# PSM-throttling: Minimizing Energy Consumption for Bulk Data Communications in WLANs

Enhua Tan[1], Lei Guo[1], Songqing Chen[2], and Xiaodong Zhang[1]

[1]Dept. of Computer Science and Engineering
The Ohio State University
Columbus, OH 43210, USA
{etan, lguo, zhang}@cse.ohio-state.edu

[2]Dept. of Computer Science
George Mason University
Fairfax, VA 22030, USA
sqchen@cs.gmu.edu

*Abstract*— **While the 802.11 power saving mode (PSM) and its enhancements can reduce power consumption by putting the wireless network interface (WNI) into sleep as much as possible, they either require additional infrastructure support, or may degrade the transmission throughput and cause additional transmission delay. These schemes are not suitable for long and bulk data transmissions with strict QoS requirements on wireless devices. With increasingly abundant bandwidth available on the Internet, we have observed that TCP congestion control is often not a constraint of bulk data transmissions as bandwidth throttling is widely used in practice.**

**In this paper, instead of further manipulating the trade-off between the power saving and the incurred delay, we effectively explore the power saving potential by considering the bandwidth throttling on streaming/downloading servers. We propose an application-independent protocol, called PSM-throttling. With a quick detection on the TCP flow throughput, a client can identify bandwidth throttling connections with a low cost. Since the throttling enables us to reshape the TCP traffic into periodic bursts with the same average throughput as the server transmission rate, the client can accurately predict the arriving time of packets and turn on/off the WNI accordingly. PSM-throttling can *minimize* power consumption on TCP-based bulk traffic by effectively utilizing available Internet bandwidth without degrading the application's performance perceived by the user. Furthermore, PSM-throttling is client-centric, and does not need any additional infrastructure support. Our lab-environment and Internet-based evaluation results show that PSM-throttling can effectively improve energy savings (by up to 75%) and/or the QoS for a broad types of TCP-based applications, including streaming, pseudo streaming, and large file downloading, over existing PSM-like methods.**

## I. Introduction

The Internet has been dramatically advanced and significantly changed in two aspects. First, wireless Internet accesses become pervasive with the widely deployed WiFi networks on university campus, in business enterprises, public utilities, and residential houses. Second, media content has accounted for a high percentage of the Internet traffic volume. Under these two trends, more and more people are accessing Internet media services via wireless connections, on both mobile or portable devices such as laptops, PDAs, BlueTooth devices, and stationary desktop computers.

Mobile and portable devices are usually driven by battery power. Due to the limited battery capacity, it is essential to reduce power consumption on mobile devices without degrading the performance of applications, particularly for those applications that are QoS sensitive. The basic power saving method is to put the wireless network interface (WNI) into the sleep mode when it is idle, e.g., IEEE 802.11 power saving mechanism [10]. However, 802.11 power saving mode (PSM) may increase the connection round trip time due to the lagged data reception, and thus may significantly degrade the throughput of TCP-based applications. In order to achieve a high TCP throughput, the WNI has to be active to generate timely acknowledgments for received data. As a result, a significant amount of energy is wasted on channel listening [7], [9]. For applications like TCP-based streaming media, which has strict requirements on packet delay and can quickly drain out the battery of mobile devices, it is difficult to explore the trade-offs between the power saving and the caused delay to applications.

The power saving mode can be most effectively managed if the streaming traffic flowing to a client is in a predicable pattern, such as periodic bursts. Accordingly, the client can accurately adapt to streaming traffic pattern to sleep and to work periodically. Therefore, the power consumption on the client device is minimized while the demanded high throughput is also maintained. Efforts have been made towards this goal. However, existing solutions are either expensive or inefficient. For example, a proxy-based solution [5] is proposed to buffer and shape streaming media traffic into blocks, so that the data packets arrive at the client side with predictable intervals. Although clients can transit to lower power states during the block intervals without degrading application level performance, this solution needs a dedicated infrastructure support and is protocol dependent. Furthermore, RTSP-based Windows, RealNetworks, and QuickTime streaming services have their own extensions on the standard RTSP protocols [8], which have to be implemented individually for a general purpose RTSP proxy.

A client-centric scheme [13] is proposed to reshape the TCP traffic into bursts, and put the WNI into sleep between two bursts by modifying the client TCP stack. Besides lacking specific consideration for the streaming traffic, this scheme increases the data transmission time as a trade-off, which can be a high cost for some bulk data transmissions and unacceptable for streaming media applications with stringent QoS demands.

Streaming, pseudo streaming, and file downloading are the most commonly used media delivery approaches on the Internet today [8]. These techniques typically use TCP as the transmission protocol. With the increasingly abundant Internet bandwidth, the transmission rate of media traffic is often not constrained by the TCP congestion control mechanism on the network, but by the control on the server side, which we refer to as *bandwidth throttling*, due to an increasingly high demand of server resources. With bandwidth throttling, the Internet transmission is constrained by the server side instead of the available Internet bandwidth. That is, there may be idle Internet bandwidth without full utilization. Our observations show that bandwidth throttling has been commonly adopted in practice in a large variety of TCP-based bulk data transmission applications, in which media services are the most typical.

In this paper, we aim to take this unique opportunity offered by bandwidth throttling to exploit unused Internet bandwidth for power saving at the client side in WLANs. We propose an application-independent PSM protocol, called PSM-throttling, to significantly improve the power saving efficiency for bulk data communication applications with stringent QoS requirements. In PSM-throttling, with a quick detection of the TCP flow throughput, a client can identify bandwidth throttling connections with a low cost. Since the effective data transmission rate is often much lower than the available Internet bandwidth due to bandwidth throttling at the server side, the unused network bandwidth enables us to reshape the traffic into periodic bursts with an average throughput the same as the server transmission rate. With such periodic burst transmission patterns, idle and busy phases on the network transmissions can be clearly distinguished. Thus packet arrivals can be accurately predicted at the client side. As a result, the WNI can be turned on and off at the right time, in order to minimize energy consumption without degrading the user-perceived performance. The protocol can also detect dynamic changes of the server transmission rate in time with a small cost by tuning the burst size and burst intervals to maximize client perceived throughput and minimize energy consumption. Since PSM-throttling works at the transmission layer on the client and does not affect server transmission rate, it is application independent and client-centric.

Our Internet-based evaluation results show that PSM-throttling can effectively improve energy savings by up to 75% on the WNI or the QoS for a broad types of TCP-based bulk communications, including streaming, pseudo streaming, and large file downloading, than other power saving schemes.

