

Energy-Aware Traffic Shaping for Wireless Real-Time Applications

Christian Poellabauer and Karsten Schwan
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
{chris, schwan}@cc.gatech.edu

Abstract

Sleep modes of wireless network cards are used to switch these cards into low-power state when idle, but large timeout periods and frequent wake-ups can reduce the utility of this approach. Modern processors offer the ability to switch CPU voltages or clock frequencies and therefore reduce CPU energy consumption, however, that can reduce the sleep durations of a network device, adversely affecting the achievable energy savings. This paper describes an approach in which multiple resource managers cooperate to reduce a mobile device's energy consumption. This system-level approach is based on the integrated management of a real-time CPU scheduler, the frequency scaling capabilities of a modern processor, a QoS packet scheduler, and the low-power sleep mode of a wireless network card.

1 Introduction

A necessity for the acceptance of wireless devices and applications is an acceptable lifetime of their finite energy sources. Recent research has focused on the key energy-consuming resources of a mobile device, including its display, network card, memory, and processor. The voltage and frequency scaling capabilities of mobile processors are exploited in [12, 4]. Other approaches utilize the runtime adaptivity of applications and flexibility in acceptable user-perceived qualities, as in video and audio streaming [14, 1, 2]. At the network level, previous work has investigated energy conserving mechanisms at different layers of the protocol stack, including the medium access control layer [7] or transport protocols [8].

Our work addresses system-level issues in energy management, by considering the joint manner in which multiple resources contribute to the total energy consumption of a mobile device. The specific resources and their interactions studied in this paper are two of the key contributors to total energy consumption: the CPU and the wireless net-

work card. The CPU-level energy management technique we use is frequency scaling. For the wireless card, we use its ability to operate in different states with different energy requirements, distinguishing between *active* state (transmit, receive, idle), *doze* (or sleep) state, and *off* state. In the doze state, a node consumes significantly less energy than in active state. For instance, the popular Orinoco wireless card requires 285mA in transmit mode, 185mA in receive mode and only 9mA in doze mode. Note, however, that a constraint on mode switches is defined by the fact that each switch has associated delays and energy overheads. For the Orinoco card, a switch from doze to idle mode requires approximately 250 μ s during which already idle energy is consumed. After another 250 μ s, the network device is ready to send [5].

Previous work has successfully exploited network card modes with *timeout-based* energy management approaches, where after a certain period of inactivity at the network interface, the card is placed into doze mode. Since a node whose card is in doze mode cannot transmit or receive data, the 802.11b standard defines power savings mechanisms to ensure successful communications. The DCF (Distributed Coordination Function) protocol requires that each node must wake up at the beginning of a *beacon* interval (typically 100ms), and must stay awake for a fixed period of time. During that time, the access points send a beacon signal containing a *traffic indication message (TIM)* indicating whether an access point has data buffered for this device. If such data exists, it is transmitted.

Our goal is to increase the time a network card is kept in doze mode, but within the constraints defined by the DCF protocol. In addition, we wish to minimize the number of switches between doze and active modes. This decreases a card's total energy consumption. Our technical approach integrates the management of two resources (processor, network) and their use of two energy management techniques (frequency scaling, low-power sleep mode). In particular, by using application-level knowledge about the deadlines associated with the data streams being transmitted

across mobile nodes, we use a packet scheduler to aggregate network communications while still meeting transmission deadlines. The outcome are increased network card idle times, coupled with decreased number of mode switches. A novel aspect of our research is its concurrent use of CPU frequency scaling to affect how long a network card is kept in sleep mode. We develop an approach to frequency scaling and modifications to a real-time CPU scheduler to further increase network card doze times.

2 Energy Management in Wireless Systems

Wireless cards consume a substantial percentage of a mobile device's energy. For instance, the Compaq iPAQ handheld with a StrongARM processor requires approximately 1.22W when idle, while an Orinoco Gold 11Mbps wireless card requires about 1.4W when transmitting. A key motivation for our research is the fact that this large contribution of network components to total system energy requirements is likely to increase in the future, due to current trends toward increasingly thin, continuously connected client devices.

