Realizing the Full Potential of PSM using Proxying

Ning Ding[1], Abhinav Pathak[1], Dimitrios Koutsonikolas[2], Clay Shepard[3], Y. Charlie Hu[1], Lin Zhong[3]
[1]Purdue University, [2]University at Buffalo, SUNY, [3]Rice University

***Abstract*—The WiFi radio in smartphones consumes a significant portion of energy when active. To reduce the energy consumption, the Power Saving Mode was standardized in IEEE 802.11 and two major implementations, Static PSM and Dynamic PSM, have been widely used in mobile devices. Unfortunately, both PSMs have inherent drawbacks: Static PSM is energy efficient but imposes considerable extra delays on data transfers; Dynamic PSM incurs little extra delay but misses energy saving opportunities.**

In this paper, we first analyze a one-week trace from 10 users and show that more than 80% of all traffic are Web 2.0 flows, which are of very small sizes and short durations. Targeting these short but dominant flows, we propose a system called Percy, to achieve the best of both worlds (Static and Dynamic PSM), *i.e.*, to maximize the energy saving while minimizing the delay of flow completion time. Percy works by deploying a web proxy at the AP and suitably configuring the PSM parameters, and is designed to work with *unchanged* clients running Dynamic PSM, and *unchanged* APs and Internet servers. We evaluate our system via trace-driven testbed experiments. Our results show that Percy saves 40-70% energy compared to Dynamic PSM configurations of Nokia, iPhone and Android, while imposing low extra delay that can hardly be perceived by users.

I. Introduction

Despite the incredible market penetration of smartphones, their utility has been and will remain severely limited by their battery life [6], [8]. A major fraction of the energy consumption in smartphones comes from the wireless network interface controller (WNIC). For example, the base energy consumption of Nokia N900 is about 12 mW while the WiFi radio consumes 765 mW during receiving and 1000 mW during sending. Since WiFi has been increasingly used by smartphones to offload traffic from the cellular network [2], reducing WiFi energy consumption is becoming essential for maximizing the battery lifetime of smartphones.

The IEEE 802.11 standard [3] defines a Power Saving Mode (PSM) for WiFi devices to reduce the energy consumption due to wireless communication. The PSM mechanism allows the WNIC of a client to spend most of the time at a low power state (Power Saving Mode or PSM) and to switch to a high power state (Constant Awake Mode or CAM) when network traffic is present. Since in PSM the WiFi radio cannot communicate, the main challenge associated with the PSM mechanism is how the mobile client will accurately determine when to switch from CAM to PSM and vice versa in order to maximize energy savings without delaying data transfers. Unfortunately, the two existing PSM implementations, Static and Dynamic PSM, fail to efficiently address this challenge, leading to either severe data transfer delays or missed energy saving opportunities. As a result, PSM is generally considered to be unusable *during* a data transfer [1], especially for short flows [5].

In this paper, we show how deploying a web proxy at the AP can achieve the best of the both worlds (Static and Dynamic PSM), *i.e.*, maximize the energy saving while minimizing the overhead on the data transfer delay. Our design exploits a key observation of Dynamic PSM: the relatively large PSM timeout value is dictated by an attempt to accommodate the varying RTTs to different Internet servers that a client downloads data from. By using a proxy behind the AP to intercept the TCP connection, the PSM timeout can be set to a value only slightly larger than the delay between the client and the proxy. This has two immediate benefits: (1) it allows the client to quickly and safely enter PSM while the proxy is performing a data transfer from a remote server, and (2) it allows the client to quickly enter PSM after the proxy flushes the data to the client.

We design our system, Percy, to work with clients running Dynamic PSM, and *unchanged* APs and Internet servers. The major challenge in designing such a system is how to set the PSM parameters at the client so as not to incur a significant increase on the packet delivery time. We distinguish two cases in addressing this challenge based on the flow characteristics (flow size and duration). (i) *Web 2.0 flows* [7]: Our analysis of a 10-phone trace of over 38,000 flows collected over a one-week period confirms that the majority of the network traffic generated on the smartphones consists of short flows of size smaller than 100 KB due to Web 2.0 applications such as instant messaging, social networking, searching, and web surfing. For these flows, Percy's capability to quickly enter PSM saves significant portion of energy without sacrificing network performance. (ii) *Long flows*: For long flows, the Percy proxy periodically (at beacon times) flushes buffered packets to the client to avoid buffering too many packets and thus limits the extra delay on the flow completion time. Hence, our scheme also supports relatively large file downloading.

