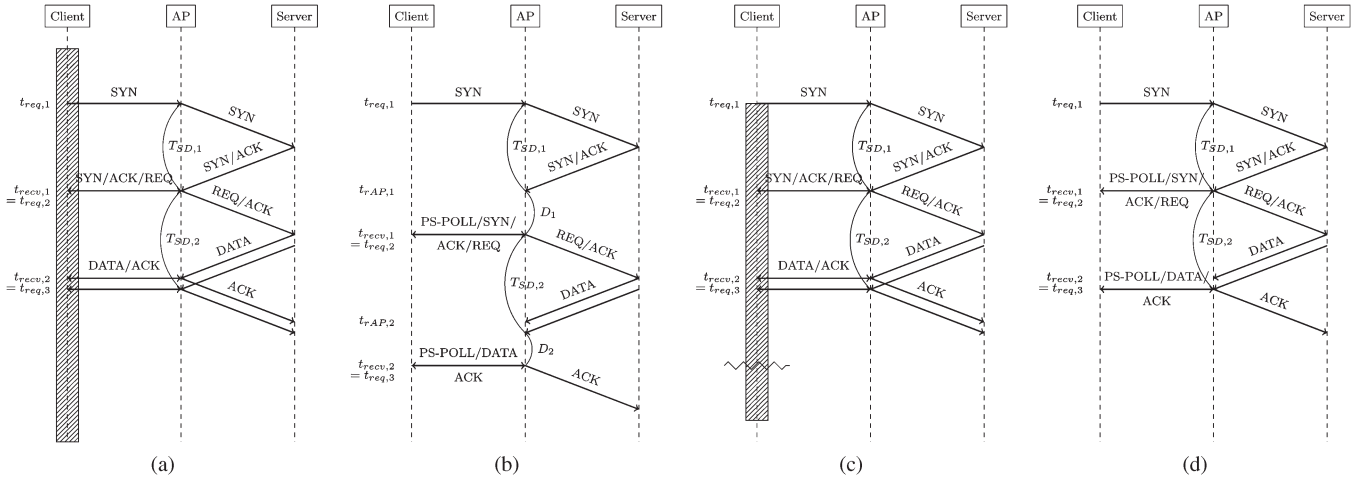


Fig. 2. TCP communication between various clients and a server with $T_{\mathrm{comm,recv},j}$ and $T_{\mathrm{comm,req},j}$ (not shown for clarity). Rectangles represent the time the device is awake with idle portions shaded. Also shown is extra delay, which is the time between data availability at the AP and reception at the client. (a) CAM client: the device receives information as soon as it is available at the AP, where there is no extra delay. (b) Static PSM client: the wake-up times are determined by beacons from the AP, which happen periodically every 100 ms. (c) Dynamic PSM client: delay is removed compared with the static PSM case, but the device sleeps less. (d) Ideal PSM client: there is no extra delay and no extra awake time.

extra awake and delay times A_k and D_k, respectively. We define a per-response PSM penalty metric $X_k = \gamma D_k + (1-\gamma)A_k$, and we then pick $t_{\mathrm{wake},k+1}$ attempting to minimize X_{k+1}. If we knew beforehand the exact server delay of the next request, this minimization would be trivial, and we would select $t_{\mathrm{wake},k+1} = t_{\mathrm{sleep},k} + T_{\mathrm{SD},k+1}$.

Unfortunately, server delays are uncontrollable and subject to much variance; thus, we cannot directly minimize X_{k+1} for arbitrary $T_{\mathrm{SD},k+1}$. Instead, we provide a connection-dependent upper bound for X_{k+1} as follows:

$$X_{k+1} \leq \rho X_k + \Delta T_{\mathrm{SD},k} \tag{4}$$

where $\Delta T_{\mathrm{SD},k} = |T_{\mathrm{SD},k+1} - T_{\mathrm{SD},k}|$ is the change in server delay, and $0 < \rho \leq 1$ is a bound parameter for merging speed. In other words, we provide an upper bound on the next per-response PSM penalty based on the previous PSM penalty and the change in server delay. Consider the constant $c = \mathbb{E}(\Delta T_{\mathrm{SD},k})$ that measures the expected change in server delay. We can use this value to calculate the expected PSM Penalty. By applying the linearity of expectation to (4), we have $\mathbb{E}(X_k) \leq \rho \mathbb{E}(X_{k-1}) + c$. Continuing recursively, we find $\mathbb{E}(X_k) \leq \rho^k \mathbb{E}(X_0) + c(\rho^{k-1} + \rho^{k-2} + \ldots + 1)$. For a long flow, we have $\lim_{k\to\infty} \rho^k = 0$ and $\lim_{k\to\infty}(\rho^{k-1} + \rho^{k-2} + \ldots + 1) = 1/(1-\rho)$. Thus, we find

$$\lim_{k\to\infty} \mathbb{E}(X_k) \leq \frac{c}{1-\rho}. \tag{5}$$

Over time, we can asymptotically converge to a connection-dependent PSM penalty based on the expected change in server delay c. If the connection is relatively stable, then an individual $\Delta T_{\mathrm{SD},k}$ is small and c is small, and we can have relatively small PSM penalties. However, if the connection is unstable with uncontrollable variance in $\Delta T_{\mathrm{SD},k}$ and c is large, then we may have large PSM penalties but which are still bounded under $c/(1-\rho)$. We will discuss the choice of ρ more fully in Sections III-D and IV-B3.

Our scheme PSM-AW performs the following.

1) A device sends a request to the server.
2) The device uses previous exchange to find a range of valid values for $t_{\mathrm{wake},k+1}$ that guarantee the bound in (4) (Range will be calculated in Section III-B).
3) The device uses a window of recent history to select optimal $t_{\mathrm{wake},k+1}$ from the range of acceptable values (Section III-C).
4) The device wakes up at time $t_{\mathrm{wake},k+1}$ and switches to CAM to prepare for packet reception.
5) If more data are ready, we continue with Step 1.

B. Adaptive Wake-Up Interval

We now seek to analyze the given bound and determine a range of valid wake-up times. We add to our list of definitions S_k, the *sleep time*, which is the duration that a device will sleep before waking up to attempt to retrieve a packet. Since there will always be one request/response pair per awake/sleep cycle, our indexes i and j line up, and we have the following:

$$S_k = t_{\mathrm{wake},k} - t_{\mathrm{sleep},k-1}. \tag{6}$$

Moreover, since we sleep immediately upon sending, we observe $t_{\mathrm{sleep},k} = t_{\mathrm{recv},k} + T_{\mathrm{comm,req},k+1} = t_{\mathrm{req},k+1}$.

To determine valid wake-up times that satisfy the bound in (4), we observe that we may have to act differently depending on the result of the most recent request. Thus, we consider two possible scenarios where: 1) the device sleeps too long and wakes up "late" or 2) the device does not sleep long enough and wakes up "early."

1) Case Analysis:

a) Woke up "Late": In the first case, the device woke up "late" for the kth response because the reply arrived before the device woke up such that $T_{\mathrm{SD},k} \leq S_k$. Since the packet is ready immediately, we observe $D_k = t_{\mathrm{recv},k} - t_{\mathrm{req},k} - T_{\mathrm{SD},k} - T_{\mathrm{comm,recv,k}} = t_{\mathrm{wake},k} - t_{\mathrm{sleep},k-1} - T_{\mathrm{SD},k} = S_k - T_{\mathrm{SD},k}$. Further, since we can sleep immediately, we have $A_k = 0$.

