Avoiding the Rush Hours: WiFi Energy Management via Traffic Isolation

Justin Manweiler
Duke University
Durham, NC, USA
jgm@cs.duke.edu

Romit Roy Choudhury
Duke University
Durham, NC, USA
romit.rc@duke.edu

ABSTRACT

WiFi continues to be a prime source of energy consumption in mobile devices. This paper observes that, despite a rich body of research in WiFi energy management, there is room for improvement. Our key finding is that WiFi energy optimizations have conventionally been designed with a single AP in mind. However, network contention among different APs can dramatically increase a client's energy consumption. Each client may have to keep awake for long durations before its own AP gets a chance to send packets to it. As the AP density increases in the vicinity, the waiting time inflates, resulting in a proportional decrease in battery life.

We design *SleepWell*, a system that achieves energy efficiency by evading network contention. The APs regulate the sleeping window of their clients in a way that different APs are active/inactive during non-overlapping time windows. The solution is analogous to the common wisdom of going late to office and coming back late, thereby avoiding the rush hours. We implement SleepWell on a testbed of 8 Laptops and 9 Android phones, and evaluate it over a wide variety of scenarios and traffic patterns (YouTube, Pandora, FTP, Internet radio, and mixed). Results show a median gain of up to $2x$ when WiFi links are strong; when links are weak and the network density is high, the gains can be even more. We believe SleepWell is a desirable upgrade to WiFi systems, especially in light of increasing WiFi density.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*

General Terms

Algorithms, Design, Experimentation, Measurement

Keywords

802.11, AP, Beacon, Contention, PSM, Scheduling, WLAN

1. INTRODUCTION

Energy management in mobile devices continues to be a relevant problem. The problem is becoming pronounced, especially with the *always-connected* usage model of modern devices. Smartphones, for instance, are rapidly becoming the convergent platform for a large variety of network applications, including emails, music, videos, games, web browsing, and picture sharing [5, 13, 14]. In addition, background applications are continuously running push-based alert services [25], location based notifications [27], and periodic sensor updates [17]. This growth in network traffic is beginning to impose a heavy demand on the phone battery, to the extent that some users are already expressing dissatisfaction [10]. The inability to cope with the energy demands can be serious, and may even hinder the steady growth in the mobile computing industry.

WiFi network communication is a predominate source of energy consumption. This has been well known for many years, and a rich body of research has addressed the problems in various ways. For example, WiFi Power Save Mode (PSM) [1] is one of the early protocols that attempts to turn off the device whenever beneficial. While WiFi energy efficiency has progressively improved since PSM (with the most recent NAPman protocol [24] offering substantial gains), we find that there is still opportunity for improvement. We describe this opportunity by first describing the core ideas in PSM and NAPman, and then identifying their respective deficiencies.

Consider the scenario in which a WiFi AP intends to communicate to a battery-operated mobile client. With WiFi PSM, the client periodically wakes up to listen to advertisements from the AP. The advertisements include client identifiers for which the AP has queued packets. If a client C learns that the AP has packets for C, it wakes up the entire radio; otherwise, it continues sleeping in the low power state. Importantly, waking up the radio incurs a high energy cost, and hence, it is unproductive if the client downloads only a few packets after waking up. Therefore, to amortize the wake-up cost, PSM clients are made to wake-up less frequently, permitting multiple packets to queue up at the AP. Of course, such queuing introduces latency in PSM packet delivery. Nevertheless, since a large number of mobile applications (email, buffered video, push updates) are reasonably tolerant to latency, PSM correctly takes advantage of it. In current Nexus One phones running Android, the WiFi PSM mode wakes up in the orders of $300ms$ to download bursts of packets. This is a judicious design decision, with proven energy benefits.

Recently, authors in NAPman [24] showed the possibility of improvements with PSM. The core observation is that multiple clients (associated to the same AP) may wake up after an AP advertisement, and expect to receive their respective burst of packets. However, since the AP can transmit one packet at a time (in a round robin manner to each client), every client must remain awake for a longer duration to receive its packets. This is a source of energy wastage, and NAPman mitigates this through *virtualized APs*. Briefly, the key idea is to make each client believe that it is associated to a different AP, and thus, have their wake up windows staggered over time. The ideas from NAPman offer energy gains, while also improving the fairness among PSM and non-PSM clients.

We observe that NAPman improves PSM in the cases where an isolated AP is connected to multiple clients. In reality, multiple APs are within the wireless vicinity, and this strongly impacts the energy consumption of individual clients. Specifically, when a PSM client wakes up to download its own burst of packets, it has to share the channel with all other clients of all other APs in the vicinity. In homes or dense office areas, it is not unusual to overhear 5 to 10 other APs. Since the APs are likely to share the channel fairly between them, it is possible that a client remains awake almost 5 times longer, than it would if there was no contention with other APs. Thus, the energy wastage during network activity can be 5 times, and even more if other APs have multiple clients associated to them. We believe that PSM and NAPman can be significantly improved if the energy-wastage from network contention is alleviated. Mitigating network contention from the energy perspective is a relatively unexplored space, especially in the face of emerging applications and usage patterns. *SleepWell* is tasked to investigate and solve this problem.

The core idea in SleepWell is simple[1]. Briefly, since APs are always powered on, they monitor ongoing wireless traffic from nearby APs. Since PSM creates periodic bursts of traffic, each AP tracks the periodicity of other APs, and dynamically re-schedules its own period to minimally overlap with others. Reduced overlap reduces competition, allowing each client to download its own packets uninterrupted, and sleep when the channel is occupied by other transmissions. This bears resemblance to a distributed TDMA scheme, but executed with energy-efficiency in mind.

The main design challenges in SleepWell appear from: (1) distributedly scheduling traffic bursts to achieve quick convergence, (2) ensuring clients do not get disassociated during dynamic rescheduling, and (3) preserving channel utilization, latency, and fairness, even under traffic variation and node churn. SleepWell addresses these systematically, *while requiring no software changes at the client.* By carefully modifying the timestamps (as a part of the WiFi clock synchronization process), the SleepWell AP regulates the client's sleep and wake-up schedules. The client remains unaware of the changes in its own duty cycle; neither does it get disassociated. 802.11a/g/n standard-compatibility remains intact.

We have implemented SleepWell on a testbed of 8 laptops and 9 Nexus One phones running the Android OS. Performance results show that energy reductions vary between 38% to 51%, across a variety of real online applications, including YouTube, Pandora and Last.fm Internet radio, and TCP bulk data transfer (e.g, FTP). Moreover, as the quality of links degrade, i.e., each packet is transmitted at lower bit rates (longer time), the relative energy gains improve. In light of these results, we believe that SleepWell may be an effective solution for the future, not only to sustain a demanding suite of applications, but also to improve "immunity" to increasingly dense WiFi environments.

Our main contributions may be summarized as follows:

- **Characterize the problem of network contention and its impact on energy consumption.** Through measurements we show that the energy-wastage is severe, especially with high device densities in the environment.
- **Design a lightweight, standard-compatible system running at the AP, that isolates traffic to reduce contention.** The system requires no changes to the client, and can quickly adapt to changing traffic conditions and node churn.
- **Implement and evaluate the system on a testbed of 8 laptops (acting as APs) and 9 Android Nexus One phone clients.** Promising performance improvements provide confidence that SleepWell can be an important step towards energy management in WiFi-enabled mobile devices.

The rest of this paper expands on each of these contributions. We motivate the SleepWell design through measurements in Section 2, followed by the system design in Section 3. The system implementation and evaluation are presented in Section 4, while limitations and future work are discussed in Section 5. Section 6 surveys the related work, and the paper concludes with a brief summary in Section 7.

2. BACKGROUND AND MEASUREMENTS

We motivate SleepWell through measurements. In this section we (1) discuss our choice of platform and measurement set-up, (2) introduce the terminology for PSM operation, (3) profile the PSM behavior of a state-of-the-art smartphone, and (4) present measurement results suggesting that network contention has a dramatic impact on energy consumption, and correspondingly, battery life

Choice of Device

For the experimentation platform, we planned on choosing a state-of-the-art mobile device that would satisfy three conditions: (1) provide accessible battery contacts to connect the device to the power monitor; (2) have up-to-date WiFi hardware, including 802.11n; (3) be supported with chipsets/drivers that optimizes for device energy. Natural candidates are the Apple iPhone (version 4 supports 802.11n), highest-end HTC/Android phones, or Windows Mobile smartphones. The iPhone is unsuitable for testing due to its self-enclosed battery design [24]. The Android-based Google Nexus One seemed attractive. In particular, documentation for the Nexus One's Broadcom BCM4329 802.11a/b/g/n chipset claims "technologies to reduce active

[1]We rejected a number of involved designs, thereby potentially trading off some performance for standard-compliance and scalability.

and idle power consumption", including an on-chip power management module. We also observed that the Android OS performs *adaptive PSM*, intelligently switching between different power modes, based on (1) whether the screen is on; (2) traffic load; (3) the beacon interval, etc. Finally, when compared with a Windows 6.5 HTC phone, both network performance and energy efficiency were better with Nexus Ones. In light of these, we deemed the Android Nexus One smartphone as our measurement platform of choice.

Measurement Set-up

To measure energy consumption in the Nexus One phones, we used two power monitors from Monsoon Solutions [18]. The probes from a monitor were connected to a hand-engineered copper-wire extension of the Nexus One lithium-ion battery as shown in Figure 1. The Nexus One battery has four terminals, the outer two are respectively labeled as positive (+) and negative (−). The corresponding positive/negative terminals on the phone are connected to the meter by copper tape and the red/black DC leads. The positive battery terminal is insulated with non-conductive tape to ensure that current is only drawn through the meter. The battery is reinstalled in the unit to allow connection with the inner terminals (used to report diagnostic information to the phone). We sanity-checked this set-up by comparing our basic measurements with hardware data-sheets and other surveys in literature [24].