The remainder of this paper is organized as follows. Section II presents our observations and measurements on typical bulk data transmission applications on the Internet and lab environments. We present PSM-throttling system designs in Section

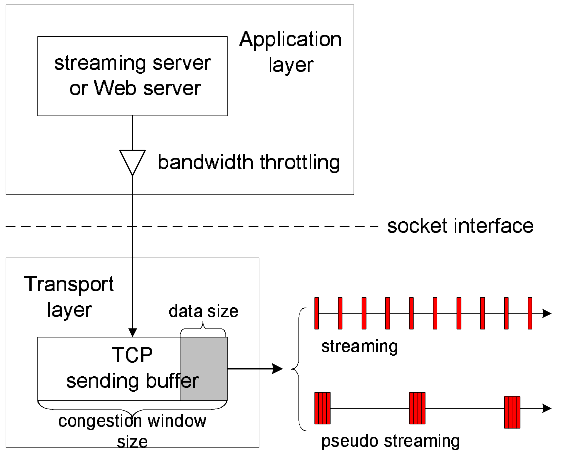


Fig. 1. Bandwidth throttling in streaming servers and Web servers

III. Based on an implemented prototype, we evaluate PSM-throttling with real experiments on the Internet environments and lab environments in Section IV. Section V outlines the related work on power saving in WLANs. Concluding remarks are made in Section VI.

## II. Bandwidth Throttling in TCP-based Media Transmissions

In this section, we characterize the performance of representative bandwidth throttling applications through Internet measurements, in order to explore the power saving space for TCP-based long duration and bulk data transmissions. We focus on the TCP-based Internet media delivery since media content is prevailing in online services and accounts for the majority of the Internet traffic [8].

Current Internet media traffic is mainly delivered via streaming, pseudo streaming, and file downloading techniques, where the bandwidth throttling is commonly used in practice.

- **Streaming**: Although traditionally UDP is the ideal protocol for streaming delivery, today TCP-based streaming accounts for more than 80% of the Internet streaming traffic, due to the wide deployment of NAT routers/firewalls and the overhead of protocol rollover [8].
  For streaming services, typically each stream is delivered at its encoding rate, even if there is more bandwidth available between the client and the server. Although Fast Cache [2] based streaming can deliver a media object with a rate up to five times of its encoding rate, it is not resource efficient and is disabled by most media services in practice [8]. Furthermore, with the increasing popularity of streaming services, a streaming server may need to serve hundreds or thousands of concurrent requests at the same time. Delivering a media object with a much higher rate than its encoding rate would significantly decrease the number of concurrent requests a server can service, and the user-perceived performance will be degraded when a burst of requests arrive.

- **Pseudo Streaming and Downloading**: Besides streaming services, many content providers and Internet media services, such as YouTube and Google video, leverage pseudo streaming techniques to deliver media content with common Web servers [6]. The transmission in pseudo streaming is essentially normal HTTP downloading. However, the client player can play the received data when a small playout buffer is fulfilled, without waiting for the complete downloading of the entire media.
  In order to serve a large number of concurrent requests, typically a pseudo streaming server has to limit the maximal throughput of each TCP downloading session, which is often much smaller than the end-to-end bandwidth between the server and its clients on the Internet.

Figure 1 illustrates the bandwidth throttling on either a streaming server or a Web server. The bandwidth throttling by a streaming server is often conducted in a fine granularity, so that the outgoing packets are evenly distributed in the stream. In contrast, the bandwidth throttling by a Web server is often conducted in a coarse granularity, and the outgoing packets may be sent with a bursty stream. Next, we present our Internet measurements to further understand the implications of bandwidth throttling on power saving.

First, we studied the widely used streaming services on the Internet, including RealNetworks media streaming and Window media streaming. All servers in our measurements, including Window media servers and RealNetworks media servers, are hosted by a CDN. For TCP-based RealNetworks media streaming, the server sends media packets with regular packet intervals. In order to test whether it is bandwidth-throttling or not (i.e., whether there is any unused bandwidth between the client and the server), we suppress the media transmission by setting the receiving window of TCP ACK to zero for 200 milliseconds at the client side, and then restore the original receiving window size to let the server send the data buffered in the TCP congestion queue. When the first packet is received, we choke the connection by sending a TCP ACK with zero receiving window size for 200 milliseconds again. Figure 2(a) shows the original data transmission sequence, while Figure 2(b) shows the sequences after our periodical choking. It shows that the TCP traffic becomes bursty, while the overall throughput keeps unchanged. The reason is as follows. As shown in Figure 1, when the TCP connection is choked by the client, the TCP layer at the server side cannot send more data. However, at the application layer, the streaming server continues to send data to the TCP layer, until the TCP congestion window is full. As a result, once the connection is unchoked, the TCP layer at the server side sends all data in the congestion window immediately. Since the average sending rate of the streaming server, i.e., the streaming rate, which is equal to the media encoding rate by default, is much smaller than the end-to-end bandwidth between the client and the server, when the buffered data is sent, no more data can be filled in the TCP congestion window in time, resulting the traffic bursts. For TCP-based Windows media streaming, we have similar observations, as shown in Figure 3(a) and Figure 3(b).

Second, we study the Internet pseudo streaming from YouTube. Figure 4 shows the time sequences of typical TCP connections of pseudo streaming media served by YouTube servers. Figure 4(a) shows the sequence without our interferences. The figure indicates that the traffic of YouTube is already bursty, due to the coarse granularity scheduling of packet sending for each connection.

Although the traffic bursts in pseudo streaming provide potentials to save energy by scheduling the on and off of the WNI, such bursts are not periodic and it is difficult for the client to predict the arriving time and finishing time of a burst. However, with a synchronized choking and unchoking on the client side, it is possible to predict packet arrivals in a high accuracy. Figure 4(b) shows the situation after our periodic choking is applied with a period of 200 milliseconds. The figure shows that both the burst length and the interval are approximately periodic with our choking scheme. Thus, the client can sleep and wake up at the right time and the energy consumption on the WNI can be minimized.