Thus, when $T_{\mathrm{SD},k} \leq S_k$, we have the following relations:

$$\begin{aligned} D_k &= S_k - T_{\mathrm{SD},k} \\ A_k &= 0. \end{aligned}$$

b) Woke up "Early": In the second case, the device woke up "early" for the kth response because the reply arrived after the device woke up such that $T_{\mathrm{SD},k} > S_k$. Since the device stayed awake and was ready to receive the packet when it arrived, we have $D_k = 0$. Moreover, we have $A_k = t_{\mathrm{sleep},k} - t_{\mathrm{wake},k} - T_{\mathrm{comm},k} = T_{\mathrm{SD},k} - S_k$.

Thus, when $T_{\mathrm{SD},k} > S_k$, we have the following relations:

$$\begin{aligned} D_k &= 0 \\ A_k &= T_{\mathrm{SD},k} - S_k. \end{aligned}$$

2) Subcase Analysis: Thus, we consider the four different combinations of waking up early and late that can occur in the kth and the $(k+1)$th packet.

a) Late, Late: Here, $T_{\mathrm{SD},k} \leq S_k$, and $T_{\mathrm{SD},k+1} \leq S_{k+1}$. In this subcase, $A_k = A_{k+1} = 0$, $D_k = S_k - T_{\mathrm{SD},k}$, and $D_{k+1} = S_{k+1} - T_{\mathrm{SD},k+1}$. We could then rewrite (4) as follows:

$$\begin{aligned} & X_{k+1} \leq \rho X_k + \Delta T_{\mathrm{SD},k} \\ \Leftrightarrow \quad & \gamma D_{k+1} + (1-\gamma) A_{k+1} \leq \rho\gamma D_k + \rho(1-\gamma) A_k \\ & \qquad + \Delta T_{\mathrm{SD},k} \\ \Leftrightarrow \quad & D_{k+1} \leq \rho D_k + \frac{1}{\gamma} \Delta T_{\mathrm{SD},k}. \end{aligned} \tag{7}$$

We first recognize that satisfying $D_{k+1} \leq \rho D_k + \Delta T_{\mathrm{SD},k}$ will satisfy (7). Because the device wakes up late for the $(k+1)$th packet, the worst case for D_{k+1} occurs when the server delay decreases drastically, such that $\Delta T_{\mathrm{SD},k} = T_{\mathrm{SD},k} - T_{\mathrm{SD},k+1}$. We consider the following calculations to determine the range of acceptable sleep times:

$$\begin{aligned} & D_{k+1} \leq \rho D_k + \Delta T_{\mathrm{SD},k} \\ \Leftrightarrow \quad & S_{k+1} - T_{SD,k+1} \leq \rho D_k + T_{\mathrm{SD},k} - T_{SD,k+1} \\ \Leftrightarrow \quad & S_{k+1} \leq \rho D_k + T_{\mathrm{SD},k} \\ \Leftrightarrow \quad & S_{k+1} \leq \rho D_k + S_k - D_k \\ \Leftrightarrow \quad & S_{k+1} \leq S_k - (1-\rho) D_k. \end{aligned}$$

Thus, to maintain the bound in (4), we can choose

$$S_{k+1} \leq S_k - (1-\rho) D_k. \tag{8}$$

b) Late, Early: Here, $T_{\mathrm{SD},k} \leq S_k$, and $T_{\mathrm{SD},k+1} > S_{k+1}$. In this subcase, we have $A_k = D_{k+1} = 0$, $D_k = S_k - T_{SD_k}$, and $A_{k+1} = T_{\mathrm{SD},k+1} - S_{k+1}$. We can then rewrite (4) as follows:

$$\begin{aligned} & X_{k+1} \leq \rho X_k + \Delta T_{\mathrm{SD},k} \\ \Leftrightarrow \quad & \gamma D_{k+1} + (1-\gamma) A_{k+1} \leq \rho\gamma D_k + \rho(1-\gamma) A_k + \Delta T_{\mathrm{SD},k} \\ \Leftrightarrow \quad & A_{k+1} \leq \frac{\rho\gamma}{1-\gamma} D_k + \frac{1}{1-\gamma} \Delta T_{\mathrm{SD},k}. \end{aligned} \tag{9}$$

Because the device wakes up early for the $(k+1)$th packet, the worst case for this scenario occurs when the server delay increases drastically and $\Delta T_{\mathrm{SD},k} = T_{\mathrm{SD},k+1} - T_{\mathrm{SD},k}$. Continuing algebraically as before, we find that we can choose S_{k+1} to maintain the bound in (4) according to

$$S_{k+1} \geq S_k - \left(1 + \frac{\rho\gamma}{1-\gamma}\right) D_k. \tag{10}$$

Combining the two inequalities from (8) and (10), we find a range to choose S_{k+1} if we woke up late for the kth packet, i.e.,

$$S_k - \left(1 + \frac{\rho\gamma}{1-\gamma}\right) D_k \leq S_{k+1} \leq S_k - (1-\rho) D_k. \tag{11}$$

c) Early, Late: Here, $T_{\mathrm{SD},k} > S_k$ and $T_{\mathrm{SD},k+1} \leq S_{k+1}$. In this case, $D_k = A_{k+1} = 0$, $A_k = T_{\mathrm{SD},k} - S_k$, and $D_{k+1} = S_{k+1} - T_{\mathrm{SD},k+1}$. We can rewrite (4) as

$$\begin{aligned} & X_{k+1} \leq \rho X_k + \Delta T_{\mathrm{SD},k} \\ \Leftrightarrow \quad & \gamma D_{k+1} + (1-\gamma) A_{k+1} \leq \rho\gamma D_k + \rho(1-\gamma) A_k + \Delta T_{\mathrm{SD},k} \\ \Leftrightarrow \quad & D_{k+1} \leq \frac{\rho(1-\gamma)}{\gamma} A_k + \frac{1}{\gamma} \Delta T_{\mathrm{SD},k}. \end{aligned} \tag{12}$$

Because the device wakes up late for the $(k+1)$th packet, the worst case for this subcase occurs with a drastic decrease in server delay. Solving for S_{k+1} as before, we can maintain the bound in (4) by choosing S_{k+1} according to

$$S_{k+1} \leq S_k + \left(1 + \frac{\rho(1-\gamma)}{\gamma}\right) A_k. \tag{13}$$

d) Early, Early: Here, $T_{\mathrm{SD},k} > S_k$, and $T_{\mathrm{SD},k+1} > S_{k+1}$. In this subcase, $D_k = D_{k+1} = 0$, $A_k = T_{\mathrm{SD},k} - S_k$, and $A_{k+1} = T_{\mathrm{SD},k+1} - S_{k+1}$ such that (4) becomes

$$\begin{aligned} & X_{k+1} \leq \rho X_k + \Delta T_{\mathrm{SD},k} \\ \Leftrightarrow \quad & \gamma D_{k+1} + (1-\gamma) A_{k+1} \leq \rho\gamma D_k + \rho(1-\gamma) A_k + \Delta T_{\mathrm{SD},k} \\ \Leftrightarrow \quad & A_{k+1} \leq \rho A_k + \frac{1}{1-\gamma} \Delta T_{\mathrm{SD},k}. \end{aligned} \tag{14}$$

Because the device wakes up early for the $(k+1)$th packet, the worst case for this subcase is a drastic increase in server delay. Again, solving for S_{k+1}, we satisfy the bound in (4) by choosing S_{k+1} according to

$$S_{k+1} \geq S_k + (1-\rho) A_k. \tag{15}$$

Combining the two inequalities from (13) and (15), we find a range to choose S_{k+1} if we woke up early for the kth packet, i.e.,

$$S_k + (1-\rho) A_k \leq S_{k+1} \leq S_k + \left(1 + \frac{\rho(1-\gamma)}{\gamma}\right) A_k. \tag{16}$$

3) Summary: Thus, we have two ranges from which to choose S_{k+1} that satisfies our bound in (4).

