A Power-Aware Scheduler For Streaming Multimedia Clients

by

Michael Gundlach

(Under the direction of David K. Lowenthal)

Abstract

Mobile computers consume significant amounts of energy when receiving streamed multimedia data. The wireless network interface card (WNIC) consumes a large portion of this energy. One way to reduce the energy consumed is to transmit the packets to clients in a predictable fashion. Specifically, the packets can be sent in *bursts* to clients, who can then switch to a lower power *sleep* state between bursts. This technique is especially effective when the bandwidth of a multimedia stream is small.

This paper investigates techniques for saving energy in a multiple-client scenario, where clients may be either (1) viewing streaming multimedia or (2) receiving background (e.g., web) data. Given the real-time requirements of multimedia traffic, we differentiate network traffic into two types: multimedia and background. We then generate different global schedules for multimedia packets and background packets. Several difficulties arise with multiple clients, including delays, *droughts* and bandwidth limitations. Despite this, multimedia clients using our schedule save significant energy with few missed packets, while general clients achieve reasonable end-to-end latency. For example, our schedule saves over 90% of the energy for ten multimedia clients viewing 56kbps video streams. With an additional 220kbps of web traffic, end-to-end latency averages 32 ms, while multimedia clients save 86.9%.

Index words: multimedia, scheduling, energy, power-aware, wireless networking

A Power-Aware Scheduler For Streaming Multimedia Clients

by

Michael Gundlach

B.S., The University of Georgia, 2002

A Thesis Submitted to the Graduate Faculty
of The University of Georgia in Partial Fulfillment
of the
Requirements for the Degree

Master of Science

Athens, Georgia

2002

A Power-Aware Scheduler For Streaming Multimedia Clients

by

Michael Gundlach

Approved:

Major Professor: David K. Lowenthal

Committee: Scott A. Watterson
John A. Miller

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
December 2002

Table of Contents

Chapter 1

Introduction

One significant source of consumed energy on mobile computers is the wireless network interface card (WNIC); in fact, it is often the single largest power drain in a mobile client. This is especially true in streaming multimedia applications; packets arrive at a frequent and steady rate, requiring the WNIC to remain in *idle* or *receive* mode, both of which use significant amounts of power.

Fortunately, by buffering packets for a client and sending them to the client in *bursts* at regular intervals, the client can keep its WNIC in *sleep* state for much of the duration of the stream; this state consumes an order of magnitude less power. This is especially effective when the bandwidth of the stream is low, in which case a large fraction of the overall energy can be saved.

We have designed a scheduling policy that bursts packets to clients and implemented it in a proxy. Specifically, our proxy:

- allows any number of clients to save energy by receiving data in bursts from a wireless access point,
- allows both multimedia and general clients to participate,
- effectively uses *delay compensation algorithms* to deal with wireless access point routing delays, and
- handles *droughts*, where no data is sent to a client for a period of time.

Our results show that a single 56kbps multimedia client that is connected to the proxy can use as little as 8.7% of the energy of a client that keeps its WNIC in high-power mode exclusively (a *naive client*). When ten 56kbps multimedia clients are connected to the proxy, they use an average of 6.9% with little or no packet loss. When an average of 220kbps of background (e.g., web) traffic is transmitted by the proxy as well, multimedia clients use an average of 13.1% while general clients use 8.4%, with an average of 32 ms end-to-end latency for their data.

Overall, we found that significant energy can be saved for several low bitrate streams up to the bandwidth of the wireless network. Moreover, this can be done with little effect on end-to-end latency of general clients downloading background data. For higher bitrate streams, effective energy saving is still possible, but becomes more difficult when general streams are added.

The rest of the paper is organized as follows. Chapter 2 discusses related work. Chapter 3 describes the design and implementation of our scheduling policy in a proxy, as well as the implementation of a simulator used for determining energy savings. Chapter 4 describes our experiments and discusses the results. Chapter 5 summarizes and discusses future directions for this research.

Chapter 2

Related Work

In the last decade, there has been significant research in power-aware computing; most aspects of an operating system have been studied from the perspective of power. For example, scheduling has been investigated [21, 10]; the basic idea is to use fine-grain control of the clock speed to give the appearance of full computational power at peak times, but less computational power at other times; this saves battery power. Also, disk spindown can be used to save energy [12, 7, 22, 1, 16]; in general, algorithms attempt to determine when there is a large time period in which there are no disk requests. In some architectures, memory banks can be powered down [6, 15]. In addition, energy can be saved by allowing the battery to recover [4]. Finally, routing can be performed in a power-aware manner [18, 5, 13].

Recent work has advocated managing energy explicitly as a resource in the operating system [23, 20, 8]. Another approach is to have the OS cooperate with applications to save energy [9]. Kravets investigated power-aware mechanisms for end-to-end communication in wireless networks [14].

Other related approaches are to use the energy saving mechanisms defined by 802.11b or the new 802.11e with Whitecap technology [17]. The former is not a good match for multimedia [2], and the latter is still upcoming.

Previous work in bursting data to clients is detailed in [2, 3], which provides energy savings for a single multimedia client for Quicktime, Real, and MS Media. While the system effectively saves energy for a single client, the scheduling policy

developed in that work caused high rates of collision and packet loss for multiple clients. Our work redesigns the algorithm for multiple clients while also improving the single-client case by handling *droughts*. In addition, our algorithm supports situations in which *both* multiple multimedia clients and general clients execute concurrently.

Chapter 3

Implementation

This chapter describes our implementation. Section 3.1 gives an overview, and Section 3.2 describes our scheduling policy and its implementation. Section 3.3 describes *delay compensation* algorithms used on clients to adjust to routing delays in the wireless access point. Section 3.4 describes the conditions that can cause a client to receive *no* data during a burst interval, and how the proxy responds to this situation. Section 3.5 describes how we compare different delay compensation algorithms using a simulator.

3.1 Overview

Our implementation is within a proxy that is interposed between servers and clients. It buffers incoming data, and at regular intervals transmits it in a burst to the appropriate client. There are two classes of clients that connect to the proxy in our experiments: *multimedia clients*, which are mobile computers that request and receive exactly one multimedia stream from the proxy, and *general clients*, which are mobile computers that do some other type of communication with the proxy, but do not request a multimedia stream.

We collect a trace of the wireless-side activity using a packet sniffer running on a mobile computer known as the *monitoring station*; this trace is read post-mortem in order to determine energy used per client. We assume that a wireless network interface card (WNIC) can be in *sleep*, *idle*, *receive*, or *transmit* mode. *Sleep* mode

uses a very small amount of power, and during this time no data can be received or transmitted. The remaining modes use a relatively large amount of power, with *receive* and *transmit* modes somewhat larger than that used by *idle* mode [19, 11]. We therefore refer to *sleep* mode as *low-power mode* and all other modes as *high-power mode* for the remainder of the paper. A client saves energy by transitioning its WNIC between high- and low-power modes according to our scheduling policy, which is described next.

3.2 Scheduling Policy

This section will describe the design and implementation of our energy-conserving scheduling policy. We use a proxy placed between the server and the wireless access point to carry out this policy. Section 3.2.1 discusses the design of our policy, and Section 3.2.2 details our implementation of the policy.