Energy management is being performed at multiple layers of a system: the physical layer, the protocol stack, the operating system, and the application. With a rising number of energy management techniques available at all of these layers, it is increasingly important to provide coordinated multi-resource and cross-layer approaches to energy management. Key approaches to energy management are: (a) reducing the amount of work (or avoiding work with little or no 'profit'), (b) 'smart' scheduling of resources (e.g., bursty disk or network accesses), and (c) reducing data reads and writes, e.g., to and from networks and disks. In the 802.11b standard, mobile devices can announce to an access point that they wish to switch to a low-power doze or sleep mode. An access point will then periodically inform all sleeping devices if inbound data is pending at the access point. In the case of the Orinoco wireless card, the transmit mode consumes 1.4W, the idle mode requires 0.8W, however, the doze mode requires only 45mW, which suggests that maximizing the times a device can be placed in doze mode can be very effective for the preservation of energy. Besides communication, computation is another significant source of energy consumption. Here, clock frequencies and CPU voltages of modern mobile processors can be reduced with dynamic frequency scaling (DFS) and dynamic voltage scaling (DVS).

The approach to energy management proposed in this work is summarized as follows:

- Real-time data streams have *ready times* t_r (earliest allowable transmission time) and *deadlines* t_d (latest allowable transmission time) associated with each

packet. The task of a QoS packet scheduler is to transmit packets according to their scheduling attributes. Best-effort packets (i.e., no associated deadlines) are transmitted *after* real-time packets. If no schedulable packets are in the device's packet queue, the wireless network card is switched to a low-power 'doze' mode.

- When a packet is placed into the device's packet queue, the device's *watchdog timer* t_w is set to the packet's deadline minus an adjustable *offset* t_o : $t_w = t_d - t_o$. This delays the packet's transmission to the latest possible time, increasing the likelihood of *bursty* transmissions, i.e., if multiple packets are in a device's queue, t_w is set such that these packets can be transmitted together as one burst. The offset value is dynamically adjustable and ensures that data transmission is successful even under network medium contention: If packet deadlines are missed, the offset is increased and if packet deadlines are met consistently over a period of time, the offset is decreased.
- When t_w expires, the device is woken up and the packet scheduler transmits all queued packets with $t_r \leq t_{curr}$, where t_{curr} is the current time. After transmission, the device is switched back to doze mode (i.e., no timeout periods).
- Dynamic frequency scaling is coordinated with this approach such that when the packet scheduler's queue is empty, the lowest possible CPU clock frequency is used to delay the generation of new packets. As soon as the first packet is generated, the system switches to the highest possible clock frequency, in order to increase the number of packets queued (i.e., increasing the burstiness of data transmission) and to accelerate data transmission.
- The real-time CPU scheduler is adjusted such that tasks that have a low probability of packet generation are preferred while the packet scheduler's queue is empty, and tasks that have high probability of packet generation are preferred when the queue is non-empty, again, to increase the burstiness of data transmission.

The goals of these steps are twofold: (1) increase the burstiness of data transfer in order to reduce the number of switches between low- and high-power modes of the wireless card, and (2) minimize the potentially negative effects of dynamic frequency scaling on the energy efficiency of traffic shaping. The resulting advantages are increased burst sizes, a reduced number of costly switches between doze and idle modes, the elimination of costly timeouts, and increases in the communication device's sleep times by 'accelerating' data transmissions.

3 CPU and Packet Scheduling

3.1 DWCS CPU Scheduling

The traditional UNIX scheduler has been shown to have unacceptable performance for multimedia applications [10]. This has led to the development of new scheduling approaches, including those based on reservations and on proportional share resource allocations [13]. To efficiently support real-time applications, we use a hard real-time CPU scheduler, called DWCS (Dynamic Window-Constrained Scheduler) [18]. DWCS assigns to each process i the following attributes: a period T_i , a service time C_i , and a window-constraint x_i/y_i . Using these attributes, DWCS **attempts** to service a process for at least C_i time units in a period of T_i time units, and it **guarantees** that it will service a process in $y_i - x_i$ periods in a window of y_i periods if the *CPU utilization* is less than or equal to 100%. Thus, the minimum CPU utilization of process i is determined by

$$U_i(\min) = (1 - \frac{x_i}{y_i}) * \frac{C_i}{T_i}. \quad (1)$$

The period T_i of process i is used to set a deadline until the scheduler has to service process i for at least C_i time units. If the process misses its deadline more than x_i times in a window of $T_i * y_i$, the scheduler *violated* the real-time guarantees to this process. Each process can run for at most its assigned service time in its period, unless it is marked as *work-conserving*, in which case it is possible to schedule this process several times within its period as long as CPU utilization allows. Table 1 summarizes the precedence rules used by DWCS to find the next process to be scheduled. Details about the DWCS algorithm can be found in [18].

Table 1. Precedence rules.