We evaluate our system via trace-driven testbed experiments. Compared to Dynamic PSM implementations on three popular smartphones: Nokia N900, iPhone 4 and Android Nexus One, our evaluation shows that Percy saves 45% to 67% energy with an average extra delay of 50 ms.

II. 802.11 Power Saving Modes

The WiFi radio on mobile devices consumes a significant portion of the total energy. To reduce the energy consumption, wireless network interface controllers (WNICs) are usually designed to support various power modes. The Constant Awake Mode (CAM) allows for normal operation but consumes high power. On the other hand, the Power Saving Mode (PSM) consumes very little power but disables communication. Since many protocols like TCP require nodes to be able to receive

packets at any moment, mechanisms that handle communication outages in PSM must be provided.

A. Static PSM

In [3], Static PSM was first standardized. With Static PSM, the AP buffers the incoming packets for a device in PSM and notifies the device by setting its corresponding Traffic Indication Map (TIM) bit in the next beacon. The device wakes up at the beginning of every beacon period and checks the TIM. If the TIM indicates buffered data at the AP, the device switches to CAM and sends PS-Poll to poll every packet until the AP indicates no more buffered packets by setting the More Data flag to 0. The device then enters PSM right away. For outgoing traffic, the device can wake up at any time to send packets and immediately switch back to PSM afterwards. By entering PSM aggressively, Static PSM keeps the WNIC in CAM only when the device is transmitting or receiving, hence achieves high energy efficiency.

However, devices using Static PSM suffer from considerable delay on data transfer for two reasons. First, the delay involved in receiving data via polling every packet can be high for interactive applications. Second, since the device immediately enters PSM after sending or receiving, packets that arrived shortly later will have to be buffered at the AP until the next beacon. Considering a TCP flow with 5 ms RTT, after a phone sends SYN, the SYN/ACK packet arrives 5 ms later when the phone has already entered PSM, thus can only be received at next beacon. Since the beacon interval is typically set to 100 ms, with Static PSM a short RTT flow behaves like a flow with 100 ms RTT. This is called the "RTT round up effect". Such an effect can severely elongate short flows.

B. Dynamic PSM

More recently, Dynamic (or Adaptive) PSM has gained popularity in mobile devices such as smartphones. Under Dynamic PSM, instead of entering PSM right after transmitting/receiving, the device will stay in CAM for a pre-defined duration called PSM timeout. This prevents the device from going to sleep in the middle of a data transfer, *i.e.,* when the PSM timeout is longer than RTT. As with static PSM, during sleeping in PSM, the device will wake up at periodic intervals to receive beacons from the AP to check the TIM. If its corresponding bit in TIM is set, it will switch to CAM and receive the packets. The device notifies the AP of any transition to CAM or PSM by sending a Null data frame with the Power Management bit set to 0 or 1, respectively.

Compared to Static PSM, Dynamic PSM improves the network performance in two ways: (1) it avoids sending PS-poll frames while retrieving data from the AP; (2) lingering in CAM for a PSM timeout periods avoids the RTT round up effect. In fact, with appropriate configuration, Dynamic PSM can keep the device in CAM during most flows, thus achieving the best network performance. However, the delay in entering PSM under Dynamic PSM can lead to extra WNIC awake time, and hence reduced energy saving.

Intuitively, the PSM timeout should be at least as large as the maximal inter-packet gap, *i.e.,* the flow RTT, as otherwise the flow will suffer the RTT round up effect and its duration can be significantly enlarged. Since the device may connect to Internet servers with varying RTTs over time, the PSM timeout value needs to be set conservatively large to accommodate most RTTs. This is the rational behind the configuration in Nokia N900 and Android Nexus One, where the default PSM timeout is set to 200 ms.

