Title
The Energy Endoscope: Real-time Detailed Energy Accounting for Wireless Sensor Nodes

Authors
Stathopoulos, Thanos
McIntire, Dustin
Kaiser, W J

Publication Date
2007-11-15

The Energy Endoscope: Real-time Detailed Energy Accounting for Wireless Sensor Nodes

Thanos Stathopoulos Dustin McIntire William J. Kaiser

Center for Embedded Networked Sensing
UCLA, Department of Electrical Engineering
{thanos@cs.ucla.edu, dustin@seas.ucla.edu, kaiser@ee.ucla.edu}

Abstract

This paper describes a new embedded networked sensor platform architecture that combines hardware and software tools providing detailed, fine-grained real-time energy usage information. We introduce the LEAP2 platform, a qualitative step forward over the previously developed LEAP [13] and other similar platforms. LEAP2 is based on a new low power ASIC system and generally applicable supporting architecture that provides unprecedented capabilities for directly observing energy usage of multiple subsystems in real-time. Real-time observation with microsecond-scale time resolution now enables direct accounting of energy dissipation for each computing task as well as for each hardware subsystem. This new hardware architecture is exploited with our new software tools, etop *and* endoscope*. A series of experimental investigations provide high-resolution power information in networking, storage, memory and processing for primary embedded networked sensing applications. Using these results obtained in real-time we show that for a large class of wireless sensor network nodes, there exist several interdependencies in energy consumption between different subsystems. Through the use of our measurement tools we demonstrate that by carefully selecting the system operating points, energy savings of over* 60% *can be achieved while retaining system performance.*

1 Introduction

Low energy operation continues as a fundamental limiting challenge for Wireless Sensor Network (WSN) systems. Most recently, the critical demand for high energy efficiency operation has increased yet further with the introduction of the new class of 32-bit processor-based WSN nodes that meet new application requirements for support of high performance sensors and complex algorithms [13, 7, 1, 11, 10]. While these platforms present high peak operating power demand, prior work has shown that design objectives focusing on high energy efficiency along with proper scheduling of platform components yield low average energy operation [13]. However, as will be shown here, high efficiency may be achieved only if platform energy usage is measured at runtime, thereby revealing the effects of not only unpredictable application demands, but also of contention between platform components and computing tasks.

This paper describes a new hardware and software architecture that forms an Energy Endoscope with the first (to our knowledge) integrated, low-power, real-time energy monitoring for WSN and other embedded systems. In contrast to prior systems that measure energy only at low temporal sample rates and determine the average over all platform components [16, 5, 2, 8], this new architecture resolves energy usage at real time and for each hardware subsystem. The Energy Endoscope relies on a unique hardware and software architecture solution that includes a new Energy Management and Accounting (EMAP2) ASIC combined with operating system kernel features that nearly eliminate the prohibitively large energy and computing overhead that high rate energy accounting would levy on conventional platforms. Experimental results will demonstrate that energy optimization requires this capability and that it must be integrated with the platform in order to reveal the large amplitude energy excursions that can only be observed with high rate sampling at runtime. Further, as experimental results will show, only with this data can a system balance the conflicting demands of interdependent subsystems to avoid unnecessary energy usage. It is very important to note that the first applications of the Energy Endoscope have immediately shown benefits for large energy optimization without loss in WSN node performance.

The Energy Endoscope enables energy optimizations that are adaptive to the many characteristics of applications

which can only be known during runtime field operation. Applications now being developed with this new platform include seismic sensor networks, structural health monitoring networks, marine sensor systems, and biomedical monitoring. These important applications share increasingly common WSN requirements for support of high performance sensor systems, on-demand use of either long range or broadband wireless communication, and complex signal processing, sensor fusion, and networking algorithms.

The primary contribution of this paper is the introduction of a new platform architecture and a new set of hardware and software energy measurement tools. We utilize the unique capabilities of our hardware and software tools to provide unprecedented energy consumption visibility. We have used our tools for real-time profiling of several important subsystems such as CPU, memory, storage and networking. We show that there exist drastic differences in power dissipation over time depending on the choice of storage device and also reveal the significant effect of CPU power dissipation on storage and networking operations. We also profile a network file transfer operation between two LEAP2 nodes and discover the optimal CPU speeds for both the sender and the receiver, resulting in overall energy savings above 60%.

