Appears in IEEE International Conference on Multimedia and Expo (ICME'03)

BUFFERING APPROACH FOR ENERGY SAVING IN VIDEO SENSORS

Wanghong Yuan, Klara Nahrstedt

Department of Computer Science
University of Illinois at Urbana-Champaign
{wyuan1, klara}@cs.uiuc.edu

ABSTRACT

Networked video sensors need to execute two dependent periodic tasks: video encoding and transmission. The dependency and periodicity often result in small idle intervals of CPU and wireless network interface card (WNIC). In this paper, we present a *sender-buffering* approach to exploit such idle intervals for energy saving. Specifically, a video sensor encodes frames in a timely fashion, but buffers encoded frames and transmits them in bursts at longer intervals. In doing so, it (1) accumulates short WNIC idle intervals into longer ones, during which the WNIC can enter the lower-power sleep mode, and (2) slows down the CPU by avoiding CPU idle intervals, which are resulted from both early completion of frame encoding and waiting for frame transmission. Our experimental results show that the buffering approach can save 32-80% CPU energy and 35-54% WNIC energy, while increasing the overall end-to-end transmission delay by at most 2 frames.

1. INTRODUCTION

Wireless video sensor networks significantly extend our ability to monitor and control remote objects (e.g., for surveillance or highway traffic management). Each sensor runs on a battery-powered mobile device, and needs to execute two tasks, an *encoder* and a *transmitter*. The encoder compresses captured raw images into proper video formats such as MPEG or H263, while the transmitter sends video frames across wireless networks to the processing center. Each task is typically implemented as a periodic thread (or process). These two threads are dependent on each other through the producer-consumer relation: the transmitter waits for frames from the encoder; the encoder starts to encode the next frame only after the transmitter sends out previously encoded frames. They also consume different system resources: the encoder consumes a lot of CPU time, while the transmitter consumes a lot of wireless bandwidth.

This work was supported by NSF under CCR 02-05638 and CISE EIA 99-72884, and by NASA under NAG 2-1250. We would like to thank the GRACE project members for informative discussions on energy saving, and thank Daniel Sachs and Kai Chen for help on the implementation.

One important design goal in sensor networks is to save energy for long operation time. To provide energy saving capability, hardware components of mobile device often support multiple operation states, trading off performance and power. For example, processors and wireless network interface cards (WNICs) can operate with three modes: *active*, *idle*, and *sleep*, where the sleep mode consumes much less power [1, 8, 7]. In addition, processors can run at multiple speeds (frequencies/voltages) in the active mode; they consume less energy at a lower speed [1].

In the system level, therefore, we can save energy either by setting components into low-power modes (dynamic power management, or DPM) [8] or by slowing down the CPU (dynamic voltage scaling, or DVS) [9, 10, 6, 5]. The DVS overhead (i.e., the time for changing CPU speed) is often negligible, while the DPM overhead (i.e., the time for switching modes, especially from sleep to active) is much larger. For example, the time for changing an AMD CPU speed is less than 120 μs [1], while the time for switching a WaveLan card from sleep to active is about 40 ms [8].

In the application level, however, the dependency and periodicity of the encoder and transmitter tasks may result in energy waste for two reasons. First, the CPU and WNIC both have an idle interval in each period, since the encoding and transmission of a frame usually complete before the next frame is available. DVS can be used to avoid the CPU idle interval by slowing down the CPU. However, the WNIC idle interval is often smaller than the DPM overhead for switching to sleep [7]; therefore, the WNIC cannot enter the lower-power sleep mode. Second, the CPU is idle during the frame transmission, since the encoder must wait until the transmitter sends out the current frame. DVS cannot avoid such CPU idle intervals; neither can DPM since they are smaller than the DPM overhead.