TABLE I
PSM PARAMETERS FOR VARIOUS PSM IMPLEMENTATIONS AND PERCY.

Name	Nokia	iPhone	Android	Static	Percy
Beacon interval	N/A	N/A	N/A	N/A	N/A
PSM timeout	200ms	95ms	200ms	0	30ms
Listen interval	2	0	0	0	0

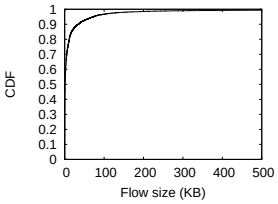

Fig. 1. CDF of flow sizes.

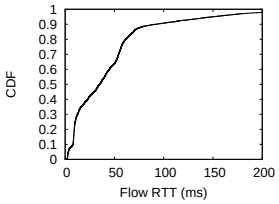

Fig. 2. CDF of flow RTTs.

C. Listen Interval

Another important parameter common to both Static and Dynamic PSM is the listen interval I_{listen}, which determines the frequency the WNIC wakes up at to receive beacons. For example, if I_{listen} is set to 2, it means while in PSM, the phone will wake up to receive every third beacon. The listen interval determines the PSM sleep granularity for incoming traffic, since all incoming data in between will be buffered by the AP until the WNIC's next wakeup time.

Table I summarizes the PSM parameters and their default configurations on three popular smartphones, as well as for Static PSM (ignore the last column for now). We can see that there is no clear trend on how to configure these parameters; each device uses a different set of values.

III. WEB 2.0 FLOWS ARE SHORT

In this section, we analyze two network characteristics of smartphone traffic traces, size and RTT. The analysis motivates and guides the design of Percy.

Smartphone Traffic Trace. We have collected a traffic trace from 10 iPhone users at Rice University over a one-week period during April 17 to April 23, 2011. The trace consists of a total of 38069 flows from 195 different IP addresses and to 2812 servers. The number of flows, servers, and IPs per phone ranges from 1319-7084, 209-869, and 5-47, respectively.

Analysis. Figure 1 shows the Cumulative Distribution Function (CDF) of the flow sizes of all the flows. We make two observations. First, the majority of the flows are of very small sizes: 46.6% flows have sizes smaller than 1 KB, 89.1% smaller than 30 KB, and 96.7% smaller than 100 KB. Second, the RTTs experienced by the flows vary significantly. While 87.6% of the flows have RTTs less than 75 ms, the remaining flows have RTTs up to 500 ms.

To facilitate the analysis in the next section, we denote flows of sizes smaller than 4 KB, between 4 KB and 12.8 KB, and between 12.8 KB and 30 KB as "1-window flows", "2-window flows" and "3-window flows", respectively. The reason for choosing these boundaries is, as the names indicate, 4 KB,

12.8 KB, and 30 KB are the maximum amount of data that can be transferred within 1, 2, 3 TCP windows, respectively, with Max Segment Size (MSS) of 1448 bytes and an initial congestion window of 3 MSSes. In our trace, 1, 2, and 3-window flows make up 58.4%, 16.9%, and 8.5% of the total number of flows, respectively. This shows that small flows dominate smartphone network traffic and should be the focus of the WNIC power management.

IV. Percy Design

In this section, we show how a simple system, called Percy, which uses a transparent web proxy behind the AP to split TCP flows, can maximize the energy saving of PSM without incurring high delay in flow completion time. We first describe the Percy architecture, then analyze how the PSM parameters should be configured to maximize the dual objectives: energy saving and low extra delay.

A. Architecture

Percy assumes *unchanged* Internet servers and *unchanged* mobile clients, and is implemented in a transparent web proxy behind the AP. The proxy performs the following tasks: (1) It intercepts each TCP connection originated on the mobile client. In particular, it finishes TCP handshake and receives the HTTP request. (2) It establishes a new TCP connection with the destination server and relays the HTTP request to the server. (3) It performs data transfer from the Internet server which may involve several TCP windows depending on the transfer size. (4) It regulates the relaying of the packets back to the WiFi client in a way that minimizes the extra flow delay and maximizes energy saving from exploiting Dynamic PSM.