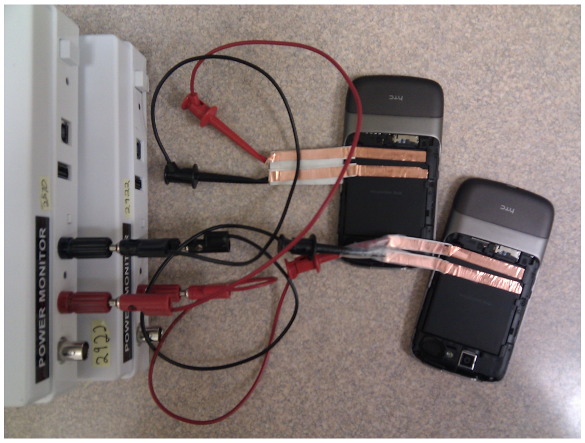


Figure 1: Shows experimental set-up with Nexus One phone connected to power meter via copper tape and DC leads. The phone is entirely powered by the power meter. The computer, connected via USB, records current and voltage at 5000 hertz.

Terminology

To ground our discussion of WiFi power consumption, it is necessary to consider how a modern, power-optimized device uses 802.11 networks from an energy perspective. We first review relevant terminology in Table 1. We will use this terminology in the subsequent discussion.

PSM Energy Profiling

We survey the energy behavior of PSM, as it dwells at (or transitions between) different power levels in response to network activities. The power values are taken with the screen off, WiFi associated to a nearby AP, bluetooth/GSM/3G radios disabled (airplane mode), and minimal background application activity. While our analysis is primarily grounded on our Nexus One measurements, we observed similar behavior on an older Windows Mobile device (albeit with different exact power drawn).

Table 1: WiFi Power Save Terminology

WiFi Power Save Mode (PSM):
A suite of polling-based power-optimizations specified by the IEEE 802.11 standard and incorporated in all WiFi implementations [1].
Constant Awake Mode (CAM):
When a PSM-capable WiFi device temporarily disables PSM to minimize latency for interactive traffic.
Beacon Interval:
Fixed time duration between two successive AP beacons, typically $100ms$ (APs continuously transmit these beacons).
Traffic Indication Message (TIM):
Virtual "bitmap" embedded in every AP beacon. Indicates which PSM clients should poll to receive queued unicast download packets.
Listen Interval:
How often a client chooses to wake up to listen for one AP beacon. Listen intervals are an exact multiple of the beacon interval.
PS-Poll:
Client notification to its AP that it is awake and ready to receive a queued packet. Issued immediately after a client recognizes its own ID in a TIM.
More Data Flag:
Flag embedded in download unicast data packets that specifies whether more data packets are queued at the AP for a PSM client. Once this flag is set to false, the client may immediately return to sleep.

Figure 2 shows the anatomy of a Nexus One PSM client, tasked to stream music from the Pandora service (Figure 2(a) shows a screenshot for the power meter, while Figure 2(b) zooms into a time segment of the measurement). At the beginning, the radio is in **PSM Deep-Sleep**, at $\approx 10mW$. In this mode, clients are only able to wakeup and receive scheduled beacons from an associated AP (wakeups shown by spikes that reach up to $\approx 250mW$). Once a client receives a TIM advertisement notifying pending traffic (around t=$1.2s$), it transitions to the highest power level and sends a PS-Poll to retrieve a queued packet. This transition from deep-sleep to **High Power** state incurs an additional wake-up energy cost ($\approx 600mW$). The client then receives a burst of data packets and responds with ACKs (around t=$1.25s$), all of which also incur high energy.

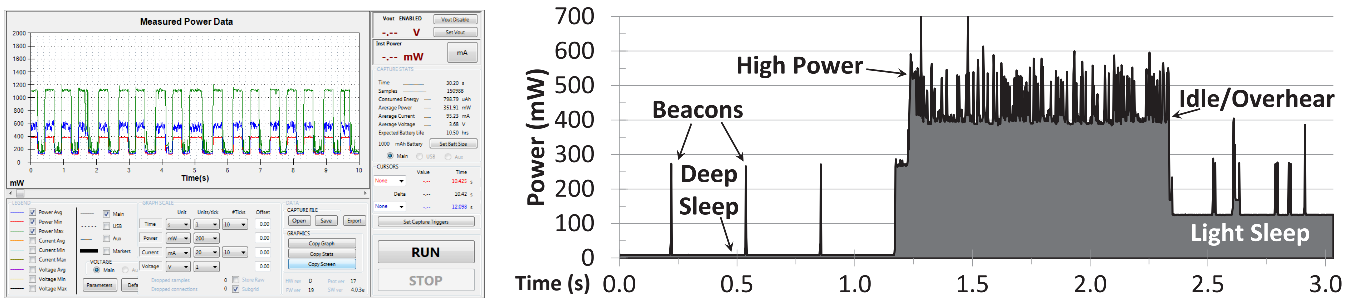


Figure 2: (a) Screenshot from Monsoon power meter; (b) Power draw over time for Pandora music streaming.

Now, while waiting for the next packet, the client cannot power down because the radio hardware continues to "overhear" packets from other APs/clients. Thus, the client dwells in this **Idle/Overhear** state, periodically transitioning to high power for transmitting/receiving its own packets. The cost of overhearing ($\approx 400mW$) is less than receiving because overheard packets are dropped at the radio, saving computation energy higher up in the protocol stack.

Once the queued packets have been downloaded (and the AP has disabled the `MORE_DATA` flag indication), the client does not go back to deep-sleep immediately; it attempts to amortize the wake-up/shutdown cost over multiple download bursts—with a guess that more traffic will soon follow. The client transitions to **PSM light-sleep** ($\approx 120mW$) in anticipation of efficiently waking up for subsequent bursts. In this state, it continues to periodically wake up and receive AP beacons, but is "deaf" to contending traffic. Later, once enough time has passed since the last download, the client shuts down to deep-sleep. Shutting down to deep-sleep (not shown) also incurs a high energy cost, similar to waking up.

Observe that these dwell-times and transitions between energy levels fundamentally define the energy-efficiency of the system. Where appropriate, we will show how SleepWell alters the PSM behavior so that it dwells longer in lower-energy states, while remaining as agile with respect to upload/download packets.

Impact of Network Contention on Energy

Energy-consumption is a function of a large number of parameters (hardware, traffic, bit rates, mobility, topology, density, etc.). Measuring over all permutations of this parameter space is difficult. We have narrowed down the space to a smaller set of common-case scenarios, and report measurements from them.

Methodology: We used a mix of Dell and Lenovo laptops as APs. We deployed fully-connected AP/client topologies to ensure that measured power from any client would be reflective of all other clients in the network. Monsoon Power monitors were attached to two arbitrarily-chosen Nexus One phone clients. Presented results reflect measurements taken from both phones. Link bitrates made to mimic measured link qualities from the Engineering building at Duke University, between 48 and 72 Mbps (recall that the Nexus One supports 802.11n bitrates). To generate realistic traffic, we used a software tool called `Tcpreplay` [30]. This tool allowed us to record packets for any arbitrary Internet download, and later *replay* the packet arrival sequence and timing within our local testbed. This allowed for repeatable experiments across different types of traffic, including `Pandora` and `Last.fm` music streaming and `YouTube` videos (Table 2). We also generated some synthetic TCP traffic using `Iperf`, representative of bulk data transfer, such as an FTP or HTTP download.

Table 2: Experimental Traffic Workload

Trace	*Packets / s*	*Bandwidth*
`Iperf`	*Link Saturation*	
`YouTube`	48.8 pps	555.9 Kbps
`Pandora`	14.3 pps	160.2 Kbps
`Last.fm`	13.6 pps	153.0 Kbps

Results: Figure 3 shows the variation of total energy consumption with increasing network contention (all flows performing TCP downloads). With increasing number of AP-client links, and correspondingly elevated channel saturation, PSM clients are forced to stay awake in the idle/overhear mode ($\approx 400mW$) for longer proportions of time. Thus the energy required to complete the same network workload increases.

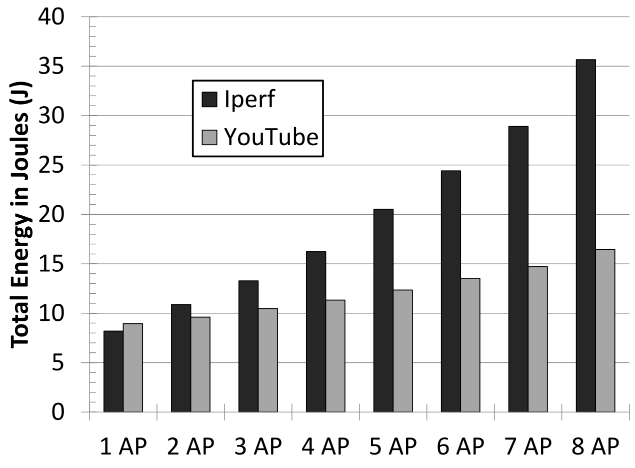


Figure 3: Energy consumed by a client under bulk data transfer and YouTube replay with varying contention (i.e., increasing number of APs in the vicinity).

Figures 4 zooms into the results and breaks down the proportion of time spent at each energy level. Also, we separate out the traffic patterns – an 8 MB TCP bulk data transfer (measured with Iperf) and a YouTube session via Tcpreplay[2]. To present an unbiased result, Figure 4(a) captures 90s of bulk download measurement, even though all transfers completed before 90s. This accounts for the system-wide deep-sleep energy consumed after a transfer completes. For YouTube (Figure 4(b)), we highlight a 60s portion of the trace, covering periods of buffering and playback. There was no network activity during playback, hence, clients were in deep-sleep by the end of the trace. Even with a fair balance between wake-up and sleep, we see that network contention forces a client to spend a much lower fraction of time in the efficient PSM sleep modes. SleepWell is designed to evade network contention, returning clients to light-sleep mode as quickly as possible.

