Managing the 802.11 Energy/Performance Tradeoff with Machine Learning

Claire Monteleoni **Hari Balakrishnan** **Nick Feamster** **Tommi Jaakkola**
MIT Computer Science and Artificial Intelligence Laboratory
32 Vassar St.
Cambridge, MA 02139
{*cmontel, hari, feamster, tommi*}@*csail.mit.edu*

***Abstract*—This paper addresses the problem of managing the tradeoff between energy consumption and performance in wireless devices implementing the IEEE 802.11 standard [1]. To save energy, the 802.11 specification proposes a power-saving mode (PSM), where a device can sleep to save energy, periodically waking up to receive packets from a neighbor (e.g., an access point) that may have buffered packets for the sleeping device. Previous work has shown that a fixed polling time for waking up degrades the performance of Web transfers [2], because network activity is bursty and time-varying. We apply a new online machine learning algorithm to this problem and show, using `ns-2` simulation and trace analysis, that it is able to adapt well to network activity. The learning process makes no assumptions about the underlying network activity being stationary or even Markov. Our learning power-saving algorithm, LPSM, guides the learning using a "loss function" that combines the increased latency from potentially sleeping too long and the wasted use of energy in waking up too soon. In our `ns-2` simulations, LPSM saved 7%-20% more energy than 802.11 in power-saving mode, with an associated increase in average latency by a factor of 1.02, and not more than 1.2. LPSM is straightforward to implement within the 802.11 PSM framework.**

I. INTRODUCTION

Energy is an important resource in mobile computing systems. Because processing, storage, display activity, and communication all consume energy, energy-saving techniques targeted at improving these subsystems have received significant attention in recent years. Impressive advances in hardware design and operating systems have greatly reduced the energy consumed by the processing and storage subsystems, and have led to the wireless network becoming a significant consumer of energy in many mobile devices. This trend is especially true for handheld mobile devices and nodes in wireless ad hoc and sensor networks.

Most wireless network interfaces consume energy not only while transmitting or receiving data, but also when they are simply awake. Therefore, to save energy, most modern wireless interfaces support a *power saving mode* (PSM). In abstract terms, the PSM primitive allows an interface to be in one of two states, SLEEP or AWAKE. SLEEP is a low-power state, but the interface cannot send or receive data in this state. In contrast, AWAKE allows data flow, but is a higher-power state. Depending on the actual device, these two states may differ in power consumption by between a factor of 10 and 50. For instance, in some current 802.11 cards, the ratio is about a factor of 20 (1 W v. 50 mW) [3], [4].

With the PSM primitive, power-saving algorithms can save energy by keeping the wireless interface in the SLEEP state for as long as possible. A SLEEPing device periodically wakes up and polls its neighbors (either an access point in "infrastructure" mode, or a neighboring node in "ad hoc" mode) for packets.[1] To avoid excessive packet loss, the neighbor must therefore buffer packets for each SLEEPing receiver. Then, the neighbor sends these buffered packets when it receives a poll from a waking receiver.

Power-saving algorithms built on top of the PSM primitive introduce a tradeoff between the amount of energy saved and the degree of performance degradation. If a device awakens and finds no data buffered for it, then it could have slept for longer and saved some more energy. On the other hand, if any packets are buffered when the interface awakens, then the latency to obtain those packets would be larger than if the network interface had been awake instead of asleep. This increased latency degrades not just the latency of the on-going data transfers, but often the throughput as well.

This paper addresses the problem designing an algorithm by which a device can decide when to SLEEP and when to be AWAKE. Our goal is to devise an algorithm that manages the tradeoff between energy consumption and data transfer latency in a principled, well-specified way, such that users or application designers can specify

[1] This is an abstract model: some implementations first have the neighbor advertise information before the polls occur.

their desired operating point. Our motivation for a principled trade-off is motivated by Krashinsky and Balakrishnan's work on the Bounded SlowDown (BSD) algorithm [2], where they demonstrate that the IEEE 802.11's non-adaptive polling time strategy [1] degrades both the latency and the throughput of TCP transfers. As in their work, we focus our algorithm on Web-like workloads, mainly because it is the dominant workload today for many mobile devices.

We develop a PSM algorithm called **LPSM (Learning PSM)** to determine a device's sleep/awake schedule. LPSM adapts the schedule to network activity by maintaining a bank of "experts". Each expert is a deterministic setting of the polling time, with an associated time-varying weight that gets updated by the algorithm in response to observations of current network activity. The weights form a probability distribution, and at each time, the polling time chosen is the weighted sum of the experts' times. LPSM is based on a recently developed machine learning algorithm called *Learn-*α [5], [6], which has the attractive property that it makes no assumptions on the statistical distribution of the activity being learned. In our context, LPSM's use of Learn-α makes no assumptions on the distribution of packet arrivals and network activity.

The first contribution of this paper is to show how online machine learning can be used to solve the wireless power-saving problem. The key to this solution is to define a *loss function* that the Learn-α algorithm uses in determining how to update the weights of the experts every time the mobile device awakens. If the device awakens and there is no data present, then the weights of the experts are carefully adjusted such that the next sleep time is longer. Conversely, if any packets were present, the opposite adjustment is made.

The second contribution of this paper is a performance evaluation of LPSM using trace-driven simulations. We compare the performance of both the non-adaptive 802.11 PSM and the BSD algorithm to LPSM. Our experimental results to date have shown that for a Web-like request/response workload, LPSM saves 7%-20% more energy than 802.11 in power-saving mode, with an associated increase in average slowdown of 2%, and not more than 20%. LPSM is straightforward to implement within the 802.11 PSM framework.

The rest of this paper is organized as follows. In Section II, we survey related work in power saving and machine learning. Section III gives a formal definition of the problem and an overview of our approach. Section IV gives the LPSM algorithm. Section V presents several results from trace-driven `ns-2` simulations and trace-based analysis of LPSM, and Section VI concludes the paper with a discussion of our results.

II. Related Work

Using trace-driven simulations, Krashinsky and Balakrishnan [2] show that the 802.11 PSM algorithm, which uses a fixed polling interval (typically 100 ms) to wake up and check for data, causes response latency for Web-like transfers to be as bad as $2.2\times$ longer than in the absence of any power-saving algorithm. To better manage the tradeoff in question, they proposed BSD, an algorithm that uses an adaptive control loop to change polling time based on network conditions. The algorithm uses a parameter, p, and guarantees that the response latency does not ever exceed $(1 + p)$ times the response latency without power-saving. Within that constraint and assuming adversarial traffic arrivals, BSD guarantees that the energy consumption is minimized. In contrast, LPSM does not attempt to guarantee bounded latency under adversarial traffic arrivals; instead, our approach is to explicitly encode a tradeoff between energy and latency and give an online learning algorithm that manages this tradeoff.

Simunic *et al.* formulate the wireless power-saving problem as policy learning in a Markov Decision Process (MDP) [7]. Their algorithm is not an online algorithm since the linear programming algorithm used to resolve the policy over any given time period requires access to data over that entire period. They also assume that network activity is stationary. In the MDP there is fixed distribution governing the selection of next states, given the current state and action. For any fixed policy such as the optimal policy in this framework, the network activity is modeled as a Markov process. This model is not an ideal one for a mobile node, since the network activity need not conform to a Markov process of any finite order k.

Simunic *et al.* refer to Chung *et al.* [8] for the solution in non-stationary environments. That work proposes "policy interpolation"; however, it still assumes that the underlying process is Markovian, even though it may initially *appear* non-stationary due to a lack of observations. They then propose to learn the associated MDP parameters sequentially [8]. Another machine learning approach to this problem was proposed Steinbach, using Reinforcement Learning [9]. This approach also imposes the assumption that network activity has the Markov property which, as discussed above, is unrealistic.

These previous learning approaches differ from ours in that LPSM does not make any Markovian or stationarity assumptions, nor require any a priori knowledge of the statistical process being learned. LPSM is also simpler to implement in the 802.11 framework compared to these previous learning approaches.