This paper presents a *sender-buffering* approach to save energy for video sensors. Specifically, the video sensor encodes a frame every period in a timely fashion, but delays video transmission by buffering encoded frames and sending them in bursts at longer intervals. In doing so, it saves both CPU and WNIC energy. First, it combines short, per-period WNIC idle intervals into longer ones, during which

Table 1. Power and DPM overhead of a WaveLan card

active power	idle power	sleep power	DPM overhead
p_{trn} = 1.5W	p_{idl} = 1W	p_{slp} = 0.1W	t_{slp} = 40ms

Table 2. Speed and relative power of an Athlon CPU

freq. (MHz)	300	500	600	700	800	1000
voltage (Volt)	1.2	1.2	1.25	1.3	1.35	1.4
relative power	0.22	0.36	0.47	0.6	0.74	1

the WNIC can enter the lower-power sleep mode. Second, it slows down the CPU by avoiding CPU idle intervals, which are resulted from both early completion of frame encoding and waiting for frame transmission.

Similar buffering approaches have been used to save energy in the *receiver side* [5, 2]. For example, Im et al. [5] and Chandra et al. [2] used buffers to save CPU and network energy, respectively, for clients. In contrast, our buffering approach aims the *sender side* in sensor networks, and saves *both* CPU and WNIC energy.

The rest of the paper is organized as follows. Section 2 introduces system models and assumptions. Section 3 describes the buffering algorithm. Section 4 presents experimental evaluation. Finally, Section 5 summarizes the paper.

2. SYSTEM MODELS

Each video sensor runs on a mobile device with a single CPU and WNIC. The WNIC supports four operation modes: *transmission*, *receiving*, *idle*, and *sleep*, where the sleep mode has much less power. We refer to transmission as the *active* mode, since we focus on frame transmission in video sensors. DPM can be used to transit the WNIC between different modes [8]. The time for transition between active and idle is small and negligible for task execution. However, the time for transition from sleep to active is large and hence is the major DPM overhead. The WNIC can enter sleep only when its idle interval is larger than the DPM overhead. Table 1 shows the power parameters of a Lucent WaveLan wireless card as an example [8].

The CPU can run at a set of frequencies, $\{f_1, ..., f_{max} | f_1 < \cdots < f_{max}\}$. The average CPU power at frequency f is $p(f) \propto C_L f V_f^2$, where C_L is the load capacitance and V_f is the voltage at frequency f [3]. Table 2 shows the frequencies, voltages, and power (relative to $p(f_{max})$) of an AMD Athlon CPU as an example [1]. In this paper, we assume that when the CPU is idle at frequency f, its power is same as in running mode, i.e., $p(f)$. This assumption holds since the load capacitance C_L is roughly constant during idling and running. Furthermore, we do not use DPM to put the idle CPU into the lower-power sleep mode, because the DPM overhead is typically larger than the video encoding periods (and consequently CPU idle intervals).

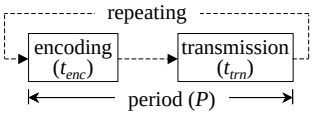

Fig. 1. Task dependency and timing constraint

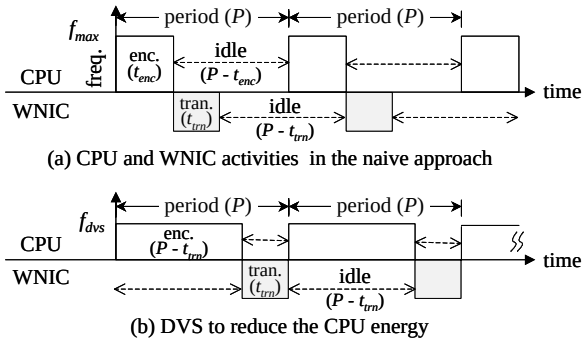

Fig. 2. The naive and DVS approaches

3. BUFFERING ALGORITHM

3.1. Naive Approach

The encoding and transmission tasks of a video sensor are *dependent* on each other (Figure 1): the transmitter must wait until the encoder completes a frame encoding; the encoder can encode the next frame only after the transmitter sends out the current frame. We use t_{enc} and t_{trn} to denote the average frame encoding and transmission times, respectively, at the maximum CPU frequency. We assume that (i) t_{enc} is inverse-proportion to the frequency, since the number of CPU cycles for encoding a frame is roughly constant at different frequencies, and (ii) t_{trn} is determined by frame size and wireless bandwidth only. It does not change with frequency, because cycles for transmission (e.g., for transport protocol processing) are negligible.