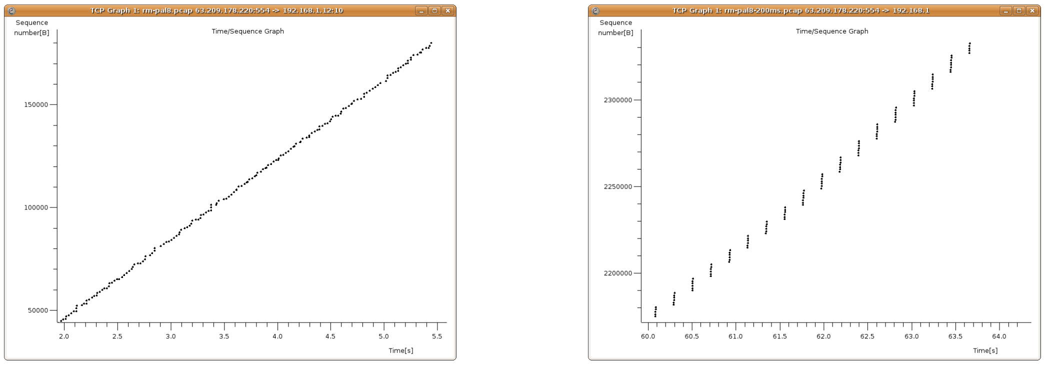
Although the above experiments confirm that bandwidth throttling is common in the Internet applications and it could be leveraged to save more power at the client side by choking and unchoking corresponding TCP flows, inappropriately flow choking may lead to unacceptable penalty. Figure 5(a) shows the time-sequence of a typical TCP connection for HTTP downloading served by a Apache Web server. Figure 5(b) shows the corresponding result after the traffic is shaped by periodic choking and unchoking. As shown in Figure 5(b), although the reshaped traffic becomes bursty and more energy could be saved, the TCP throughput is actually reduced. The reason is that in this case, the server does not use bandwidth throttling to limit the transmitting rate at the application layer. Choking the connection will cause the server to pause the data transmitting, but the TCP transmitting rate cannot be increased after unchoking, and the overall throughput is decreased. Therefore, choking and unchoking must be carefully used to reshape the traffic. Unless bandwidth throttling is detected, traffic reshaping via choking/unchoking is not encouraged.

Besides Internet experiments, we have also conducted experiments on Windows media server, RealNetworks media server, and Apache Web server in the lab by simulating the Internet environment with NIST Net emulator [1]. All the experimental results are consistent and confirm our observations.

## III. PSM-throttling Protocol Design

Our study in the last section shows that 1) bandwidth throttling commonly exists in various Internet applications, which implies that there are great potentials for further power savings on the WNI at the client side, and 2) the simple

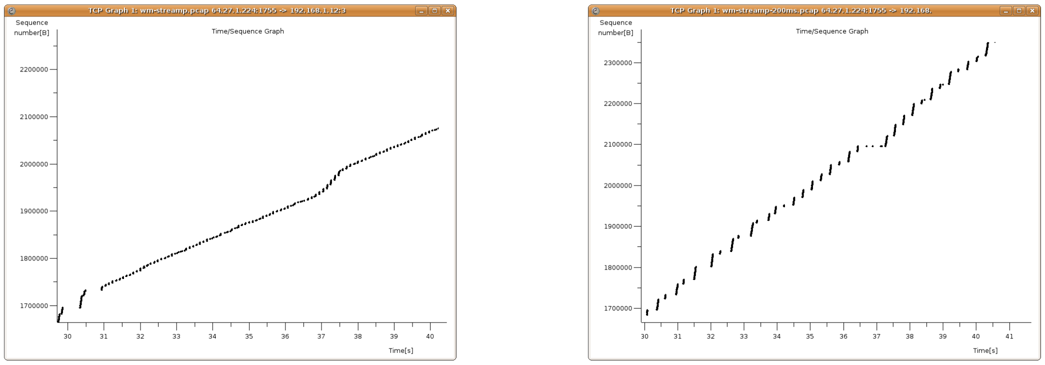
[1]http://www-x.antd.nist.gov/nistnet/



(a) Bandwidth throttling without reshaping

(b) Bandwidth throttling with reshaping

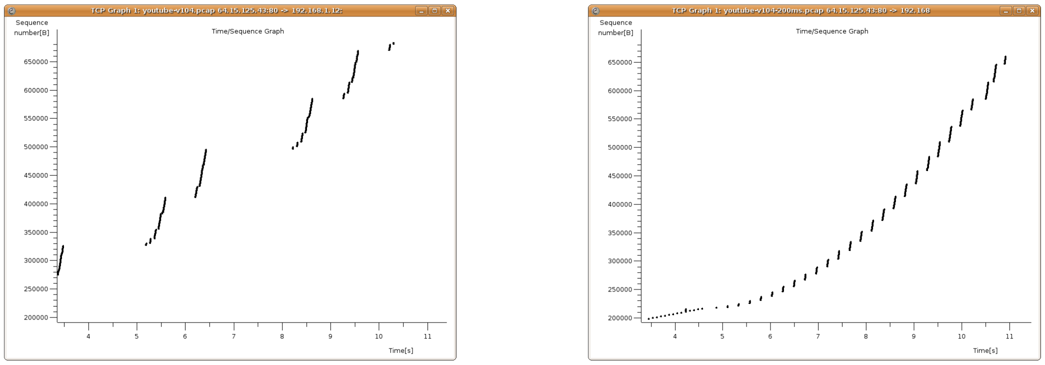
Fig. 2. Time-sequence graph of TCP-based streaming media by RealNetworks media server



(a) Bandwidth throttling without reshaping

(b) Bandwidth throttling with reshaping

Fig. 3. Time-sequence graph of TCP-based streaming media by Windows media server



(a) Bandwidth throttling without reshaping

(b) Bandwidth throttling with reshaping

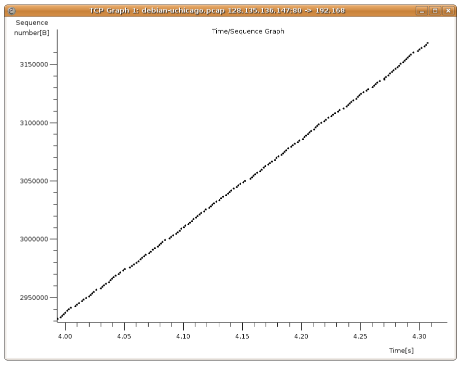
Fig. 4. Time-sequence graph of YouTube streaming

choking and unchoking scheme to reshape the traffic may affect the performance of the running application. Therefore, to explore this power saving space, first, the bandwidth-throttling must be correctly detected in time. Second, the client must be able to predict the packet arrivals accurately so that the WNI can be turned on and off at the right time.