Earliest deadline fi rst (EDF)
Equal deadlines, then order tightest window-constraint fi rst
Equal deadlines and zero window-constraints, then order highest window-denominator fi rst
Equal deadlines and equal non-zero window-constraints, then order lowest window-numerator fi rst
All other cases: fi rst-come fi rst-serve

3.2 DWCS Packet Scheduling

In this paper, we use the same scheduler, DWCS, as a packet scheduler. The end of a period T_i for a stream i specifies a deadline by which the transmission of a packet has to be initiated. The periods of the CPU scheduler and the packet scheduler are identical, in order to ensure that

packet generation and packet transmission are synchronized (e.g., for video streaming, the period corresponds to the inverse of the frame rate). However, the periods of the packet scheduler are *phase-delayed* with the periods of the CPU scheduler, as shown in Figure 1. This phase Θ determines

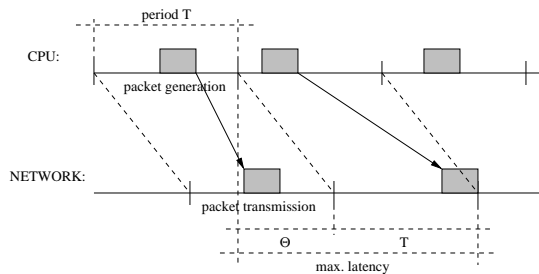


Figure 1. CPU and packet scheduling.

the latency of a packet – from packet generation to transmission – acceptable to the user: $l_{max} = \Theta + T$. For example, a period of 30ms and a phase of 15ms result in a maximum latency of 45ms for a packet. The actual end-to-end latency is further determined by the transmission delays (wireless and wired links) and the reception and processing delays at the client. This paper focuses on sender-side deadlines and on UDP packet transmissions, which support lossy communication as expressed in the window-constraints of both the CPU and packet schedulers.

4 Integrated Energy Management

4.1 Traffic Shaping for Increased Burstiness

The first element of our approach increases the burst size of packet transmissions over a wireless link. Figure 2 shows a scenario where 3 streams are submitted periodically, all with identical periods ($T=200ms$) and packet sizes (5kBytes). All measurements are performed on a Compaq iPAQ H3870 with an Orinoco Gold 11Mbps wireless card and the power measurements are performed with a Picotech ADC-100 PC oscilloscope (2 channels, 100kSamples/s, 12 bit resolution). In this scenario, as soon as a packet is submitted to the device’s packet queue, the device immediately tries to deliver the packet at the earliest possible time (i.e., *ready time*). In keeping with standard energy management techniques, the device is kept in doze mode whenever possible. In this scenario, the network card needs to wake up 3 times each period and transmit a packet. Note the ‘peaks’ at 90 and 190ms, which occur when the device wakes up to receive a beacon signal from the access point. Figure 3 shows the same scenario, but packet transmission is delayed in order to increase the likelihoods of large bursts. When a packet is entered into the packet queue, the transmission

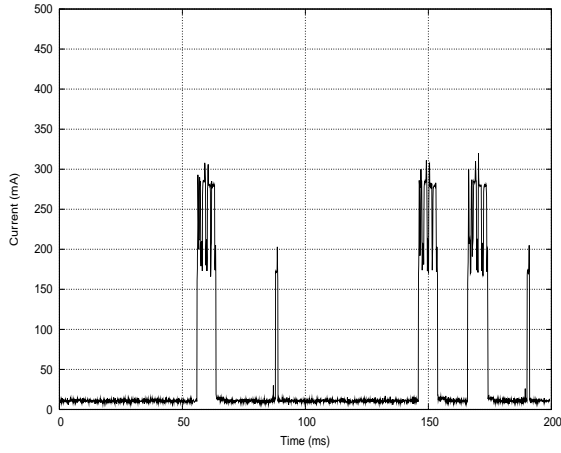


Figure 2. No traffic shaping.

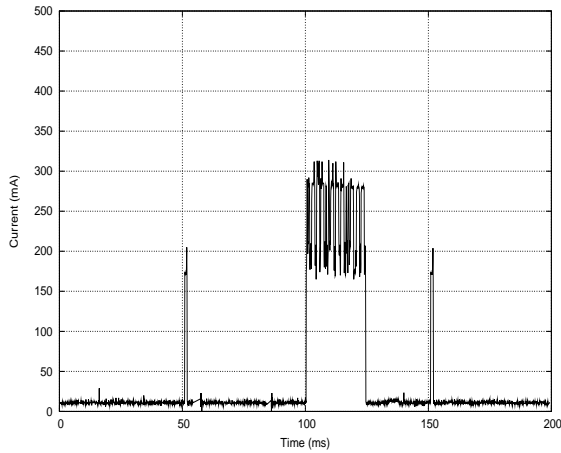


Figure 3. Bursty communication.