3.2.1 Design

The proxy's primary purpose is to transform regular multimedia streams into bursted streams, scheduling bursts so that each multimedia client is given equal bandwidth. The proxy also schedules background traffic so that it is not forwarded during any multimedia client's burst.

We define a *rendezvous point* (RP) as a moment in time at which the proxy agrees to begin sending data to a client. The scheduling policy (see Figure 3.1) creates a series of alternating multimedia and general rendezvous points. Each multimedia client is assigned exactly one multimedia RP. The proxy buffers all UDP multimedia data for a given multimedia client in that client's *packet queue* until the client's RP arrives; at that point the proxy transmits all of the data in the packet queue to the client in a burst, marking the last packet so the client knows when to transition

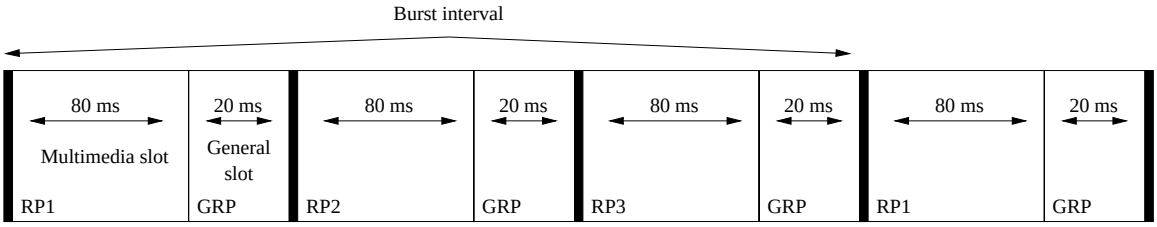

Figure 3.1: Example scheduling policy allowing three multimedia clients.

its WNIC to low-power mode. The multimedia clients are served in a round-robin fashion: each multimedia RP is assigned to a different client, and the amount of time between RPs for a given client (referred to as the *burst interval*) is the same for each client (see Figure 3.1). For example, in Figure 3.1, RP_2 is assigned to multimedia client 2 and arrives every 300 ms. The time between a multimedia RP and the following general RP is referred to as a *multimedia slot*.

The proxy buffers all incoming background data (including TCP communication to or from multimedia clients) in a single *general queue* until a general RP arrives, at which point it transmits all of the data contained in the general queue in a burst. The time between a general RP and the following multimedia RP is referred to as a *general slot*. If at any time during the general slot additional background data arrives, it will be sent out immediately by the proxy.

The proxy's burst interval, the size of a multimedia slot, the size of a general slot, and the maximum number of multimedia clients are set upon initialization of the proxy and are static. These four variables precisely define a scheduling policy for the proxy. For example, in Figure 3.1, the burst interval is 300 ms, the multimedia slot size is 80 ms, the general slot size is 20 ms, and there can be a maximum of three multimedia clients. The benefit of this policy is that WNICs on general clients need only be in high-power mode during general slots. A multimedia client also must

keep its WNIC in high-power mode during general slots in case TCP data is sent, in addition to the time it is receiving its multimedia burst. This consumes more energy than buffering TCP data in the packet queue, as the WNIC would then be in low-power state during general slots; Section 5 discusses this further.

The schedule guarantees that the proxy will begin streaming multimedia data to a multimedia client when the client's RP arrives (the thick lines in Figure 3.1). However, in the event that a particular multimedia burst is too large to fit in the multimedia slot (possible because the bit rate is variable), the proxy may continue transmitting the multimedia burst up to the next multimedia RP. This delays the transmission of background data, possibly preventing any from being transmitted during that general slot. This reduces the chance of multimedia jitter but can increase background data latency. Note that if the multimedia burst is finished before the end of the multimedia slot, the schedule may *not* send background data before the general RP arrives; this would cause most clients to miss these packets (their WNICs would be in *sleep* mode). This means that bandwidth is wasted due to the proxy sitting idle after each multimedia burst until the next general RP. We are currently investigating allowing the size of slots to adapt dynamically to the amount of data being transmitted (see Section 5).

3.2.2 Implementation

We use a multithreaded producer-consumer model for the low-level proxy implementation. The code is divided into two threads: input (the producer) and output (the consumer). The two threads run in a round-robin fashion, using semaphores for passing control.

The output thread is responsible for transmitting all buffered data. When a multimedia RP arrives, it transmits all of that multimedia client's data from that client's packet queue, marking the last packet in the type-of-service field in the IP

header. This indicates that the burst is finished and the client may transition its WNIC to low-power mode. The output thread then allows the input thread to run until the next general RP. When that arrives, the output thread reacquires control from the input thread, transmits all packets buffered in the general queue [1], then gives temporary control to the input thread. If at any time before the end of the general slot more background data arrives, the input thread allows the the output thread to reacquire control and send this data, at which point the output thread again releases control. At the end of the general slot, the output thread reacquires control and begins transmitting the data for the next multimedia client. Note that if the entire general slot is spent sending background data, the input thread is not allowed to run, and some background data may remain unsent until the next general RP.

The input thread runs whenever the output thread has idle time between bursts. It simply listens for incoming data on all open sockets, placing the data into the appropriate queue for later transmission by the output thread. It also parses new connections to determine if they are multimedia or general requests. Because the input thread is the producer, it cannot be starved by the output thread due to high-bandwidth traffic: the output thread may at times have so much data to send that it does not release control, but will eventually consume all the available data and allow the input thread to run.

In order to guarantee that the input thread releases control before the next RP, the output thread sets a timer and then blocks using its semaphore. The input thread checks the timer after each small section of code so that when the output thread

[1]Before sending each packet, the output thread verifies that the send will not block, to prevent the proxy from missing hard rendezvous points. If the send would block, the output thread gives control to the input thread for a short time (less than the time remaining in the general slot), then checks again to see if a send would block. It repeats this process until it is able to send the remaining data in the queue or until the general slot ends.

needs to run, the input thread will release control. The output thread sets the timer one millisecond earlier to ensure that it will be running before the RP; it then spins calling `gettimeofday()` until the next RP arrives. This solution allows the output thread to transmit bursts on time with better than millisecond accuracy. Previous work [2] has used the `nanosleep()` system call to sleep until the RP; however, `nanosleep()` is reported to sleep at least 10 ms longer than requested, which causes the proxy to send its bursts late at each RP. Our solution therefore results in better power savings than the call to `nanosleep()`.

3.3 Delay Compensation Algorithms

A client's WNIC must be transitioned to high-power mode to receive bursts of data. If this is done early, some energy will be wasted while waiting for the burst to arrive; if it is done late, packets will be missed. This would not be an issue if every burst arrived according to the bursting schedule. However, while the proxy usually adheres to the bursting schedule with better than millisecond accuracy, delay may at times be observed by the client. Because data is routed through at least one network node (the wireless access point), the exact travel delay for each packet cannot be known with certainty. Furthermore, the receipt of a packet at the proxy causes an interrupt which is immediately handled by the operating system; large spikes in incoming data can starve the proxy application for a second or more. For these reasons, we use a *delay compensation algorithm* on the client to determine when to transition the WNIC out of low-power mode.