Our learning algorithm is from the field of *online learning* algorithms. Littlestone and Warmuth [10] designed online learning algorithms that have access to a fixed set of experts, for which they were able to bound the cumulative error (regret) of the algorithm over any fixed period, relative to the error that would have been incurred by using just the best expert over that period. Herbster and Warmuth [11] extended this work to algorithms whose cumulative error they upper bounded relative to the error of the hindsight-optimal (cumulative error minimizing) k-partition of the time period; i.e., the best partition of the time period into k (variable length) segments and selection of the best expert for each segment. Similar algorithms have been applied to such problems as paging [12], gambling [13] and portfolio management [14].

We will consider algorithms that treat network activity as non-stationary, although possibly composed of variable-length periods that exhibit stationarity in some sense. These algorithms are parameterized by the switching-rate of the non-stationary process. In the context of wireless networks, this value cannot be known by a mobile node *a priori*. Thus we will use a new algorithm to learn the switching-rate parameter online, simultaneous to learning the target concept: the polling time in this context [5], [6].

III. Overview and Methods

The intuition behind LPSM is as follows. The IEEE 802.11 PSM standard is a deterministic algorithm polling at fixed polling time, $T = 100$ ms. We treat that as one "expert," who always claims that 100 ms is the polling time that should be used. Clearly the ability to consult a set of experts, in which each expert is a deterministic algorithm using a different T value as its polling time, would enable a more flexible algorithm compared to operating with just one polling time. LPSM uses an adaptive randomized online algorithm that maintains a probability distribution over a set of such deterministic experts. The algorithm will compute its polling time as a function all the experts, subject to this distribution. It will adaptively update its distribution over experts, based on current network activity.

Not only is this approach more flexible than that of the 802.11 PSM standard, it also seems to be a promising alternative to approaches like BSD that are based on an adaptive control framework. While BSD adaptively updates its polling time, T, based on network conditions, it does so with only one starting T value. The algorithm we propose would instead maintain updates on a set of n T values, instead of just one. Although the form of updates are different than those used by BSD, the essential intuition is that this allows the learning algorithm to explore across a range of T's simultaneously, and should thus allow it to choose a good instantaneous T value to manage the tradeoff. Since exploration for the sake of learning incurrs loss, this algorithm should be better than previous approaches at adapting to whatever traffic pattern is presented.

Additionally, since our proposed algorithm allows rapid switching between experts, it is able to handle sudden changes in the observed process, which might happen when there is a sudden burst of network activity. Moreover, the model we propose for a node using this algorithm, is similar to 802.11 PSM in mainly sleeping except for polls, although the sleep times would be of variable length. After retrieving any packets that may have arrived from the neighbor's buffer, it will only stay awake if the link continues to be active.

To update the distribution maintained over experts, the intuition is that different experts should gain or lose favor based on current network activity. Upon awakening, if many packets had been buffered, then the algorithm should adjust and sleep for less time. On the other hand if only a few packets were received, the algorithm can sleep for longer in order to save energy. Thus after each observation, each expert gets scored according to how effective its proposed sleep time would be in the current network conditions. The score is based on the loss function that we will define in the next section, but encodes how well the polling time that the expert proposes would handle the current network conditions, with the goal of simultaneously minimizing energy and slowdown.

A. Machine Learning Background

The algorithms of the type discussed above have performance guarantees with respect to the best expert in the expert set given. These guarantees make no assumptions on the set of experts, as they are worst case guarantees computed as if the expert set is just a black box, and could even contain algorithms that are adversarial. Thus when applying such learning algorithms to a specific problem, we can achieve additional gains from the performance guarantees, by choosing experts that are actually helpful for the problem domain in question. In the wireless energy management problem, we use n experts, each corresponding to a different but fixed polling time: $T_i : i \in \{1 \ldots n\}$. The experts form a discretization over the range of possible polling times.

Unlike many previous problems where online learning has been applied, our problem imposes the constraint that the learning algorithm can only receive observations, and perform learning updates, when the mobile device is awake. Thus our subscript t here signifies only wake times, not every time epoch during which bytes might arrive.

To apply this type of online learning algorithm to this problem, we instantiate the prediction function using the weighted mean. Thus the algorithm's polling time T_t, i.e. its prediction of the current amount of time it ought to sleep for is:

$$T_t = \sum_{i=1}^{n} p_t(i) T_i \tag{1}$$

where $p_t(i)$ is its current distribution over experts that favors those who have recently predicted accurately.

To build intuition for the algorithm we use, we will describe the use of the `Fixed-share` algorithm, due to [11], in this problem. The algorithm we will use employs a collection of `Fixed-share` algorithms in a hierarchy. The collection is managed by a simple `Static-expert` algorithm [10]. Since a direct application of the `Static-expert` algorithm is also used in our comparisons we will commence with the description of this algorithm, followed by the `Fixed-share`.

In the `Static-expert` algorithm, which does not model switches between experts, the distribution over experts is updated at each training iteration as follows:

$$p_t(i) \;\; = \;\; \frac{1}{Z_t} \, p_{t-1}(i) \, e^{-L(i,t-1)} \tag{2}$$

where $L(i,t)$ denotes the loss of expert i at time t, and Z_t normalizes the distribution. The loss captures how the prediction (polling time) relates to the observations and we will describe it in detail in the next section. All the algorithms here are modular in terms of the loss function in the sense that they can accomodate any loss function given as an input to the algorithm. The definition of the loss function can be tailored separately to each application based on its specific objectives. In probabilistic terms, we can relate the loss function also to predictive probabilities.

As illustrated in Figure 1, if we define the probability of predicting observation y_t with expert T_i as $P(y_t|T_i) = e^{-L(i,t)}$, then the update equation above is consistent with Bayesian estimation, with the identity of the current best polling time (or expert, i) as the unknown variable.

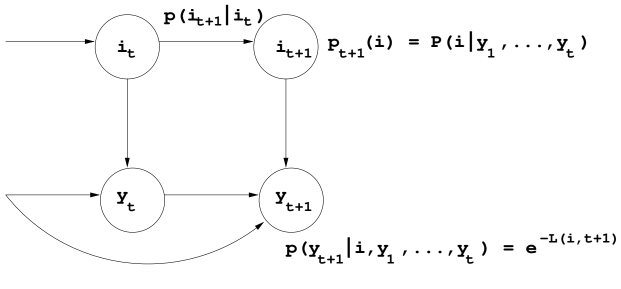


Fig. 1. A generalized Hidden Markov Model (HMM) of probability of the next observation, given past observations, and the current best expert.

The `Fixed-share` algorithm involves modeling the probability that the current best expert switches from one time-step to the next, and updates are thus:

$$p_{t;\alpha}(i) \;\; = \;\; \frac{1}{Z_t} \sum_{j=1}^{n} p_{t-1}(j) \, e^{-L(j,t-1)} \, P(i|j;\alpha) \tag{3}$$

where Z_t normalizes the distribution. We initialize the weighting to uniform $p_1(i) = 1/n$, since the learner has no *a priori* knowledge about the experts, so introducing an arbitrary preference for a given expert could hurt performance if that expert turns out to be a poor predictor. After each update, we renormalize the weights to sum to one, and thus a probability distribution over the experts is maintained.

The probability of switching between experts is defined as:

$$P(i|j;\alpha) = \begin{cases} (1-\alpha) & i = j \\ \frac{\alpha}{n-1} & i \neq j \end{cases} \tag{4}$$

$0 \leq \alpha \leq 1$ is a parameter that indicates how likely it is that switches will occur in the model's estimate of the current best expert. That is, it is the model's estimate of how often the deterministic polling time corresponding to the current best choice changes, in light of the current rate of packet arrivals, or level of traffic burstiness. Using this model in the weight updates, each expert shares a fraction of its weight with the pool of weights. This ensures that no expert will accrue a weight exponentially close to zero, which allows for quick weight-recovery in the event of a switch in the current observations, i.e., network activity.

The α parameter determines how rapidly the algorithm adapts to changes: high values correspond to assuming the observation process changes rapidly, whereas values close to zero assume the process is nearly stationary. In the ideal case, the `Fixed-share` algorithm should use the setting of α that matches the true switching-rate of the non-stationary distribution being observed. Without prior knowledge of the process however, it is hard to set this parameter beforehand. Thus in the algorithm we describe next, this quantity is learned online.