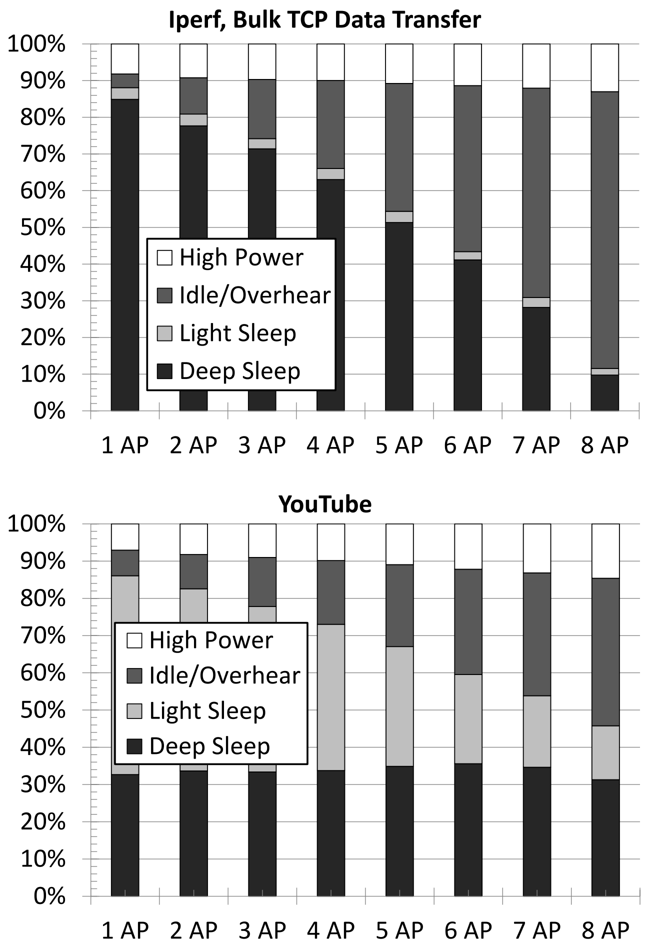


Figure 4: Proportion of time spent in each power activity level. (a) 8 MB TCP Iperf; (b) YouTube w/ Tcpreplay.

3. SLEEPWELL DESIGN

The SleepWell design firmed up after multiple rounds of testing and modifications. To convey some of the rationale in the final design, we first describe a basic version of SleepWell under the following assumptions.

1. All APs have saturated traffic.
2. Each AP has one client.
3. All APs are running SleepWell.

Thereafter, we relax the assumptions, and modify SleepWell to be applicable to real-world networks.

3.1 Basic SleepWell

SleepWell has 3 main modules, namely, (1) *traffic monitoring*, (2) *traffic migration*, and (3) *traffic preemption*.

(1) Traffic monitoring.
At bootstrap, each AP behaves similar to standard 802.11 – when their respective PSM clients wake up, the APs contend for the channel and send packets to them. However, a SleepWell AP also listens for ongoing beacons, and identifies which other APs are within its collision domain (we consider hidden terminals later in Section 5). Observe that beacons are transmitted at base rate, and hence, are audible over the carrier sensing zone of an AP [31]. Each AP assimilates this information into a traffic map that captures when each of its contending APs start their beacon intervals. The maps can clearly be different at different APs, depending on the AP's neighborhood. Figure 5 shows an example topology, and the corresponding maps assimilated by AP1 and AP3 (AP2's map is identical to AP1). Since PSM transmission bursts will immediately follow a beacon, these bursts are likely to overlap, forcing APs to waste energy due to traffic contention. SleepWell aims to avoid this contention through traffic migration.

(2) Traffic migration.
Given n other contending APs in the traffic map, each AP computes its *fair share* of the channel. The fair share is expected to be at least $\frac{1}{(n+1)}$. Each AP also computes its *actual share* of the channel as the time from its beacon to the immediate next (in the clockwise direction). If an AP's actual share is less than its fair share, and assuming that the AP has saturated traffic, the AP is said to be *unsatisfied*. Now, each unsatisfied AP looks into its traffic map and finds the largest inter-beacon interval, not including its own beacon – denote the start and end points of this interval as T_{start} and T_{end}. If this interval is twice that of the AP's fair channel share, then the AP moves its own beacon to the mid-point of this interval. However, if the interval is shorter, the AP migrates its beacon to a time T, such that $T_{end} - T = \frac{1}{(n+1)}$. Essentially, every SleepWell AP greedily migrates its traffic, claiming at least its fair share from the largest available interval. If this migration encroaches on another AP's traffic and fair share, the other AP should also attempt to migrate. On the other hand, if there is more time available, the AP shares the excess equally with the AP which owns the now-preceding beacon. We present an example to better capture the operation.

Consider Figure 5. All SleepWell APs use a 100ms beacon interval (denoted by a circle). For AP1's network view, the fair share is $\frac{100}{3} = 33.33ms$, and the length of the largest segment is $84ms$ (i.e., $T_{start} = 16$ and $T_{end} = 0$). Assume AP1's beacon is currently at 70. Thus, AP1 moves its PSM

[2]Results from `Pandora` and `Last.fm` music streaming (not shown) are consistent with `YouTube`, albeit with some variations in energy levels due to lower packet injection rate from the server.

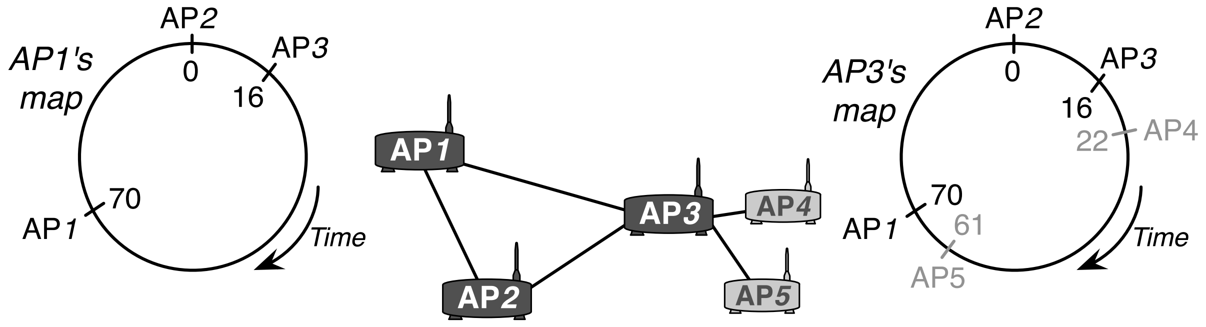


Figure 5: AP1 and AP3's traffic maps during bootstrap (AP2's map, not shown, is identical to AP1's). The circle denotes one BEACON_INTERVAL of $100ms$. The ticks on the circle denote when an AP has overheard beacons from other APs, as well as the time of its own beacon. The traffic maps clearly depends on the neighborhood.

beacon from 70 to 58, the mid-point of the largest segment (Figure 6(a)). AP3 observes the new position of AP1 and prepares to make its own move. AP3's fair share is $\frac{100}{5} = 20ms$, and largest segment is 39 (i.e., $T_{start} = 61$ and $T_{end} = 0$). Since the largest segment is less than twice of the fair share, AP3 migrates from 16 to 80 (Figure 6(b)), thus claiming its fair share, and forcing AP5 to move from time 61. Similarly, AP2 observes AP1 at 58 and AP3 at 80, and moves to the mid-point of 80 and 58, which is 19 (Figure 6(c)). Observe that from AP1 and AP2's perspectives, the traffic map begins to exhibit more uniformity in beacon separation. AP3's neighbors also perform the same operation (not shown), making AP3's traffic map uniform as well.

Over time, we expect all APs to converge to a reasonably uniform traffic map, thereby reducing the energy wastage from contention. In some cases, cyclical re-adjustment patterns may slow or break convergence, especially in large, dense network graphs where adjacent APs share highly-divergent views of the local neighborhood. To recover, we detect such cases, and trigger a randomization step – poorly converged nodes temporarily assume a random beacon assignment, breaking the non-converging cycle. Of course, this is a heuristic and may not converge to the optimal solution (the optimal beacon positioning corresponds to a TDMA schedule, and is thus NP-Complete [22], reduced from graph coloring). However, Monte Carlo simulations of 10,000 topologies and traffic patterns show that convergence is quick, reliable, and results in substantially better beacon placements than random assignment. We report these results in Section 4.

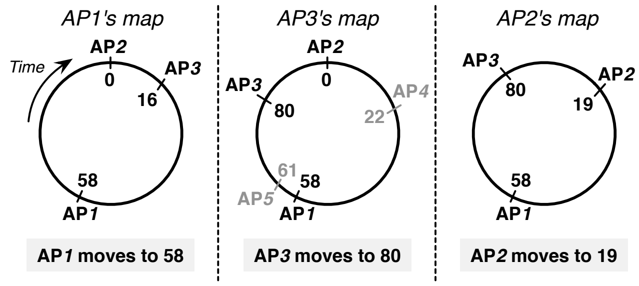


Figure 6: APs 1, 2, and 3 migrate their traffic per the SleepWell heuristic. Over time, the beacons are spread in time, alleviating contention between APs.