We divide these algorithms into two broad classes: *non-adaptive* and *adaptive* (see Figure 2). Non-adaptive algorithms leave the WNIC in high-power mode long enough to estimate an initial transition point; they then set each transition point one burst interval after the *previous transition point*. They may use the arrival time

of the first burst as the original transition point, or may shift the transition times earlier by any fixed amount less than one burst interval. Figure 2 shows a non-adaptive algorithm that has shifted the transition point more than half of a burst interval early. It does not adjust to transitioning the WNIC early nor to missing packets.

It is our experience that non-adaptive algorithms often produce poor energy savings and large packet loss. If the algorithm makes a poor initial estimate, it will either often spend too much time waiting for a burst or may often miss packets from a burst. For this reason, we examined the benefits of using an adaptive delay compensation algorithm.

The intuition behind adaptive algorithms is as follows. If a burst arrives earlier or later than expected, it is most likely a change in access point delay between the proxy and the client. Most likely, several subsequent bursts will arrive according to the same schedule. Adaptive algorithms therefore set each transition point a fixed amount after the *arrival time of the previous burst*, so that they may immediately adjust to these changes. If the algorithms transitions in the middle of a burst, it will estimate the arrival time of the first packet based on the average amount of time between packets in the burst[2]. In order to reduce missed packets, the amount is slightly less than a burst interval; we refer to the difference as the *early transition amount* (see Section 4.2). Figure 3.2 shows both classes of algorithms pictorially.

[2]Adaptive delay compensation algorithms rely on accurate knowledge of packet arrival times in order to make an accurate estimate about the next burst arrival. On a slow client machine, several packets could queue in the operating system buffers before being read by the delay compensation software, resulting in several packets appearing to arrive simultaneously. This would have the greatest impact on calculating a new estimate after missing packets, as the remaining packets in the burst do not give any information about the average time between packets. A slow client machine such as a PDA might have to make coarser adjustments in the event that it misses packets, such as estimating an entire multimedia slot early for the following burst, rather than computing a more precise estimate.

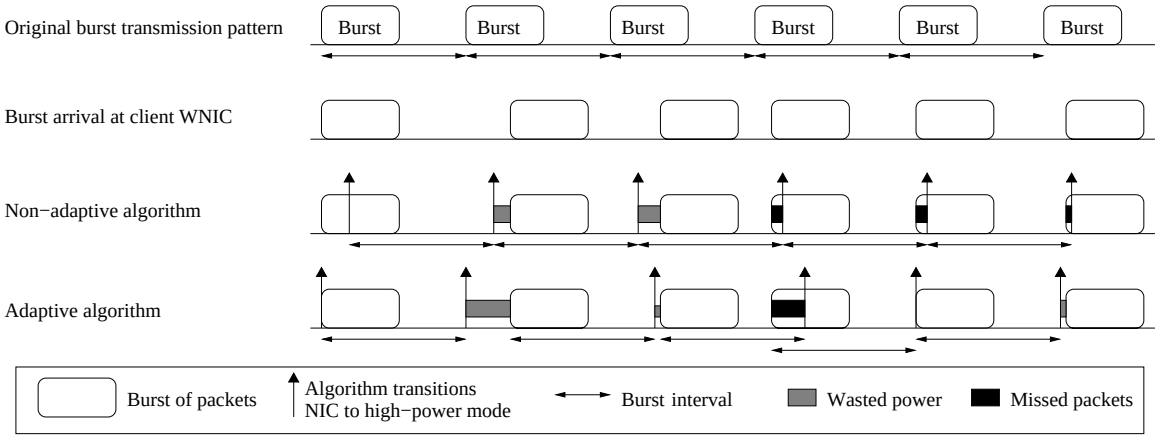

Figure 3.2: Delay compensation algorithms compensating for irregular burst arrival times due to network delays. Non-adaptive algorithms choose a fixed transition point, whereas adaptive algorithms dynamically adapt to network delays.

As the delay compensation algorithms discussed rely on the fact that the client is receiving a burst of video data, a slight modification to the algorithms are necessary when run by general clients. For our experiments, the general clients simply listened for any multimedia client's burst and used that to choose an original transition point and, for adaptive algorithms, to make regular adjustments as necessary. However, in actual use, it could be the case that a general client was the only client using the proxy. The proxy could instead mark the initial packet of background data bursts in the same way as it marks the final packet of multimedia bursts (discussed in Section 3.2.2), and general clients could use this information to align themselves to background data burst arrivals. We feel that the performance of that algorithm would be similar enough to the current solution that it is not necessary to implement it for our experiments.

3.4 Drought

A multimedia client relies on the marked packet at the end of a multimedia burst to know when to transition its WNIC to low-power mode. If that packet is not present, the client keeps its WNIC in high-power mode for an entire burst interval until the next marked packet for that client arrives. This occurs if there is no data in the client's packet queue when its multimedia RP arrives; we refer to this as a *drought*.

There are two situations in which a multimedia client may experience several droughts during a session. First, low-bitrate clients with small burst intervals will experience regular droughts throughout the session. This situation occurs because there are very few packets per second in low-bitrate streams, so very short burst intervals such as 50 ms result in a single packet received per *several* burst intervals. The second situation occurs when the bandwidth use of all clients approaches the maximum for the wireless medium. A client experiencing heavy packet loss due to collisions notifies the server, which seems to respond by changing the client's stream bandwidth. However, the results are unexpected: the client experiences several periods during which it receives no data (which is an extended drought), followed by short periods of full- to near-full bandwidth data reception. The reason for these extended droughts, which can vary from one to tens of seconds, is unknown.

A simple but effective solution to the drought problem that prevents the client from keeping its WNIC in high-power mode for an entire interval is to send an empty marked packet to the client in the event of a drought, indicating that no data will be arriving during this multimedia slot (this has very small overhead.) In an actual implementation, a daemon controlling the WNIC would intercept this packet and transition the WNIC to *sleep* mode, then discard the packet; however, we did not implement a daemon for our experiments[3]. Because sending an empty packet

[3]We chose not to implement a daemon in order to keep the experiments simple; however, this daemon would consume a small additional amount of energy. In actual implementation,

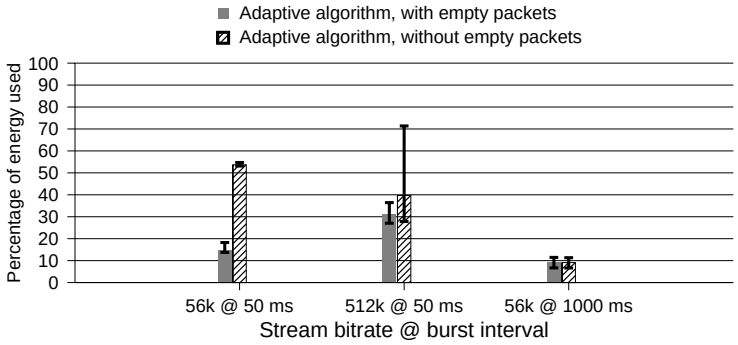

Figure 3.3: Comparison of energy usage for an adaptive delay compensation algorithm in ten client experiments before and after inserting empty packets into the trace. The bar represents the average energy usage for the ten clients; the superimposed line represents the minimum and maximum energy usage.

without a daemon to intercept it caused RealOne to crash, we instead explicitly inserted these packets post-mortem before simulation.