This paper is organized as follows: Section 2 describes the LEAP2 platform and the EMAP2 while Section 3 describes our software tools. Section 4 presents experimental discoveries when profiling power dissipation of a single node, with a focus on CPU-intensive and storage-intensive workloads. In Section 5 we profile a networking application that exercises several important subsystems over two nodes and also experimentally derive the optimal CPU speeds that minimize energy consumption. We present related work in Section 6 and conclude in Section 7.

2 The LEAP2 Architecture

In this section we provide an overview of current techniques for energy accounting and also describe the LEAP2 hardware architecture.

Traditional energy accounting techniques in WSNs as well as in mobile computing rely on external device support—such as oscilloscope sampling or data acquisition systems [16, 5]—or on internal device support such as "fuel gauge" circuits [2], or peripheral circuit modules [8]. Devices such as an oscilloscope provide high sampling rates and as such can acquire and display power dissipation data in real-time. However, they are external devices and may not probe *all internal* subsystem paths. Thus, they are impractical for use in deployed systems that need compact, low-power, real-time energy consumption information. A fuel gauge [2] or an integrated peripheral solution [8] is sufficiently low-power to be included in actual field applications; however, those devices cannot meet real-time constraints as they are limited either by their internal sampling rate or by the speed of the communications bus.

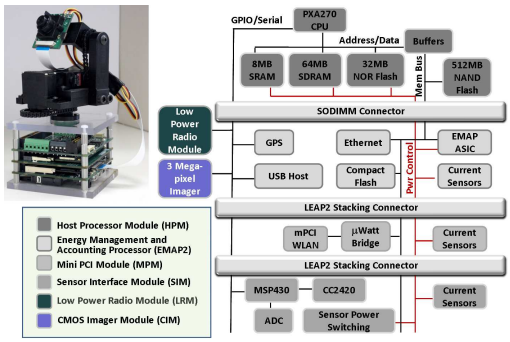

Figure 1. The LEAP2 hardware architecture.

Power dissipation in previously developed systems is typically measured at the power supply or between the power supply connector and the node itself [16, 5, 2, 8]. This enables *system-wide* power dissipation and energy consumption information. However, for the purposes of energy optimization, it is important to know the contribution of *each subsystem*—e.g CPU, memory or storage—to the total system energy consumption. With only single-channel, combined energy information, resolving energy contributions associated with each subsystem may not be feasible in general due to uncertainties resulting from subsystem interdependence and resource contention. A typical approach [16] is to use microbenchmarks that exercise individual subsystems. By carefully reconstructing the event timeline, the system energy consumption can be attributed to the subsystem under investigation. This approach is therefore not real-time as post-facto processing is required. In addition component power dissipation is assumed to be *constant* throughout the duration of an operation [9]. This approximation may be adequate for very low-power components or very simple architectures. However, for high-performance WSN nodes where embedded processors may support dynamic voltage and frequency scaling, power draw depends on several factors such as CPU utilization, operating system support for on-demand frequency scaling, I/O intensity and others; therefore constant power dissipation cannot be safely assumed. Finally, we note that even though per-subsystem resolution could be achieved with an oscilloscope and a peripheral probe, in reality this solution would not be practical, in terms of scalability, ease of deployment and energy consumption for actual field deployments where energy dissipation measurements are most important.

2.1 LEAP2 Platform Overview

The second generation Low power, Energy Aware Processing (LEAP2) platform architecture overcomes these

limitations and enables Energy Endoscope capability, providing an integrated, real-time detailed energy monitoring, capable of resolving energy usage at the level of each processing task and platform hardware component. This LEAP2 platform, shown in Figure 1 bears a similarity to LEAP and other previous energy aware architectures [13, 10, 15] in that a high performance host processor and low power preprocessor are present. However, LEAP2 includes both a new Energy Management and Accounting Processor (EMAP2) ASIC device and new energy aware operating system kernel components. In contrast to previous platforms including LEAP, LEAP2 with the EMAP2 ASIC provides a qualitative and unprecedented advance in high accuracy, low overhead energy measurement of platform computing, storage, sensing, and communications devices at granularity levels previously unachievable.