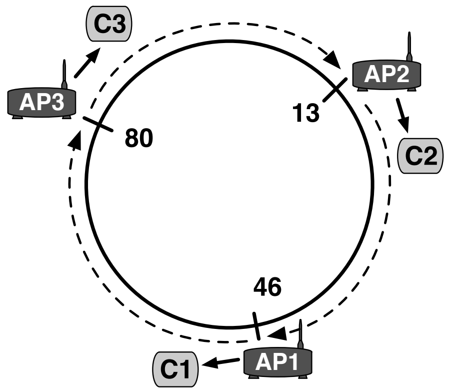


Figure 7: SleepWell APs distributedly stagger their beacons to reduce contention. Each AP preempts its traffic to honor another AP's schedule.

(3) Traffic preemption.

With 802.11, a client wakes up at the PSM beacon times and downloads packets until its AP turns off the `MORE_DATA` flag, indicating no more traffic. Continuous downloads at different APs induce continuous contention, resulting in significant energy wastage. Spreading the PSM beacons apart, as performed by SleepWell, will evade contention for some time, but the bursts will soon "spill" into the next bursts, reintroducing contention. To avoid this, SleepWell employs a simple preemptive mechanism. When AP_i observes that its traffic is likely to "spill" into AP_j's, it turns off the `MORE_DATA` flag in the subsequent data packet, forcing its client to go to sleep until the next listen interval. This permits AP_j's transmissions to progress without competition, reducing time to completion. When AP_i's client wakes up at the next PSM beacon, AP_i transmits the pending packets. Now the other APs preempt their respective transmissions, allowing AP_i to use the channel without contention. This is indeed a loose form of TDMA, where clients "avoid the rush hours" and sleep instead. Figure 7 shows the steady state operation with an example.

Observe that AP_i need not always preempt its traffic for AP_j. It is possible that AP_j's client is not awake in the same `BEACON_INTERVAL` as AP_i's (recall that PSM clients periodically wake up every `LISTEN_INTERVAL`, e.g., once in 3 BEACON_INTERVALs for Nexus One phones). In such a scenario, AP_i should detect the opportunity and use up AP_j's slot. Of course, the detection mechanism needs to be robust to ensure that AP_i does not mistakenly encroach into AP_j.

SleepWell is designed to handle this situation. When AP_i approaches AP_j's time slot, it looks for (1) any PS-Poll from any of AP_j's PSM clients; (2) AP_j's download packets with the MORE_DATA flag enabled; or (3) an ACK from one of AP_j's clients. (1) and (2) may not be always feasible as high bitrate transmissions may prevent overhearing at AP_i. (3) is more robust, as ACKs are transmitted at a lower bitrate, often at half the transmission rate of the preceding unicast packet. In case all of these techniques fail, SleepWell defaults to a simple inference scheme. AP_i looks into AP_j's prior beacons to see if AP_j has pending traffic for any of its clients (recall that the beacon TIM embeds pending traffic information). When the TIM is not set, AP_i does not preempt its own traffic, and continues transmission through AP_j's slot. This ensures channel utilization.

This concludes the description of Basic SleepWell under the three assumptions of saturated traffic, single client, and no legacy APs. We now relax these assumptions to fit real world scenarios.

3.2 Coping with Traffic Dynamics

The traffic migration heuristic in Basic SleepWell has deficiencies. While, upon convergence, each AP certainly receives its fair share of the channel, the heuristic may not cope well with dynamic traffic offered by different APs. The issues arise in 2 main scenarios: (1) Consider the case where an AP has n neighbors, but one of its neighbors has $m > n$ neighbors (e.g., in Figure 5, AP1 has 3 neighbors, but AP3 has 5). Here, AP1 could be satisfied by $1/3$ channel share, assuming that AP2 and AP3 will also consume $1/3$ each. However, AP3 may only be able to consume $1/5$, opening up some *slack* in channel time. Basic SleepWell may not be able to consume this slack. (2) In the same topology of Figure 5, if AP2 has little traffic (requiring less than $1/3$ channel time), Basic SleepWell will again fail to exploit this slack. We modify SleepWell to better "absorb" the slack, and thereby cope with dynamic traffic patterns.

Algorithm 1 shows pseudocode for the modified SleepWell. Although the pseudocode seems involved, the key idea is simple. In face of varying traffic demands, we require SleepWell APs to advertise the minimum of the *needed* channel share and the *available* channel share. In Figure 5 for instance, AP3 will advertise $1/5$ if it has adequate traffic to fill up its own slot. Otherwise, if it has queued traffic only for, say $1/7$ channel time, it advertises $1/7$. Knowing this information, the traffic map can be updated to additionally reflect the burst following each PSM beacon. This facilitates efficient traffic migration. SleepWell now computes the maximum slack interval as the separation between the end of the i^{th} burst to the $i+1^{th}$ beacon. Now, the actual migration rule also changes. If AP1 recognizes that AP3 is taking up less than its fair share, then AP1 computes the slack and attempts to redistribute it among APs that need more. In this case, the slack is $\frac{1}{5} - \frac{1}{7} = \frac{2}{35}$, which distributed between AP1 and AP2 becomes $\frac{1}{35}$. AP1 updates its fair share as $\frac{1}{3} + \frac{1}{35} = 0.36$. Using this fair share, and the largest interval computed from burst-to-next-beacon, SleepWell migrates its traffic according to the original rule. AP2 does the same, and the system is expected to still converge. If AP3 later changes its traffic advertisement, or an AP joins or leaves the network, AP1 and AP2 can adapt accordingly.

Algorithm 1 Traffic-Aware Beacon Adjustment

```
1: Input: $P$: Set of all peer APs
2: if $satCounter >$ CONVERGENCE_THRESHOLD then
3:     $satCounter \leftarrow 0$
4:     $newBeaconTime \leftarrow$ Rand() $\cdot$ BEACON_INTERVAL
5: else
6:     $satCounter \leftarrow satCounter + 1$
7:     $fairShare \leftarrow$ BEACON_INTERVAL$/(|P|+1)$
8:     $share \leftarrow fairShare$
9:     $gapEnd \leftarrow$ BeaconTime(.) + TrafficAdvert(.)
10:    $slack \leftarrow 0$
11:    for all AP $p_1 \in P$ do
12:        $gap \leftarrow$ BEACON_INTERVAL
13:        for all AP $p_2 \in P$ do
14:            $s \leftarrow$ BeaconTime($p_2$) - BeaconTime($p_1$)
15:            $gap \leftarrow \min(s, gap)$
16:        $midpointGap \leftarrow gap/2$
17:        $trafficGap \leftarrow gap$ - TrafficAdvert($p_1$)
18:        $slack \leftarrow \max(trafficGap, slack)$
19:        $available \leftarrow \max(midpointGap, trafficGap)$
20:        $available \leftarrow \min(available, gap - \epsilon)$
21:        if $available > share$ then
22:            $share \leftarrow available$
23:            $gapEnd \leftarrow$ BeaconTime($p_1$) + $gap$
24:    $expectedShare \leftarrow fairShare + slack/|P|$
25:    $share \leftarrow \max(expectedShare, share)$
26:    $newBeaconTime \leftarrow gapEnd - share$
27: Return $newBeaconTime$
```

3.3 Seamless Beacon Re-adjustment

In describing Basic SleepWell, we assumed that APs can re-adjust beacons at will. This is non-trivial in actual systems because 802.11 PSM does not have provisions to inform the client about new beacon timings. Incorporating this capability would require client-side changes, and SleepWell intends to avoid it. Existing schemes such as NAPman [24] employ virtualized APs [9], a method that makes a single AP advertise multiple beacons with different SSIDs. This effectively defeats passive scanning; clients must actively scan for APs by issuing a PROBE_REQUEST for a known SSID, and then re-associate to the BSSID of the virtual AP. To re-position a client again, the existing association must be dropped, forcing the client to re-associate through active scanning again. This is a heavyweight process, with significant time spent in the idle/overhear and high power activity levels, exacerbating the energy consumption in clients. Prior work has shown that the cost of associations can dominate the energy consumption in WiFi [6].

SleepWell APs can re-position clients without client-side changes and re-associations. The key idea here is to manipulate the TSF timestamps in advertised beacons. As mandated by the 802.11 standard, clients treat these timestamps as authoritative, and correspondingly update their clocks to a new beacon schedule. By advertising different beacon schedules to different clients, SleepWell APs move clients between beacons until a desirable distribution is reached. Consider the example in Figures 5 and 6 where AP1 intends to move its client from 70 to 58. Given that AP1 and its client are clock-synchronized, AP1 advertises the time as $12ms$ ahead of its current time. The client updates its clock accordingly. At absolute time $t = 58$ the client believes that it has reached $t = 70$, and wakes up to receive packets. Now, the AP also transmits a beacon at 58, effectively re-positioning its client seamlessly. We believe this technique is lightweight and scalable, and may be useful to other protocols (including NAPman [24]) that require traffic isolation among clients.

3.4 Multiple Clients per AP

For ease of explanation, we assumed one client per AP in Basic SleepWell. In reality, SleepWell can operate seamlessly with multiple clients associated to the same AP. Specifically, when AP_i has its transmission slot, it will transmit to each of its clients in the way the packets are queued up for them. At the cost of a little more complexity, the AP can create additional beacon schedules within its own time slot, ensuring that its own clients do not contend with each other. This can be accomplished by performing the same SleepWell beacon adjusting operations. Of course, this is aligned with the core beacon staggering ideas in NAPman [24], and hence, we do not claim this to be SleepWell's contribution. Nonetheless, SleepWell's technique of lightweight beacon re-adjusting can make this inter-client scheduling more efficient than NAPman. We briefly outline the mechanism next.

Inter-client SleepWell: The goal here is to disperse clients across different beacons. SleepWell APs predict when their own clients are expected to wakeup for a beacon (depending on the per-client `LISTEN_INTERVAL`). Since not all clients will wakeup within the same beacon interval, there can be opportunities to direct some clients to change their beacon wakeup schedules independently from other peers. However, when a pair of clients systematically share the same wakeup schedule, SleepWell unicasts an extra beacon to a particular client. This produces the desired schedule change.