Note that we have designed our proxy so that non-participating clients (those who are unaware of the power-saving capabilities available) can still use the wireless access point to communicate. If the proxy sent empty packets to every multimedia client, these naive clients would crash. Therefore, in an actual implementation, the daemon on the client would need to notify the proxy of its existence before the proxy could send empty packets to the client.

The comparison of energy usage for an adaptive delay compensation algorithm before and after empty packet insertion is shown in Figure 3.3. Each pair of bars represents an experiment with 10 clients; the first is an example of a low bitrate stream and a small interval (50 ms), and the second experiment is an example where

this daemon could also act as a "deburster" for the client software, transforming the bursts of data back into a stream before forwarding it to the client software. This would allow the software to have a small buffer as usual, removing a step in configuring a client computer to use our energy-saving software.

the maximum bandwidth for the network (about 4 Mbps) was reached. The effects of drought can be significant, costing significant energy across all clients (in the low bitrate case) or unfairly affecting a few clients (in the high bitrate case, shown by the large error bars). The third pair of bars shows that drought does *not* occur with a low bitrate stream and a large interval, because at least one packet is sent per burst interval.

3.5 Simulator

As mentioned earlier, a packet sniffer on a monitoring station generates a trace of all incoming and outgoing packets on the wireless side of the access point. The simulator reads this trace post-mortem, calculating how much time a client's WNIC has spent in high- and low-power mode and how many bytes its WNIC has transmitted or received. The simulator then reports packet loss for each client, as well as how much energy the client would use by transitioning its WNIC between modes according to a given delay compensation algorithm. This is compared to the naive client, which keeps its WNIC in high-power mode.

For calculating how much energy a client uses, we assume that the energy and time required to transition the WNIC from *sleep* to *idle* mode and vice versa is 0. We also assume that the power cost to have the WNIC in *sleep* mode is 0. The basic energy usage formula is therefore

$$T * P_i + B * P_b,$$

where T is the number of seconds spent with the WNIC in a high-power mode (including *idle*, *receive*, and *transmit*); P_i is the energy cost per second spent in *idle* mode; B is the total number of bytes transmitted or received; and P_b is the energy cost per byte. P_b is determined by subtracting the energy cost for idling from the energy cost for receiving, then dividing by the average throughput of the

wireless network (which separate experiments show to be approximately 4 Mbps). For example, a 2.4GHz WaveLan DSSS card uses 1319 mJ/s when idle and 1425 mJ/s when receiving [19, 11]. Receiving 4Mb would take at least one second, using 106 mJ above idle costs; so receiving one byte would use 212 picojoules above idle costs.

Chapter 4

Performance

This chapter describes our experiments and presents our results. Section 4.1 describes the experimental setup; Section 4.2 shows the results of experiments using a single multimedia client. Section 4.3 examines experiments using ten multimedia clients, and Section 4.4 adds background traffic.

4.1 Experimental Setup

See Figure 4.1 for a graphic representation of the experimental setup. In each experiment, multimedia clients requested a video from the proxy in Real format. The basic results of the work with Real should in general apply to Quicktime and MS Media; while there are some differences, they do not affect the general principles of saving energy nor end-to-end latency. This was the case in our prior work [2].

Each of our experiments compared clients using our algorithms to a naive client that leaves its wireless network interface card (WNIC) in high-power mode for the duration of the video. HTTP requests comprised all of the background traffic. The monitoring station, mobile web client, and mobile multimedia clients were various laptops using 11Mbps Orinoco PCMCIA WNICs. The monitoring station ran tcpdump (version 3.6.3, using libpcap 0.6) to capture data about each packet on the wireless network. The mobile web client made web requests to an Apache web server on a Dell dual processor (Xeon 933MHz) server with 1.5 GB of memory, running FreeBSD 4.5-STABLE. We ran RealServer 8.01 on a Dell 330 equipped with a

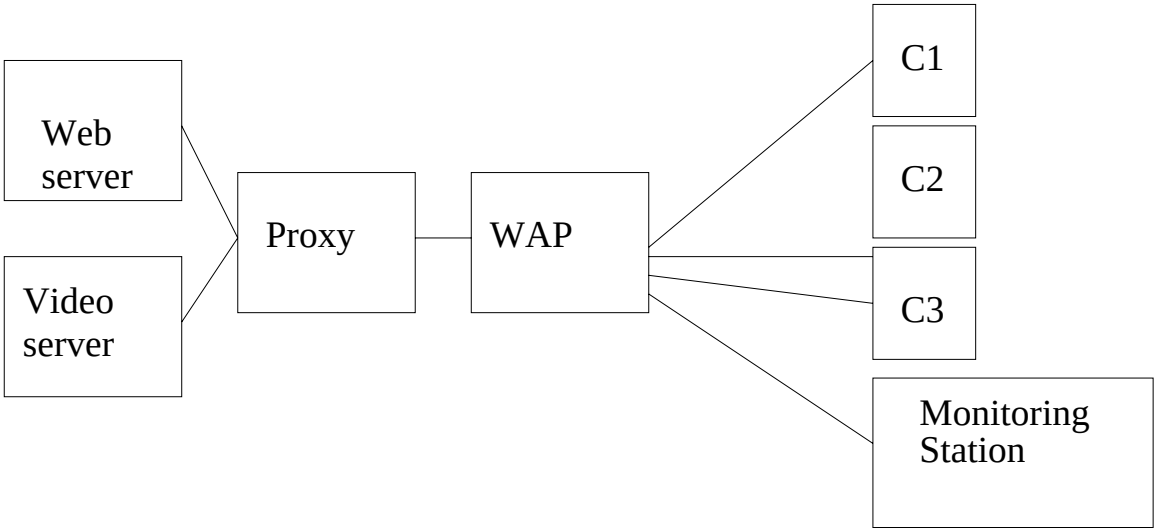

Figure 4.1: Experimental setup.

1.5GHz Pentium 4 processor with 512 MB RDRAM, running Microsoft Windows 2000 Server SP2. The multimedia server, web server, and access point were connected over 100Mbps Fast Ethernet. Each client laptop ran RealOne 2.0; six ran Windows 2000 Professional, three ran Windows 98, and one ran Windows ME. The multimedia clients requested a two minute trailer for the movie *The Wall*, encoded by Adobe Premier 6.0 at 56kbps, 128kbps, 256kbps, or 512kbps. Because the encoder could not perfectly match the requested bitrates, the effective bitrates of these streams are 34kbps, 80kbps, 225kbps, and 450kbps respectively. As these streams are representative of multimedia streams that might be encountered in actual use, we will refer to the streams by their nominal rather than effective bitrates.