At the foundation of the LEAP2 platform lies its energy management and accounting capabilities. On LEAP2 this feature is integrated into a dedicated ASIC, implemented in a micro-power antifuse-based field programmable gate array (FPGA). The FPGA has very low quiescent current—compared to SRAM-based FPGAs—of approximately $250uA$. The EMAP2 ASIC, shown in Figure 2 performs continuous real-time energy monitoring, sophisticated power scheduling, and device resource multiplexing across the entire LEAP2 platform while requiring less than $6mW$. Through the EMAP2 ASIC, LEAP2 peripherals may be scheduled for use only when needed and detailed energy information is gathered during their operation.

Energy usage information for individual platform subsystems including computational resources such as the PXA270 microprocessor, memory subsystems such as the SDRAM and SRAM, storage subsystems such as NOR flash and NAND flash, peripheral subsystems such as the Ethernet, 802.11, USB, Imaging, Compact Flash, and external sensors modules is available at millisecond accuracies. In addition, the EMAP2 ASIC energy data and scheduling controls are available to the host processor through a high bandwidth memory bus interface thereby minimizing measurement overhead issues. This enables the host processor to obtain energy usage information across a wide range of devices at millisecond intervals and with a minimal overhead. These features provide LEAP2 with a unique platform monitoring and control capability that can enable significant energy consumption optimizations. The following section provides details on the EMAP2 ASICs three critical functions on the LEAP2 platform: resource multiplexing, power scheduling, and energy accounting.

2.2 The EMAP2 ASIC

LEAP2 is an advance over previous architectures [13, 15, 11, 10] through inclusion of a Resource Multiplexor that enables an expanded set of peripheral and sensor interfaces

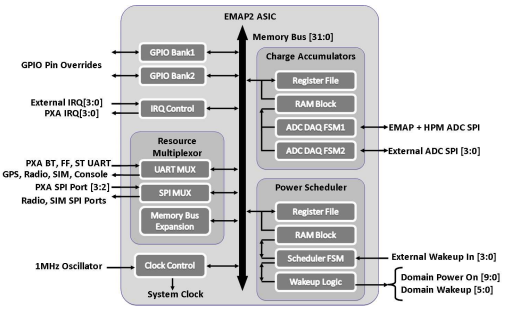

Figure 2. The EMAP2 ASIC.

and dynamic scheduling and energy measurement of power domains that may be added to or removed from the platform through an expandable energy management bus integrated into the LEAP2 stacking connectors.

Due to the large number of peripheral devices integrated into the LEAP2 platform and also expandable on the interboard stacking connector, a Resource Multiplexor was necessary. For point-to-point and point-to-multipoint buses such as UARTs, SPI ports and for traditional parallel memory buses this resource sharing feature is critical since the host processor is often insufficiently equipped with enough hardware resources to allow dedicated one-to-one connections. In the EMAP2 ASIC, serial bus multiplexor ports are implemented as a partially connected, switched network. Device ports are redundantly assigned to a subset of the host processor's available ports. This increases the probability of successfully mapping all active device ports to the host processor. A fully connected network was implemented, but found to require a substantial additional hardware investment over the partially connected implementation. In addition to the resource multiplexor, the EMAP2 ASIC provides a dynamic power scheduler capability (clocked by a configurable input signal driven from a micro-power silicon oscillator requiring only $70uA$ at $3.3V$), providing configurable scheduler resolution from $1us$ to $10ms$ based upon the configured input clock rate.

The EMAP2 ASIC design addresses the challenges arising from the competing requirements for microsecond-resolution sampling and large data acquisition storage rate, low energy operation, and low processor computing overhead. The EMAP2 energy accounting module consists of a charge accumulation finite state machine (FSM), two ADC data acquisition FSMs, and a large RAM block containing the charge accumulation data. The EMAP2 may address up to six Texas Instruments TLV2548 8-channel ADCs providing a total of 48 charge accumulation channels. Eight channels are allocated to the EMAP2 module's peripherals such as Ethernet, USB, quick capture camera, and compact flash. Eight additional channels are allocated for the host processor module (HPM) PXA270 core, SDRAM, NAND