B. Learn-α Algorithm

We will now describe the `Learn-`α algorithm [5], [6]. The additional benefit of using `Learn-`α is that it does not require any prior knowledge or assumptions about the level of non-stationarity. The `Fixed-share` algorithm uses the parameter α to indicate how likely the model believes switching is to occur between which expert is currently predicting best. The `Learn-`α algorithm learns this quantity online, simultaneously to learning and adapting its polling time.

Before stating the details, we give a conceptual view of the `Learn-`α algorithm in Figure 2. The algorithm updates a distribution over α-experts, and each α-expert maintains a distribution over polling times.

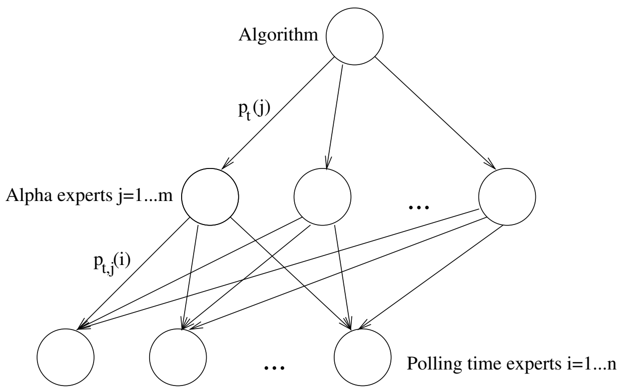


Fig. 2. The hierarchy maintained by the `Learn−`α algorithm, to simultaneously learn the current best polling time, and the degree of nonstationarity in network activity.

In the `Learn−`α algorithm, we maintain m α-experts. The α-experts are `Fixed-share` sub-algorithms that each maintain a distribution over the experts, $p_{t,j}(i) = p_{t;\alpha_j}(i)$, initialized to the uniform distribution. Here α_j is the jth sub-algorithm's definition of the probability that the optimal polling time changes from one discrete T_i to another, and thus $p_{t,j}(i)$ is updated according to Equation 3. The `Learn−`α algorithm chooses its current polling time T_t, i.e., its prediction of the current amount of time it ought to sleep, using the weighted mean of the α-expert predictions:

$$T_t \quad = \quad \sum_{j=1}^{m} p_t^{top}(j)\, T_{t,j} \tag{5}$$

where $p_t^{top}(j)$ is the distribution the algorithm maintains over α-experts, and $T_{t,j}$ is the prediction of the jth sub-algorithm of how long to sleep for, computed as the weighted mean over the experts:

$$T_{t,j} \quad = \quad \sum_{i=1}^{n} p_{t,j}(i)\, T_i \tag{6}$$

The sub-algorithms do not actually sleep for this amount of time though, as the polling time is chosen by the top-level. The algorithm's polling time is thus

$$T_t \quad = \quad \sum_{j=1}^{m}\sum_{i=1}^{n} p_t^{top}(j)\, p_{t,j}(i)\, T_i \tag{7}$$

We define the loss $L(\alpha_j, t)$ of the jth sub-algorithm, or α-expert, as:

$$L(\alpha_j, t) = -\log \sum_{i=1}^{n} p_{t,j}(i)\, e^{-L(i,t)} \tag{8}$$

This form has the Bayesian interpretation that the "regret" of the jth sub-algorithm is the negative log-likelihood of the weighted sum over experts of the current observation given the prediction of that expert, where the weighting is given by the jth sub-algorithm's current distribution over experts.

Using the per expert loss function $L(i, t)$ that we will define below, we update $p_t^{top}(j)$, the distribution over α-experts, according to:

$$p_t^{top}(j) \quad = \quad \frac{1}{Z_t} p_{t-1}^{top}(j) e^{-L(\alpha_j, t-1)} \tag{9}$$

where $p_1^{top}(j) = 1/m$, $p_{1,j}(i) = 1/n$, and $p_{t,j}(i)$ is updated according to (3), using $\alpha = \alpha_j$.

C. Optimal Discretization

The number, m, of α_j values to use in the algorithm need not be a parameter, as we can use a discretization algorithm for computing the optimal set of values, based on recent work on regret-optimal discretization [6]. Based on this work, only a finite number of α values need to be used in order come close enough to the optimal setting, while minimizing the regret from using too many settings. The discretization is based on uniformly minimizing the regret incurred by the `Learn−`α algorithm. To use the regret-optimal discretization with respect to any fixed number of learning iterations $\mathcal{T}$, let α^* be the unknown optimal switching rate, i.e. the setting of the parameter α that (in hindsight) best describes the sequence of observations to be predicted. Then for any threshold $\delta > 0$, we find a minimal set of discrete switching rate parameters $\{\alpha_j\}$ such that the additional regret due to discretization (shown in [6]), $\mathcal{T} \min_j D(\alpha^* \| \alpha_j)$, does not exceed δ for any $\alpha^* \in [0, 1]$. Here $D(\cdot\|\cdot)$ is the relative entropy between Bernoulli distributions. The value of δ controls the number of discrete levels m and is chosen to balance the ability of the `Learn−`α algorithm to identify the best discrete choice of the switching rate and the additional loss due to discretization. The optimal a priori choice of δ^* in this sense is $1/2\mathcal{T}$, and the detailed computation of $m(\delta^*)$ is given in [6].

IV. LPSM: Learning PSM algorithm

A. Objective Function

The algorithm as stated is complete except for the choice of loss function, $L(i, t)$. Note however that the performance bounds on the algorithm hold regardless of the choice of loss function. In this application to the wireless power-saving problem, we instantiate the loss function as follows. The objective at each learning iteration is to choose the polling time T_t that minimizes both the energy usage of the node, and the network latency it introduces by sleeping. We define the loss function so as to reflect the tradeoff inherent in these conflicting goals. Specifically, we will design a loss function that is directly proportional to appropriate estimates of these two quantities. It is important to note that the algorithm is modular with respect to this function, so while we suggest several loss functions that are proportional to the energy versus slowdown tradeoff, there are many possible functions. If one wishes to model the tradeoff differently, one need only specify an objective function, and the algorithm stated above will learn to optimize that objective.

The BSD [2] approach to managing this tradeoff uses an objective of:

$$\min E \quad : \quad L \leq (1+p)L_{opt} : p > 0 \tag{10}$$

for a scalar threshold L_{opt}. Our approach diverges from this in not applying a threshold on either of the quantities, but instead proposing a function that encodes the tradeoff between latency and energy.

The observation that the learning algorithm receives upon awakening is the number of bytes that arrived while it slept during the previous interval. We denote this quantity as I_t, and the length of time that the node slept upon awakening at time t, as T_t. We modeled energy usage proportional to $\frac{1}{T_t}$. This is based on the design that the node wakes only after an interval T_t to poll for buffered bytes, and the fact that it consumes less energy when asleep than awake. Additionally there is a constant spike of energy needed to change from sleeping to awake state, so the more times a node polls during a given time horizon, the higher the energy consumption. An alternative model for the energy consumption would be to use the term $\frac{1}{\log T_t}$. This is a larger penalty, in order to account for additional energy used, such as while staying awake when the link is active, which is not reflected in how long we choose to sleep for at each learning iteration. For clarity, we will just show the first energy model in our equations below, but we will report on the evaluation of both models.