4.2 Single Multimedia Client

As a baseline, we first ran experiments to determine how well our proxy performs when serving a single multimedia client, as was done in [2]. For each bitrate we

performed an experiment using burst intervals of 50, 100, 200, 500, and 1000ms. The multimedia slot size was equal to the burst interval for each experiment, and the general slot size was zero, which transmits one packet per general slot. We then ran three variations of adaptive algorithms and five variations of non-adaptive algorithms on the trace. For each experiment, we report packet loss as well as percentage of energy used by each algorithm compared to the energy used by a naive client. An energy usage of 100% for an algorithm indicates that no energy was saved.

The three adaptive algorithms had different *early transition amounts* (as described in Section 3.3): 10 ms, 20 ms, and $.01B + 5$ ms, where B is the burst interval for the experiment. The third algorithm was developed empirically; the intuition is that for small burst intervals, the 5 ms will be the dominant factor, while for large burst intervals, the 1% of the burst interval will be the dominant factor. We found the third algorithm to be the most effective in all cases we tested. Hence, we present only the $.01B + 5$ ms algorithm in the rest of the paper and refer to it simply as the *adaptive algorithm*.

The five non-adaptive algorithms have shifts from -40% to 0% of a burst interval in 10% increments and represent the best possible non-adaptive algorithms (shifts from -50% to -90% always performed poorly). We refer to these five algorithms as the *0% through 40% non-adaptive algorithms*.

The results of the experiments are in Figure 4.2. We present the energy usage for three algorithms: the adaptive algorithm and the best and worst non-adaptive algorithms. There is in general no heuristic to determine *a priori* which non-adaptive algorithm will perform the best; arbitrarily choosing a non-adaptive algorithm falls within this range.

The figure shows that the adaptive algorithm performs nearly as well as the best non-adaptive algorithm in all experiments. While it usually uses 1-5% more energy than the best non-adaptive algorithm, recall that the *best* non-adaptive algorithm

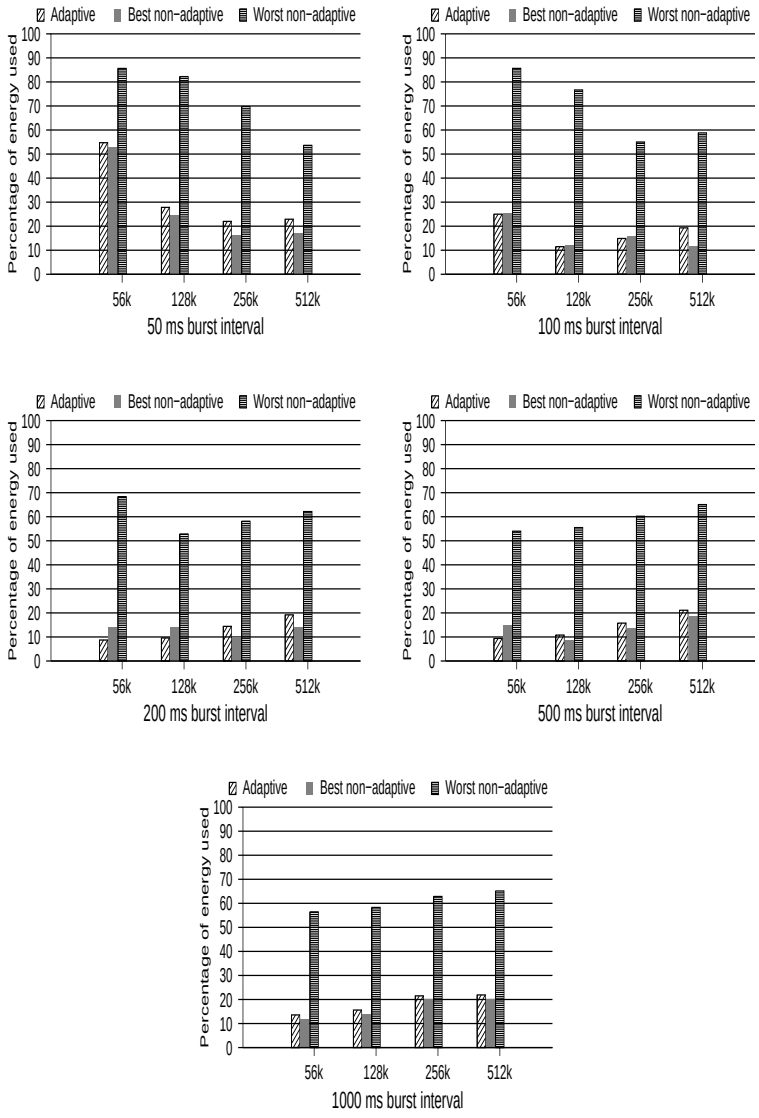

Figure 4.2: Energy used in single-client experiments. The 56kbps 50 ms and 100 ms experiments and the 128kbps 50 ms experiment falsely report a high level of energy usage; see the text of Section 4.2.

to use on a client in general *cannot* be known except by post-mortem analysis. The exception is in the low bitrate streams and small burst interval experiments (e.g., 50 ms); this is because our simulation requires at least two clients to execute our drought-handling algorithm correctly. The figure therefore reports falsely high energy usage for low-bitrate clients with short burst intervals. This is an artifact of our simulation and would not occur in actual use. Packet loss for the adaptive algorithm was never greater than 1.2% and averaged 0.2%; for the non-adaptive algorithm, it was as large as 33.6% and averaged 3.8%.

The optimal energy usage for a given stream can be determined by dividing the effective bandwidth of the stream by the effective bandwidth of the network. For example, the 512kbps stream had an effective bandwidth of 450kbps, so the optimal energy usage would be approximately 450kbps/4Mbps, or 11% of the energy used by a naive client. There are several reasons that an algorithm produces suboptimal energy usage. One is that RealOne clients must set up their streams before transitioning their WNICs to low-power mode, alternately transmitting and receiving about twenty RTSP packets via TCP to do so. In single-client experiments with large burst intervals, this handshake can represent a significant amount of the time spent with the WNIC in high-power mode. For example, the best non-adaptive algorithm for the 512kbps stream in the 500 ms experiment used 18.6% of the energy that the naive client used; however, 5.8 of the 17.3 seconds spent with the WNIC in high-power mode were spent executing the handshake. Excluding this, the client only used 12.6% of the energy that the naive client used, which is much closer to the optimum energy usage. For the 56kbps client in the 500 ms experiment, the best non-adaptive algorithm used 12% of the energy, but only 1.3% excluding the handshake, which is much closer to the optimal energy usage of .85%. As the movie length increases, this handshake energy cost is amortized.

Another cause of suboptimal energy usage is when an algorithm transitions the WNIC to high-power mode too early, wasting energy until the burst arrives. This is often the case with non-adaptive algorithms, which can potentially spend nearly the entire burst interval with the WNIC in high-power mode. Occasionally, it also occurs with adaptive algorithms. For example, with a 50 ms burst interval, the adaptive algorithm (which transitions early by 1% of the burst interval + 5 ms) transitions the WNIC to high-power mode 5.5 ms early 20 times per second, or 110 ms per second (an 11% energy usage). With a 1000 ms burst interval, however, it transitions the WNIC 15 ms early once per second (only a 1.5% energy usage). Note, however, that the experiments with larger burst intervals do not necessarily report better energy savings, because they have longer handshake times.