- Woke up late for the kth packet, choose from

$$S_k - \left(1 + \frac{\rho\gamma}{1-\gamma}\right) D_k \leq S_{k+1} \leq S_k - (1-\rho) D_k.$$

- Woke up early for the kth packet, choose from

$$S_k + (1-\rho) A_k \leq S_{k+1} \leq S_k + \left(1 + \frac{\rho(1-\gamma)}{\gamma}\right) A_k.$$

C. History-Based Fine-Tuning

Once we are given the range of acceptable values for S_{k+1} shown earlier, we need to determine the optimum value for

the sleep time. We consider a set Ω of all possible valid sleep times governed by the equations in Section III-B3. We also consider a window w of the most recent server delays $T = \{T_{\mathrm{SD},k-w+1}, \ldots, T_{\mathrm{SD},k}\}$. Once these sets are known, we desire to choose an element from Ω that minimizes the expected PSM penalty. Thus, we choose

$$S_{k+1} = \arg\min_{s\in\Omega} \left(\mathbb{E}[X_{k+1}]\right). \tag{17}$$

Now, $\mathbb{E}[X_{k+1}] = \gamma\mathbb{E}[D_{k+1}] + (1-\gamma)\mathbb{E}[A_{k+1}]$ by linearity of expectation. The next D_{k+1} is either 0 or $s - T_{\mathrm{SD},t}$ for a given $s \in \Omega$ and $k - w + 1 \leq t \leq k$. Assuming that the previous w values are representative of reasonable future values, they have an equal chance of occurring in the succeeding request. Thus, to calculate $\mathbb{E}[D_{k+1}]$, we add up the calculated value of D_{k+1} for the previous w observed values and divide by w, such that $\mathbb{E}[D_{k+1}] = \sum_{t=k-w+1}^{k} \max(0, s - T_{\mathrm{SD},t})/w$ for a given $s \in \Omega$. Similarly, we can calculate $\mathbb{E}[A_{k+1}] = \sum_{t=k-w+1}^{k} \max(0, T_{\mathrm{SD},t} - s)/w$. Given these values, we can then choose the $s \in \Omega$ with the smallest $\mathbb{E}[X_{k+1}]$ to be S_{k+1}.

Thus, S_{k+1} represents the amount of time that we choose to sleep for the following sleep cycle, such that we choose S_{k+1} to be the value that minimizes the expected value of the PSM Penalty. Combining our choice of S_{k+1} in (17) with the definition of S_k given in (6), we have the relation $t_{\mathrm{wake},k+1} = t_{\mathrm{sleep},k} + S_{k+1}$. From this perspective, our algorithm determines the next wake-up time $(t_{\mathrm{wake},k+1})$ so that the expected PSM penalty is minimized.

To determine the window size w, we consider the effects of different traffic types. For relatively stable connections, we want a relatively large window to allow fine-tuning to pick an appropriate wake-up time. If the window is too large, it may take a long time to fill up the window, resulting in stale data; therefore, we set a maximum window size of 30. On the other hand, unstable connections should not use old history data; thus, we want a small window size. Thus, we use the following formula to determine w:

$$w = \max\left(1, 30 - \left\lfloor \frac{\Delta T_{\mathrm{SD},k}}{c} \right\rfloor\right) \tag{18}$$

where $c = \sum_{k=1}^{w-1} |T_{\mathrm{SD},k} - T_{\mathrm{SD},k+1}|/(w-1)$ is the mean difference of successive server delays. We thus choose a window size between 1 and 30, such that smaller windows are used when the most recent change is very large compared with the mean difference. Full details on the implementation of our algorithm are given in Section III-G.

D. Determining the Adaptive ρ

We consider the effect of the merging speed parameter ρ by considering two general cases of potential traffic. In the first case, the connection is relatively stable, and traffic history is a good indicator of the future, such that we have a very low mean difference c. In this instance, a low ρ such as $\rho = 0$ would collapse the bound in (4) to $X_{k+1} \leq \Delta T_{\mathrm{SD},k}$. As a result, the range of acceptable sleep times condenses to a single value and we eliminate the possible benefits from history-based fine-tuning. A higher value of ρ closer to $\rho = 1$, however, would enlarge the bound to $X_{k+1} \leq X_k + \Delta T_{\mathrm{SD},k}$ and produce a much larger range of acceptable sleep times, allowing history-based fine-tuning to have a greater effect. Since the traffic is relatively stable, this situation is ideal, and we prefer a large ρ. We depict the effect of this situation in Fig. 3.

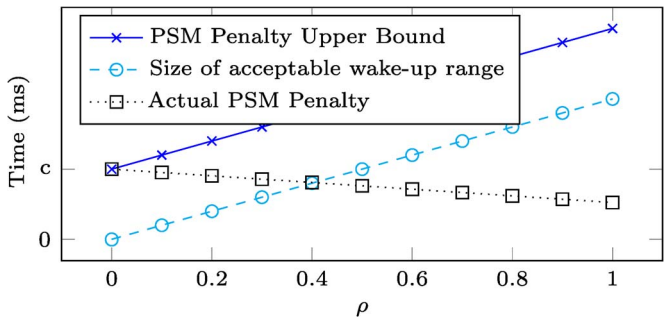


Fig. 3. Conceptual depiction of the effect of ρ for a relatively stable connection. A small ρ yields a higher PSM penalty but with a smaller upper bound. A large ρ yields a smaller PSM penalty but with a larger upper bound.

We must also consider a second general case of unstable traffic such that history is not a good indicator of the future. In this scenario, we have a large mean difference c. Since history does not indicate future performance, history-based fine-tuning would skew the results and be unhelpful. However, a low value of ρ decreases the bound on PSM penalty, which is ideal.

We now consider the possibility of using a value of ρ obtained through a statistical model. Depending on the model chosen for $T_{\mathrm{SD},k}$, we could solve for the mean difference c and in turn choose an appropriate ρ. However, real conditions do not conform well to any specific model; thus, both scenarios aforementioned occur in practice. Specifically, any given connection is likely to see periods of extended stability, followed by short periods of erratic behavior. Thus, the value of ρ ought to adapt to the observed traffic. We desire a value such that $0 < \rho \leq 1$ with ρ dependent on (decreasing monotonically with) $c = \sum_{k=1}^{w-1} |T_{\mathrm{SD},k} - T_{\mathrm{SD},k+1}|/(w-1)$. To do so, we first find an upper bound on c in Lemma 1.

Lemma 1: For arbitrary a_i and $c = \sum_{i=1}^{w-1} |a_i - a_{i+1}|/(w-1)$ across w observed values, $0 \leq c < 2\mu(w/(w-1))$.