time is determined as shown in the following code segment, where t_w is the current watchdog time, t_o is the offset, t_x is the predicted transmission time, T_x is the sum of the predicted transmission times of all previously queued packets, and q_len is the the number of queued packets that will be transmitted before the newly arrived packet (according to the DWCS packet scheduling policy). As before, t_d denotes the packet's deadline and t_r the packet's ready time.

```

1:  if (q_len == 0) {
2:      t_w = t_d - t_o;
3:  } else {
4:      T_x = 0;
5:      for (i=0; i<q_len; i++) {
6:          T_x += queue[i].t_x;
7:      }
8:      t_x = t_w + T_x;
9:      if (t_d < t_x) {
10:         prev_t_w = t_w;
11:         t_w = t_w - (t_x - t_d);
12:         restore = FALSE;
13:         for (i=0; i<q_len; i++) {

```

```

14:         if (queue[i].t_x < queue[i].t_r) {
15:             restore = TRUE;
16:             break;
17:         }
18:     }
19:     if (restore == TRUE) {
20:         t_w = prev_t_w;
21:     }
22: }
23: if (t_d < t_w) {
24:     t_w = t_d - t_o;
25: }
26: enqueue(packet);
27: }

```

Lines 1-2 describe the case when the packet queue is empty and t_w is set such that the newly arrived packet will be transmitted as late as possible. Lines 4-8 compute the predicted transmission times for all packets in the queue that will be transmitted before the newly arrived packet; this time will be used in the subsequent steps. If the scheduler would not be able to transmit the packet before its deadline, the algorithm attempts to adjust t_w such that the new packet will 'fit' into the current burst (lines 9-22). This is done by moving t_w forward by $t_x - t_d$, but only if this action will not cause any previously queued packet to be transmitted before its ready time (lines 14-17). Finally, if $t_d < t_w$, a new watchdog time is set (lines 23-25), i.e., the new packet will form a new burst.

The adaptive offset is initially set to zero; when the packet scheduler observes that deadlines are missed beyond the 'allowable' deadline misses (expressed by x/y), it is increased in steps of 1ms. Once all 'eligible' packets (i.e., the packet's ready time has expired) are transmitted, the watchdog time is set to the deadline (minus the offset) of the next packet in the queue, if the queue is non-empty. This approach can further be refined to coordinate data reception and data transmission, i.e., the watchdog time is set such that each burst is transmitted right before a beacon signal is expected from the access point. That is, a device wakes up, submits all queued packets and then remains awake to listen for a beacon signal and returns to doze mode if no data is pending at the access point. This optimization is not further considered in this paper. All packets that do not have deadlines associated are considered best-effort packets and are transmitted at the end of the current burst. Finally, this paper considers UDP streams, i.e., streams that support lossy transmission of data, e.g., video and audio streams. If either a TCP packet or a UDP packet that has been marked as 'urgent', is submitted to the device, the packet is transmitted immediately and all queued packets with $t_{curr} \geq t_r$ are transmitted.

4.2 Dynamic Frequency Scaling

Modern mobile processors support multiple clock frequencies or voltages. This paper uses frequency scaling, with voltage scaling being considered in our ongoing work.

Scaling the clock frequency has been shown to be effective in saving energy when the processor has idle times [12, 4]. Network activities that involve the CPU, including protocol processing and packet scheduling, particularly when packet fragmentation is used (as supported by IEEE 802.11b), affect how long a network card requires to transmit data and how long it can be kept in doze mode. Figure 4 shows the current drawn by the network device

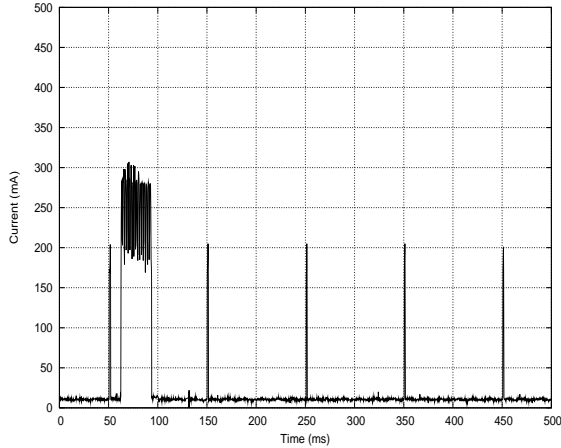


Figure 4. Packet transmission without DFS.

when a packet of size 24kBytes is transmitted over a wireless connection at the default clock frequency of 206.4MHz on the Compaq iPAQ, requiring 31ms. In contrast, Figure 5 shows the same packet being transmitted when the clock frequency is scaled down to 59MHz, requiring 55.5ms. In