Previous work [2] implemented a scheduling policy that was able to support a single multimedia client successfully. We have improved upon this scheduling policy in several ways and ran single-client experiments similar to those in [2] for comparison[1]. We present power usage as the amount of power required to receive the video data (in mJ/kB): [2] used this metric to normalize against (incorrect) stream fidelity lowering caused by their schedule. Our schedule never caused incorrect adaptation, but we list our results in millijoules per kilobyte to allow comparison. We cannot compare packet loss, as it is not reported in [2]; the percentage of packets missed by our adaptive algorithm was never greater than 1.2% and averaged .2%.

We find that our schedule and adaptive algorithm usually used between one fourth and one third of the power reported in [2][2]. For example, our 128kbps experiment with a 100 ms burst interval used 11 mJ/kB, compared to 32 mJ/kB in [2]; our 56kbps experiment with a 200 ms burst interval used 20 mJ/kB, compared to

[1]The previous work used the same video clips and burst intervals, with a general slot size of 0.

[2]The full range of power usage, excluding the three experiments which failed to insert empty marked packets, was between 20% and 41% of the power usage in [2].

62 mJ/kB; our 512kbps experiment with a 50 ms burst interval used 4 mJ/kB, compared to 18 mJ/kB.

The higher energy cost to download data using the scheduling policy in [2] is partially due to the use of the `nanosleep()` system call, as discussed in Section 3.2.2. `nanosleep()` was used in the proxy to block the output thread between bursts; however, [2] reported that `nanosleep()` tends to return at least 10 milliseconds later than requested, so the proxy was late in sending the burst to the client, who in turn awoke unnecessarily early. Our experiments show that our solution (using semaphores and a timer) works well, eliminating both the wasted power and the missed packets. Our proxy also responded correctly to droughts, reducing wasted power for low bitrate streams, while the proxy in [2] may not. These problems combined with other choices in the design of the scheduling policy made the policy in [2] unsuitable for supporting multiple clients, so we do not compare our multiple-client experiments with this scheduling policy.

4.3 Multiple Multimedia Clients

The next set of experiments examined how our scheduling policy supported multiple multimedia clients. We ran ten multimedia streams per experiment, with all clients requesting the same bitrate. The streams were unicast to the clients, so that the total bandwidth used was roughly equal to ten times the bitrate of the stream. We ran an experiment for each combination of burst interval and bitrate as in Section 4.2. The requests were spaced roughly one second apart in order to spread the multimedia traffic; transmitting identical multimedia streams simultaneously could cause large spikes of activity during high bitrate sections of the video.

Figure 4.3 shows the average, minimum, and maximum energy usage for each experiment (represented by the bar and error lines, respectively). If the proxy cor-

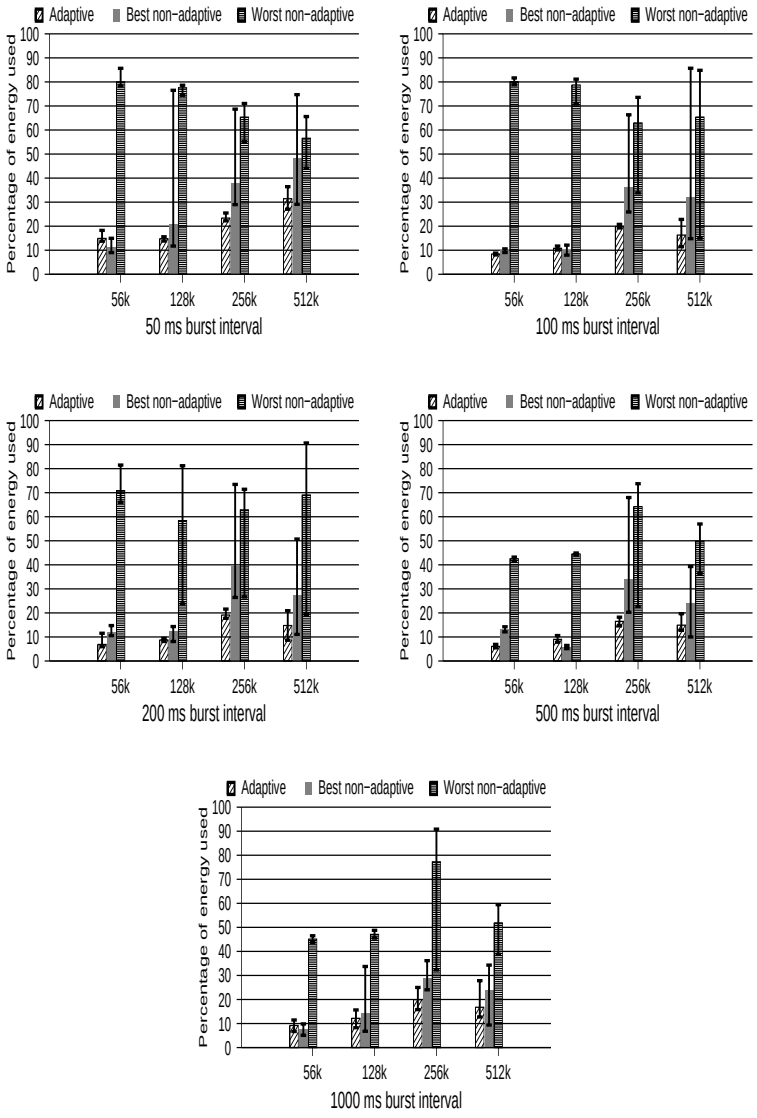

Figure 4.3: Ten multimedia clients with no background traffic. Average, minimum, and maximum energy usage for the ten clients are represented by the bar and error lines respectively.

rectly sends bursts according to the schedule, then each client should be able to receive its video independent of the rest of the traffic on the network. We found that the proxy can in fact schedule all clients successfully. Excluding experiments in which the aggregate transmitted bandwidth exceeded the maximum for the wireless network, energy savings versus single-client experiments was unaffected by having multiple clients share the network[3]. For example, in the 256kbps experiment using a 50 ms burst interval, the average energy usage by the adaptive algorithm was 23.4%, which is similar to the single-client usage of 22%. As in the single-client case, the non-adaptive algorithms are inferior. Choosing the best non-adaptive algorithm is more difficult, because each client itself may need to shift a different amount; this results in large variation between the minimum and maximum energy used per client. Packet loss for the adaptive algorithm averaged 1.7%, with a maximum of 6.2%; the loss for the best non-adaptive algorithm averaged 1.5%, but the maximum was 31.1%. Handshake time was shorter than in the single-client experiments because ten background packets could be sent per burst interval.

The 512kbps experiments exceeded the maximum effective bandwidth for the wireless medium. As a result, some clients requested adaptation from the server to a lower stream quality. As discussed in Section 3.4, these clients then received unpredictable amounts of data at unpredictable times and experienced large droughts. However, our algorithm handles drought effectively (see Figure 3.3).