The major design challenge of Percy is how to configure the PSM parameters in order to maximize the energy saving while minimizing the packet transmission delay. Since Web 2.0 flows are largely short flows, we study the parameter configuration problem by examining short flows in detail.

B. Analysis

1) Assumptions:

Bandwidth. We assume both the local and remote link effective bandwidth (*i.e.,* the actual data transfer rate) are 5 Mbps. With such rate, it takes $T_{data} = 2.4$ ms to transmit a 1500-byte data packet. For simplicity, in our analysis we do not consider transmission delay of SYN, FIN or ACK packets, which take less than 0.2 ms.

Delay. Since the phone accesses the Internet via an AP, the delay consists of two parts: the local delay between the phone and the AP or proxy, and the remote delay between the AP/Proxy and the server. Due to very little propagation delay in WLAN, we only consider the transmission delay on the local link, while the remote delay equals the sum of transmission delay to the remote server (based on effective bandwidth) and the RTT (the propagation delay).[1]

2) 1-TCP-Window Flows: We start with no proxy case. Figure 3 shows the dynamics of a TCP flow that has only 3 data packets, which fit in one TCP window. It takes two RTTs for the client to finish the downloading: 1 RTT for establishing the TCP connection with the server, and 1 RTT for sending request and receiving data. Assume the RTT to server is 50 ms (which is the average RTT in our smartphone trace), and the PSM timeout is 200 ms (the default setting of Nokia and Android phone), the flow time T_{flow} and awake time T_{awake} can be calculated as below:

$$T_{flow} = 2RTT + 3T_{data} = 107.2ms$$
$$T_{awake} = T_{flow} + T_{PSMtimeout} = 307.2ms$$

The result shows that for a 1-window flow with an average RTT and the default PSM timeout, the NIC awake time is three times of the flow time! We reiterate that the reason the default PSM timeout on Nokia and Android is set to 200 ms is because the phone may communicate with many Internet servers which have varying RTTs. In other words, if we were certain that the phone only connects to servers with short RTTs, the PSM timeout could have been set to be much smaller.

The above observation motivates the design of Percy: if we deploy a local proxy at the AP which has a small, constant delay to the client, and set the client PSM timeout to be short, as shown in Figure 4, there will be two immediate benefits in terms of energy saving: (1) the short PSM timeout allows the client to enter PSM while the proxy is performing data transfer from the remote server which can take long (about two RTTs in this case); and (2) it significantly reduces the overhead to enter PSM after the flow completion.

Consider the previous example. Assume a local Percy proxy is deployed behind the AP to intercept HTTP connections and buffer received data. The RTT between the proxy and the server is still 50 ms. Since now it is certain that the phone only communicates with the Percy proxy, we can set the PSM timeout to a much smaller value, 30 ms. We will justify this new timeout later.

Figure 4 shows that Percy effectively creates an idle period of 100 ms on the client, which is larger than the 30 ms PSM timeout thus long enough for the client to enter PSM. This means even for a flow with only 3 packets, Percy can enable the phone to enter PSM and save energy!

Wakeup delay. Note that at the moment the proxy finishes

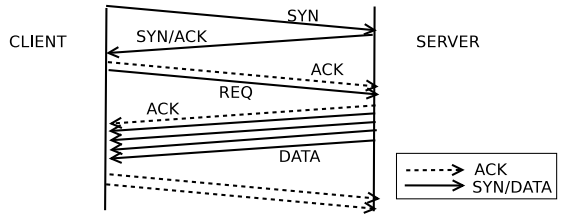

Fig. 3. Packets dynamics of 1-Window flows without proxying.

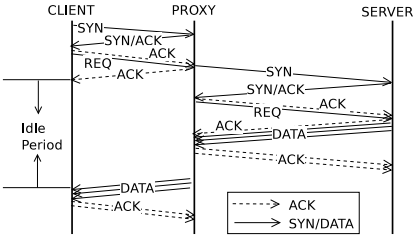

Fig. 4. Packet dynamics of 1-Window flows with proxying.