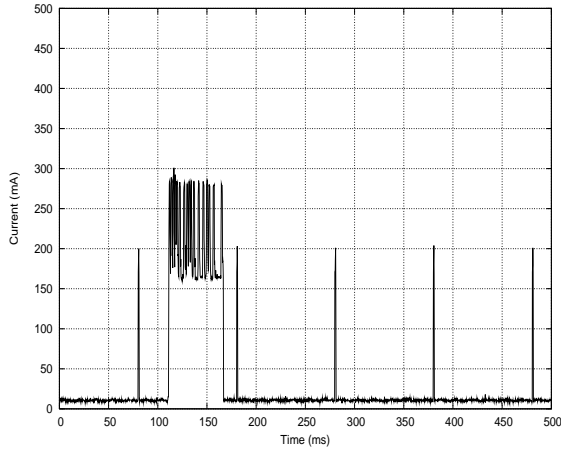


Figure 5. Packet transmission with DFS.

the example shown here, at 59MHz, the network card has to remain active 78% longer than compared to the case with 206.4MHz. This means that frequency scaling – and simi-

larly voltage scaling – affects adversely the doze mode approach for wireless network cards. Further, slowing down the clock frequency also reduces the burstiness of data transmission. The reason is that once the first packet is entered into a packet queue, it takes longer, possibly beyond the deadline of the first packet, until other applications can generate more packets. Figure 6 explains this situation in

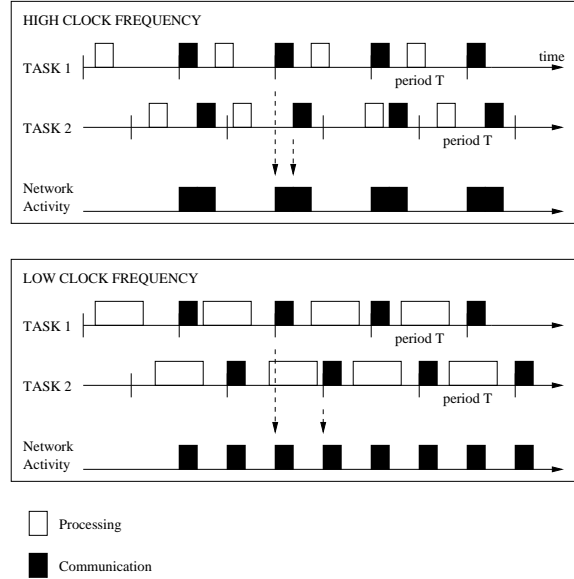


Figure 6. Effect of DFS on burstiness.

detail. Consider two tasks that generate packets with identical periods, but with a phase delay. When task 1 submits a packet, the transmission is delayed until the packet’s deadline. In the meantime, task 2 also generates a packet, which is placed in the scheduler queue behind task 1’s packet and both are transmitted together. In the snapshot shown in the graph, the network is active 4 times. Now consider the case with a low clock frequency, the execution of tasks 1 and 2 requires more time, leading to a scenario where task 2 cannot generate the packet in time such that it could be send at the same time as task 1’s packet, and therefore will be transmitted at its own deadline. The result is that the network has to be active twice as frequent. We address this problem by modifying the way frequency scaling is used.

Original Method. Clock frequencies are typically computed such that the *rest utilization* of the CPU is exploited by slowing down task execution. With DWCS, we compute a new clock frequency whenever the system utilization changes, e.g., when tasks join or leave the run queue. The current utilization of all tasks is computed with

$$U = \sum (1 - \frac{x_i}{y_i}) * \frac{C_i}{T_i}. \quad (2)$$

C_i is the service time allocated to task i at the default clock

frequency and this service time increases when the clock is slowed down. For each clock frequency n , a *scaling factor* k_n can be obtained by executing a sample processing-intensive code at both the default frequency f_{max} and f_n and dividing the measured run-times: $k_n = C_n/C_{max}$. This is repeated for each available clock frequency (or core voltage) for a given processor. The goal of frequency scaling is to get as close to 100% utilization as possible, i.e.,

$$U_{100\%} = \sum (1 - \frac{x_i}{y_i}) * \frac{C_i * k'}{T_i} \quad (3)$$

where k' is the yet unknown scaling factor. To guarantee that best-effort tasks are not starved, we can replace $U_{100\%}$ with $U_{x\%}$, e.g., $U_{95\%}$. Then k' can be determined with:

$$k' = \frac{U_{95\%}}{\sum (1 - \frac{x_i}{y_i}) * \frac{C_i}{T_i}} \quad (4)$$

The resulting k' is compared to the previously obtained scaling factors, and the scaling factor k_n closest to k' ($k_n \leq k'$) is selected, and the clock frequency is adjusted to frequency n .