Figure 4.3 shows that the adaptive algorithm in 512kbps experiments usually has a larger difference between the maximum and minimum energy usage than in experiments using other bitrates. This is not due to drought; rather, it is a direct

[3]As the naive client in the ten-client experiments receives ten times the data as in the single-client experiments, the same energy usage in Joules by an algorithm will result in a lower *percentage* of energy used compared to the naive client. This difference is small, as time with the WNIC in high-power mode rather than the amount of data received is the dominating factor in energy usage.

result of packets being dropped by the wireless network. If the marked packet at the end of a burst is dropped, the client will keep its WNIC in high-power mode until the next burst arrives. This accounted for between 1/3 and 2/3 of the total energy consumed in the worst case. We are currently investigating ways to detect the dropping of marked packets.

4.4 Adding Background Traffic

We next examined our scheduling policy when handling multimedia and general clients simultaneously. There were three patterns that we used to represent varying amounts of background traffic, labeled *light*, *medium*, and *heavy*; they had average bitrates of 220kbps, 1.4Mbps, and 2.8Mbps respectively. Light traffic was generated by requesting small text files in rapid succession. Medium traffic was generated by requesting 1-2 megabyte files in succession, with little delay between downloads. Heavy traffic was generated by requesting 20 megabyte files in succession, with short delays between downloads. The HTTP requests were generated by a script to ensure that the pattern remained identical throughout the experiments.

As discussed in Section 3.2.1, the static nature of the bursting schedule results in suboptimal use of available bandwidth on the wireless medium. If the total bandwidth of the general slots in the schedule is less than the bandwidth of the background data arriving at the proxy, the proxy will be forced to buffer the data until it can be sent. There may be unused time after the end of each multimedia client's burst, but the schedule cannot use this time to transmit background data. A dynamic schedule could alleviate this problem. To study the effectiveness a static schedule, we perform experiments for each traffic pattern using three different values for *general slot weight*, which is the percentage of the schedule devoted to transmitting background data. For example, the schedule in Figure 3.1 has a general slot weight of

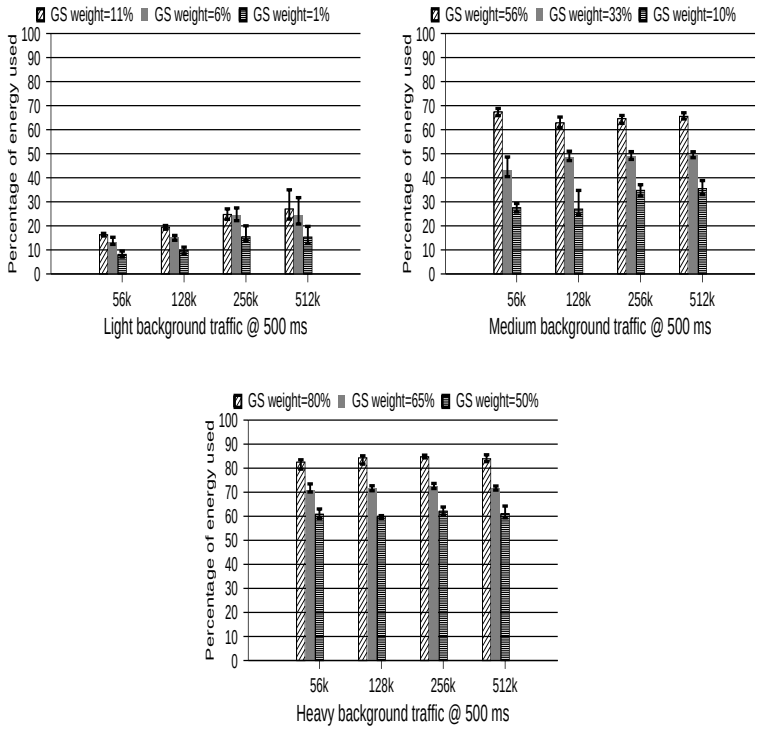

Figure 4.4: Average, minimum, and maximum energy usage for ten multimedia clients with background traffic on the network. The general slot weight is the time devoted to the general slot in the schedule.

20% (60 of the 300 ms in the bursting schedule are reserved for background data). For each traffic pattern, we ran experiments where the general slot weight was larger than, smaller than, and equal to the size necessary to support the background traffic. These general slot weights were calculated using our observation that the effective bandwidth of our wireless network is approximately 4Mbps.

Figure 4.4 shows results for the ten 56kbps video clients in the experiment, using the adaptive algorithm on a 500 ms burst interval. Packet loss for the adaptive algorithm averaged 1.5% with a maximum of 6.7%; non-adaptive algorithms averaged 0.7% with a maximum of 38.6%. Fewer packets were missed in medium and heavy

traffic experiments than in light ones because the client's WNIC was in high-power mode for a greater percentage of time due to larger general slot weights.

Each client is required to keep its WNIC in high-power mode for the duration of the general slot in case there is data to be received. This results in the general slot size being the dominant factor in energy usage for most experiments. It is not possible to simply minimize the general slot size, as background data will not be able to travel through the proxy: Figure 4.5 shows that as the general slot weight decreases, the end-to-end latency of background traffic increases. When the general slot weight is too small to support the amount of background traffic on the network, the latency quickly escalates.

If enough packets arrive at the proxy at once (which is often the case when a large file is requested via TCP), the burst of interrupts can starve the proxy application for a second or more. This results in a small increase in energy usage and cannot be resolved simply by inserting empty marked packets into the stream. A real-time operating system or a distributed proxy might make this occurrence less likely.

Figure 4.5 shows energy usage and average end-to-end latency for the general clients in each experiment. In general, smaller general slots use less energy but cause higher end-to-end latency. As seen from the figure, the light and medium traffic patterns can be handled by setting the general slot size large enough. However, for a heavy bandwidth background traffic pattern (with an average bandwidth of 2.8Mbps), it is unclear if the data is supportable at any general slot size. As the heavy bandwidth experiments had frequent proxy starvation due to large amounts of data arriving at once at the proxy, the proxy had difficulty following the bursting schedule, causing less predictable results.

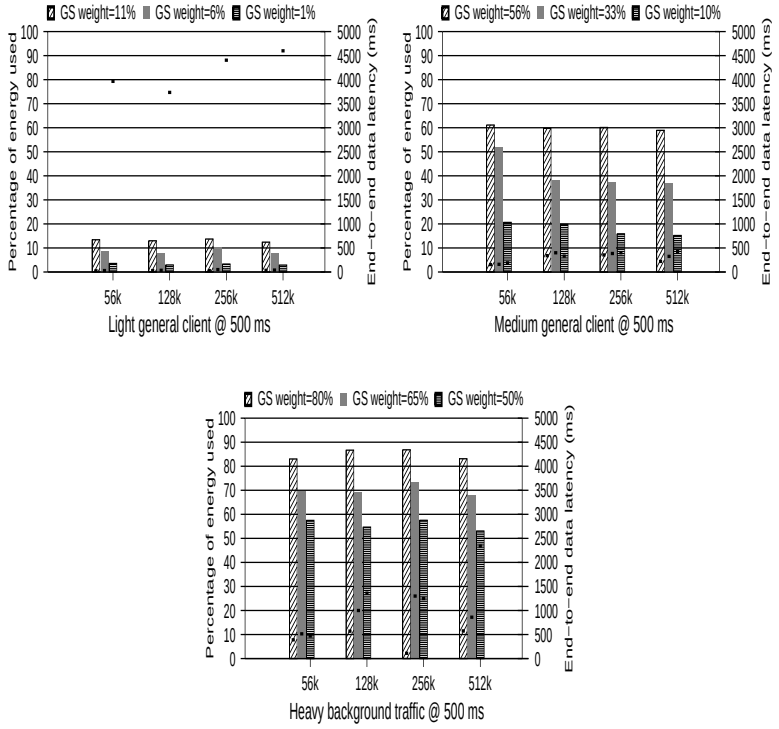

Figure 4.5: Energy usage (bars) and transmission latency (dots) for general clients, using the adaptive algorithm in 500 ms burst interval experiments. Dots correspond to the right-side axis.