[1]Note that when the phone directly downloads from servers through the AP, the AP does not buffer data. So the transmission delay counts once instead of twice.

downloading from the remote server and is ready to flush packets to the phone, the phone is likely still in PSM and therefore can not receive the packet until the next beacon. We denote this potential delay as PSM wakeup delay D_{wakeup}, which takes a random value between 0 and the listen interval. Its value solely depends on the timing the incoming packets arrive at the AP and the next wakeup beacon time.

We next analyze the impact of proxying on the flow time. With proxying, the flow consists of three stages: (i) The phone connects to the proxy and sends HTTP request. Since the proxy is local, this stage takes negligible time. (ii) The proxy downloads data from the server. It takes 1 RTT for the proxy to establish the connection, 1 RTT to send request and receive data, plus $3T_{data}$ to receive 3 packets. (iii) The proxy sends data to the phone. It takes a wakeup delay D_{wakeup} for the phone to wake up, and $3T_{data}$ to transfer the packets. Among these three stages, the phone's WNIC is awake for $3T_{data}$ at the third stage to receive data, and for $T'_{PSMtimeout}$ of 30 ms after the first and third stage to enter PSM. Thus, with Percy the flow time T'_{flow} and the awake time T'_{awake} are:

$$\begin{aligned} T'_{flow} &= 2RTT + +3T_{data} + D_{wakeup} + 3T_{data} \\ &= (114.4 + D_{wakeup})ms \\ T'_{awake} &= 3T_{data} + 2T'_{PSMtimeout} = 67.2ms \end{aligned}$$

Compared to $T_{flow} = 107.2ms, T_{awake} = 307.2ms$ in no-proxying case, the WNIC awake time is reduced from 307.2 ms to 67.2 ms, a 78.1% saving! The flow time is elongated by 7.2 ms plus a wakeup delay D_{wakeup}. Since D_{wakeup} is randomly distributed from 0 to the listen interval, we can limit its upper bound by setting the listen interval to 1 beacon interval, which is 100ms. In this case, the average flow time increase is about half the beacon interval, or 50 ms, which is insignificant as it is much smaller than that can be perceived by the user.

We justify why we use 30 ms PSM timeout instead of a smaller value. As discussed in Section II-B, the condition to avoid the RTT round up effect is to set the PSM timeout larger than the maximal inter-packet gap, *i.e.,* the transmission delay for a data packet, 2.4 ms. Technically a PSM timeout of 5ms would suffice. However, setting timeout to 5 ms means the phone will enter PSM for all flows longer than 5 ms, and thus suffer the wakeup delay of on average 50 ms. For example, without proxying, a 1-window flow with 10 ms RTT (to the server) will last for 20 ms. With a proxy, it would take on average 50 ms for the flow to complete, a 2.5-time slowdown! However, if we set PSM timeout to 30 ms, the phone will keep awake during the flow, as a result with proxying the flow time is still around 20 ms. In summary, setting PSM timeout to 30 ms prevents performance degradation for very low RTT flows (*e.g.,* accessing local web servers).

3) Flows with a few TCP windows: In general, for a flow with n packets and $w(n)$ windows, the flow time of no proxy and proxying cases can be similarly derived as follows:

$$\begin{aligned} T_{flow} &= w(n)RTT + nT_{data} \\ T'_{flow} &= w(n)RTT + 2nT_{data} + D_{wakeup} \end{aligned}$$

To download the same amount of data, Percy elongates the flow time by the packet's transmission delay nT_{data} plus the wake up delay D_{wakeup}. As shown in Section III, 89.1% of flows transfer less than 30 KB data, which takes 48 ms with a rate of 5 Mbps. For those flows, together with the wake up delay, the average elongation is around 98 ms on average. We believe such a delay can hardly be perceived by human in normal web browsing.

Periodic flush. For the remaining 14% flows that transfer more than 20 KB data, if the proxy simply sends all data at the very end, the extra transmission delay can be large. To handle this problem, Percy uses a "periodic flush" strategy to flush the received data to the phone when it buffers a certain amount of data for any flow. If we set the threshold to 20 KB, the average extra flow time brought by Percy is limited to around 83 ms, thus effectively bounding the slowdown on relatively large transfers.