A naive approach is to encode a frame and transmit it within each period P. Figure 2-(a) shows CPU and WNIC activities in the naive approach, where the CPU runs at the maximum frequency. In each period, CPU and WNIC both have an idle interval, because the encoding and transmission of a frame complete before the next frame is available. However, CPU and WNIC cannot enter sleep during such idle intervals, which are smaller than the DPM overheads. Equation (1) and (2) give CPU and WNIC energy consumption in each period, respectively:

$$E_{nav}^{cpu} = p(f_{max}) \times P \quad (1)$$

$$E_{nav}^{nic} = p_{trn} \times t_{trn} + p_{idl} \times (P - t_{trn}) \quad (2)$$

3.2. DVS Approach

We can use DVS to reduce CPU energy by slowing down the CPU. Specifically, the CPU can run at a lower frequency

$f_{dvs} = \frac{t_{enc}}{P-t_{trn}} f_{max} < f_{max}$. At frequency f_{dvs}, a frame encoding needs $P - t_{trn}$ time units since $f_{dvs} \times (P - t_{trn}) = f_{max} \times t_{enc}$; the transmitter then uses the remaining time t_{trn} in the period to send the encoded frame (Figure 2-(b)). Compared to the naive approach, the DVS approach reduces CPU energy and consumes the same WNIC energy:

$$E^{cpu}_{dvs} = p(f_{dvs}) \times P < E^{cpu}_{nav} \quad (3)$$

$$E^{nic}_{dvs} = p_{trn} \times t_{trn} + p_{idl} \times (P - t_{trn}) = E^{nic}_{nav} \quad (4)$$

The above DVS approach avoids CPU idle intervals resulted from early completion of frame encoding. However, it cannot avoid CPU idle intervals resulted from waiting for frame transmission, as shown in Figure 2-(b).

3.3. Buffering Approach

To save more energy, we propose a *sender-buffering* approach for video sensors. A video sensor still encodes a frame every period in a timely fashion, but delays the transmission by buffering frames and sending them in bursts at longer intervals (Figure 3). Specifically, let's assume that the buffer size is k frames. In each period, the encoder encodes a frame and stores it in the buffer. When the buffer has k frames (i.e., every k periods), the transmitter sends all buffered frames in batch.

The buffering approach shapes CPU and WNIC activities (Figure 4). First, it combines short WNIC idle intervals with length $(P - t_{trn})$ into longer ones with length $k(P - t_{trn})$. Such aggregate idle intervals are larger than the DPM overhead; so, the WNIC can enter sleep. Second, the encoder can start to encode the next frame before the transmitter sends out previous frames. Hence, it avoids CPU idle intervals resulted from waiting for frame transmission, and can run the CPU at a lower frequency $f_{buf} = \frac{t_{enc}}{P} f_{max} < f_{dvs}$. The buffering approach saves more energy. Equation (5) and (6) give CPU and WNIC energy consumption in every k periods, respectively:

$$E^{cpu}_{buf} = p(f_{buf}) \times kP < kE^{cpu}_{dvs} \quad (5)$$

$$E^{nic}_{buf} = p_{trn} \times kt_{trn} + p_{slp} \times k(P - t_{trn}) < kE^{nic}_{dvs} \quad (6)$$

where we assume that the WNIC power during transition from *sleep* to *active* is same as in *sleep*.