Proof: We formally define mean $\mu = \sum_{i=1}^{w} a_i/w$ and mean difference $c = \sum_{i=1}^{w-1} |a_i - a_{i+1}|/(w-1)$, where a_i is a generic value to represent $T_{\mathrm{SD},k}$. Now, because $\sum_{i=1}^{w}(\mu - a_i) = 0$ and $a_i > 0$, we have $\sum_{i=1}^{w} |\mu - a_i| < w\mu$.

We can now show the following:

$$\begin{aligned} 0 \leq c &= \sum_{i=1}^{w-1} \frac{|a_i - a_{i+1}|}{w-1} = \sum_{i=1}^{w-1} \frac{|a_i - \mu + \mu - a_{i+1}|}{w-1} \\ &\leq \sum_{i=1}^{w-1} \frac{|a_i - \mu|}{w-1} + \sum_{i=1}^{w-1} \frac{|\mu - a_{i+1}|}{\mathrm{w}-1} \\ &\leq \sum_{i=1}^{w} \frac{|a_i - \mu|}{w-1} + \sum_{i=1}^{w} \frac{|\mu - a_i|}{w-1} < 2\mu\frac{w}{w-1}. \end{aligned}$$

Thus, $0 \leq c < 2\mu(w/w-1)$. □

Since Lemma 1 shows that $0 \leq c < 2\mu(w/w-1)$, we can use the following relation to choose ρ such that $0 < \rho \leq 1$ and ρ decreases monotonically with c:

$$\rho = 1 - \frac{c}{2\mu\frac{w}{w-1}} \tag{19}$$

where w is the window size of the observed times, and c and μ are the mean difference and mean of the server delay times, respectively. We consider the effects of this adaptive ρ in Section IV-B3 and then use it in our experiments.

E. Choosing γ

Another important parameter in our scheme is γ, which directly illustrates the trade off in choosing between energy efficiency and delay performance. We recall that the PSM penalty is calculated with $X = \gamma D + (1-\gamma)A$. Thus, a low γ heavily penalizes awake time, and we then spend more time sleeping. Conversely, a high γ heavily penalizes delay time; therefore, we spend less time sleeping so that we wake up early enough to avoid any delay.

Since the choice of γ directly influences how much time we spend sleeping, we may choose γ based on the actual power consumption of our NIC. For instance, we may choose a low γ to maximize sleeping if our device has a NIC with relatively large power consumption, and a high γ to maximize delay performance if our NIC has low power consumption, and we are not as concerned with energy efficiency.

However, the raw power consumption does not tell the whole story. The user may care about battery lifetime as well. For instance, if the device's battery is low (or if the device has a full battery but the user will be unable to charge the device for a long time), the user may want to conserve energy at all costs and select a low γ.

As a result of these potentially conflicting inputs to determine the choice of γ, we leave the choice to the user and simply provide a range of options. In Section IV-B1, we look at the effects of γ on awake and delay times. For our experiments in Section V, we choose $\gamma = 0.7$ to favor a reduced delay.

F. Multiple Flows

In practice, a single device is very rarely occupied with only one flow. Rather, many flows may be in operation over a given time interval. Given this fact, our scheme intelligently handles multiple flows with the following simple heuristics.

- *Opportunistic Retrieval*: While waiting on a packet for Flow A, a packet for Flow B may arrive, which we may not have been expected until later. In this case, PSM-AW simply receives the packet.
- *Delayed Sleeping*: After we finish processing a packet for Flow A, we may predict a packet for Flow B to arrive shortly after. Instead of sleeping for only short duration, PSM-AW opts to stay awake to wait for Flow B. The threshold time for deciding whether to sleep is calculated by comparing the energy saved by sleeping versus the energy cost to wake up. In our experiments, we use a Linksys WPC55AG NIC and select a threshold of 7 ms, such that we will stay awake for Flow B if the packet is predicted to arrive in the next 7 ms.

G. Summary

Overall, our scheme works in a simple two-step process. First, we determine a range of wake-up times that satisfy the performance bound given in (4). Then, we select the time within this range that minimizes the expected PSM penalty. Both of these steps occur at the client; thus, no AP modification is necessary. More specifically, we choose S_{k+1} according to Algorithm 1.

Algorithm 1 Calculate $T_{\mathrm{SD},k+1}$

▷ Calculate the acceptable wakeup range
if $T_{\mathrm{SD},k} \leq S_k$ **do**
  $l \Leftarrow S_k - (1 + (\rho\gamma/1-\gamma))D_k$
  $h \Leftarrow S_k - (1-\rho)D_k$
else
  $l \Leftarrow S_k + (1-\rho)A_k$
  $h \Leftarrow S_k + (1 + (\rho(1-\gamma)/\gamma))A_k$
end if
▷ Calculate c based on up to last w values
$c \Leftarrow (\sum_{k=1}^{w-1} |T_{\mathrm{SD},k} - T_{\mathrm{SD},k+1}|)/(w-1)$
▷ Calculate μ based on up to last w values
$\mu \Leftarrow (\sum_{k=1}^{w} T_{\mathrm{SD},k}/w)$
▷ Fine Tuning: Choose next sleep time that minimizes expected PSM Penalty
$\min \Leftarrow$ MAX_INT
for $s = l : h : 1$ ms **do**
  EX $\Leftarrow \mathbb{E}[X_{k+1}]$
  if EX $\leq \min$ **then**
    $S_{k+1} \Leftarrow s$
    $\min \Leftarrow$ EX
  end if
end for
▷ Update w—Slowly grow history
$w \Leftarrow \min(w+1, \max(1, 30 - (\Delta T_{\mathrm{SD},k}/c)))$

As we can see, the algorithm is fairly efficient. We can compute the lower and upper bounds on S_{k+1} in constant time, and c and μ in $\mathcal{O}(w)$ time. Since w is small, this also is quick. The most complex part of the algorithm is searching through possible s. Each computation of $\mathbb{E}[X_{k+1}]$ is also $\mathcal{O}(w)$, but we may have to do as many as $h - l + 1$ such computations. However, in practice, we can use simple heuristics (such as noticing that the values of $\mathbb{E}[X_{k+1}]$ are concave up) to limit the search space, thus making the entire algorithm efficient.

IV. Simulation Studies

We now seek to validate the previous analysis using exhaustive simulation results.

A. Simulation Setup

We perform our simulations in a custom discrete event simulator built with MATLAB. The simulator is capable of using sample server delays from a collected trace or generating its own sample server delays. We then simulate the sending