Adjusted Method. The original method is adjusted in the following way. The clock frequency is kept at its maximum while the packet scheduler queue is non-empty and while packets are transmitted. Once the queue is empty, we switch to a lower clock frequency to exploit idle CPU time. Here, k' is calculated as follows:

$$k' = \frac{U - U'}{\sum (1 - \frac{x_i}{y_i}) * \frac{C_i}{T_i} - U'} \quad (5)$$

U' is obtained by measuring and averaging the time from the arrival of the first packet in the device's queue until the end of the transmission of the last packet in the next burst. It expresses the portion of the CPU utilization during which the device is operated at the maximum clock frequency and equation (5) computes the scaling factor (and therefore the clock frequency) for the remaining portion of the CPU utilization. Besides ensuring that the burst size is maximized, this further reduces the delays during packet transmission. Figure 7 shows an example for packet transmission with our approach, where computations are performed with frequency f_{59} until packets are enqueued, then the clock frequency is switched to f_{max} .

Burst Sizes. If the average burst size is small (e.g., 1 packet), there is no advantage in switching the clock frequency at the time the packet is placed into the queue. We therefore monitor the actually achieved burst sizes and switch between two modes: (a) if the average burst size is 2 or higher, the clock frequency is switched to f_{max} at the time the first packet is enqueued, (b) otherwise, the clock frequency is switched to f_{max} at the time packet transmission starts.

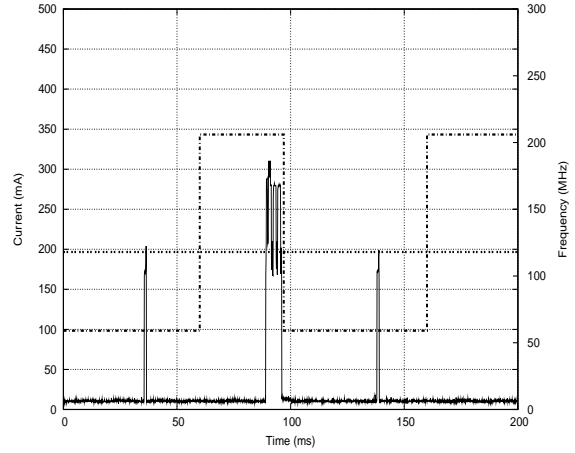


Figure 7. Adjusted DFS algorithm.

Scheduler Queue Fill Level. The ready time of a packet determines its earliest possible transmission time. It could happen that an application submits packets with ready times further in the future, preventing them from being transmitted in the current burst. In that case, our algorithm would not switch back to f_n after the burst is transmitted because packets are still pending in the queue. Here, we modify the algorithm to ensure that the CPU clock is executed at the lower frequency for at least 50% of the time, to exploit idle CPU times. This is achieved by setting the device clock to f_n after transmitting a burst and the CPU clock is set to f_{max} either when t_w expires (i.e., transmission begins) or the clock has run at f_n for at least 50% over a certain window of time (e.g., 100ms).

4.3 CPU Scheduling

Each task or each time slice of a task has a probability of network communication associated. For example, a probability of 0 may be assigned to processing-intensive tasks that do not use the network card, where a probability of 1 may be assigned to communication-intensive tasks. In general, such probabilities can be assigned to tasks through online monitoring. For example, a video capture task may require 3 time slices for each frame. While the first and second slices in each period are required for camera operation, the processing, and the compression of a video frame, the third time slice may be the one during which the frame is actually submitted to the network connection. In that case, time slices 1 and 2 will have a low probability and slice 3 has a high probability of network communication. Consider Figure 8, where the top graph shows the task or time slice communication probabilities, and the bottom shows network activity. While the network queue is empty, a CPU scheduler can choose low probability tasks

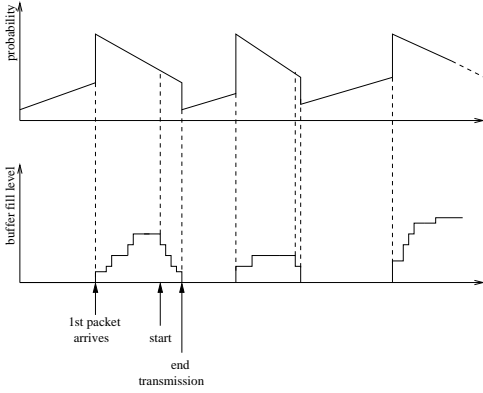


Figure 8. Probability of network activity and packet scheduler queue fill levels.