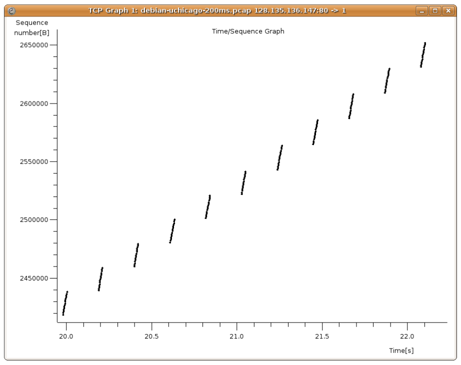
Aiming to achieve maximal power savings without degrading application level performance, in this section, we propose PSM-throttling, an efficient power saving mechanism for TCP-based bulk data communications. After presenting our detection algorithm, we will present our two level traffic burst generation algorithm. Lastly, we discuss how our proposed protocol promptly adapts to the fluctuations of the server transmission rate and network transmission rate.

## A. Bandwidth Throttling Detection

In the experiments presented in Section II, we have shown that the traffic of pseudo streaming from YouTube is already bursty, for which bandwidth throttling is easy to be identified. However, other applications, such as TCP-based streaming



(a) Bandwidth throttling without reshaping



(b) Bandwidth throttling with reshaping

Fig. 5. Time-sequence graph of TCP-based file downloading by Apache Web server

media, does not normally have traffic bursts. In order to exploit the under-utilized bandwidth on the Internet for power savings at the client side, a quick bandwidth throttling detection algorithm with low overhead is designed as follows.

First, the detection algorithm needs to measure the round trip time (RTT) between a client and its server. Thus, the TCP timestamp option is enabled in the TCP header when the client initiates a TCP connection. Once a TCP connection is established, the protocol agent monitors the flow rate and the transmission duration. If the TCP connection maintains a stable flow rate $r$ for a specific duration $T_0$ ($T_0 > 5$ seconds), the protocol agent begins to test the bandwidth utilization of this connection. The threshold $T_0$ is set based on the the default playout buffer size of widely used media players such as Windows media player [1]. The throttling test is conducted as follows. The client sends a choke ACK to the server, in the TCP header of which the receive window is set to zero. After sending the choke ACK, the client will still receive packets for a RTT, because it takes half of a RTT for the choke ACK to arrive at the server. Then after two RTTs, the client sends an unchoke ACK, and restores the original receive window size. Thus, the server will buffer the data sent from the application layer for two RTTs, and then send these buffered packets in a burst if they can be held in a congestion window. Upon receiving the first packet after sending the unchoke ACK, the protocol agent can estimate the flow rate $r'$ for the $2RTT * r$ number of bytes. If $r' \geq 2r$, that means the server only uses less than half of the end-to-end bandwidth for the data transmission, and we can exploit this potential to save energy at the client side.

In this detection algorithm, the threshold of end-to-end bandwidth to media encoding rate ratio is set to 2 above which PSM-throttling is enabled. That is, if the un-utilized bandwidth is less than the end-to-end bandwidth, PSM-throttling will not be activated. One reason is that if the un-utilized bandwidth is too small, the power saving would be trivial and may not offset the overhead. This will also be considered in the burst generation. On the other hand, some previous research has studied the relationship of the end-to-end bandwidth with the

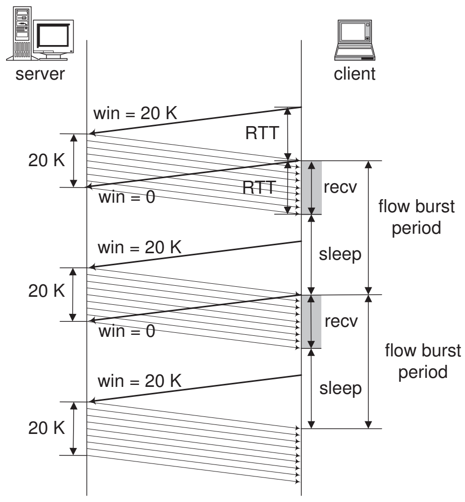


Fig. 6. TCP level burst generation with flow choking and unchoking

object encoding rate, i.e., the ideal data transmission rate. For example, Wang et al. [11] have conducted a modeling study and found TCP-based streaming can achieve good performance when the end-to-end bandwidth between the client and the server is about twice of the object encoding rate, with a small playout buffer for only a few seconds. A measurement study by Guo et al. [8] shows that transmitting the media traffic with a much higher rate than the object encoding rate helps little on the client performance while limits the system capacity for serving more client requests. Therefore, in our current design and experiment, the bandwidth to media encoding rate threshold is set to 2.

## B. Two Level Traffic Burst Generation

After bandwidth throttling is detected, we can start to reshape the traffic to form periodic packet bursts. We achieve this through well tuned two level burst generation schemes. Accordingly, the client is able to predict when a packet burst arrives, and when the packet burst terminates. In addition, the interval between bursts should be non-trivial so that the WNI can be turned off.

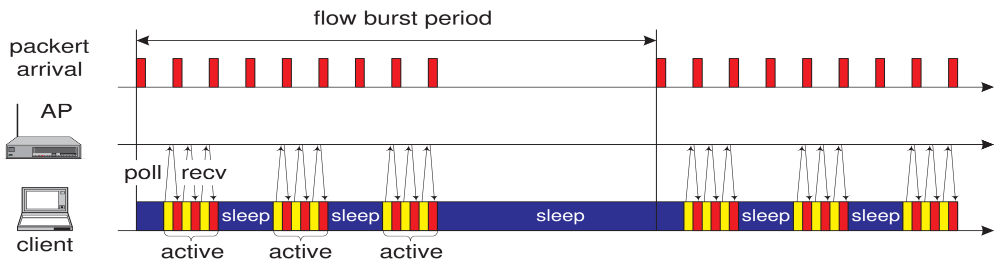


Fig. 7. MAC level reconciliation for burst generation with access point buffering

*1) Initial Traffic Burst Generation at TCP Layer:* For the TCP level burst generation, assume the connection round trip time is $RTT$, the end-to-end bandwidth between the client and the server is $BW$. Denote the media transmission rate as $r$, according to our detection algorithm, $BW > r$. Assume the duration of receiving a packet burst is $T_{recv}$ and the duration of the sleep interval is $T_{sleep}$, and the duration of a flow burst period, $T_{burst}$, is the sum of $T_{recv}$ and $T_{sleep}$. As long as the sleep duration is not trivial, this burst generation mechanism is engaged to save power in a coarse granularity.