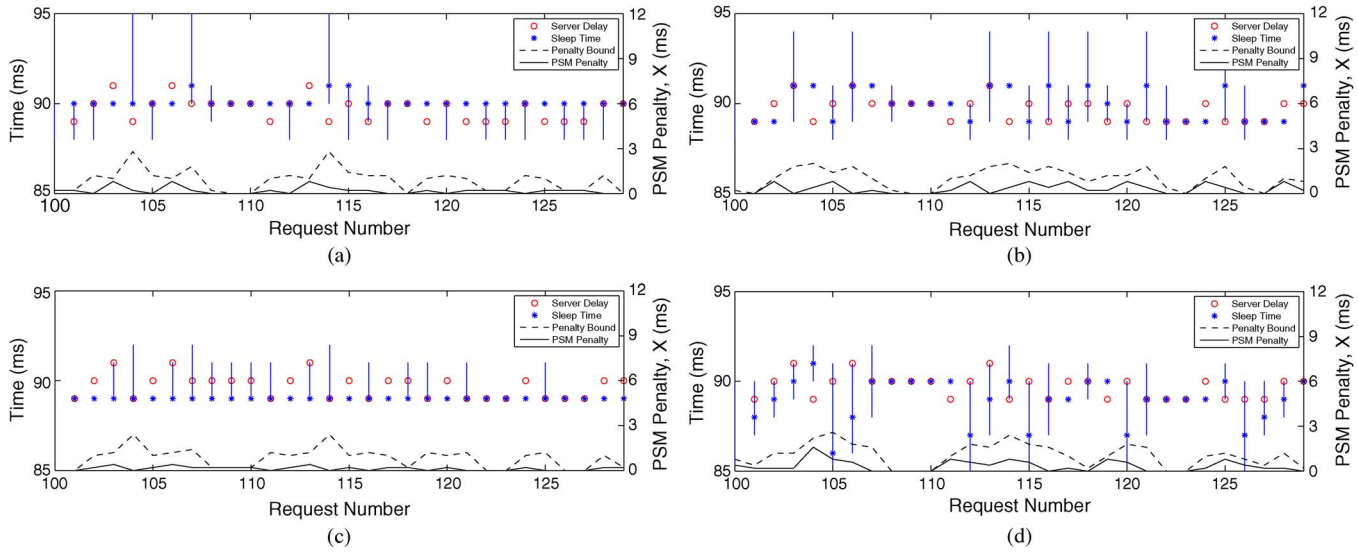


Fig. 4. Trace of 30 requests to Twitter. The bars represent the range of valid wake-up times to ensure the bound from (4), which is shown by the dashed lines. (a) $\gamma = 0.2$, with fine-tuning. (b) $\gamma = 0.2$, no fine-tuning (midpoint). (c) $\gamma = 0.8$, with fine-tuning. (d) $\gamma = 0.8$, no fine-tuning (midpoint).

and receiving of messages (using packet transmission times according to $\mathcal{N}$ (1 ms, 0.2 ms), with the simulator tracking the awake and sleep patterns for the device. From here, we calculate the total time spent awake and asleep, additional delay due to the PSM scheme, and the total PSM penalty.

For the majority of the simulations, we use a trace of 1000 requests gathered from a session with a Twitter application. When an average of many traces with varying parameters is required, we construct a traffic model with parameters similar to this trace, which is a normal distribution defined by $\mathcal{N}$ (70 ms, 2 ms). Unless specified otherwise, we use a traffic model consisting of an initial SYN packet followed by one or more requests for data, similar to the communication shown in Fig. 2.

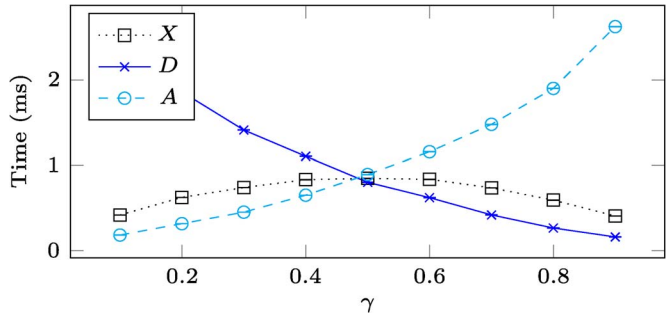


Fig. 5. X, A, and D for 50 requests of $\mathcal{N}$ (70, 2 ms) traffic. Each data point is the average of 100 simulations and shows the average value of the respective metric.

B. Observing Scheme Parameters

We use our simulator to observe the effect of our scheme parameters and to choose optimal values when appropriate.

1) The Effects of γ: We look closer at a segment of the Twitter trace to determine the effect of γ. In this simulation, we consider a subset of 30 requests from the Twitter trace and show the server delay, sleep time, and PSM penalty for each request in Fig. 4. As expected, a large value of γ causes the device to consistently wake up slightly early to avoid delay, as shown in Fig. 4(c) and (d). Conversely, a small value of γ results in waking up later, as shown in Fig. 4(a) and (b). This effect can be substantial. For example, it is shown in Fig. 4(c) that the algorithm always chooses the smallest of the possible valid sleep times, resulting in a constant actual sleep time.

Additionally, we look at the effect of γ on the actual awake and delay times. To do so, we generate 50 requests from a normal distribution $\mathcal{N}$ (70 ms, 2 ms). We then repeat the simulation 100 times for a range of γ values and plot the results in Fig. 5. As expected, a low γ results in priority given to minimizing A; thus, D remains high. Similarly, a high γ results in priority given to minimizing D; thus, A remains high.

To understand the effect of γ in real-world conditions, we consider values for energy consumption from [3], where idle listening uses about 520 mW and light sleeping uses only 120 mW. According to the data for Fig. 5, using $\gamma = 0.8$ keeps the device awake for an extra 1.9 ms per request, and using $\gamma = 0.2$ keeps the device awake for an extra 0.32 ms per request. Thus, choosing a lower gamma saves us up to $(1.9{-}0.32)$ ms $\cdot$ $(520{-}120)$ mW $= 0.632$ mJ per request. This savings can add up quickly when many requests are involved.

2) The Effects of Fine-Tuning: We also seek to determine the effect of fine-tuning on our strategy. We consider a version of the scheme that simply uses the midpoint (calculated as $\lfloor (h-l)/2 \rfloor$, where h and l are the upper and lower bounds) of the acceptable range for each request as the wake-up time, rather than the fine-tuning process described in Section III-C. Without fine-tuning Fig. 4(b) and (d), the midpoint provides a reasonable strategy but not as good as using fine-tuning [see Fig. 4(a) and (c)]. On average, using fine-tuning helps lower the average PSM Penalty, as shown in Table II. If we repeat this experiment hundreds of times, we find that fine-tuning helps lower the PSM penalty by up to 34% on average, with potentially even more improvement for unstable traffic.

3) The Effects of ρ: We now verify the claims from Section III-D regarding the effect of ρ on performance by considering 50-request traces with the server delays drawn from

TABLE II
COMPARISON OF AVERAGE X_k FOR DIFFERENT PSM-AW VERSIONS BASED ON A TRACE OF 30 REQUESTS WITH $\mathcal{N}$ (70 ms, 2 ms) TRAFFIC, AS SHOWN IN Fig. 4

Average X_k (ms)	$\gamma = 0.2$	$\gamma = 0.8$
Fine-Tuning	0.19	0.13
No Fine-Tuning	0.31	0.35

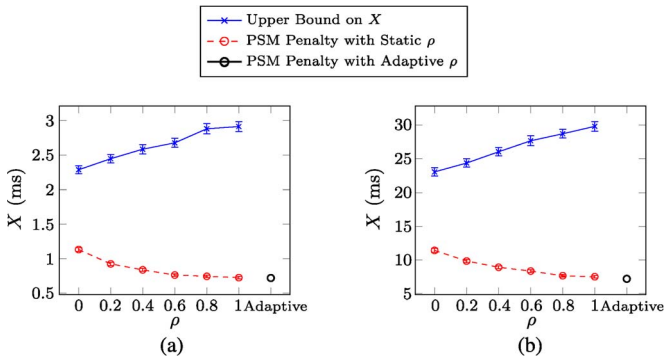


Fig. 6. PSM penalty and upper bound for simulation of 50 requests with indicated traffic pattern for varying ρ, with $\gamma = 0.7$. For normal distributions, increasing ρ increases the upper bound but will allow fine-tuning to take greater effect, resulting in a lower actual penalty. The final circle (labeled Adaptive) represents the PSM penalty obtained with an adaptive ρ. (a) $\mathcal{N}$ (70 ms, 2 ms). (b) $\mathcal{N}$ (70 ms, 20 ms).

a normal distribution in Fig. 6. In Fig. 6(a) and (b), we see that larger values of ρ result in a higher upper bound, as (5) suggests. However, because a high value of ρ also allows us to use a larger range of acceptable wake-up times, fine-tuning can strategically select a time closest to the mean; thus, our actual performance is better. Finally, we show the result from the same simulation with an adaptive ρ from Section III-D as a single circle following the other data, which performs as well as or better than the static values of ρ. In practice, the adaptive value of ρ averages 0.98 when $\sigma = 2$ ms and 0.84 when $\sigma = 20$ ms.