3.5 Compatibility with adaptive-PSM clients

SleepWell is unaffected to serve adaptive-PSM clients. Like those of traditional PSM clients, adaptive PSM downloads occur in a regular burst pattern, immediately following AP beacons. Also, it is mandatory for all clients to wake up at least once per `LISTEN_INTERVAL` to receive their AP's beacon TIM advertisement. Thus, if data is available for the client, it will initiate a stream of download packets from the AP (either through a traditional `PS-Poll` request or by temporarily disabling PSM). As with regular PSM clients, this periodic, bursty behavior is obvious and easily predictable, thereby enabling SleepWell to converge on traffic-aware beacon schedules.

4. EVALUATION

In this section, we present the implementation of SleepWell and evaluate its performance.

4.1 Implementation

We implemented SleepWell as a set of modifications to the open source `ath9k` driver for Atheros 802.11n PCI/PCI-express interfaces. Driver-level modifications were required to (1) enable dynamic adjustment of beacon timing; (2) control TSF clock values advertised in beacons; (3) enable driver interrupts to quickly receive overheard beacons and packets from adjacent BSSes; and (4) exert timely control on the `MORE_DATA` flag for outbound traffic. Our implementation provides complete support for dynamic beacon adjustment, traffic migration and preemption, and multiple staggered beacons per AP[3].

4.2 Methodology

Experimental setup was consistent to that described for our earlier measurement set-up in Section 2. We deployed SleepWell on a testbed of 8 Dell and Lenovo laptops, serving as WiFi APs. Laptops were configured with Atheros chipset D-Link DWA-643 ExpressCard 802.11n WLAN interfaces. Linux kernel 2.6.3 with the `hostapd` daemon provided 802.11-compliant AP association support. Unmodified Nexus One smartphones (Broadcom BCM4329 802.11n WiFi chipset [7]) served as clients. In most tests, we tasked up to 7 AP/client pairs using Iperf TCP to create background traffic. While testing different applications (e.g., YouTube), `Wireshark` recorded packet traces, and `Tcpreplay` replayed them for different experiments (traffic characterized in Table 2). To closely model real-world behavior, we kept the phone screen on during trace collection. When replaying this trace, we turned off the screen to precisely measure the power draw.

4.3 Performance Results

Our evaluation attempts to answer the following:

1. Correctness of beacon adjusting heuristic (Fig. 8).
   Convergence via Monte Carlo simulation (Fig. 9).
2. Overall energy gain for different traffic patterns (Fig. 10).
3. Impact of network contention (Fig. 11, 12).
   Gap from the optimal case of zero-contention (Fig. 13).
4. Impact of link quality (bitrates) (Fig. 15).
5. Impact on throughput, latency (Fig. 16, 17).
   Impact on beacon spacing (Fig. 18).
6. Impact on fairness (Fig. 19).

(1) Beacon adjustment and convergence

Figure 8(a, b) shows a zoom-in view of how 2 clients (associated to distinct SleepWell APs) adjust their beacons and preempt traffic to converge on to non-overlapping traffic bursts. The graph is a 4 second segment of a TCP download. Each client periodically wakes up and stays active in the high power state, while the other remains in light-sleep during that interval. Figure 8(c) contrasts this behavior with 802.11.

[3]Due to interrupt timing limitations of the Atheros hardware, we cannot reliably support staggered beacons closer than $12ms$ of spacing (a driver interrupt must occur at least $2ms$ after a beacon and $10ms$ before the next beacon). Therefore, we can support a maximum of 8 beacons per beacon interval.

Under the same experiment settings, each 802.11 client stays awake continuously, sharing the channel with the other in fine time scales. Clearly, this is a source of energy wastage, and SleepWell mitigates it.

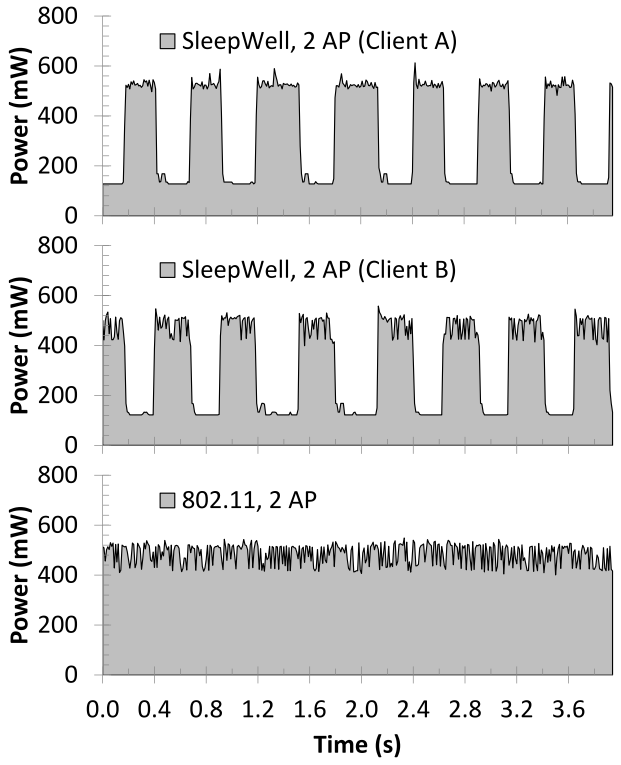


Figure 8: (a, b) 2 SleepWell clients converge to non-overlapping activity cycles, one sleeping when the other is active. (c) Under the same experimental settings, an 802.11 client stays awake for entire TCP download (energy graph of the other client is equivalent, not shown).

To evaluate convergence of these schedules, we performed Monte Carlo simulations with 1000 APs in a $1km$ x $1km$ area. Two APs were considered in range of each other within $40m$. We ran 10,000 trials with different topological configurations. To reflect an incremental deployment, we assumed a 50-50 mix of SleepWell and legacy APs. For results involving bounded traffic demand, we assumed uniform random demand between 0 and $50ms$ per beacon (up to one-half BEACON_INTERVAL). Figure 9 shows the CDF of convergence time (including the cases where randomization was necessary to break oscillations). Each convergence round corresponds to the number of BEACON_INTERVALs required to transition clients to a new wakeup schedule, equal to the longest LISTEN_INTERVAL among all clients (approximately $300ms$ for the Nexus One). Evidently, SleepWell achieves fast convergence, even with the traffic advertisement heuristic.

(2) Overall energy gain (varied traffic patterns)

Figure 10 presents overall energy consumption during (a) bulk data transfer, (b) YouTube, and (c) Pandora tests. The experiments are performed for 8 AP/client pairs within mutual contending range. As a zero-contention baseline, we show the results from a single AP/client network running 802.11. SleepWell demonstrates substantial energy savings, nearing the baseline for YouTube and Pandora. In contrast, 802.11 PSM wastes energy staying awake through much of the contending traffic, as observed earlier in Figure 8(c).

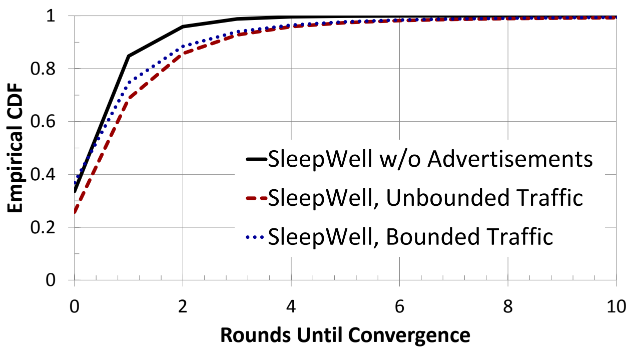


Figure 9: Adjustment rounds until a SleepWell AP reaches a converged beacon placement. 10 rounds takes $\approx 3s$.

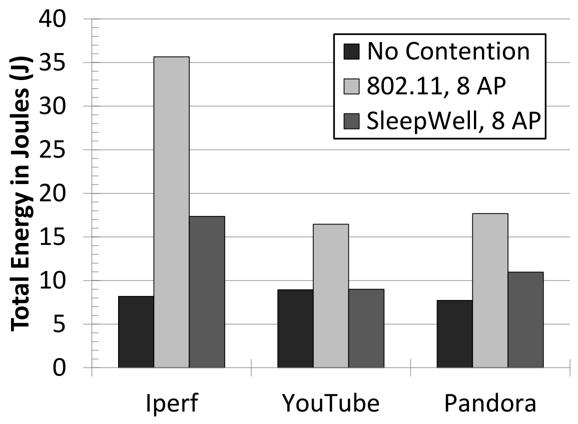


Figure 10: Overall energy performance of SleepWell.

(3) Impact of network contention

Figure 4(a) from Section 2 showed how network contention increases the fraction of time a PSM client stays in the idle/overhear state. Figure 11 demonstrates how SleepWell mitigates the problem for the same (TCP bulk transfer) experiments. As anticipated, SleepWell powers down clients to PSM light-sleep, allowing them to save energy while contending APs communicate. Thus, the duration of time spent in the light-sleep state increases with increasing contention, ultimately offering substantial energy savings. As an aside, note that the client does not go back to the deep-sleep state because it needs to wake up soon for remaining traffic. Switching to and from deep-sleep will incur a high wake-up/shutdown cost, and the hardware is designed to avoid it whenever possible.

Figure 12 shows results for the same experiment, but with a YouTube trace measured using Tcpreplay. SleepWell gains are again substantial compared to 802.11 (in Figure 4(b)).