The server traffic bursts are generated by leveraging the TCP flow control as follows. As shown in Figure 6, after bandwidth throttling detection, the client sends an unchoke ACK to the server, specifying the receive window size in the TCP header. Upon receiving the unchoke ACK, the server sends the number of bytes that the client requests to the network interface (if it is smaller than congestion window size) together in a burst. Then after the client receives the first data packet, it sends a choking ACK with zero receive window in the TCP header. Then after a RTT, the client receives the last packet the server sends in the burst. In order to let the choking ACK sent by the client block the data flow before the client receives the last packet in the burst, the burst duration $T_{recv}$ should be larger than one RTT.

The unchoke ACK must be sent before a RTT when the client wakes up to receive data. Since the unchoke ACK may be lost, the client may need to retransmit this packet when necessary. As a result, the next choke ACK cannot be sent before the client receives the first data packet for the current unchoke ACK. That is, a flow burst period must not be less than two RTTs.

Note that in the above settings, by setting the receive window size in the TCP headers of ACK packets, the client can specify the total number of bytes in a packet burst. When the entire burst has been received by the client, the client can safely put the WNI into sleep to save energy.

*2) Traffic Burst Reconciliation at MAC Layer:* Following the above protocol, we can get traffic bursts. However, this initial design would not enable us to *minimize* the power consumption on the WNI. The problem comes from the mismatch transmission speed on the Internet and the WLAN.

Typically, for most high speed Internet users, the end-to-end bandwidth between a client and its server is about 1 - 2 Mbps, while the effective bandwidth in current WiFi networks such as 802.11g can be up to more than 24 Mbps. In such a WLAN, there are still great potentials to save energy further if the WLAN is lightly loaded, because the packet transmission over wireless networks is much faster than that over the Internet: with our initial traffic burst generation algorithm, the burst traffic is transmitted with the rate of end-to-end bandwidth. If the number of packets in a burst is large, the interval of successive packets when they arrive is non-trivial. A significant amount of energy will have to be wasted to keep the WNI awake to wait for these packets if nothing further is performed.

Motivated by the speed mismatch of the Internet and the WLAN, we further refine our burst generation protocol at the MAC layer through access point buffering. The basic idea is as follows. We divide a long TCP burst into several MAC frame bursts. The duration of each MAC frame burst should be short enough so that buffering them on the access point will trivially affect the RTT estimation. Meanwhile, the duration of each MAC frame burst should be long enough so that the energy cost of mode switching of the client WNI is smaller than the energy saved for receiving this burst. For many commercial WNI products like what we use in our experiments, the mode switching overhead is about 4 ms. Thus, we set the duration of each MAC frame burst as 20 ms.

Figure 7 shows the generation of MAC level bursts. The arriving TCP bursts are buffered at the access point. The client can predict the burst arrival accurately, since the burst arrives a RTT after the unchoke ACK. The client also knows the number of bytes in this burst. Thus, the client waits for about 20 ms so that there are enough packets buffered at the access point, and then polls the access point to receive data. After receiving one MAC level frame burst, the client sleeps another 20 ms and then polls the access point. As a result, in a lightly loaded WLAN, the client can save more energy.

When the WLAN is heavily loaded or even congested, the poll message sent by the client may not be responded quickly. In this case, the client may wait for a longer time to receive the polled packet. As a result, after the client polls all packets in a MAC level burst, it may have no time to sleep, and will continue to poll the next burst, which will not outperform the

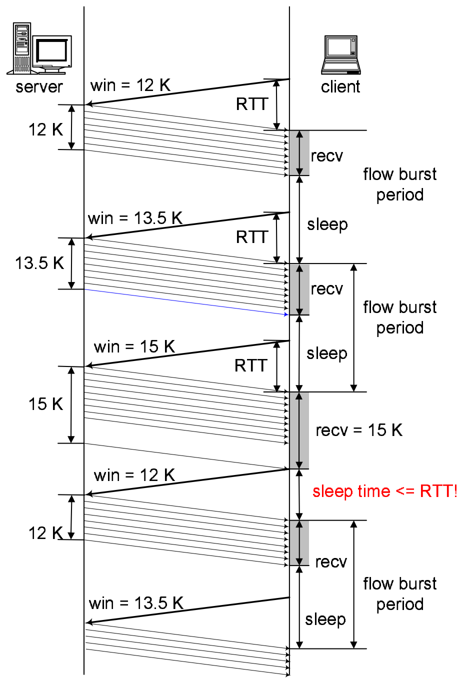


Fig. 8. Adaptation to server transmission rate

initial TCP layer traffic burst generation algorithm.

### *C. Adaptation to Network and Server Transmission Fluctuations*

On the Internet, the end-to-end bandwidth between the client and the server may fluctuate from time to time. As a result, it is difficult to predict the duration of a TCP level burst at the client side. A delay in packet receiving may cause prolonged RTT or even time out, which may affect the TCP performance significantly. On the other hand, if the client keeps awake waiting for packets in a burst, the energy consumption for idle awake time may be non-trivial, since it cannot predict burst duration accurately.

In the design of PSM-throttling, we have considered such a situation. As discussed before, a client detects when a burst ends based on the number of bytes it receives, not based on the duration of the burst, since the number of bytes in a burst is specified by the receive window size. Thus, the client only needs to keep awake receiving the specified number of packets, and the sleep time of the client WNI is the remaining time during a burst period. In this way, the network bandwidth fluctuations can be smoothed out automatically.

In addition to the network fluctuations, the server transmission rate may change, although in most bandwidth throttling transmission cases, such cases are rare. A server may increase or decrease its transmission rate suddenly during a user session. For example, in streaming media services, a user may use fast forward. In the response, the server transmits media content to the client in a higher speed (about five times of the media encoding rate for Windows media services). If our detection protocol cannot detect such changes in time, the user may experience a degraded fast forwarding experience.