in order to maximize the time the device is operated with a low clock frequency, and in addition, the network card can be kept in doze mode. When the first packet is generated and placed in the packet scheduler queue, we begin to prefer tasks with high probabilities of packet generation in order to ensure that as many packets as possible will arrive in the packet queue to maximize burstiness. Further, the DFS mechanism switches the clock frequency to the highest possible clock frequency. Once the packets are transmitted, the clock frequency is set to low and low-probability tasks are preferred. Table 2 shows the precedence rules used by the modified DWCS CPU scheduler to find the next process to be scheduled. The new rule (shown in bold) ensures

Table 2. Modified precedence rules.

Earliest deadline fi rst (EDF)
Equal deadlines, then order lowest probability of network activity first
Equal deadlines and equal probabilities, then order tightest window-constraint fi rst
Equal deadlines and zero window-constraints, then order highest window-denominator fi rst
Equal deadlines and equal non-zero window-constraints, then order lowest window-numerator fi rst
All other cases: fi rst-come fi rst-serve

that tasks with low probability of network activity are given preference over other tasks. Note, that this rule is executed frequently (i.e., task deadlines are identical), since (a) periods for multimedia streaming are likely to be 'close' to each other (e.g., corresponding to transmission rates of audio and video), (b) periods in the DWCS CPU scheduler are 'aligned' such that *hyper periods* can be formed, which facilitate the computation of CPU utilization or clock fre-

quencies, and (c) the timing parameters in DWCS are not chosen randomly, but as multiples of 'jiffies', the unit of time used in Linux (e.g., 10ms).

4.4 Overhead Considerations

By running the device at two different clock frequencies (f_{max} and f_n), it is possible that we do not fully exploit CPU idle times. However, this is acceptable when we compare the potential loss in energy savings for running the CPU at higher clock frequencies with the gains in energy savings for aggregating larger bursts and faster data transmissions. Consider a situation where a handheld is run at both the maximum clock frequency (206.4MHz) and the lowest possible clock frequency (59MHz). If the utilization is 100%, then the difference in energy consumption is 13.1mJ for the execution of the same code at 206.4 and 59MHz. The handheld used in these experiments has 12 different clock frequencies, i.e., between two neighboring frequencies, the difference in energy consumption is only slightly more than 1mJ. Since our algorithm uses f_{max} for at most 50% of the time, the differences in energy consumption are 6.55mJ (between the highest and lowest possible clock frequencies) and 0.5mJ (between neighboring clock frequencies) in the worst case. In contrast, consider the energy costs for the transmission of data over a wireless link at the default frequency and the lowest clock frequency. For example, a packet of size 19.3kBytes takes 25ms to be transmitted, i.e., the wireless card is active for that period of time. At 59MHz, even though the costs for the actual transmission are identical, the card has to stay in idle mode longer because of the delays caused by the the lower clock frequency. Here, the card has to stay active for 80ms, resulting in an additional energy consumption of 44mJ. Again, the difference for neighboring clock frequencies would be about 3.7mJ. That is, if the clock frequency is set to a higher value than necessary, then the loss of energy savings for the CPU are outweighed by a gain in energy savings for the network transmission by a factor of at least 7.4 in this example. The higher the communication (more or larger packets), the higher this factor.

5 Related Work

There has been substantial work on power management for mobile devices, including low-power modes for disks and networks [6, 3], power-aware scheduling policies [16], and energy management techniques for wireless communication [14, 15]. The authors in [7] describe a modification to the power saving mechanism in the IEEE 802.11 Distributed Coordination Function. Their aim is to remove the overheads associated with the TIM windows and to increase

the available bandwidth for data transmission. In the PAMAS [17] approach, separate control and data channels are used, where the control channel is used to determine when and how long to turn off a wireless network card. The approach suggested in this paper addresses the efficient integration of multiple such energy management techniques in order to prevent one technique from negating the results of another. In [19, 9], the authors address the integration of resource management across different layers of a system. The approach introduced in this paper complements work done for server-side traffic shaping, e.g., in [3]. It also has similarities to the approach in [11], where disk access pattern burstiness is increased in order to decrease the time a disk has to be kept in high-power active mode.

6 Conclusions and Future Work

This paper introduces an approach to increase the time a network card can be kept in doze mode and to reduce the number of switches between doze and active modes. It does so by delaying packet transmission to achieve large burst sizes, and by cooperatively managing CPU scheduling, dynamic frequency scaling, and packet scheduling. One important conclusion from this work is that it is essential to go beyond single-resource techniques for energy management and to investigate multi-resource techniques for quality and energy management.