We modeled the latency introduced into the network due to sleeping for T_t ms as proportional to I_t. In other words, there is increased latency for each byte that was buffered during sleep, by the amount of its wait-time in the buffer. Since our learning algorithm does not perform measurement or learning while asleep, and only observes the aggregated number of bytes that arrived while it slept, we have to approximate the amount of total latency its chosen sleep time introduced, based on the individual buffer wait-times of each of the I_t bytes. To estimate the average latency that each of the I_t buffered bytes would have experienced, without knowledge of the byte arrival times, we can use the maximal entropy assumption. This models all the bytes as arriving at a uniform rate during the sleep interval, T_t, under which assumption the average wait-time per byte would be $\frac{T_t}{2}$. Thus the total latency introduced by sleeping for T_t is approximated by the number of bytes that arrived in that time, times the average wait-time per byte, yielding $\frac{T_t}{2} I_t$.

```
Algorithm LPSM
```
Initialization:
$\forall j,\ p_1(j) \leftarrow \frac{1}{m}$
$\forall i, j,\ p_{1,j}(i) \leftarrow \frac{1}{n}$
Upon tth wakeup:
$T_t \leftarrow$ number of ms just slept
$I_t \leftarrow \#$ bytes stored at neighbor
Retrieve buffered data
For each $i \in \{1 \ldots n\}$:
  Loss[i] $\leftarrow \gamma \frac{I_t T_i^2}{2T_t} + \frac{1}{T_i}$
For each $j \in \{1 \ldots m\}$:
  AlphaLoss[j] $\leftarrow -\log \sum_{i=1}^{n} p_{t,j}(i)\, e^{-\text{Loss}[i]}$
  $p_{t+1}(j) \leftarrow p_t(j) e^{-\text{AlphaLoss}[j]}$
  For each $i \in \{1 \ldots n\}$:
    $p_{t+1,j}(i) \leftarrow \sum_{k=1}^{n} p_{t,j}(k)\, e^{-\text{Loss}[k]}\, P(i|k;\alpha_j)$
  Normalize $P_{t+1,j}$
  PollTime[j] $\leftarrow \sum_{i=1}^{n} p_{t+1,j}(i)\, T_i$
Normalize P_{t+1}
$T_{t+1} \leftarrow \sum_{j=1}^{m} p_{t+1}(j)$ PollTime[j]
Goto sleep for T_{t+1} ms.

Fig. 3. Conceptual view of Algorithm LPSM.

The form of our proposed loss function is thus

$$L(t) = \gamma \frac{T_t I_t}{2} + \frac{1}{T_t} \quad : \gamma > 0 \tag{11}$$

In our weight updates however, we apply this loss function to each expert i, indicating the loss that would have accrued had the algorithm used T_i instead of T_t as its polling time. So the equivalent loss per expert i is:

$$L(i,t) = \gamma \frac{I_t T_i^2}{2T_t} + \frac{1}{T_i} \tag{12}$$

where the first term scales I_t to the number of bytes that would have arrived had the node slept for time T_i instead of T_t, under the assumption discussed above that the bytes arrived at a uniform rate. Note that the objective function is a sum of convex functions and therefore admits a unique minimum.

The parameter $\gamma > 0$ allows for scaling between the units of information and time, as well as the ability to encode a preference for the ratio between energy and latency that the particular node, protocol or host network favors. This can be seen as follows. The two optimizations we are trying to perform are minimizing energy usage, and minimizing additional latency caused by buffering. These are

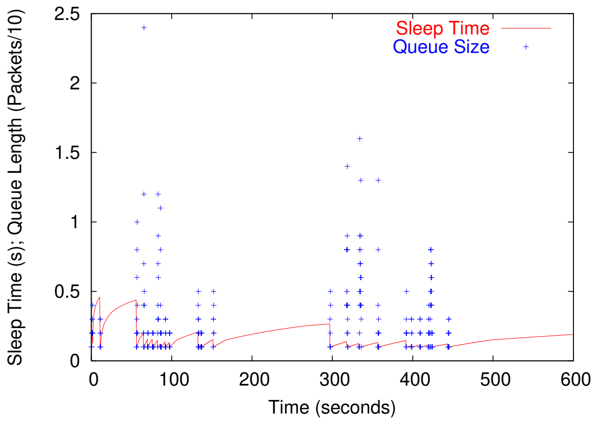


Fig. 4. Evolution of sleep times with LPSM ($1/T$).

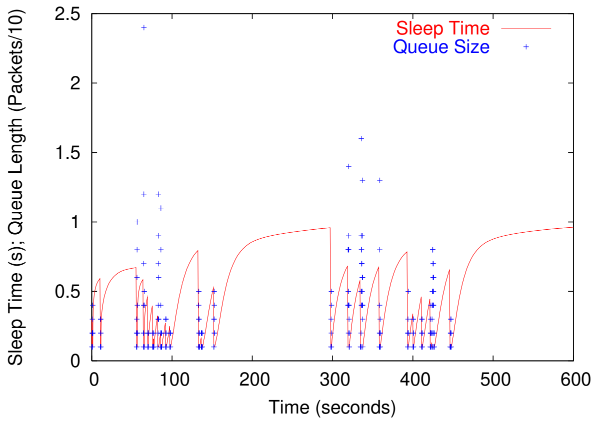


Fig. 5. Evolution of sleep times with LPSM ($1/\log T$).

conflicting goals. This can be formulated as an energy minimization problem, subject to the constraint that latency be upper bounded by a threshold. Then γ is the Lagrange multiplier enforcing that constraint. To clarify the relation between using latency as a constraint and including it as a term in the loss, note that increasing γ will monotonically increase the effect of latency on the loss, which the algorithm seeks to minimize.

Note that this is one of many possible loss functions that are proportional to the tradeoff that must be optimized for this application.

To summarize, the LPSM algorithm proceeds as shown in Figure 3.[2]

V. Performance evaluation

In Section V-A, we study the performance of LPSM with respect to the tradeoff between energy savings and performance degradation using trace-driven simulations. We also compare the performance of LPSM to previously proposed power-saving algorithms [2]. In Section V-B, we use loss function units to study the behavior of LPSM and other online learning algorithms on traces of real-time Web activity.

A. LPSM Performance: Energy/Latency Tradeoff

In this section, we study the performance of LPSM with respect to the energy/latency tradeoff. After describing the setup of our `ns-2` simulation, we compare the performance of LPSM with 802.11 static PSM and BSD [2].

1) Evaluation Framework: The simulation framework that we used to evaluate LPSM is the same as that which has been used to used to evaluate previous 802.11 power-saving algorithms such as BSD [2], [15]. We briefly summarize that framework here. We use a simple 3-node topology: a mobile client accesses a Web server via an 802.11 access point. The bandwidth between the mobile host and the basestation is 5 Mbps and the latency is 0.1 ms; the bandwidth between the basestation and the Web server is 10 Mbps and the latency is 20 ms. As in previous work [2], we do not model the details of the 802.11 MAC protocol. Rather, we model sleep times with some simple modifications to `ns-2`[15]. A sleeping device does not forward any packets, but buffers them until the device wakes up again. A device wakes up whenever it has data to forward to the access point and sleeps for the interval determined by LPSM. It remains awake after polling, only if the link remains active. Since the 802.11 PSM framework assumes that nodes wakeup and poll only on 100 ms boundaries, for the sake of synchronization issues between sleeping and awake nodes, we rounded the sleep interval determined by LPSM to the nearest 100 ms. We modeled energy consumption using previous estimates [2]: 750 mW when the interface is awake, 50 mW when the interface is asleep, and 1.5 mJ for each beacon.

We report results from an experiment that involved simulating 100 Web transfers from the client to the Web server over approximately 1.5 hours of simulation time. To verify the robustness of our results, we also ran 4 independent experiments of 500 Web transfers, each over approximately 8.5 hours of simulation time. As in previous work [2], we modeled Web browsing behavior using the `ns-2` HTTP-based traffic generator, using `FullTcp` connections. In this model, a Web transfer consists of several steps: (1) a client opens a TCP connection and sends a request; (2) after some delay, the server sends a response and several embedded images; the client may open up to 4 parallel TCP connections to retrieve these images; (3) the client waits for some amount of "think time" before making the next request. Also as in previous work [2], we randomly selected the parameters for each request based on an empirical distribution [16], and we

[2]Implementation is optimized to be more compact.

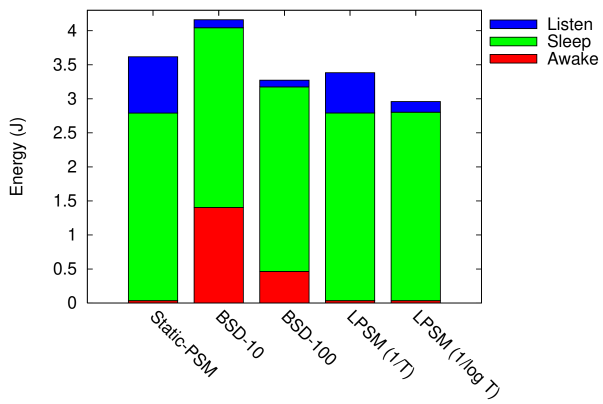


Fig. 6. Average energy usage per page for various PSM algorithms. Results from 100-page trial.