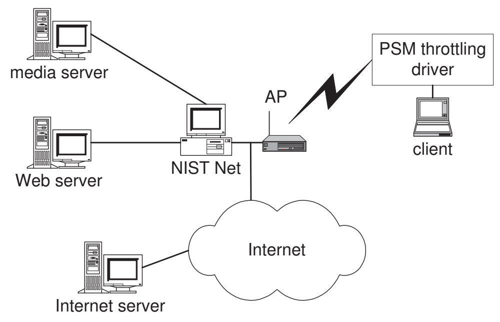


Fig. 9. Implementation testbed

In order to quickly detect the server transmission rate variations, in PSM-throttling, the receive window size in the TCP ACK headers is dynamically adjusted. Initially, the receive window is set based on the average transmission rate estimated during bandwidth throttling detection. PSM-throttling monitors the sleep time, as well as the average throughput during a flow burst period. If the sleep time falls below a RTT, it is a signal that the receive window size is so large that it takes too long to receive the entirety of specified bytes. PSM-throttling will then decrease the receive window size by two packets in order to recover from this situation. If the observed average throughput is the same or even higher than the average throughput in the previous period, it is highly likely that the receive window size is still less than the server-side buffer size. PSM-throttling will increase the receive window size in order to drain off the server-side buffer. Figure 8 illustrates this scheme. We will evaluate its effectiveness in Section IV.

## IV. Performance Evaluation

In this section, we evaluate PSM-throttling based on a prototype we have implemented. Figure 9 shows the architecture of the prototype system. The client runs Linux with kernel 2.6.18, equipped with D-Link DWL-G520 wireless card (Atheros chipset). We have implemented our protocol based on the `madwifi` driver `0.9.2`. In order to emulate the bandwidth, the round trip time, and the loss rate of the Internet bulk data transmission, we run NIST Net emulator 2.0.12b under Linux kernel 2.4.27. The experiments are run with the prototype system to access Windows media streaming, RealNetworks media streaming, YouTube pseudo streaming, and common HTTP downloading. For Windows and RealNetworks media streaming, we use MPlayer v1.0rc1, a movie player on Linux. MPlayer can support Windows media service with MMS and RealNetworks media service with RTSP protocol. The encoding bit rates of media objects used in Windows streaming, RealNetworks streaming, and YouTube pseudo streaming evaluations are 330 Kbps, 350 Kbps, and 300 Kbps, respectively. For YouTube, we test a YouTube video with the Adobe Flash Player plug-in for Linux. Due to page limit, we only present our representative results conducted on the Internet, the results of experiments in the lab environment are similar and omitted.

Four different metrics are used for the performance comparisons of PSM-throttling with other four mechanisms. We measure the average TCP throughput, the energy consumption, the total object transmission time, and the total awake time of the WNI when the experiments are conducted. PSM-throttling (denoted as *PSM-T* on the figure) is evaluated against Continually Aware Mode (denoted as *CAM*), the Client-Centered (denoted as *CC*) power saving approach [13], 802.11 power saving mode (denoted as *PSM*), and a PSM adaptive (denoted as *PSM-A*) approach used in commercial WNIs.

Among the four approaches we have implemented for performance comparisons, with Continually Aware mode (CAM), the client WNI always keeps awake even it is idle for a long time. CAM can provide the best TCP performance since it does not delay any packet. However, it consumes a significant amount of energy due to continuous idle awake time. In contrast, with 802.11 power saving mode (PSM), the client WNI only wakes up to listen to the beacon message. When there is a traffic notification in the beacon, the client polls the access point to receive data, and then returns to sleep mode again. Due to its significant impact to TCP throughput by increasing the round trip time, PSM is rarely used in practise although its power consumption is often quite low. We evaluate this scheme here in order to have an idea on whether we can achieve the minimum power consumption in our experiments.

The client-centered power saving approach (CC) aims to reduce the energy consumption on TCP downloading without increasing the product of energy and transmission delay, i.e., `energy`$\times$`delay`, which could save energy consumption with the cost of increased transmission time.

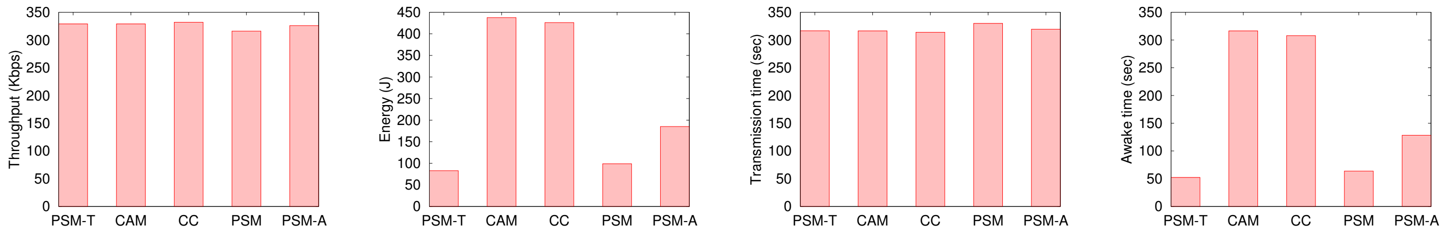
The PSM adaptive (PSM-A) approach has been widely used in commercial products recently by switching between PSM and CAM mode adaptively. Initially, the overhead of mode switching on the WNI is non-trivial. With the help of advanced hardware technologies, this overhead has been significantly reduced. For example, as reported in 2003 [3], the mode switching overhead could be as high as 100 ms for Cisco Aironet wireless cards. In contrast, our experiments on latest Atheros chipset WNI show this overhead is only about 4 ms. As a result, recently many manufacturers of laptops and PDAs have adopted this PSM adaptive method to save energy with little network performance degradation. For example, with this trivial mode switching overhead, the system built-in wireless cards in IBM ThinkPad laptops can automatically go to sleep after it is idle for 75 ms, and wake up when receiving a traffic notification beacon.

Figures 10(a), 10(b), 10(c) and 10(d) show the throughput, the energy consumption, the object transmission time, and the WNI awake time, for Windows streaming media services in different power saving approaches. As shown in Figure 10(a) and Figure 10(b), PSM-throttling achieves the maximal throughput (same as that of CAM) and the minimum energy consumption (about 25% of that in CAM) among all five approaches. Compared to the most advanced mechanism used in commercial products, the energy consumption of PSM-throttling is only about 50% of PSM-A due to traffic reshaping in PSM-throttling. This amount of power savings is due to the minimum awake time of the WNI in PSM-throttling as indicated on Figure 10(d) (in fact, the sleep time is often comparable to awake time for bulk data transmission, and thus does not contribute much for power consumption). For Windows media streaming, PSM-throttling does not increase the transmission time. Figure 10(c) shows that the transmission time for all five approaches are similar (except for PSM). This is because the streaming server transmits data in a low rate continuously.