Next, we analyze the selection of the buffer size k and the corresponding performance impact. To put the WNIC into sleep during its idle intervals, the aggregate idle interval in k periods should be larger than the DPM overhead, i.e., $k(P - t_{trn}) \geq t_{slp}$. Hence, we get $k \geq \frac{t_{slp}}{P-t_{trn}}$. That is, the minimum required buffer size is $k = \lceil \frac{t_{slp}}{P-t_{trn}} \rceil$. Given such a buffer size k, the buffering approach increases the overall end-to-end transmission delay by $(k-1) \times P$ time units (Figure 3). However, it does not affect frame encoding, since the encoder still encodes a frame every period in a timely fashion.

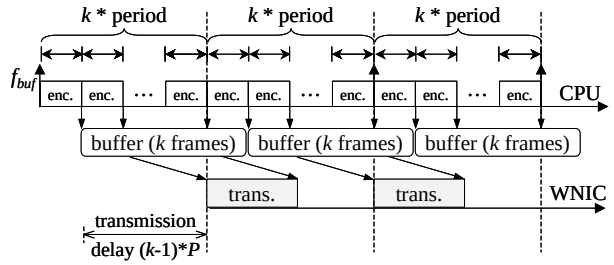

Fig. 3. The buffering approach

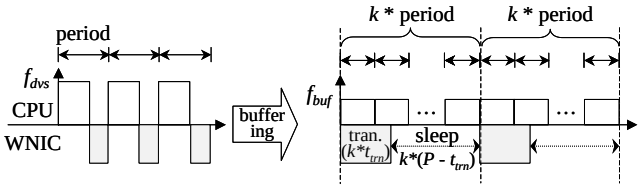

Fig. 4. Buffering to shape the CPU and WNIC activities

4. EXPERIMENTAL EVALUATION

4.1. Experiment setup

Although we developed the sender-buffering technique for small sensors, our prototype is implemented on the PC architecture. Specifically, we use H263 encoder as the video sensor, which reads images from a local file, encodes them to H263 frames, and sends them via UDP to a receiver. The H263 encoder runs on HP N5470 laptop, and the receiver runs on IBM T23 laptop. Both laptops have Red Hat Linux 7.2, and are connected via two 11 Mbps Lucent WaveLan cards (ad hoc mode). The HP laptop has a single AMD Athlon CPU with six different speeds (Table 2).

We have implemented a kernel module to adjust the CPU speed [10]. We simulate the WNIC DPM control by assuming that the WNIC enters sleep whenever its idle intervals are larger than its DPM overhead. To identify such idle intervals, we modify the WaveLan driver to trace the beginning and ending of transmission. In our experiments, we have not measured the actual energy consumption, because we currently do not have power meters like PowerScope [4]. Instead, we calculate CPU and WNIC energy consumption by using the power parameters in Table 1 and 2 (assuming $p(f_{max}) = 1$ Watt).

4.2. Results

The H263 encoder can operate with three different configurations by varying parameters of quantization and motion vector search. At the highest CPU frequency, we profile the H263 encoder with the input image file `mobile.cif`, which contains 300 images, to probe average frame encoding time (t_{enc}) and transmission time (t_{trn}). Table 3 shows the probing results.

Table 3. Configurations and corresponding t_{enc} and t_{trn}

configuration	*conf*-1	*conf*-2	*conf*-3
encoding time t_{enc} (ms)	26.7	22.8	18.0
transmission time t_{trn} (ms)	11.7	13.5	27.3

For each configuration, the H263 encoder can run at period 40, 50, 60, 100, and 150 ms. We first calculate the minimum required buffer size, which enables the WNIC to sleep, for the above configurations and periods. The results (Figure 5-(a)) imply that the buffering approach increases a small end-to-end transmission delay (by at most 2 frames in our experiments).

Next, we run the H263 encoder with *conf*-3 and period 50 ms to compare energy consumption in the naive, DVS, and buffering approaches. The results (Figure 5-(b)) show that the buffering approach significantly reduces the CPU and WNIC energy (e.g., saving 32% CPU energy and 44% WNIC energy compared to the DVS approach).