and NOR flash, and SRAM. The EMAP2 and HPM accumulation channels are provided by the first of the two data acquisition modules and interface the ADCs over SPI port1. The remaining 32 channels are controlled via the second data acquisition module which can interface up to four additional ADCs over SPI port2. The parallel SPI acquisition channels reduce acquisition latency and required SPI bus clock frequency. The host processor interfaces with the energy accounting module through a set of configuration registers and the charge accumulation data is read directly from the RAM block. Host reads from the charge accumulation RAM is arbitrated by the energy accounting module with priority given to the host processor reads. Write from the ADC data acquisition FSMs may be delayed by the memory arbiter during host processor access. The current sensor sample period is configurable through a 24-bit configuration register providing ADC sample rates from 25 KHz to less than 0.25 Hz. All SPI bus data samples transfers are pipelined such that commands for sequential channels conversions are interleaved with the readback of the current channel data to minimize latency. Readback from the last active channel on a given ADC causes the ADC powerdown command to be issued, reducing ADC idle power.

The EMAP2 architecture is also designed to reduce processor overhead associated with energy accounting. The host processor reads charge accumulation and accumulation count information directly from the accumulation RAM block in 32-bit read cycles. Since the EMAP2 is able to provide accumulation RAM block access in typically less than $40ns$, data readback is extremely low overhead. The charge accumulation data is aliased to two separate host processor memory address windows providing a rapid, incremental charge reading capability necessary for low overhead energy accounting. When all of the 48 EMAP2 channels are enabled and accumulating, host processor readback of accumulation data and count will require only 192 bus cycles (taking less than $8us$) assuming no bus arbitration loss. Even when using $1ms$ readback periods, the host processors memory access overhead is then less than 0.8%.

3 Energy Measurement Software Tools

Using the capabilities of the EMAP2 ASIC, we developed two energy measurement software tools: *Etop*, a user-space energy consumption observation tool and *endoscope*, a low-overhead kernel-space energy measurement tool.

3.1 Etop

Etop is a user-space tool that enables rapid observation of energy consumption, when running an arbitrary set of processes. Using *etop*, an application developer can quickly ascertain the energy consumption of a particular operation

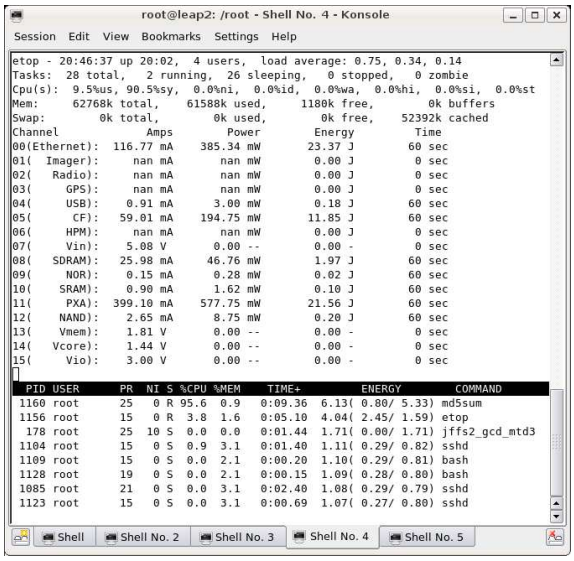

Figure 3. Etop screenshot displaying per-subsystem power and energy information as well as per-process energy consumption.

(e.g. file copy, network transfer or sensor acquisition) on all the subsystems observable by the EMAP2 ASIC.

Etop is based on the well-known UNIX program *top*. *Etop* adds two additional capabilities: per-subsystem current, power and energy information and per-process energy accounting. Figure 3 presents a screenshot of *etop*, with the top part displaying per-subsystem information and the bottom part displaying per-process energy consumption.

Etop's per-subsystem information is directly linked to the output of the EMAP2. On every refresh cycle (a user-controlled parameter that *top* provides, with a default of 3 sec), *etop* presents current, power and energy consumption information of individual subsystems. It also provides voltage information for the four voltage-only channels. Current and power values are by default averaged over the refresh period; maximum values can also be displayed.

To provide *etop* with per-process energy accounting, we introduced a modification to the Linux OS scheduler so as to record energy consumption information, in a manner similar to ECOsystem [17]. Specifically, we augmented the *task_struct* structure (that contains information about a specific process) with two energy containers: one for user-level information and one for kernel-level information. On every scheduler tick, Linux determines whether the current process time slice executed in user or kernel mode and charges an appropriate container accordingly. This information is used by Linux (and *top*) to determine the fraction of CPU time spent in kernel (system) mode and in user mode. We read the EMAP2 charge values from all channels on every scheduler tick and charge the corresponding user- or kernel-

level process energy container. As a result, we can disambiguate between energy consumption that occurred when a process was performing work in support of a user mode task and when the system was performing work during the process time slice in support of kernel mode tasks. Both these values are exported using the */proc* interface, in a manner similar to */proc/<pid>/stat*. *Etop* then reads those values and displays them in standard *top*-like column format.