Figures 11 shows the corresponding results for RealNetworks streaming media services. Similar to Windows media streaming, PSM-throttling achieves the same throughput as that of CAM while it consumes approximately the minimum amount of energy among all five approaches. Again, it is only about 25% of that in CAM and the most advanced PSM-A. PSM degrades the QoS experienced by the user because the throughput is reduced to be below 310 Kbps, while PSM-throttling can maintain the desired media streaming quality although its power consumption is a bit higher than that of PSM. Comparing Figure 11(b) to Figure 10(b), we find that the energy consumption of PSM-A in Windows media streaming is much smaller than that in RealNetworks media streaming. The analysis reveals that for TCP-based streaming, Windows media services send media data to the client in block. Thus, the TCP streaming traffic of Windows media services is already bursty. So the client in the PSM adaptive mode can go to sleep to save energy after 75 ms of a burst. In contrast, in RealNetworks media streaming, the server transmits media data to the client packet by packet evenly. Since the interval between two successive packets is small, the WNI of the client cannot sleep.

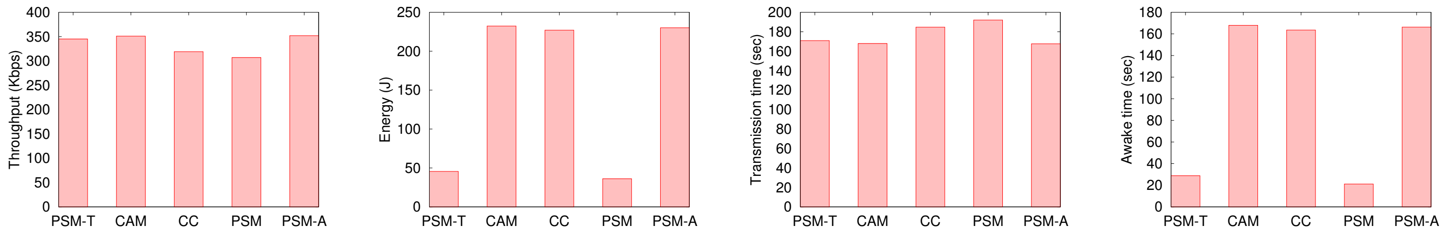
Figures 12 shows our experimental results when YouTube video is accessed with different power saving approaches. Figure 12(a) shows that the throughput of PSM-throttling is comparable to that of CAM. On the other hand, Figure 12(b) shows the energy consumption of PSM-throttling is much less than CAM and PSM adaptive. Our evaluation results also show that the throughput and energy savings in CC are the worst among the five. This is due to the largest transmission time and awake time as shown in Figure 12(c) and Figure 12(d). These results indicate that this scheme may not be suitable for YouTube video services.

For HTTP downloading, Figures 13 shows the corresponding results achieved by different power saving approaches. In terms of power savings, Figure 13(b) shows PSM-throttling is among the lowest of all schemes. The throughput achieved by



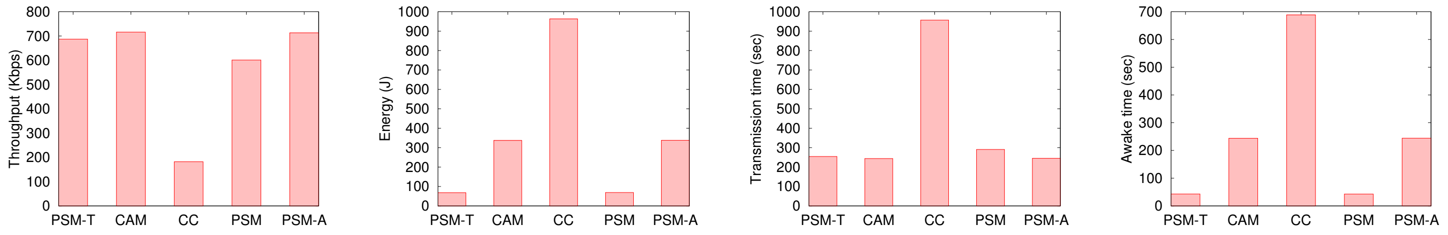
(a) Throughput (Kbps) (b) Energy consumption (mJ) (c) Transmission time (ms) (d) Awake time (ms)

Fig. 10. TCP-based Windows media streaming



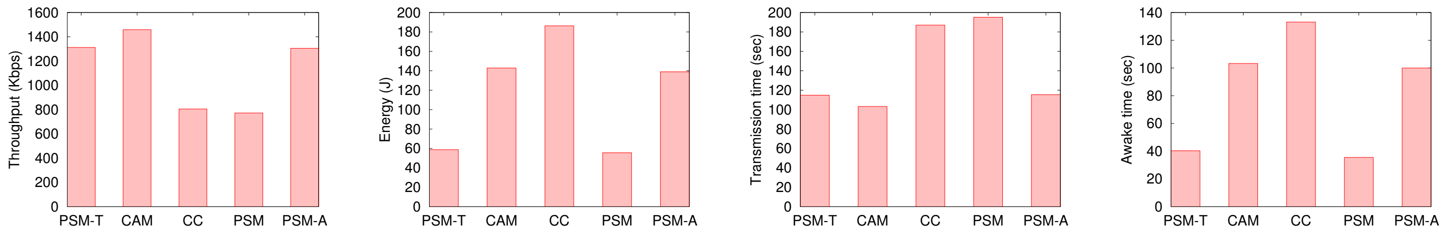
(a) Throughput (Kbps) (b) Energy consumption (mJ) (c) Transmission time (ms) (d) Awake time (ms)

Fig. 11. TCP-based RealNetworks media streaming



(a) Throughput (Kbps) (b) Energy consumption (mJ) (c) Transmission time (ms) (d) Awake time (ms)

Fig. 12. YouTube pseudo streaming



(a) Throughput (Kbps) (b) Energy consumption (mJ) (c) Transmission time (ms) (d) Awake time (ms)

Fig. 13. HTTP downloading with bandwidth throttling

PSM-throttling and PSM-A is close to CAM while PSM and CC get the worst throughput as shown in Figure 13(a).

Our evaluation results show that PSM-throttling is very effective for the widely used Internet TCP-based bandwidth throttling services. However, for non-bandwidth throttling services, although it can save power, it may increase the delay due to degraded throughput. Thus, a trade-off must be carefully balanced if PSM-throttling is used in such applications.