4) Longer Example: Finally, we observe the performance of the selected parameters for the full Twitter trace in Fig. 7. Here, we use $\gamma = 0.7$ to place moderate importance on the delay. We also use fine-tuning and an adaptive ρ. We notice that the trace is relatively stable, except for a few minor disruptions. PSM-AW accurately utilizes fine-tuning during the stable periods to select the best wake-up time. During the disruptions, PSM-AW adjusts the window size w and the merging parameter ρ to bound the PSM penalty.

V. PERFORMANCE EVALUATION

A. Experimental Setup

To fully compare the performance of PSM-AW to other schemes, we perform a series of experiments. To perform these experiments, we create a working implementation of PSM-AW by modifying the open-source MadWifi device driver. This custom driver can be configured to use static PSM, Dynamic PSM, or PSM-AW as its PSM. We then use this driver on a Dell Latitude E5400 laptop equipped with a Linksys WPC55AG NIC. For parameters, we choose $\gamma = 0.7$ to slightly favor reduced delay, fine-tuning, and an adaptive ρ. When we need to measure performance to a server with specific delay parameters as in Section V-C2, we set up another laptop with an unmodified driver and the ability to introduce additional delay.

B. Implementation Details

To implement PSM-AW, we modify the following modules of the MadWifi device driver, as shown in Fig. 8.

- *ieee80211_var*: In MadWifi, this module is responsible for keeping track of various data related to each device. For PSM-AW, we modify it to keep track of all the current possible destinations and ongoing flows that a client might be using. Specifically, it keeps track of previous server delays (up to the window size w), the most recent A_k and D_k values, and other flow state information.
- *ieee80211_input*: The input module handles all incoming messages to a device. We modify it to calculate the server delay of the previous request.
- *ieee80211_output*: In MadWifi, the output module handles all outgoing packets. The PSM-AW modified version performs the bulk of the work for our scheme and is responsible for calculating the bound of acceptable wake-up times and for performing history-based fine-tuning to choose the best wake-up time that satisfies the PSM-AW bound. The results of the calculations are stored in the flow data.
- *ieee80211_power*: This module handles the power state of the device by either sleeping or waking up the device and informing the AP of any changes as necessary. We modify the module for PSM-AW by monitoring the flow data to see when power status changes are necessary. When there is no network activity, this module aggregates information from all active flows and determines whether or not to sleep. If any flow is currently waiting for a packet (as in the case when the NIC wakes up early), this module will prevent the device from sleeping. Otherwise, it determines the earliest time that any flow needs to wake up and sleeps accordingly.

We note that all of the necessary modifications are done at the client and no modification to the AP is required. In this manner, a working implementation of PSM-AW can be deployed with relative ease.

C. Comparison to Existing Schemes

We now compare PSM-AW to existing PSM schemes, including dynamic PSM and static PSM. We consider two dynamic PSM schemes with timeouts of 95 and 200 ms, which are labeled Dyn-95 and Dyn-200, respectively. For our direct comparisons, we are mostly concerned with measuring the awake time and the flow time. Specifically, a good scheme will have a low total awake time—the amount of time the NIC is in an awake state—and thus use little energy. Similarly, a good scheme will have a low total flow time—the total amount of time a device waits for outstanding packets—and thus allow the user to experience little delay.

1) A Simple Study: We first consider a simple study. On an isolated network, we set up a simple web server that serves

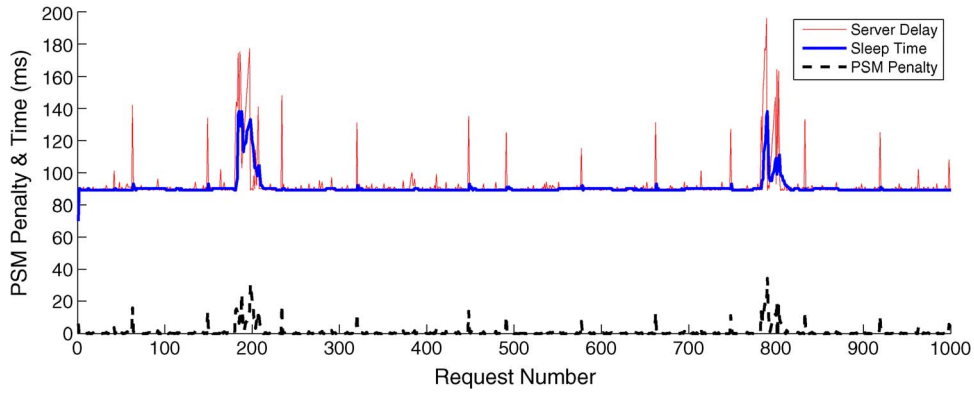


Fig. 7. Trace of 1000 requests to Twitter.

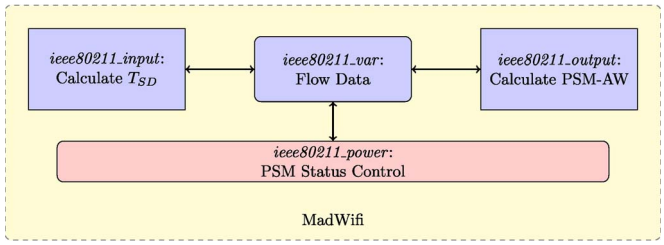


Fig. 8. Diagram of PSM-AW implementation in MadWifi.

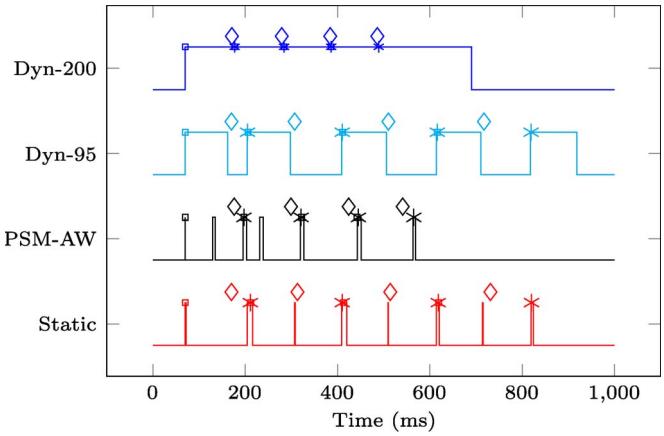


Fig. 9. This figure shows a simple HTTP flow with a SYN packet, two TCP windows, and a FIN packet for the four tested schemes. The solid lines represent the awake (high) and sleep (low) status of the device. The square represents the request made by the client, the diamond is when the response is available at the AP, and the asterisk represents the reception of the response at the client.

a static website. We then equip one laptop with a modified MadWifi driver capable of dynamic PSM, static PSM, and PSM-AW. We then make a single HTTP request for the website with each scheme, and observe the results. The entirety of the HTTP flow is four separate requests, including a SYN packet, two TCP windows, and a FIN packet.