Chapter 5

Summary and Future Work

This paper has described a novel scheme to save energy for several clients, both multimedia and general. Whereas streaming multimedia to mobile clients normally requires the WNIC on a client to remain in high-power mode at all times, our scheme schedules transmissions to clients so that they may transition their wireless network interface cards (WNICs) into low-power mode between bursts, thereby saving energy.

We found that our scheduling algorithm is effective. First, it supports multiple clients, matching per-client energy savings seen in a single-client scenario. Second, it effectively handles delays and droughts that can occur quite often. Third, it minimizes the time the multimedia client spends in an expensive *idle* mode. Finally, it supports general clients; end-to-end latency is reasonable if the wireless network bandwidth is not exceeded.

Specifically, our results show that our scheduling algorithm uses as little as 8.7% of energy compared to a naive client on a 56kbps stream. It averages as little as 6.9% energy usage for ten 56kbps streams. With an additional 220kbps of background data being transmitted, ten 56 kbps streams use an average of 13.1% and general clients use an average of 8.4% with an average of 32 ms end-to-end latency for their data.

Our system is highly effective at saving energy in many cases. We are currently relaxing the fixed nature of our scheduling policy and moving to a dynamic policy; this should provide improved energy savings under changing network conditions. One way to do this is to broadcast a new schedule on a regular basis. This avoids

setting the multimedia slot size large enough for the highest-fidelity client, which wastes bandwidth in slots for low-fidelity clients. It also allows adaptation to the changing bandwidth requirements of general clients.

In the current implementation of the proxy, a multimedia client's TCP connection to the proxy is buffered in the general queue along with general traffic, as this data is sent more frequently and therefore has lower latency. This causes the multimedia client's WNIC to have to transition out of low-power mode during all general slots, in case there is TCP communication from the proxy (forwarded from the server.) Moving the multimedia client's TCP packets into its multimedia slot would use less power but introduce greater latency into the client's TCP communication; the two schemes have not yet been compared to determine how greatly power and latency are affected. It may be the case that the greater latency is acceptable, as the TCP connection is mostly used for setup of the stream and then remains unused.

The adaptive algorithm can be slightly improved by detecting that a marked packet is dropped, by seeing the proxy transmit general data without first sending a marked packet. This can avoid leaving the WNIC in high-power mode for an entire burst interval.

Bibliography

[1] M. Baker, S. Asami, E. Deprit, J. Ousterhout, and M. Seltzer. Non-volatile memory for fast, reliable file systems. In *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating System (ASPLOS)*, 1992.

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

[3] Surendar Chandra. Wireless network interface energy consumption implications of popular streaming formats. In *Multimedia Computing and Networking (MMCN '02)*, Jan 2002.

[4] C. F. Chiasserini and R. R. Rao. Pulsed battery discharge in communication devices. In *Proc. 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM '99)*, 1999.

[5] A. Datta, A. Celik, J. G. Kim, D. E. VanderMeer, and V. Kumar. Adaptive broadcast protocols to support power conservant retrieval by mobile users. In *ICDE*, pages 124–133, 1997.

[6] V. Delaluz, A. Sivasubramanian, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin. Scheduler-based DRAM energy management. In *Proc. Design Automation Conf. (DAC '02)*, Jun 2002.

[7] F. Douglis, P. Krishnan, and B. Bershad. Adaptive disk spin-down policies for mobile computers. In *Proc. 2nd USENIX Symp. on Mobile and Location-Independent Computing*, 1995.

[8] Carla S. Ellis. The case for higher-level power management. In *Proc. 7th Workshop on Hot Topics in Operating Systems*, Mar 1999.

[9] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *Symposium on Operating Systems Principles*, pages 48–63, 1999.

[10] K. Govil, E. Chan, and H. Wasserman. Comparing algorithm for dynamic speed-setting of a low-power CPU. In *Mobile Computing and Networking*, pages 13–25, 1995.

[11] Paul J. M. Havinga. *Mobile Multimedia Systems*. PhD thesis, Univ. of Twente, Feb 2000.

[12] D. P. Helmbold, D. D. E. Long, and B. Sherrod. A dynamic disk spin-down technique for mobile computing. In *Mobile Computing and Networking*, pages 130–142, 1996.

[13] R. Kravets and P. Krishnan. Power management techniques for mobile communication. In *Proceedings of the 4th International Conference on Multimedia Computing and Networking (MOBICOM '98)*, pages 157–168, Oct 1998.

[14] R. Kravets, K. Schwan, and K. Calvert. Power-aware communication for mobile computers. In *Proc. 6th International Workshop on Mobile Multimedia Communications*, Nov 1999.

[15] A. R. Lebeck, X. Fan, H. Zeng, and C. S. Ellis. Power aware page allocation. In *Architectural Support for Programming Languages and Operating Systems*, pages 105–116, 2000.

[16] K. Li, R. Kumpf, P. Horton, and T. E. Anderson. A quantitative analysis of disk drive power management in portable computers. In *USENIX Winter*, pages 279–291, 1994.

[17] Cirrus Logic. Whitecap2 wireless network protocol — white paper. Technical Report, 2001.

[18] S. Singh, M. Woo, and C. S. Raghavendra. Power-aware routing in mobile ad hoc networks. In *Mobile Computing and Networking*, pages 181–190, 1998.

[19] M. Stemm, P. Gauthier, D. Harada, and R. H. Katz. Reducing power consumption of network interfaces in hand-held devices. In *Proc. 3rd International Workshop on Mobile Multimedia Communications*, September 1996.

[20] A. Vahdat, A. Lebeck, and C. Ellis. Every joule is precious: The case for revisiting operating system design for energy efficiency. In *Proc. 9th ACM SIGOPS European Workshop*, 2000.

[21] M. Weiser, B. Welch, A. J. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *Operating Systems Design and Implementation (OSDI '94)*, pages 13–23, 1994.

[22] John Wilkes. Predictive power consumption. Technical Report HPL-CSP-92-5, Hewlett-Packard Labs, Feb 1992.

[23] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. ECOSystem: Managing energy as a first class operating system resource. In *ASPLOS X*, October 2002.