While the approach introduced in this paper addresses energy management by tight integration of CPU scheduling, traffic shaping, and frequency scaling, our future work will extend this approach to different layers of a system and across protection and network boundaries. That is, to efficiently manage the energy of a system, all layers (application, middleware, operating system, network, etc.) have to cooperate and possible effects of one energy management technique on the results of all others have to be considered.

References

- [1] K. Barr and K. Asanovic. Energy Aware Lossless Data Compression. In *Proc. of the First International Conference on Mobile Systems, Applications, and Services*, May 2003.
- [2] S. Chandra, C. S. Ellis, and A. Vahdat. Managing the Storage and Battery Resources in an Image Capture Device (Digital Camera) using Dynamic Transcoding. In *Proc. of the Third ACM International Workshop on Wireless Mobile Multimedia (WoWMoM)*, August 2000.
- [3] S. Chandra and A. Vahdat. Application-specific Network Management for Energy-aware Streaming of Popular Multimedia Formats. In *Proc. of the USENIX Annual Technical Conference*, 2002.
- [4] D. Grunwald, P. Levis, C. B. M. III, M. Neufeld, and K. I. Farkas. Policies for Dynamic Clock Scheduling. In *Proc. of the 4th Symposium on Operating System Design and Implementation (OSDI)*, 2000.
- [5] P. J. M. Havinga and G. J. M. Smit. Energy-Efficient Wireless Networking for Multimedia Applications. *Journal on Wireless Communications and Mobile Computing*, 2001.
- [6] D. P. Helmbold, D. D. E. Long, and B. Sherrod. A Dynamic Disk Spin-down Technique for Mobile Computing. In *Proc. of the Intl. Conference on Mobile Computing and Networking*, 1996.
- [7] E. Jung and N. Vaidya. An Energy-Efficient MAC Protocol for Wireless LANs. In *Proc. of IEEE Infocom*, June 2002.
- [8] R. Kravets and P. Krishnan. Application-driven Power Management for Mobile Communication. *Wireless Networks*, 6(4):263–277, 2000.
- [9] P. Mohapatra, J. Li, and C. Gui. QoS in Mobile Ad hoc Networks. *Special Issue on QoS in Next-Generation Wireless Multimedia Communications Systems in IEEE Wireless Communications Magazine*, June 2003.
- [10] J. Nieh, J. G. Hanko, J. D. Northcutt, and G. A. Wall. SVR4 UNIX Scheduler Unacceptable for Multimedia Applications. In *Proc. of the 4th Int. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV '93)*, Lancaster, UK, November 1993.
- [11] A. E. Papathanasiou and M. L. Scott. Energy Efficiency through Burstiness. In *Proc. of the Fifth IEEE Workshop on Mobile Computing Systems and Applications*, October 2003.
- [12] P. Pillai and K. G. Shin. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *Proc. of the 18th ACM Symposium on Operating Systems Principles*, 2001.
- [13] T. Plogemann, V. Goebel, and P. Halvorsen. Operating System Support for Multimedia Systems. *Computer Communications Journal, Special Issue on Interactive Distributed Multimedia Systems and Telecommunications Services (IDMS '98)*, 1998.
- [14] C. Poellabauer and K. Schwan. Energy-Aware Media Transcoding in Wireless Systems. In *Proc. of the Second IEEE Intl. Conference on Pervasive Computing and Communications (PerCom 2004)*, March 2004.
- [15] D. Qiao, S. Choi, A. Jain, and K. G. Shin. MiSer: An Optimal Low-Energy Transmission Strategy for IEEE 801.11a/h. In *Proc. of the ACM/IEEE Intl. Conference on Mobile Computing and Networking*, September 2003.
- [16] S. Saewong and R. Rajkumar. Practical Voltage-Scaling for Fixed-Priority RT-Systems. In *Proc. of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, May 2003.
- [17] S. Singh, M. Woo, and C. S. Raghavendra. Power-aware Routing in Mobile Ad Hoc Networks. In *Proc. of MOBICOM '98*, October 1998.
- [18] R. West, K. Schwan, and C. Poellabauer. Scalable Scheduling Support for Loss and Delay Constrained Media Streams. In *Proc. 5th Real-Time Technology and Applications Symposium*, Vancouver, Canada, 1999.
- [19] W. Yuan, K. Nahrstedt, S. Adve, D. Jones, and R. Kravets. Design and Evaluation of a Cross-Layer Adaptation Framework for Mobile Multimedia Systems. In *Proc. of the SPIE/ACM Multimedia Computing and Networking Conference*, January 2003.