As we can see from the results in Fig. 9, both Dyn-95 and Dyn-200 are awake much longer than necessary, with Dyn-95 also incurring significant delays by sleeping immediately before the response would arrive. Static PSM also experiences very significant delays, although it spends very little time awake. PSM-AW better responds to the actual traffic pattern and experiences very little additional delay and very little awake time.

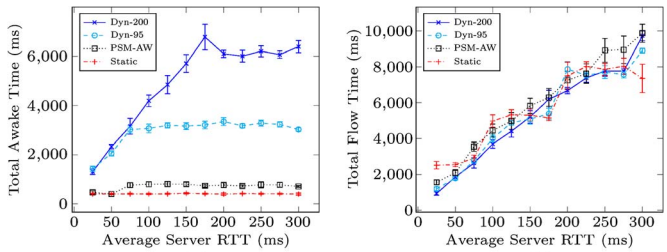


Fig. 10. Experimental results. We adjust the RTT of the server to show the performance for each scheme. Each data point is an average of ten trials with a 95% confidence interval shown. (a) Total awake time for varying RTT. (b) Total flow time for varying RTT.

TABLE III
DETAILS OF SERVERS USED INSection V-C3

Server	Download Size (KB)	Average T_{SD} (ms)
amazon.com	221	60
mp3lemon.com	28	242
google.com	13	91
cnn.com	109	114

2) Variable Server Delay: In real networks, we may initiate connections with servers that vary greatly in their server delays. We desire to see the effect of these varying server delays on different power saving schemes. To do so, we repeatedly request an entire website from a local server with all its assets, i.e., 12 files in all. We adjust the RTT introduced by the server to test different server delays. As shown in Fig. 10(a) and (b), PSM-AW and static PSM provide the smallest total awake time and thus save the most power. Dyn-200 and Dyn-95 spend a significantly longer time awake, as they do not sleep immediately after the flow ends or pauses. Further, PSM-AW offers this savings in awake time without significantly increasing the total flow time, as required by static PSM.

3) Multiple Flow Performance: Finally, we compare the overall performance of PSM-AW to existing schemes in the presence of multiple flows. To do so, we observe each scheme while performing multiple concurrent flows, namely accessing four different websites. The details of the connections can be seen in Table III, which shows the server name, the amount of data downloaded, and the average server delay. Fig. 11 confirms the results from Section V-C2 by showing that PSM-AW requires less awake time than Dynamic PSM without excessive additional flow time. On average, PSM-AW is only

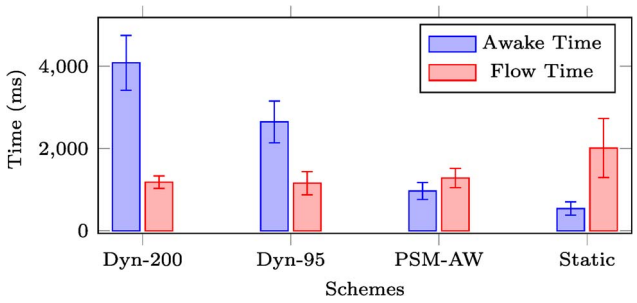


Fig. 11. Comparison of schemes for a simple web session involving four different flows to different servers. Each data point is an average of ten trials with a 95% confidence interval shown. PSM-AW requires significantly less awake time than the Dynamic PSM schemes, but it does not take as long to complete the flow as with static PSM.

awake for 23.6% (36.3%) of the time Dyn-200 (Dyn-95) needs to be awake, while having a total flow time of only 63.6% of static PSM.

Using the same method as in Section IV-B1, we can estimate the actual energy savings of PSM-AW. Since PSM-AW is awake for 962 ms, it consumes $962\ \text{ms} \cdot (520 - 120)\ \text{mW} = 0.38$ J more energy than if the device were asleep the whole time. Similarly, Dyn-200 and Dyn-95 use 1.6 and 1.1 J more, respectively, whereas Static uses 0.21 J more. Thus, PSM-AW can save considerable energy over a short scenario compared with Dynamic PSM, while still having a reasonable flow time.

VI. Related Work

A. Power Efficiency in Wi-Fi

Much work has been done to increase the efficiency of PSM, attempting to allow devices to sleep for long time intervals while still maintaining a small delay. In [4], the bounded-slowdown (BSD) protocol is first presented. The paper describes how static PSM effectively rounds the RTT for arbitrary connections up to the beacon interval by forcing the client to sleep until the succeeding beacon period. To mitigate this rounding, Krashinsky and Balakrishnan presented BSD, which adapts to RTT for arbitrary connections by selectively skipping beacons and sleeping for longer time intervals. Similarly, Perez-Costa and Camps-Mur [5] sought to limit delay by altering the transmission of PS-Poll packets in static PSM, whereas Qiao and Shin [6] sought to determine the optimum sleep sequence to bound the delay of downlink packets. PSM-AW, on the other hand, seeks to bound both the delay and the awake time.

More recently, Ding [7] also sought to maximize power savings from PSM techniques. To do so, Ding introduced a proxy at the AP to complete a data exchange while the initiating client sleeps. This allows the client to sleep while data are transferred across the network and then only be awake for the final data transfer. While this method may introduce significant power savings, it still has the effect of rounding up short RTTs to the nearest beacon interval and may result in significant delays for otherwise brief connections. Similarly, Dogar *et al.* [8] utilized the AP to buffer additional packets for long data transfers to allow more sleep time for the client, so long as the final transfer still finishes at the same time. PSM-AW seeks to maintain the power savings from such methods while also introducing a bound for the delay seen by the end client. Another consideration in Wi-Fi power efficiency is the impact of contention on channel access. In [9]–[13], ways to schedule clients within an AP to minimize contention were considered, whereas in [3], the effects of multiple nearby APs were considered. While these works may eliminate unnecessary power consumption due to idle listening, they still suffer from the RTT roundup problem. Chen *et al.* [14] considered the effects of mobility on PSM and predict the optimal time to deliver packets based on a user's proximity to the AP. Moreover, Perez-Costa and Camps-Mur [15], [16] considered changes to traditional static PSM to account for updates based on 802.11e QoS enhancements. Finally, Liu and Zhong [17] discussed hardware improvements in wireless NICs, which potentially allow for more sleep opportunities. Most of these schemes require modification of the AP to achieve their goals. PSM-AW, however, is a client-side-only solution, requiring minimal changes to the system.

Still other work targets voice over IP (VoIP) applications specifically. In [18], SiFi dynamically detects silent periods over a VoIP connection and suppresses those packets, allowing the devices to sleep. GreenCall [19] considers ear-to-ear delay in VoIP calls and selectively chooses sleep and wakeup schedules to minimize playback jitter. PSM-AW, however, works with generalized connections.

Outside the realm of Wi-Fi, Humar *et al.* [20] propose a new model to consider not only the operating energy of deploying base stations but also the total embodied energy, which include manufacturing and fabrication costs as well. This work is complementary to ours as we seek to consider the larger implications of our energy savings decisions. Their work shows that producing more low-energy devices may not actually be helpful, whereas we explore the relation between energy saving on client devices and resulting delay performance.