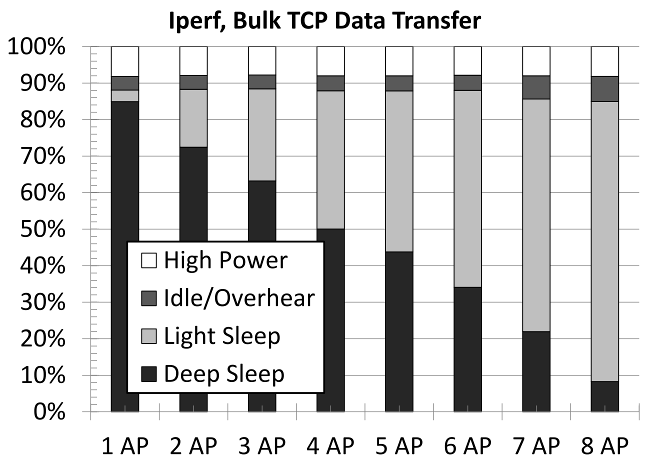


Figure 11: 8 MB Iperf TCP download. With higher contention, SleepWell spends a larger fraction of time in light-sleep, whereas, 802.11 spends most of the time in the idle/overhear state (see Fig. 4a).

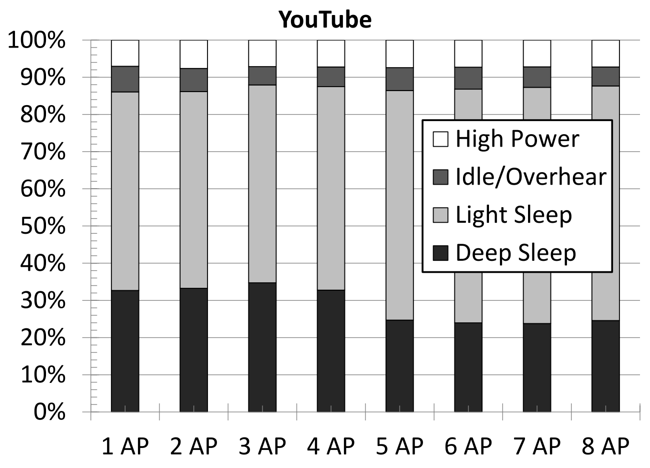
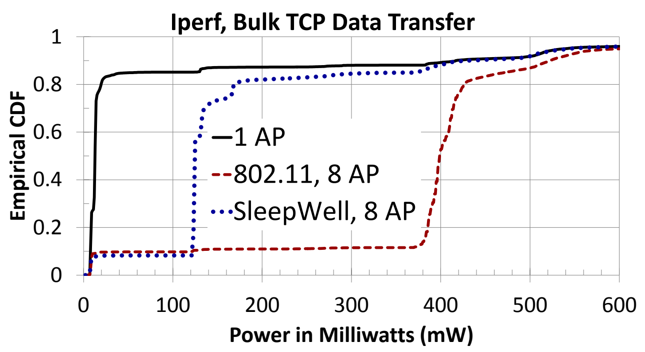
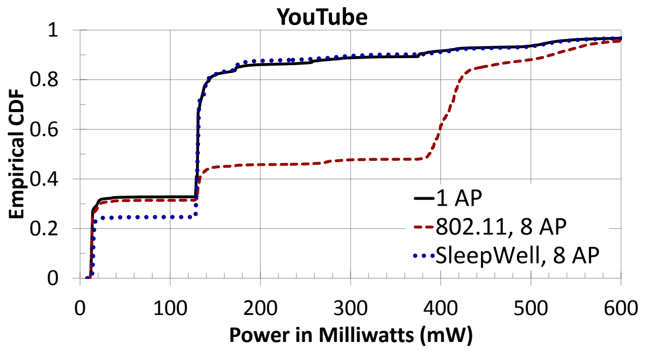


Figure 12: Proportion of SleepWell time spent in each activity level with YouTube traffic. Compare to Figure 4.





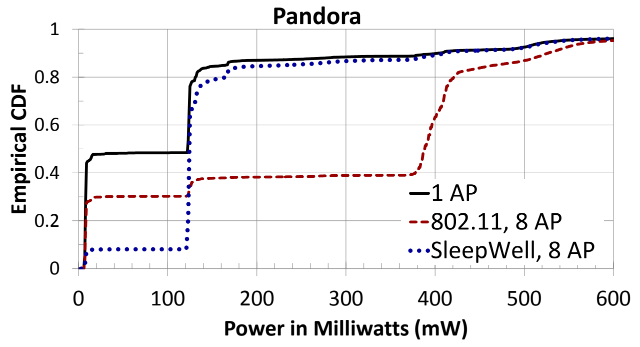


Figure 13: (a) Iperf, (b) YouTube, (c) Pandora. CDF comparison of instantaneous power showing that SleepWell better matches the zero-contention curve.

Further, due to the bursty nature of the `YouTube` trace[4], clients spend more than 50% of the time in the PSM light-sleep mode even without contention. Under SleepWell with rising contention, the individual traffic bursts becomes systematically desynchronized. The overall proportion of time spent in light-sleep (instead of deep-sleep) only increases marginally. As shown, SleepWell clients using YouTube have nearly complete energy-immunity to at least 7 saturated links worth of traffic.

Performance gap from the case of zero-contention: Instead of categorizing power draw into different energy-states, Figure 13 directly compares 802.11 and SleepWell's instantaneous power draw under 8-AP contention. The zero-contention scenario is used as the lower bound. Graphs show (a) TCP bulk transfer, (b) `YouTube`, and (c) `Pandora`. Note that the SleepWell CDF remains closer to that of 1 AP (i.e., zero contention), except in the proportion of time spent in light versus deep-sleep.

Every Client Running YouTube: Figure 14 presents results from a scenario where all links download `YouTube` traffic. To model realistic environments as closely as possible, we used an extended-length `YouTube` trace ($\approx 20\ min$), consisting of a user watching a series of movie trailers. The trace includes time spent selecting a series of video for playback, buffering, and watching the trailer, all in realistic proportions. As in all traces, we captured this trace from using the YouTube application on the Nexus One – this captured smartphone-specific buffering behaviors. We ran the trace in a loop with Tcpreplay on all AP/client pairs, varying the start time independently for each client. For this test, we used 9 phone clients distributed among 6 APs. The measured clients were

[4]YouTube clients buffer videos in smalls bursts to optimize for users that will not play the entire video, or will skip forward.

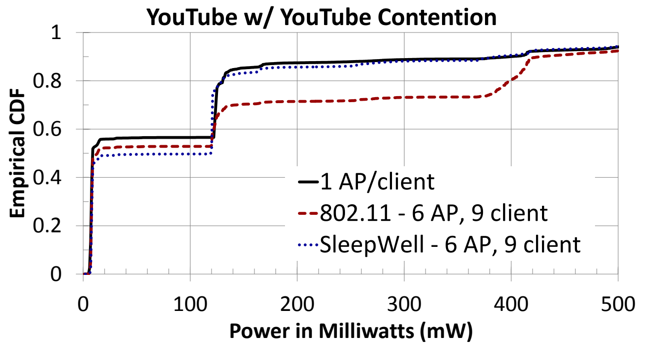


Figure 14: CDF of instantaneous power under homogeneous YouTube traffic (client shown running YouTube with contention from YouTube clients). 18 Mbps link bitrates.

associated to distinct APs, and were the only client for their respective APs. Due to the relatively low server-side bitrate (optimized to ensure that there is never too much buffered video) 9 `YouTube` clients were not sufficient to saturate the (high bitrate) wireless channel. Thus, we used 18 Mbps fixed bitrates, very much reflective of residential/public environments. Evidently, SleepWell consistently outperforms 802.11 PSM, although the margin is smaller due to small bursts of contention. We envisage the gain to increase if the number of APs increase or the link qualities degrade.

(4) Impact of link quality (bitrates)

Figure 15 shows energy performance of an 8 MB bulk data transfer with 4 contending links at varying link bitrates. At low bitrates, all transmissions inflate in time, forcing 802.11 to spend considerably more time in the idle/overhear state. SleepWell does not incur this cost, as it sleeps through the long durations of neighboring traffic. Thus, relative gains grow as links degrade to 18 Mbps and lower.

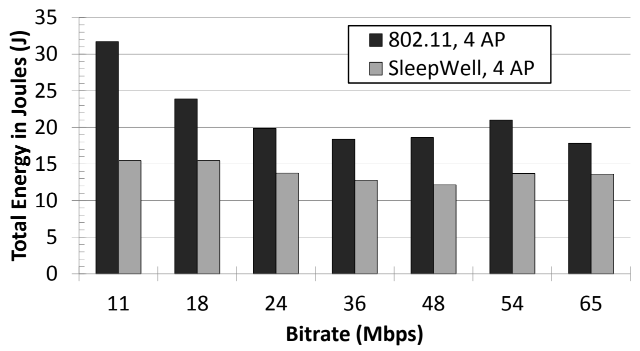


Figure 15: Bulk data transfer on 4 AP/client testbed.

(5) Impact on throughput, latency

Figure 16 shows per-link TCP throughput in a 4-link topology. SleepWell's performance is certainly comparable to 802.11. When clients are backlogged with traffic, recall that they would continue download until the start of another PSM burst. As a result, the channel remains well utilized. When the traffic is unsaturated in some clients, the SleepWell traffic advertisements help in re-distributing the slack among backlogged clients. This minimizes wasteful gaps, allowing high channel utilization.