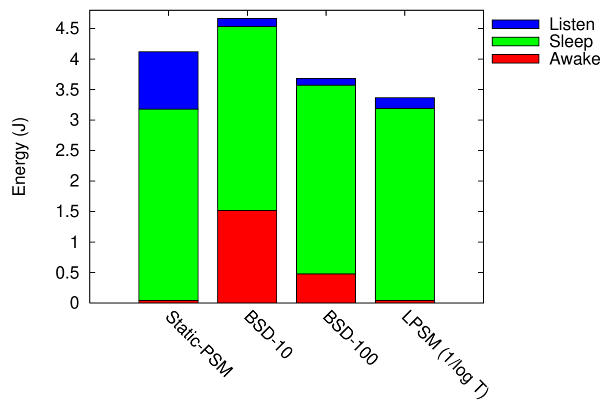


Fig. 8. Average energy per page for various PSM algorithms. Results from 4 independent 500-page trials.

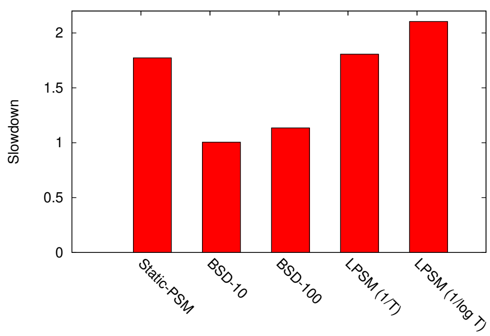


Fig. 7. Average slowdown over 802.11 without power-saving for various PSM algorithms. Results from 100-page trial.

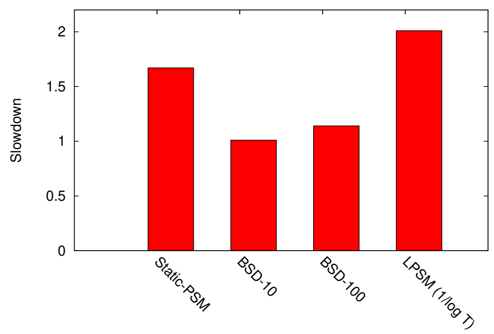


Fig. 9. Average slowdown over 802.11 without power-saving for various PSM algorithms. Results from 4 independent 500-page trials.

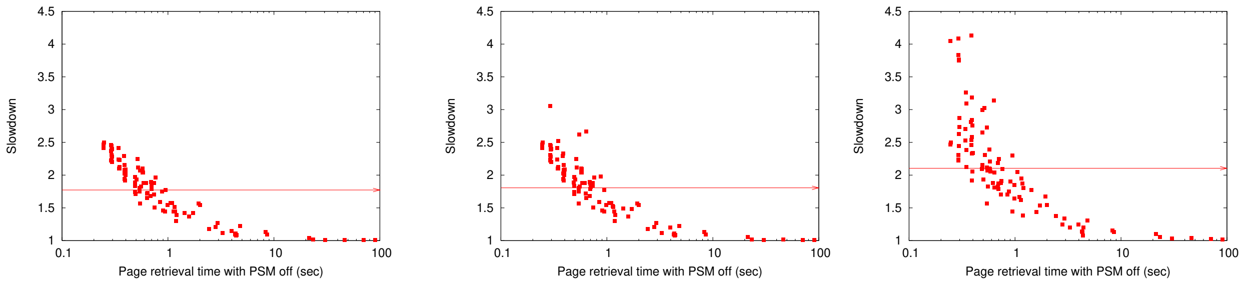
limited client think time to 1000 seconds.

In these experiments, we configured LPSM with 12 experts spanning the range from 100 ms to 1.2 seconds, at regularly spaced intervals of 100 ms. Thus, the lowest expert, 100 ms, matched the polling time of the current 802.11 PSM standard, and the highest expert was 1.2 seconds. Since a convex combination is upper bounded by its maximum value and lower bounded by its minimum value, we were assured of only using polling times within this range.

As discussed in Section IV, the learning function is modular with respect to the loss functon, as it will seek to minimize whichever loss function one uses to encode the energy/slowdown tradeoff. We experimented with the effects of two different loss functions: (1) the loss function defined in Figure 3, with the second term as $1/T$; and (2) a loss function that uses $1/\log T$ as its second term, to penalize energy usage more severely. For the first loss function, we used $\gamma = 1/(1.2 \cdot 10^5)$; for the latter, we used $\gamma = 1/(1.2 \cdot 10^3)$.

2) Algorithm Behavior: Figures 4 and 5 show the evolution of sleep times over a portion of the simulation trace. When LPSM discovers that packets are queued at the access point, it quickly reduces the sleep interval to 100 ms. During periods of inactivity, LPSM continually increases the sleep time each time the device wakes up and discovers that no packets are queued for it. These two figures also illustrate how the choice of loss function affects the behavior of the learning algorithm. When the learning algorithm has a loss function that has $1/T$ as its energy term, it backs off much more slowly than when it uses a loss function with $1/\log T$ as its energy term. This makes sense: the latter loss function places relatively more importance on the energy term, the penalty incurred for waking up when no packets are queued for the device.

3) Performance: Energy vs. Latency: Figures 6 and 7 characterize the energy savings and slowdown of LPSM relative to static PSM, which we ran under the same simulation, using the implementation by [15]. As in previous work, the slowdown is the ratio of the latency incurred by the PSM algorithm to the latency incurred by using



(a) Static PSM.

(b) LPSM ($1/T$)

(c) LPSM ($1/\log T$)

Fig. 10. Average slowdown vs. time to download that page without power-saving. Average slowdown for the entire experiment is shown with a horizontal line. LPSM ($1/T$) imposes only a slight increase in slowdown over static PSM. Using a loss function with the energy term $1/\log T$ saves more energy at the cost of increased slowdown; however, it never increases slowdown over static PSM by more than a factor of 2 for a given page.

no PSM whatsoever. Our first result is that LPSM consistently decreased per-page energy consumption. For a slight increase, 2%, in slowdown over static PSM, LPSM, with $1/T$ as the energy model, reduces overall energy consumption by about 7% (from 3.62 J to 3.38 J), and energy due to beaconing by almost 30% (from 0.83 J to 0.59 J). On the other hand, LPSM with $1/\log T$ as its energy term, reduces overall energy consumption by nearly 20% (from 3.62 J to 2.95 J) and energy consumption due to beaconing by more than 80% (from 0.83 J to 0.16 J), while increasing slowdown over static PSM by only a factor of 1.19.

To verify the robustness of our results, we also ran longer simulations for LPSM with $1/\log T$ as the energy term. Figures 8 and 9 show results averaged over 4 independent 500-page experiments, each one starting the traffic simulator with a different random seed. We also ran static PSM in each of these four simulations, and averaged its results, for a fair comparison. This experiment shows similar results: energy savings of over 18% (from 4.12 J to 3.36 J) and an 82% reduction in energy consumption due to beaconing (from 0.94 J to 0.17 J), while increasing slowdown over static PSM by a factor of only 1.2.

Figures 6-9 also compare the performace of LPSM with BSD. We ran BSD in the same simulation for both scenarios described above, using the original BSD implementation [15], including averaging over the four runs with the same set of random seeds for traffic generation as used on LPSM. LPSM shows an energy reduction versus most settings of BSD, though higher slowdowns. Notably, the LPSM using $1/\log T$ to model energy, had deeper energy savings than all the previous BSD settings, though it increased slowdown more. It is important to note that LPSM can work well even if the distribution generating the observations of network activity changes with time. Since in simulation, traffic was generated using a stationary distribution, we would expect only better results against BSD in practice. Additionally, in both settings, we ran with m optimized for a timescale of 45 minutes. We could expect better results, especially in the longer trials, had it been optimized for the length of the trial. Yet these results lend validity to using LPSM even under resource limitations.