We also evaluate the effectiveness of PSM-throttling in adaptation to server transmission fluctuations by playing an RMVB (RealMedia Variable Bitrate) video. Figure 14 shows that PSM-throttling is able to detect the change of server transmission rate and achieve the similar transmission fluctuations as CAM when the media encoding rate changes.

## V. Other Related Work

With pervasive wireless Internet accesses, research issues of power saving and utilization on mobile devices have been paid attention. For example, a self-tuning power management

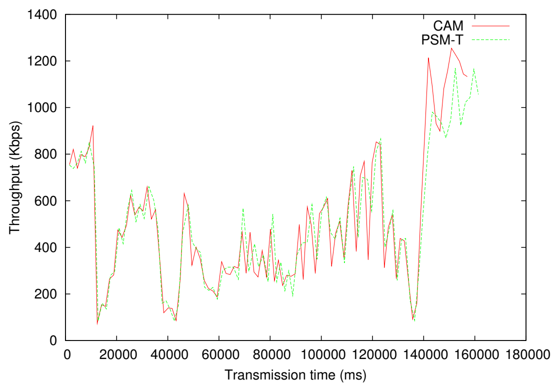


Fig. 14. PSM-throttling adaptation to server transmission fluctuations

approach to adapting the behavior of a station's WNI to the access pattern and the intent of its applications is proposed in [3], and a cooperative relay service to exploit the idle communication power of WNI to improve network throughput is studied in [7].

As power saving is critical to applications with long and bulk data transmissions, a number of studies have focused on Internet media applications. However, existing power saving studies for media traffic mainly focus on UDP-based media data transmissions. For example, work in [4] characterizes the traffic patterns of Windows, Real, and QuickTime streaming media services, and analyzes the implications on the energy consumption on the WNI under varying stream bandwidth and network loss rates. This paper shows 802.11 PSM does not offer any energy savings for multimedia streams over 56 Kbps for commercial access points. A number of packet prediction algorithms have been proposed to put the WNI into sleep and wake up the WNI based on the prediction of packet arrivals, such as history-based prediction strategies [4] and linear prediction-based strategy [12]. In addition to the proxy-based work in [5], a priority-based bulk scheduling for proxy buffering is proposed in order to provide delay assurance and achieve power efficiency simultaneously [14].

However, TCP accounts for more than 90% media traffic on the present Internet [8]. Due to the TCP congestion control mechanism, the power saving for TCP-based streaming is more difficult than UDP-based streaming, although a number of studies have been conducted in order to reduce power consumption for TCP-based communications. The effect of prolonged connection round trip time on Web traffic has been studied, and a bounded slowdown algorithm to save energy and bound the throughput reduction within a specified range is proposed [9]. While these existing studies aim to reduce power consumption for applications constrained by the TCP congestion control mechanism, nowadays on the Internet, for many TCP based long and bulk data communications, such as streaming, pseudo streaming, and file downloading, the TCP congestion control is no longer a constraint with the increasingly abundant Internet bandwidth. Instead, the transmission rate is controlled by the server due to the high resource demand per stream in media delivery [8]. Targeting such applications, instead of manipulating the trade-off between the power saving and application performance, our PSM-throttling scheme aims to utilize under-utilized bandwidth to minimize the power consumption of bandwidth throttling applications without degrading user-perceived performance.

## VI. Conclusion

Effectively saving the limited battery power of mobile devices is a key issue for improving increasingly pervasive wireless Internet accesses. Instead of further addressing the trade-off between power-saving and the incurred delay of data communications, we have explored a unique opportunity from bandwidth throttling that has been widely adopted in practice. Accordingly, in this paper, we have presented our design and implementation of a new power saving protocol, called PSM-throttling, to reduce power consumption on wireless devices in bandwidth throttling bulk data communications. Our experimental evaluation results show that PSM-throttling has the following merits: (1) it can minimize the power consumption of the WNI without degrading the application performance; (2) it is client-centric and does not demand any infrastructure support; and (3) it is application independent, and is highly effective for both typical and emerging Internet applications.

## References

[1] Buffer settings in Windows media player. http://support.microsoft.com/?scid=kb;en-us;q257535.

[2] Fast streaming with windows media 9 series. http://www.microsoft.com/.

[3] M. Anand, E. Nightingale, and J. Flinn. Self-tuning wireless network power management. In *Proc. of ACM MOBICOM*, Sept. 2003.

[4] S. Chandra. Wireless network interface energy consumption implications of popular streaming formats. In *Proc. of MMCN*, 2002.

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

[6] L. Guo, S. Chen, Z. Xiao, and X. Zhang. Analysis of multimedia workloads with implications for Internet streaming. In *Proc. of WWW*, May 2005.

[7] L. Guo, X. Ding, H. Wang, Q. Li, S. Chen, and X. Zhang. Exploiting idle communication power to improve wireless network performance and energy efficiency. In *Proc. of IEEE INFOCOM*, April 2006.

[8] L. Guo, E. Tan, S. Chen, Z. Xiao, O. Spatscheck, and X. Zhang. Delving into internet streaming media delivery: a quality and resource utilization perspective. In *Proc. of ACM SIGCOMM/USENIX IMC*, Oct 2006.

[9] R. Krashinsky and H. Balakrishnan. Minimizing energy for wireless Web access with bounded slowdown. In *Proc. of ACM MOBICOM*, Sept. 2002.

[10] LAN/MAN Standards Committee of the IEEE Computer Society. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Technical Report IEEE Std 802.11, the Institute of Electrical and Electronics Engineers, Inc., 1999.

[11] B. Wang, J. Kurose, P. Shenoy, and D. Towsley. Multimedia streaming via TCP: An analytic performance study. In *Proc. of ACM Multimedia*, 2004.

[12] Y. Wei, S. M. Bhandarkar, and S. Chandra. A client-side statistical prediction scheme for energy aware multimedia data streaming. *IEEE Transactions on Multimedia*, 8(4), 2006.

[13] H. Yan, R. Krishnan, S. A. Watterson, D. K. Lowenthal, and K. Li. Client-centered energy and delay analysis for TCP downloads. In *Proc. of IWQoS*, June 2004.

[14] H. Zhu and G. Cao. A power-aware and QoS-aware service model on wireless networks. In *Proc. of IEEE INFOCOM*, Mar. 2004.