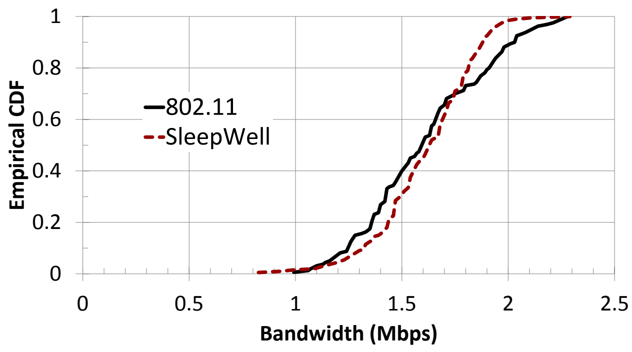


Figure 16: TCP throughput on 4 AP/client testbed. Distribution reflects per-link goodput across all links.

Figure 17 presents per-packet latency under heavy contention, as measured through ICMP pings from the AP. After a PSM wakeup, SleepWell clients experience little contention from other links, and are able to receive and reply to probes faster than 802.11 clients. Further, the inflection point at $307ms$ (one listen interval) reflects that $\approx 95\%$ of SleepWell probes are received before the end of the timeslot following the probe. Even though 802.11 remains awake longer, the latency is still greater due to high network contention following beacons.

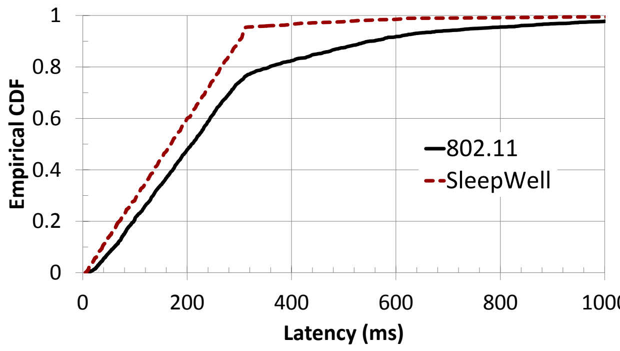


Figure 17: Per-packet latency on 8 AP/client testbed. Latency measured as 10 ICMP pings per second on one link, 7 others contend with TCP.

To characterize performance for large-scale networks, we looked in the results of the Monte Carlo simulations. The goal was to observe the beacon placements, and compute the channel share that each client was getting. The per-client channel share directly relates to the throughput expected at that client. Figure 18 shows that after convergence SleepWell (a) provides a near-universal improvement to beacon spacing; (b) provides spacing improvements irrespective of network density; and (c) enables APs to satisfy a greater proportion of their traffic load (for this graph, all APs are given a random traffic demand, sampled as uniform random `[0, BEACON_INTERVAL / 2]`). In contrast, random beacon placements lead to inequitable and inefficient distribution of the channel resources, leading to contention and wastage.

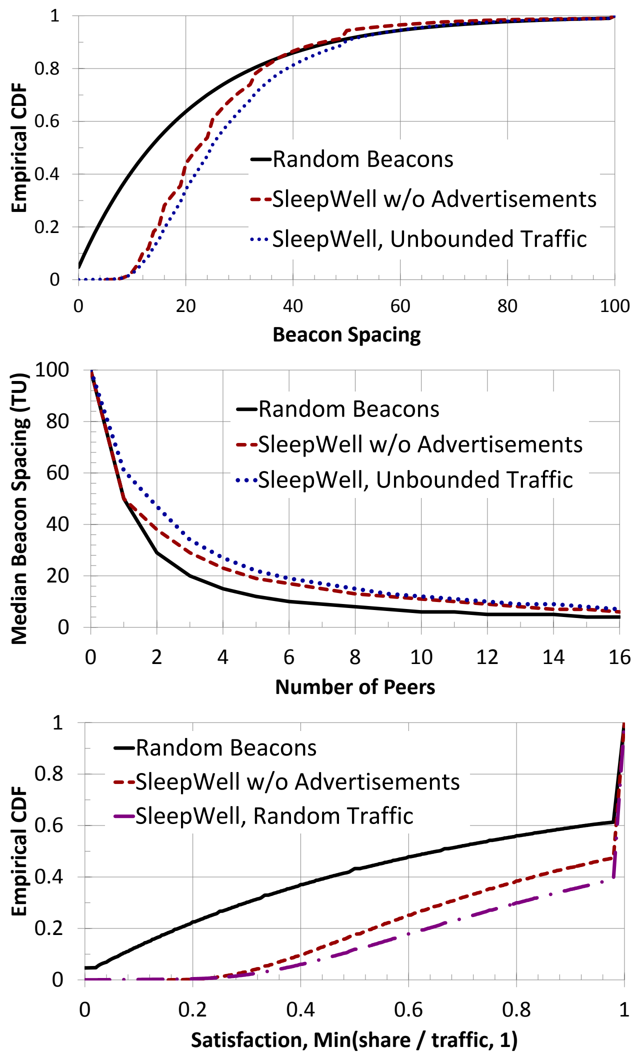


Figure 18: Performance of beacon adjustment: (a) CDF of beacon spacing; (b) spacing by network density. (c) CDF of proportion of an AP's traffic that can be satisfied in the separation between its beacon and that of the next AP.

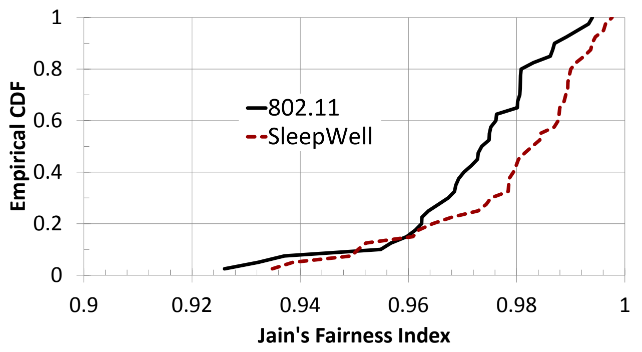


Figure 19: TCP Jain's fairness on 4 AP/client testbed. Note X-intercept at 0.9.

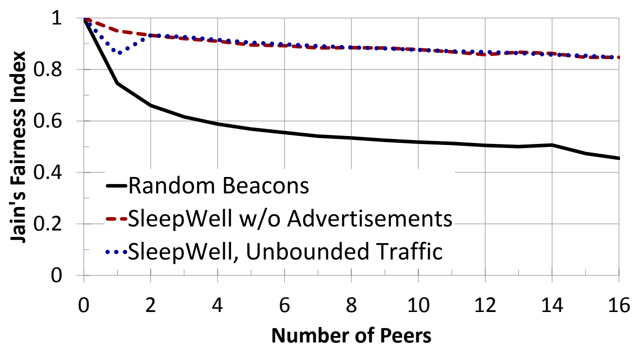


Figure 20: Jain's fairness for simulated beacon shares with unbounded traffic.

(6) Impact on fairness

Figure 19 presents 4-link testbed results, showing that SleepWell is able to allocate throughput slightly more equitably than 802.11. Figure 20 presents Monte Carlo simulation results confirming that the beacon adjustment heuristic results in a more equitable spacing.

5. LIMITATIONS AND DISCUSSION

In this section, we discuss practical challenges for a SleepWell deployment.

Impact of Hidden Terminals: Hidden terminals may impact SleepWell to a greater degree than a randomized beacon placement. In rare cases, SleepWell may synchronize the transmissions of AP-to-AP hidden terminal pairs, as both APs may migrate their beacon to the same perceived-vacant airtime. This may cause collisions, forcing a client to stay awake longer, and thereby, increasing the energy overhead. While hidden terminals are mostly mitigated by carefully tuning the carrier sense threshold and bitrates [8], SleepWell can adopt counter-measures to alleviate the problem. Specifically, since the hidden APs will also impose bursty traffic, a SleepWell AP may observe that its download packets are failing despite a high SNR to its client. The download SNR can be inferred from SNR of upload packets, coming back over a roughly symmetric link. At this point, the SleepWell AP can assume a "virtual beacon" on its traffic map and re-adjust its own beacon as per the protocol heuristic. In other words, the hidden terminal may be treated as another contending AP, only its beacon/traffic advertisements are indirectly inferred. If rare occasions present an excessive number of hidden terminals, SleepWell may not be able to cope. SleepWell APs may gracefully degenerate to 802.11, by randomizing their beacon schedule and disabling traffic preemption.

Incremental Deployability: Thus far, our discussion has assumed all APs in the wireless vicinity to be running SleepWell. For practicality, SleepWell must be and is incrementally deployable; it is also able to coexist with legacy access points with fixed beacon schedules and no traffic preemption. SleepWell APs treat legacy APs identically for the purpose of beacon placement. Although the latter will not re-adjust to obtain their fair share of the beacon interval, they can still be expected to have bursty PSM traffic starting with a PSM beacon. Thus, the time period immediately following their beacon is best avoided. SleepWell includes these APs in the traffic maps, and computes the expected share calculation assuming an advertised share of infinity. The system still converges from our Monte Carlo simulations, using 50% legacy APs.

Interactive Traffic: SleepWell is not indended for interactive, highly latency sensitive traffic (e.g., VoIP). PSM explictly forgoes support for low-latency operation for energy savings; SleepWell is subject to the same pitfalls.

TSF Adjustment: We believe our mechanism for adjusting the TSF clock (to migrate clients to a new beacon schedule) has no side effect. However, we cannot guarantee this to be universal among all devices.