Moreover, LPSM offers several unique advantages over existing approaches. First, because LPSM's determination of appropriate sleep times is based on a loss function, rather than a single parameter, LPSM provides designers sufficiently more flexibility than BSD in exploring the energy/performance tradeoff. It should be possible to simulataneously reduce both energy and slowdown from that of previous approaches, by appropriate choice and calibration of the loss function. Second, because LPSM uses a learning algorithm designed to react well under nonstationarity, we would expect LPSM to perform better than BSD when the distribution generating traffic changes over time, a situation not modeled in this simulation, but which is realistic in practice. These hypotheses deserve further attention and present interesting possibilities for future work.

Figure 10 shows the slowdown behavior of various power-saving algorithms for individual Web page downloads. LPSM imposes only a moderate increase in slowdown over static PSM for various Web pages: only a 2% increase on average, but never more than 20%. Changing the loss function in LPSM to favor energy reduction more will obviously increase the slowdown, as shown in Figure 10(c), but even using this loss function never incurs a slowdown of more than a factor of 2 for any given Web page. For longer Web transfers, both LPSM algorithms introduce only negligible slowdown. Some shorter Web transfers impose longer slowdown times than static

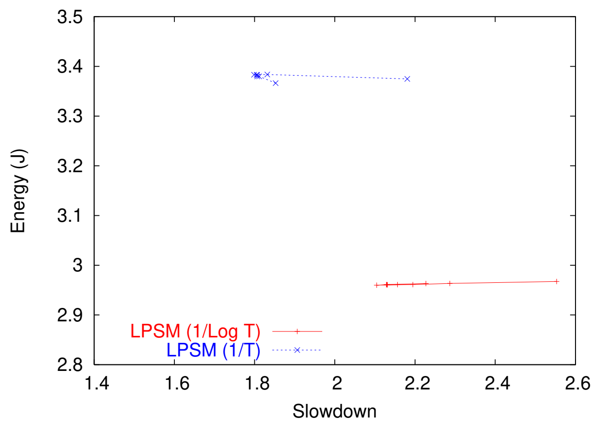


Fig. 11. Performance of LPSM with different loss functions. Modeling energy usage as $1/T$ is shown by the top curve, and as $1/\log T$ by the bottom curve. Points on a curve indicate varying settings of γ.

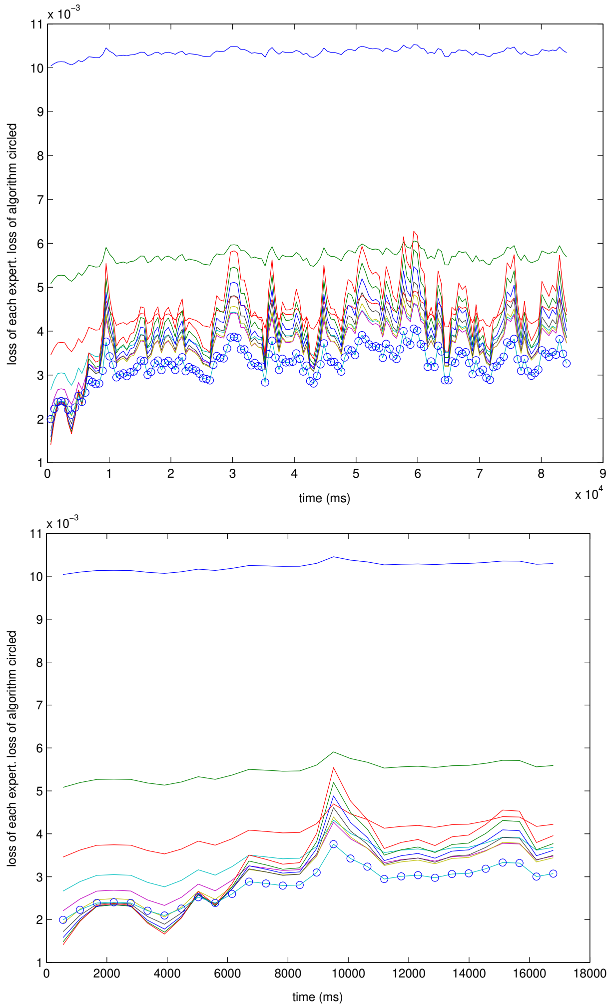


Fig. 12. These figures show the loss of each expert as a function of time. The circled path is the loss of the algorithm. The bottom figure zooms in on the earlier iterations.

PSM, but these download times are short anyway, so the additional slowdown that LPSM imposes is not drastic.

Varying the latency constraint in the Lagrange optimization (i.e., varying γ) and varying the functional form of the loss function give LPSM flexibility with respect to the tradeoff between energy and latency. For example, Figure 11 shows the performance of LPSM for various settings of γ and the loss function; each point in this figure represents a 100-page simulation experiment. Clearly, calibrating the correct γ value is important for achieving good slowdown performance for a given level of energy consumption. Additionally, varying the functional form has significant effects on the behavior of the learning algorithm: using $1/T$ as the energy term in the loss function achieves better slowdown performance than using $1/\log T$, at the cost of more energy consumption. Again, note that LPSM using either loss function allows significant energy savings over static PSM with only moderate additional slowdown.

B. Trace-based Analysis of LPSM and Online Learning Algorithms

We also ran LPSM on traces of real network activity [16]. This is different than the trace-driven simulation because the distribution for traffic generation was created by averaging over the traces [15]. So running the LPSM algorithm on the traces themselves, outside of a simulation, would allow it to observe real network activity, with potentially less stationarity of traffic than from the fixed distribution used in the simulation environment. We present these results to further illustrate the behavior of LPSM versus 802.11 PSM as well as other online learning algorithms in this problem domain. We first illustrate the behavior of `Fixed-share`. `Learn-`α then runs with the current best linear combination of the existing `Fixed-share` sub-algorithms' distributions over experts at any given time. The `Fixed-share` algorithm we illustrate below is running with $\alpha = 0.01$.

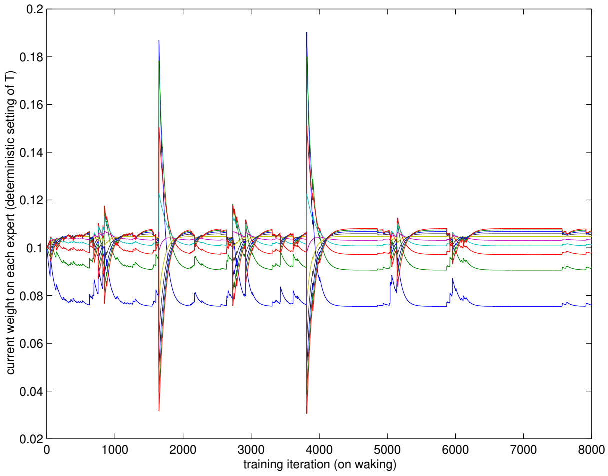
We used publicly available traces of network activity from a UC Berkeley home dial-up server that monitored users accessing HTTP files from home [16] (the same trace we used to synthetically generate Web traffic for the `ns-2` simulations). Because the traces only provided the start and end times, and number of bytes transferred for each connection, per connection we smoothed the total number of bytes uniformly over 10 ms intervals spanning its duration. In the network trace experiment results below, we ran with 10 experts spanning the range of 1000 ms at regularly spaced intervals of 100 ms, and $\gamma = 1.0 \times 10^{-7}$, calibrated to attain polling times within the range of the existing protocol.

1) Behavior of Online Learning Algorithms: Figure 12 shows the loss, on one trace, of each of the experts as a function of time, measured at the algorithm's current polling times, where the circled path is the loss of the algorithm. Since the algorithm allows for switching between the experts, it is able to maintain loss close to or better than the loss of the best current expert, even though our results show that the identity of the best current expert changes with time. The bottom graph, which zooms in on the early phases of the run, shows that it takes some time before the algorithm is doing as well as the best current expert. Note however that since the weights start as uniform, the initial polling time is actually the mean of the polling times of each of the experts, so even in the early iterations the algorithm already beats the worse half of the experts, and still tracks the average expert.