Finally, we run the H263 encoder with *conf*-3 and various periods to evaluate the relationship between energy and period. The results (Figure 6) show that, in the buffering approach, energy consumption is *not* linear to periods (and hence device operation time, $period \times frames$). The reason is that a longer period may cause longer idle intervals, during which CPU and WNIC can operate at lower-power modes. Another interesting result is that, when the period is greater than 100 ms, the buffering approach consumes the same energy as the DVS approach. The reason is that the DVS approach already runs the CPU at the minimum frequency (since CPU utilization is very low), and switches the WNIC into sleep during its idle interval (which is larger than the DPM overhead) in each period. This indicates that the buffering approach is more effective when the video sensor has short periods, which is the common case.

5. CONCLUSION

This paper presents a *sender-buffering* approach to save CPU and WNIC energy for mobile video sensors. In this buffering approach, each sensor buffers encoded frames and transmits them in bursts to exploit small CPU and WNIC idle intervals. As a result, it can slow down the CPU and switch the WNIC into the lower-power sleep mode, thereby saving energy. Our experiments show that compared to the naive and DVS approaches, the buffering approach can save 32-80% CPU energy and 35-54% WNIC energy, while increasing the end-to-end transmission delay by at most 2 frames.

6. REFERENCES

[1] AMD. Mobile AMD Athlon 4 processor model 6 CPGA data sheet. http://www.amd.com, Nov. 2001.

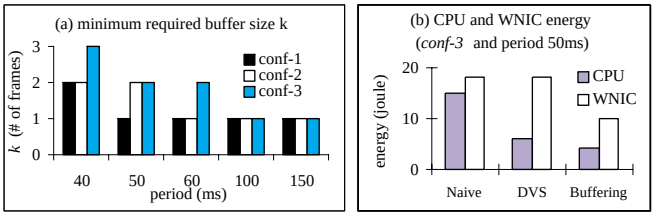

Fig. 5. (a) Minimum buffer size, and (b) Energy comparison

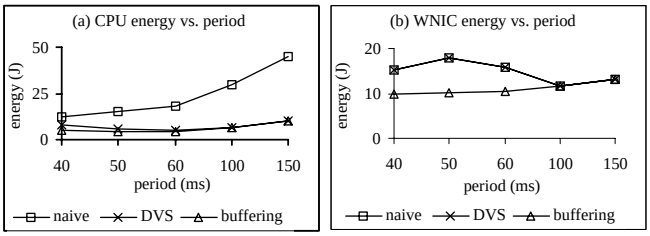

Fig. 6. Energy consumption over periods

[2] 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), Monterey, CA*, June 2002.

[3] A. Chandrakasan, S. Sheng, and R. W. Brodersen. Low-power CMOS digital design. *IEEE Journal of Solid-State Circuits, Vol. 27*, pages 473–484, Apr. 1992.

[4] J. Flinn and M. Satyanarayanan. PowerScope: A tool for proling the energy usage of mobile applications. In *Proc. of 2nd IEEE Workshop on Mobile Computing Systems and Applications*, Feb. 1999.

[5] C. Im, H. Kim, and S. Ha. Dynamic voltage scheduling technique for low-power multimedia applications using buffers. In *Proc. of 2001 International symposium on Low power electronics and design*, Aug. 2001.

[6] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *Proc. of International Symposium on Low-Power Electronics and Design (ISLPED'98)*, 1998.

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

[8] T. Simunic, L. Benini, A. Acquaviva, P. Glynn, and G. Micheli. Dynamic voltage scaling and power management for portable systems. In *Proc of Design Automation Conference (DAC 2001)*, June 2001.

[9] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *Proc. of Symposium on Operating Systems Design and Implementation (OSDI'94)*, Nov. 1994.

[10] 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 SPIE Multimedia Computing and Networking Conference*, Jan. 2003.