Feasibility of the Fair Share, Proof Sketch: To ensure that beacon adjustments may converge, the sum of expected fair shares must not exceed the BEACON_INTERVAL within a one-hop neighborhood of each SleepWell AP. For n other peer APs in a clique topology, a basic SleepWell AP (unaware of traffic) computes its fair share as $\frac{1}{(n+1)}$. The complete traffic-aware SleepWell degenerates to basic, in the case that no AP has excess channel share (slack). The sum of fair shares for APs with unbounded demand, $\sum_0^n \frac{1}{(n+1)} = (n+1) \cdot \frac{1}{(n+1)} = 1$, yields a completely-utilized BEACON_INTERVAL. Instead, assume there exists one AP with nonzero slack s. A peer SleepWell AP with unbounded traffic demand will increases its fair share to $\frac{1}{(n+1)} + \frac{s}{n}$. In aggregate, $\frac{1}{(n+1)} - s + \sum_1^n [\frac{1}{(n+1)} + \frac{s}{n}] = \frac{1}{(n+1)} - s + \frac{n}{(n+1)} + s = 1$. Again, the BEACON_INTERVAL is fully utilized. In general, let an AP i have a needed share n_i where $s_i = \max(0, \frac{1}{(n+1)} - n_i)$. Let $s^* = \max(s_0, s_1, \ldots, s_n)$. Total of expected fair share across the clique is $\sum_{i=0}^n \min[n_i, \frac{1}{(n+1)} + \frac{s^*}{n}] \leq 1$. Further, note that non-clique topologies are more loosely constrained. For each clique subgraph of the larger network topology, APs with neighbors outside the clique will compute an expected fair share strictly less than the clique-only fair share.

Ensuring Convergence: Consider the graph G of all APs in the network, with edges reflective of each AP's one-hop neighborhood. There may be some cycle $C \subseteq G$ of APs such that a beacon schedule adjustment by an AP $a \in C$ will eventually trigger another another for a. To break such a readjustment cycle, a increments a counter upon each schedule change. Each time a's counter exceeds a CONVERGENCE_THRESHOLD, it randomly reassigns its beacon schedule (in our simulations, we choose this threshold twice the number of one-hop neighbors, to avoid instability in dense subgraphs). After a long period without any schedule changes, the counter may be reset to 0, ensuring that the next schedule change (e.g., triggered by a legitimate change in network traffic or topology) does not invoke an unnecessary perturbation. Results in Section 4 confirm that this randomization heuristic quickly breaks the readjustment cycle, finding a stable beacon schedule (guaranteed to exist, as discussed above) and enabling universal convergence in all simulations. Although required for complete convergence, the randomization was only triggered by less than 1% of nodes in all traffic scenarios (with dense, 1000-node topologies used to increase the probability of cyclic instability). While our simulations have shown this technique to be effective, we provide no formal proof that a livelock readjustment pattern will never occur. However, irrespective of convergence, SleepWell APs will still continuously provide complete 802.11 AP services for all associated clients.

6. RELATED WORK

Substantial prior work has considered mechanisms to reduce the energy cost for mobile devices. In the interest of space, we sample a subset of them.

WiFi PSM sleep optimization: A number of solutions have considered augmenting PSM behaviors for improved efficiency. [12, 15] propose client-side techniques for adaptive PSM, enabling clients to switch between PSM and fully-awake CAM modes as a function of traffic load. Catnap [11] exploits the discrepancy between wired and wireless bandwidth. [4] considers proxies to reduce the cost of application polling. [29] employs traffic shaping on TCP to make flows bursty, and thus more suitable for efficient PSM delivery. μPM [16] leverages prediction to enable a wireless interface to sleep opportunistically over short durations. Most closely aligned to SleepWell, NAPman [24] considers inter-client beacon staggering to improve the energy efficiency of mobile clients through reduced contention. SleepWell is complementary, extending the core beacon staggering idea to the network. In conjunction with NAPman, the total energy gains can be higher.

WiFi Duty Cycling: A number of projects have considered duty cycling the WiFi radio into a deeper sleep state to avoid power drawn when idle. Wake-on-Wireless uses a secondary low-power radio interface for signaling traffic to reduce energy consumption while idle [26]. Cell2Notify uses cellular radios to forward notifications of incoming VoIP calls, waking up the WiFi radio to receive the call just in time [2]. Context-for-Wireless predicts WiFi availability from nearby cell towers [21]. Blue-Fi correlates the presence of nearby Bluetooth devices with WiFi availability [3]. Breadcrumbs predicts the availability of WiFi from personal mobility profiles [19]. Turducken [28], CoolSpots [20], and Tailender [6] consider the use of heterogeneous radios for data transfer, only enabling the highest-powered WiFi radio when it is most appropriate for the traffic load. Each of these techniques enables a complete shutdown of the WiFI interface over long timescales. SleepWell is complementary in reducing energy consumption during those periods in which the WiFi interfaces are enabled and in active use.

Sensor network TDMA: Scheduled channel access has often been considered for energy savings in sensor networks. S-MAC enables nodes to synchronize sleep schedules with their peers, and accordingly sleep through the peers' transmissions [32]. Z-MAC multiplexes CSMA and TDMA [23], and partly achieves the best of TDMA and CSMA. SleepWell bears resemblance to these high level ideas, however, the system is designed in response to a set of completely different challenges and constraints.

7. CONCLUSION

We summarize SleepWell with an analogy. Big cities in the US and other countries face heavy rush hours due to masses of people commuting to office. If work times were flexible, different companies could potentially stagger their office hours to reduce this rush. Reduced rush would open up more free time for all, and yet, the total working hours can remain unaffected. This intuition underlies the design of SleepWell.

Given that Internet traffic can tolerate a reasonable amount of latency/flexibility, SleepWell APs adjust their activity cycles to minimally overlap with others. Each client frees up time to sleep, ultimately resulting in promising energy gains with practically negligible loss in performance. Our testbed implementation and thorough evaluation gives us confidence that SleepWell is actually viable, and hence, worth considering as a revision to current 802.11 PSM.

Acknowledgments

We sincerely thank our shepherd, Anmol Sheth as well as the anonymous reviewers, for their invaluable feedback. We also thank Kip Coonley (Duke ECE) for assistance in the phone energy measurement set-up. Finally, we are grateful to NSF for partially funding this research through the following grants: CNS-0916995 and IIS-910846.

8. REFERENCES

[1] Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. *IEEE Std 802.11*, 2007.

[2] Y. Agarwal et al. Wireless wakeups revisited: energy management for voip over wi-fi smartphones. In *MobiSys*, 2007.

[3] G. Ananthanarayanan and I. Stoica. Blue-Fi: enhancing Wi-Fi performance using bluetooth signals. In *MobiSys*, 2009.

[4] T. Armstrong et al. Efficient and transparent dynamic content updates for mobile clients. In *MobiSys*, 2006.

[5] A. Balasubramanian, R. Mahajan, and A. Venkataramani. Augmenting mobile 3G using WiFi. In *MobiSys*, 2010.

[6] N. Balasubramanian et al. Energy consumption in mobile phones: a measurement study and implications for network applications. In *IMC*, 2009.

[7] Broadcom. BCM4329 product brief. *www.broadcom.com*.

[8] M. Brodsky and R. Morris. In defense of wireless carrier sense. In *SIGCOMM*, 2009.

[9] R. Chandra et al. A location-based management system for enterprise wireless LANs. In *NSDI*, 2007.

[10] ChangeWave Research. New smart phone owners tell us what they really think. May 2010.

[11] F. Dogar, P. Steenkiste, and K. Papagiannaki. Catnap: exploiting high bandwidth wireless interfaces to save energy for mobile devices. In *MobiSys*, 2010.

[12] M. Edmund, E. Nightingale, and J. Flinn. Self-Tuning Wireless Network Power Management. In *MobiCom*, 2003.

[13] H. Falaki et al. A First Look at Traffic on Smartphones. In *IMC*, 2010.

[14] H. Falaki et al. Diversity in smartphone usage. In *MobiSys*, 2010.

[15] R. Krashinsky and H. Balakrishnan. Minimizing energy for wireless web access with bounded slowdown. In *MobiCom*, 2002.

[16] J. Liu and L. Zhong. Micro power management of active 802.11 interfaces. In *MobiSys*, 2008.

[17] P. Mohan et al. Nericell: rich monitoring of road and traffic conditions using mobile smartphones. In *SenSys*, 2008.

[18] Monsoon Solutions Inc. *http://www.msoon.com/LabEquipment/PowerMonitor/*.

[19] A. Nicholson and B. Noble. Breadcrumbs: Forecasting mobile connectivity. In *MobiCom*, 2008.

[20] T. Pering et al. Coolspots: Reducing the power consumption of wireless mobile devices with multiple radio interfaces. In *MobiSys*, 2006.

[21] A. Rahmati and L. Zhong. Context-for-wireless: context-sensitive energy-efficient wireless data transfer. In *MobiSys*, 2007.

[22] S. Ramanathan. A unified framework and algorithm for channel assignment in wireless networks. *Wireless Networks*, 5(2):81–94, 1999.

[23] I. Rhee et al. Z-MAC: a hybrid MAC for wireless sensor networks. *IEEE/ACM ToN*, 2008.

[24] E. Rozner, V. Navda, R. Ramjee, and S. Rayanchu. NAPman: Network-Assisted Power Management for WiFi Devices. In *MobiSys*, 2010.

[25] A. Schulman et al. Bartendr: a practical approach to energy-aware cellular data scheduling. In *MobiCom*, 2010.

[26] E. Shih, P. Bahl, and M. Sinclair. Wake on wireless: An event driven energy saving strategy for battery operated devices. In *MobiCom*, 2002.

[27] T. Sohn et al. Place-its: A study of location-based reminders on mobile phones. In *UbiComp*, 2005.

[28] J. Sorber, N. Banerjee, M. Corner, and S. Rollins. Turducken: hierarchical power management for mobile devices. In *MobiSys*, 2005.

[29] E. Tan et al. PSM-throttling: Minimizing energy consumption for bulk data communications in WLANs. In *ICNP*, 2007.

[30] A. Turner. Tcpreplay. *tcpreplay.synfin.net*.

[31] M. Vutukuru et al. Harnessing exposed terminals in wireless networks. In *NSDI*, 2008.

[32] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *INFOCOM*, 2002.