Figure 13 shows the evolution of the distribution that the algorithm maintains over experts.As expected, the algorithm allows for switching its weights on the experts, based on which experts would currently be performing best in the observed network activity. Changes in the burstiness level of the arriving bytes are reflected in shifts in which expert currently has the highest weight, starting even in the early learning iterations. The bottom figure, which zooms in on the earlier phases of learning, shows that the algorithm's preference for a given expert can easily decrease and increase in light of network activity. For example after several periods of being the worst expert, after iteration 1600, an expert is able to regain weight as the best expert, as its weight never went exponentially to zero.

These results also confirm that for real network traffic, no single deterministic setting of the polling time works well all the time. The algorithm can do better than any single expert, as it is a convex combination of all the experts, and the updates of the weighting allow the algorithm to track the expert that is currently best. Note that these online adjustments of which expert the algorithm currently favors are closely reflected in the evolution of sleep times graphed in Section V-A, Figures 4 and 5.

2) Competitive Analysis: To perform competitive analysis of the learning algorithm, we compare it to the hindsight best expert, as well as the current best expert at each time. Figure 14 illustrates how the learning algorithm does in relation to the best fixed expert computed in hindsight, and to the current best expert at each time epoch at which the algorithm does a learning update. The algorithm learns to track and actually do better than the best fixed expert, in the first figure. In the second figure, this scenario is an approximation to the best k-partition,(i.e. the best partition of the trace into k parts and choice of expert for each part) where k is equal to the length of the full time horizon of the trace. Note that as k increases, the optimal k-partition is harder to track,



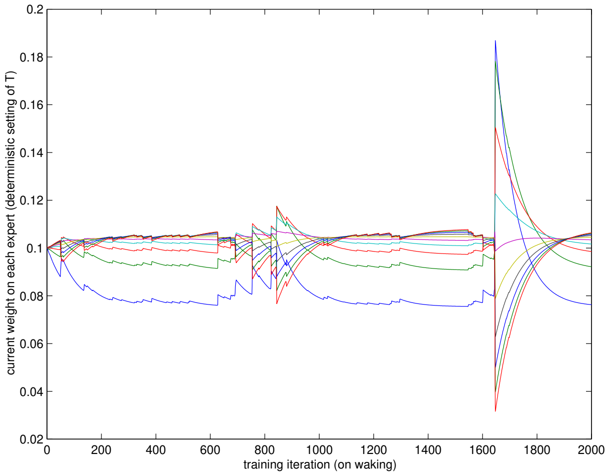


Fig. 13. These figures show the weights that the algorithm maintains on each expert, per training iteration. The bottom figure zooms in on the earlier iterations.

since it is the limit of non-stationarity of the process, so the above is close to the hardest partitioning. Here the algorithm does not beat the loss of the sequence of best experts, but is at least able to "track" it, as per the bounds [10], [11]. We see that the algorithm's performance is never too far away from that of the current best expert.

Even in the first figure, while it is unsurprising that the algorithm that allows switching between the experts can beat the single best expert, the intuition of why it performs better than the performance bound is as follows. The performance bound is a worst-case upper bound on the regret of the algorithm, which need not be met in practice. In the worst-case framework, an adversary gives the learning algorithm a bank of experts as a black box. The algorithm does not know anything about the true mechanism by which the experts make predictions. In our case however, in applying the algorithm, we were able to *choose* the set of experts to best apply to this problem domain. Thus by choosing experts that actually have a semantic:

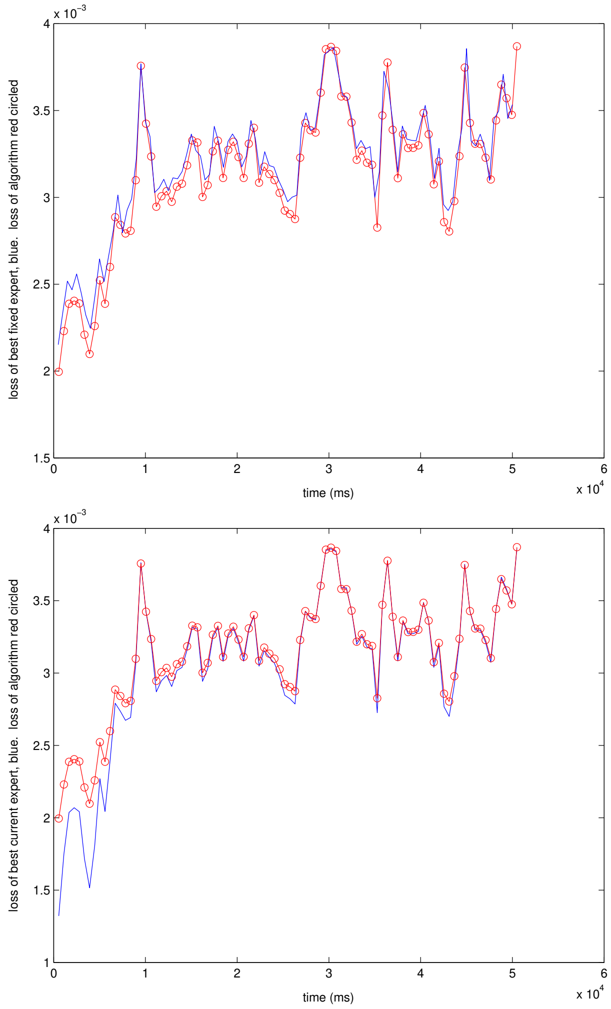


Fig. 14. Competitive Analysis. Loss of the algorithm versus time, circled. Solid is loss of the best fixed expert (top), and loss of the current best expert per training epoch (bottom).

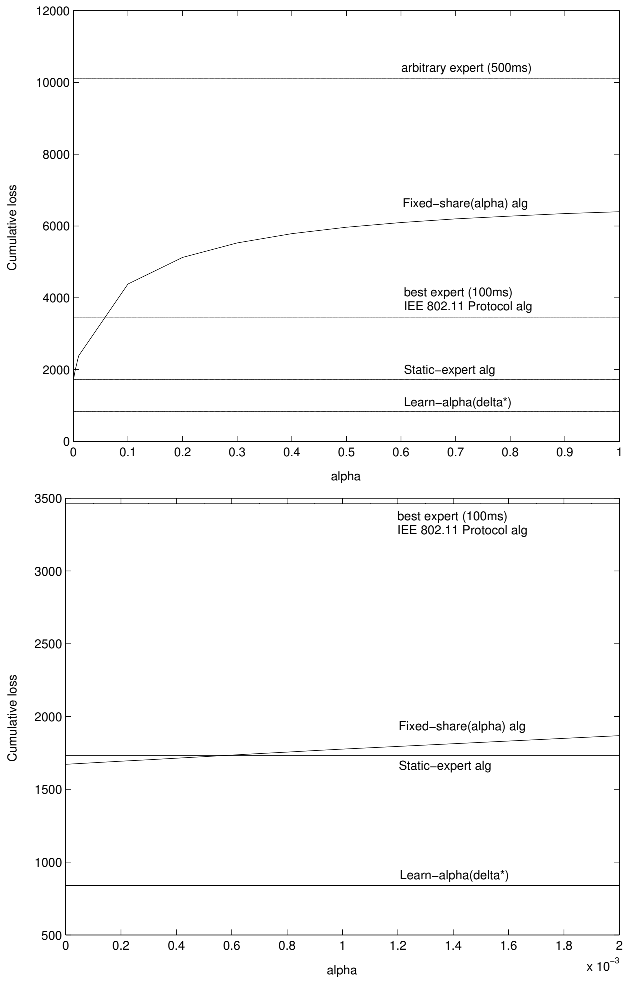


Fig. 15. Cumulative loss comparison. The curve indicates the cumulative loss of `Fixed-share` (α) along the range of α values. We compare to the cumulative loss on the same trace of the 802.11 PSM protocol, `Static-expert`, and `Learn-`α. The bottom figure zooms in on the first 0.002 of the range for α.

a discretization of the range of values we would like to consider for polling time, the algorithm can actually perform better than any of the individual experts, as it strives to be the best (cumulative loss minimizing) linear combination of the experts at any given timestep. Thus it is able to adaptively smooth over the discrete values that the experts represent, and perform with the current best smoothed value in the desired polling time range.