This development of *etop* provides both an immediately valuable and also new profiling capability. However, it is important to note that additional opportunities for extensions lie ahead. Specifically, *etop* attributes energy consumption in a *synchronous* manner. In the future, asynchronous operations, i.e. operations that do not necessarily occur when the process that caused them is running may be included. Typical examples of those operations include networking and disk I/O. On the other hand, one can assume that when a process is running, it effectively has control of the CPU and memory. As a result, when calculating the energy consumption of a process, we only factor in the PXA (CPU) channel and the SDRAM (memory) channel. An important extension of *etop* addressing the problem of accounting for asynchronous operations will be to associate the initiating process identifier with a particular operation (i.e. disk I/O or networking I/O) and then retroactively charge the appropriate process. We thus plan to develop this extended per-process energy accounting in the future.

3.2 Endoscope

Since *etop* displays system energy performance in real-time it is not optimized for in-depth detailed measurements. *Etop* is a userspace process with non-trivial memory and CPU usage (especially at refresh rates higher than 1 Hz) and as such can interfere with the measurements themselves.

To provide high-rate sampling in software with a very low energy overhead, we implemented the *endoscope* kernel module. *Endoscope* can read the EMAP2 registers at frequencies as high as the default Linux scheduler tick resolution (100 Hz in our system) and with very low CPU overhead, as no expensive user-kernel boundary crossings are required. The results are then stored in a circular buffer in kernel memory—a very fast and low-overhead operation. We utilize a standard */proc* interface for userspace data display and control purposes. To avoid frequent periodic polling of the */proc* interface from userspace, the circular buffer has sufficient memory to store several minutes' worth of *real-time* data—up to 2 minutes of continuous sampling of 16 channels at 100Hz, or 200 minutes at 1Hz. To avoid buffer overrun, an application running in userspace only needs to read from the */proc* interface at an interval that is slightly smaller than the buffer's capacity at a specific sampling rate.

We used *endoscope* extensively to collect our experimental dataset. *Endoscope*, unlike *etop*, measures energy consumption of entire subsystems rather than that associated with individual processes. As a result, when conducting our experiments, care is taken to ensure that *endoscope* measures energy solely attributed to a single application. For this development of *endoscope*, this procedure is effective for WSN systems that typically support one dominant application, or a set of applications that can be considered dominant. At the same time, as for *etop*, extensions to *endoscope* will enable increasingly complex WSN computing application support. Table 1 summarizes the capabilities and differences of our software tools.

	Etop	**Endoscope**
Execution space	User	Kernel
Kernel modifications	Scheduler for per-process data	Kernel Module
Data display interface	*top*-like	/proc
CPU usage	$1-25\%$	Negligible
Sample storage	No	Yes
Real-time data collection	Yes	Yes
Real-time data display	Yes	No
Per-subsystem resolution	Yes	Yes
Per-process resolution	Yes	No

Table 1. Qualitative comparison of *Etop* and *Endoscope*

4 Single-node Energy Profiling

In this section we focus on profiling the energy consumption of a single LEAP2 node. In particular, our experiments aim to showcase the energy consumption of critical subsystems, such as CPU, memory, storage and networking. We also outline potential optimizations that can yield better energy efficiency.

Throughout our experiments, we utilized three metrics: instantaneous power dissipation over time, total energy consumption and the energy-latency product. *Instantaneous power dissipation over time* captures variations in power dissipation and is especially important in cases where power dissipation is not uniformly distributed in time. We calculate power dissipation by converting charge values from the EMAP2 into instantaneous current dissipation and then multiplying with the corresponding voltage channel measurement. The *energy* metric is useful when optimizing energy consumption for the entire system or for a specific set of subsystems. We calculate energy consumption by integrating instantaneous power values over a specific time period. The *energy-latency product* [6] is useful when optimizing for energy consumption as well as *latency*. Several WSN applications tend to be latency-tolerant; therefore trading off increased latency for lower energy consumption is acceptable and in that case the energy consumption met-