Now we compare the WNIC awake time. The WNIC awake time with or without Percy can be calculated as below:

$$\begin{aligned} T_{awake} &= w(n)RTT + nT_{data} + T_{PSMtimeout} \\ T'_{awake} &= nT_{data} + 2T'_{PSMtimeout} \end{aligned}$$

The energy saving comes from 2 parts: (1) $w(n)RTT$: normally, the WNIC has to be kept on during the whole flow; while with Percy, the WNIC is in PSM when the proxy is downloading data from the remote server. (2) $T_{PSMtimeout}$: with Percy we can set PSM timeout to as small as 30 ms, while without Percy the PSM timeout has to be set conservatively to a large value, *e.g.,* 95 ms on iPhone, 200 ms on Android and Nokia phones.

In summary, under Percy the WNIC awake time only depends on the amount of data to transfer, while the remaining flow time is converted to WNIC sleep time. In contrast, under Dynamic PSM, the WNIC is likely to be awake throughout the flow duration.

V. Smartphone Experiment

Methodology. We conducted a trace-driven experiment to compare the performance of Percy and other PSM schemes. From the smartphone trace we picked one phone (Phone 5) with a moderate number of HTTP flows (5524 for one week), and extracted each flow's starting time, flow size and RTT as the workload for the trace-driven experiment.

Our testbed consisted of 4 HP Compaq dx2200 PCs with Celeron D 3.20 GHz CPU and 512MB RAM, a Netgear Wireless G WGR 614 v9 router, a Nokia N900 smartphone and a Monsoon Power Monitor. We used 3 PCs acting as servers, 1 acting as the proxy, and controlled their delay and bandwidth using the Linux Traffic Control utility. Since we only had 3 servers, we only considered cases of up to 3 concurrent flows. If more flows were concurrent in the trace, only the first three were considered. This excluded 407 out of 5524 flows.

We implemented a simple HTTP web system, which consists of one client, one proxy and three servers. The client, which runs on the phone, reads the workload and sends HTTP requests according to the flow starting times. It also logs every flow's actual starting and finishing time to calculate the flow time. The servers also read the trace, adjust the delay to emulate the RTT, and respond to HTTP requests sent to them. The Percy proxy, which resides between the client and

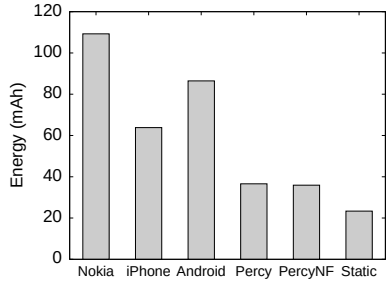

Fig. 5. Total energy consumption.

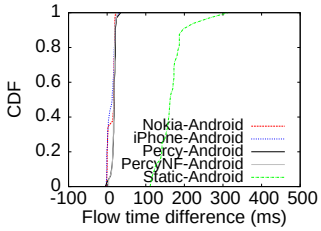

Fig. 6. CDF of flow time differences for flows shorter than 30ms.

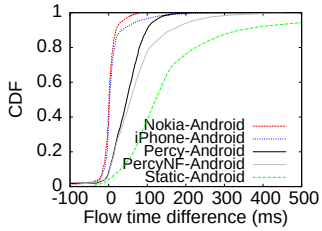

Fig. 7. CDF of flow time differences for flows longer than 30ms.

servers, intercepts HTTP requests from the phone, performs downloading and buffers data from the servers. With this system, we are able to repeat the flows in the trace in an automatic manner, with the phone's screen off and no human interference during the experiment, thus ensuring the accuracy of power measurement.

We conducted the experiment for Percy, Static PSM, and Dynamic PSM with all three configurations from Nokia, iPhone and Android. Detailed parameter settings are shown in table I. To better compare the energy consumption, all the PSM schemes were implemented on Nokia N900. We repeated each experiment three times and used the average as the results.