B. RTT Prediction and Modeling

Much work has also been done in an effort to predict end-to-end delay between a client and a server [21]–[25]. They describe various efforts to model sections of the Internet using techniques such as graphs, learning algorithms, and queuing networks. Unfortunately, all of the techniques are very limited and often resource intensive. Further, even the most accurate of prediction methods cannot predict server variability in every instance; thus, guaranteed performance bounds must account for this variability. PSM-AW, however, accounts for this variability and thus offers a connection-dependent performance bound.

VII. Conclusion

Currently, Wi-Fi PSMs are not efficient in both energy and delay efficiency due to difficulties in predicting the communication delay across a network. We propose PSM-AW, a power-saving scheme capable of achieving both low energy use and low delay by anticipating server delay and preemptively waking up to retrieve packets from the AP. This scheme allows the client more sleep opportunities as in static PSM, while avoiding

delay penalties as in dynamic PSM. In our simulations and experiments, PSM-AW achieves substantial awake time efficiency over dynamic PSM with only a small increase in total flow time.

REFERENCES

[1] *IEEE Standard 802.11-2007, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, (Revision of IEEE Std 802.11-1999)*, Jun. 2006.

[2] Android Developers, WiFiManger Class Overview. [Online]. Available: http://developer.android.com/reference/android/net/wifi/WifiManager.html

[3] J. Manweiler and R. Roy Choudhury, "Avoiding the rush hours: WiFi energy management via traffic isolation," in *Proc. ACM MobiSys*, Washington, DC, USA, 2011, pp. 253–266.

[4] R. Krashinsky and H. Balakrishnan, "Minimizing energy for wireless web access using bounded slowdown," in *Proc. ACM MobiCom*, Atlanta, GA, USA, 2002, pp. 119–130.

[5] X. Perez-Costa and D. Camps-Mur, "APSM: Bounding the downlink delay for 802.11 power save mode," in *Proc. IEEE ICC*, Seoul, Korea, 2005, vol. 5, pp. 3616–3622.

[6] D. Qiao and K. Shin, "Smart power-saving mode for IEEE 802.11 wireless LANs," in *Proc. IEEE INFOCOM*, Miami, FL, USA, 2005, vol. 3, pp. 1573–1583.

[7] N. Ding *et al.*, "Realizing the full potential of PSM using proxying," in *Proc. IEEE INFOCOM*, Orlando, FL, USA, 2008, pp. 2821–2825.

[8] F. R. Dogar, P. Steenkiste, and K. Papagiannaki, "Catnap: Exploiting high bandwidth wireless interfaces to save energy for mobile devices," in *Proc. ACM MobiSys*, San Francisco, CA, USA, 2010, pp. 107–122.

[9] E. Rozner, V. Navda, R. Ramjee, and S. Rayanchu, "NAPman: Network-assisted power management for wifi devices," in *Proc. ACM MobiSys*, San Francisco, CA, USA, 2010, pp. 91–106.

[10] Z. Zeng, Y. Gao, and P. Kumar, "SOFA: A sleep-optimal fair-attention scheduler for the power-saving mode of WLANs," in *Proc. IEEE ICDCS*, Minneapolis, MN, USA, 2011, pp. 87–98.

[11] P. Agrawal *et al.*, "OPSM—Opportunistic power save mode for infrastructure IEEE 802.11 WLAN," in *Proc. IEEE ICC Workshops*, Cape Town, South Africa, 2010, pp. 1–6.

[12] C.-H. Gan and Y.-B. Lin, "An effective power conservation scheme for IEEE 802.11 wireless networks," *IEEE Trans. Veh. Technol.*, vol. 58, no. 4, pp. 1920–1929, May 2009.

[13] Y. He and R. Yuan, "A novel scheduled power saving mechanism for 802.11 wireless LANs," *IEEE Trans. Mobile Comput.*, vol. 8, no. 10, pp. 1368–1383, Oct. 2009.

[14] X. Chen, S. Jin, and D. Qiao, "M-PSM: Mobility-aware power save mode for IEEE 802.11 WLANs," in *Proc. IEEE ICDCS*, Minneapolis, MN, USA, 2011, pp. 77–86.

[15] X. Perez-Costa and D. Camps-Mur, "AU-APSD: Adaptive IEEE 802.11e unscheduled automatic power save delivery," in *Proc. IEEE ICC*, Istanbul, Turkey, 2006, pp. 2020–2027.

[16] X. Pérez-Costa, D. Camps-Mur, and A. Vidal, "On distributed power saving mechanisms of wireless LANs 802.11e U-APSD vs 802.11 power save mode," *Comput. Netw.*, vol. 51, no. 9, pp. 2326–2344, Jun. 2007.

[17] J. Liu and L. Zhong, "Micro power management of active 802.11 interfaces," in *Proc. ACM MobiSys*, Breckinridge, CO, USA, 2008, pp. 146–159.

[18] A. Pyles, Z. Ren, G. Zhou, and X. Liu, "SiFi: Exploiting VoIP silence for WiFi energy savings in smart phones," in *Proc. ACM UbiComp*, Beijing, China, 2011, pp. 325–334.

[19] V. Namboodiri and L. Gao, "Energy-efficient VoIP over wireless LANs," *IEEE Trans. Mobile Comput.*, vol. 9, no. 4, pp. 566–581, Apr. 2010.

[20] I. Humar, X. Ge, L. Xiang, and M. Jo, "Rethinking energy efficiency models of cellular networks with embodied energy," *IEEE Netw.*, vol. 25, no. 2, pp. 40–49, Apr. 2011.

[21] M. Yang, X. Li, H. Chen, and N. Rao, "Predicting internet end-to-end delay: An overview," in *Proc. IEEE SSST*, Atlanta, GA, USA, 2004, pp. 210–214.

[22] S. Shakkottai, N. Brownlee, A. Broido, and K. Claffy, "The RTT distribution of TCP flows on the internet and its impact on TCP based flow control," Cooperative Association for Internet Data Analysis, La Jolla, CA, USA, Tech. Rep., Mar. 2004.

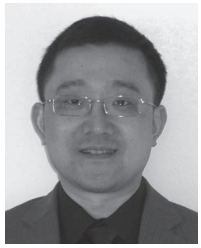
[23] S. Belhaj and M. Tagina, "Modeling and prediction of the internet end-to-end delay using recurrent neural networks," *Comput. Netw.*, vol. 4, no. 6, pp. 528–535, 2009.

[24] W. Zhang and J. He, "Modeling end-to-end delay using Pareto distribution," in *Proc. IARIA ICIMP*, San Diego, CA, USA, 2007, p. 21.

[25] L. Lam, K. Su, C. Chan, and X. Liu, "Modeling of round trip time over the internet," in *Proc. IEEE ASCC*, Hong Kong, 2009, pp. 292–297.

Brian Peck (S'12) received the B.S. degree in computer engineering from Iowa State University, Ames, IA, USA, in May 2010, where he is currently working toward the Ph.D. degree with the Department of Electrical and Computer Engineering.

His current research interests include protocols for wireless local area networks and remote situation awareness.



Daji Qiao (M'04) received the Ph.D. degree in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, in February 2004.

He is currently an Associate Professor with the Department of Electrical and Computer Engineering, Iowa State University, Ames, IA, USA. His current research interests include protocol and algorithm innovation and implementation for IEEE 802.11 wireless local area networks and wireless sensor networks, cyber security and cloud computing, and pervasive computing applications.

Dr. Qiao is a member of the Association for Computing Machinery.