3) LPSM advantages: Now we focus on `Learn-`α, the algorithm used for LPSM, which maintains a bank of sub-algorithms of the type analyzed above. The strength of this algorithm is that it learns α online while performing the learning task, and the empirical gains of this are shown by the next figures. Figure 15 compares the cumulative losses of the various algorithms on a 4 hour trace, with observation epochs every 10 ms. This corresponds to approximately 26,100 training iterations for the learning algorithms. Note that in the typical online learning scenario, $\mathcal{T}$, the number of learning iterations, or time horizon, for which one would like to benchmark the performance of the algorithm, is just the number of observation epochs. In this application, the number of training epochs need not match the number of observation epochs, since the application involves sleeping during many observation epochs, and learning is only done upon awakening. Since in these experiments the performance of the learning algorithms (`Static-expert`, `Fixed-share`, and `Learn-`α) are compared by each algorithm using n experts spanning the range of 1000 ms at regularly spaced intervals of 100 ms, in order to get an *a priori* estimate of $\mathcal{T}$, we assume a mean sleep interval of 550 ms, i.e. the mean of the experts. It is important to note however that the polling times of each of the runs graphed are set by the algorithm being run, and so for the learning algorithms, the polling times change online.

Since we calibrated our objective function (by choice of γ) by calibration with the existing protocol, it is no surprise that the deterministic setting of 100 ms, used by the existing IEEE 802.11 PSM protocol, was the best expert. `Static-expert`, the online learning algorithm with relative loss bounds against the best fixed expert, achieved lower cumulative loss however, since it can predict with the best linear combination of the experts, as opposed to any one of their values, i.e., at each time-step it can choose the optimal smoothed value over the desired range, as opposed to being limited by the discretization imposed by the experts.

On this particular trace, the optimal α for `Fixed-share` turns out to be extremely low, and so for most settings of α, one would be better off using a `Static-expert` model. Yet as the second graph shows, there is a value for α below which it is beneficial to use `Fixed-share`, which highlights the strength of the `Learn-`α algorithm, since without prior knowledge of the stochastic process to be observed, there is no optimal way by which to set α.

VI. Discussion and Conclusion

We proposed the use of online machine learning to manage the well-known energy/performance tradeoff in wireless networks, and applied a recently developed machine learning algorithm, Learn-α, to the 802.11 wireless LAN PSM. The idea behind LPSM is to set up n "experts," each corresponding to a particular deterministic value of the sleep cycle, with a device sleeping for a duration equal to a weighted sum of these deterministic times. The weights of these n independent experts are updated according to a loss function, which for 802.11 combines the energy and latency losses incurred by waking up at any time.

The method of updating the weighting depends upon the algorithm's model of the level of non-stationarity of the observed process, e.g. current network activity. Unlike previous online learning algorithms, Learn-α does not take this quantity as a parameter that must be set beforehand. Instead it learns this quantity online: the "switching-rate" of current network activity, simultaneous to learning the best current polling time.

Our experimental results, based on trace-driven simulation and trace-based analysis of Web client traces, show that for a Web-like request/response workload, LPSM (using the particular loss function we chose) can save 7%-20% more energy than 802.11 in power-saving mode, with an associated increase in average slowdown by a factor of at most 1.2.

The performance of our algorithm tracks the performance of the best of its input candidate time values, or best sequence of such fixed candidate polling times, chosen for different segments of time. Based on the nature of these guarantees, in future work we propose to run simulations in which traffic is generated by a non-stationary stochastic process, as opposed to a stationary distribution, and compare the results with previous approaches.

Since the LPSM implementation we suggest is the same as 802.11 PSM, other than a polling time that changes adaptively, and remaining awake while the link is active, integration into 802.11 PSM would be relatively straightforward. In order to help synchronize between sleeping and awake nodes LPSM chooses sleep durations that are rounded to the nearest 100 ms multiple of the time computed by `Learn-`α.

The complexity of the algorithm, can be $O(mn)$, or $O(m+n)$ if parallel computation is supported, since the α-expert updates can be run in parallel. In comparison to related work, this is slightly more complex, but note that n, the number of candidate polling times, can be chosen as a small constant, without degrading performance, as it just defines the level of discretization for a fixed range of possible sleep times. To obtain tight performance guarantees, the optimal number of α-experts, m^*, is computed based on the timescale along which one would like to benchmark performance. However m, can also be held constant, when computation is an issue, and our empirical results verified that this did not significantly degrade performance.

We note that the loss function we use in LPSM may be improved in the future, to obtain a different tradeoff between energy and latency. In our idealized model, we assume that bytes arrive uniformly during the sleep interval. If the protocol for delivering packets to a node maintained the times at which the packets originally arrived, the loss function can be made more accurate. The fact that LPSM did rather well despite this simplifying assumption, augurs well for its performance when finer-grained information on arrivals is available.

Another extension to LPSM would be to estimate the energy that would be consumed in retrieving the buffered bytes, and factor that into the loss function as well. Note that the algorithm is modular, in that any objective function that is proportional to the energy/performance tradeoff may be used. The implementation methods mentioned above, which would not add too much complexity, should further improve the performance of LPSM over 802.11 PSM. We note that previous approaches to this problem did not save significant energy compared to 802.11 PSM, yet even our initial application of this algorithm was able to save substantial energy.

References

[1] IEEE, "Computer society LAN MAN standards committee.," in *IEEE Std 802.11: Wireless LAN Medium Access Control and*

Physical Layer Specifications, August 1999.

[2] R. Krashinsky and H. Balakrishnan, "Minimizing energy for wireless web access with bounded slowdown," in *MobiCom 2002*, Atlanta, GA, September 2002.

[3] L. Feeney and M. Nilsson, "Investigating the energy consumption of a wireless network interface in an ad hoc networking environment," in *INFOCOM 2001*, Anchorage, Alaska, April 2001.

[4] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris., "Span: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks," *To appear in: ACM Wireless Networks Journal*, vol. 8, no. 5, September 2002.

[5] C. Monteleoni, "Online learning of non-stationary sequences," in *AI Technical Report 2003-011, S.M. Thesis*, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, May 2003.

[6] C. Monteleoni and T. Jaakkola, "Online learning of non-stationary sequences," in *Neural Information Processing Systems 16*, Vancouver, Canada, December 2003.

[7] T. Simunic, L. Benini, P. W. Glynn, and G. De Micheli, "Dynamic power management for portable systems," in *Proc. ACM MOBICOM*, Boston, MA, 2000, pp. 11–19.

[8] E. Chung, L. Benini, and G. De Micheli, "Dynamic power management for non-stationary service requests," in *Proc. DATE*, 1999, pp. 77–81.

[9] C. Steinbach, "A reinforcement-learning approach to power management," in *AI Technical Report, M.Eng Thesis*, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, May 2002.

[10] N. Littlestone and M. K. Warmuth, "The weighted majority algorithm," in *IEEE Symposium on Foundations of Computer Science*, 1989, pp. 256–261.

[11] M. Herbster and M. K. Warmuth, "Tracking the best expert," *Machine Learning*, vol. 32, pp. 151–178, 1998.

[12] A. Blum, C. Burch, and A. Kalai, "Finely-competitive paging," in *IEEE 40th Annual Symposium on Foundations of Computer Science*, New York, New York, October 1999, p. 450.

[13] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "Gambling in a rigged casino: the adversarial multi-armed bandit problem," in *Proc. of the 36th Annual Symposium on Foundations of Computer Science*. 1995, pp. 322–331, IEEE Computer Society Press, Los Alamitos, CA.

[14] D. P. Helmbold, R. E. Schapire, Y. Singer, and M. K. Warmuth, "On-line portfolio selection using multiplicative updates," in *International Conference on Machine Learning*, 1996, pp. 243–251.

[15] R. Krashinsky, "Implementation of BSD and static PSM, and `ns-2` http traffic simulation," in *http://www.cag.lcs.mit.edu/∼ronny/wireless_psm/readme.html*, 2002.

[16] U.C. Berkeley, "U.C. Berkeley home IP web traces," in *http://ita.ee.lbl.gov/html/contrib/UCB.home-IP-HTTP.html*, 1996.