Energy saving. Figure 5 shows the total energy consumption under each PSM configuration. Among the three Dynamic PSM configurations, Nokia and iPhone consume the highest and lowest amount of energy, respectively. Percy and PercyNF (Percy without periodic flushing) consume much less energy than the three Dynamic PSM schemes: 67% less than Nokia, 58% less than Android and 45% less than iPhone. As expected, Static PSM is the most energy efficient among all schemes.

Network performance. Figures 6, 7 plot the CDF of the flow time differences between each scheme and the Android PSM configuration, for flows shorter and longer than 30 ms, respectively. For short flows, Nokia, iPhone, and Percy, incur a very small penalty. For Percy and PercyNF the flow time increase is always less than 35 ms with a median increase of 18ms. Such increases keep the total flow time under 50 ms. In contrast, Static PSM increases the flow time by 163 ms in the median case, with a max increase of more than 300 ms.

For flows longer than 30 ms, Percy increases the flow time by up to 210 ms, with a median increase of 47 ms. The corresponding values with PercyNF are 449 ms and 57 ms, respectively. Thus we observe that periodic flushing helps performance in case of large flows. With Static PSM, the flow time elongations are much larger, 125 ms in the median case, and more than 300 ms for 14.2% of the flows.

VI. Related Work

Our work is closely related to several previous works on WiFi power management of mobile devices. [5], [9] propose new PSM protocols to minimize the energy consumption with bounded slowdown. However, the savings largely depend on the packet arrival gaps and come at the cost of elongating the RTT. In contrast, under Percy the phone offloads downloading to a local proxy which converts the packets' arrival gaps into idle periods long enough to enter PSM. Catnap [4] also deploys a proxy to combine packet gaps and enable devices to sleep during transfers. Essentially, Catnap exploits the bandwidth difference between high rate local link and slow Internet link, and hence requires large data transfers to gain energy benefits. Percy works by exploiting the RTT difference between the local proxy and remote server, hence does not rely on bandwidth difference and is effective for the dominating Web 2.0 short flows.

VII. Conclusions

The WiFi radio in smartphones consumes a significant portion of power when active and thus, wireless devices implement a variety of PSM strategies to save energy. Unfortunately, both major existing PSM implementations (Static and Dynamic PSM) suffer from inherent drawbacks: Static PSM may impose significant delays on data transfers while Dynamic PSM may miss energy saving opportunities.

In this paper, we presented a system called Percy to improve the energy savings for Web 2.0 flows, the dominating traffic on smartphones, while at the same time minimizing the extra delay of the flow completion time. Percy deploys a proxy at the unchanged AP and works with unchanged Internet servers and unchanged clients with suitably configured PSM parameters. Our testbed experiments show that Percy reduces the energy consumption by 44-67% compared to various Dynamic PSM configurations (on Nokia, Android, and iPhone phones) while incurring a minimal flow time elongation for the majority of the flows compared to Static PSM.

Acknowledgment. This work was supported in part by NSF grants CNS 0905331 and NetSE 1012831.

References

[1] M. Anand, E. B. Nightingale, and J. Flinn. Self-tuning wireless network power management. In *Proceedings of ACM MobiCom*, 2003.

[2] A. Balasubramanian, R. Mahajan, and A. Venkataramani. Augmenting mobile 3g using wifi. In *Proceedings of ACM MobiSys*, 2010.

[3] I.-S. S. Board. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specification. *Electronics*, 1999(802.11), 1997.

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

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

[6] V. Namboodiri and L. Gao. Towards energy efficient VoIP over wireless LANs. In *Proceedings of ACM MobiHoc*, 2008.

[7] T. Oreilly. What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software. *Communications and Strategies, No. 1, p. 17, First Quarter 2007*.

[8] T. Pering, Y. Agarwal, R. Gupta, and R. Want. Coolspots: reducing the power consumption of wireless mobile devices with multiple radio interfaces. In *Proceedings of ACM MobiSys*, 2006.

[9] D. Qiao and K. Shin. Smart power-saving mode for IEEE 802.11 wireless LANs. In *Proceedings of IEEE INFOCOM*, 2005.