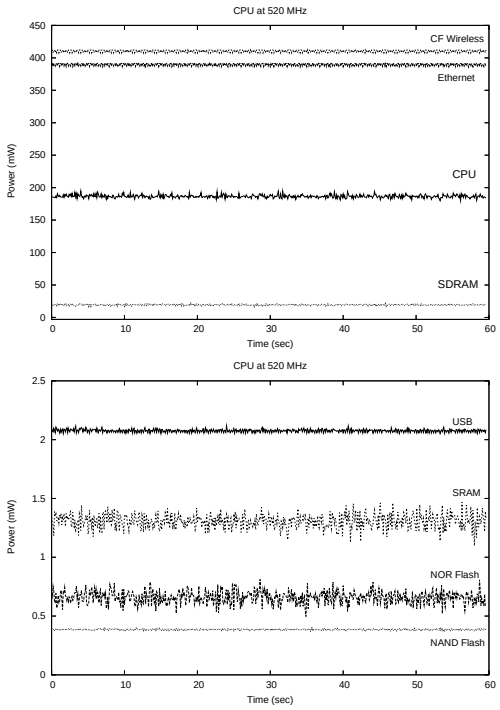

Figure 4. Power dissipation over time for all active subsystems: a) CPU, SDRAM, Ethernet and CompactFlash Wireless and b) USB, SRAM, NOR flash and NAND flash, with the CPU operating at 520 MHz.

ric is more useful. However in cases where specific latency bounds need to be met, or when optimizing for both energy consumption and performance the energy-latency product is the more appropriate metric. Ideally, the desired operating point of a system or application is the one that minimizes *both* energy consumption and the energy-latency product.

Our experimental data was collected in real-time, by using *endoscope* with a sampling rate of 10Hz. Each experiment was repeated 5 times. We report averages when presenting energy (Joules) and energy-latency product (Joules*sec) values and use instantaneous values from a single experiment when presenting power values.

4.1 System-wide Energy Profile

We initially focus on the energy consumption of the entire system, in order to establish a baseline for the energy usage of each subsystem, for different CPU speeds. Therefore, in this set of experiments our LEAP2 node only runs the Linux OS, without any additional applications (aside from the standard Linux daemons, and sshd), or any network traffic. As a result, we consider this to be our "idle" case for all individual subsystems. We ran 5 experiments lasting 60 seconds each at every CPU speed. We used the *userspace* Linux CPU governor to manually change the CPU speed before each experiment.

Figure 4 shows the power dissipation over time for eight subsystems with the CPU operating at 520 MHz. The network interfaces dominate the system power dissipation, with each interface requiring approximately $400mW$ of power. More importantly, both interfaces are *idle*, i.e. they are not transmitting or receiving any substantial amount of data (apart from MAC- and routing-level control packets). The current version of LEAP2 only supports CompactFlash (CF) wireless cards. The CF interface itself draws about half of this power ($200mW$), even when the wireless card is not turned on. Future versions of LEAP2 will include a miniPCI which is expected to provide significant savings in power dissipation, in addition to being able to support 801.11g speeds. Furthermore, reductions in power dissipation could be achieved by taking advantage of the low-power modes that miniPCI Linux drivers support.

The computational and memory subsystems (CPU, SDRAM, SRAM) use considerably less power when idling compared to the networking subsystems. Nevertheless, the power dissipation of the CPU is considerable, especially considering the fact that the CPU is not performing any actual work. Running the CPU at 104 MHz results in a significantly reduced power dissipation of $25mW$, slightly higher than that of SDRAM. At 520 MHz CPU power dissipation raises to $180mW$, more than an order of magnitude higher than SDRAM and comparable to the network interfaces. Comparing dynamic RAM power dissipation with static RAM power dissipation, we notice that SRAM requires an order-of-magnitude less power to operate—less than $1.5mW$ of power when idling. This very low power dissipation is a well-documented property of static RAM which we were able to experimentally verify using *endoscope*. Finally, there is a very considerable difference in power dissipation between the storage subsystems and the computation and networking subsystems. This is in contrast to mote-class devices, where the power dissipations of the storage subsystem and the MCU are comparable [12]. Section 4.3 further explores the storage subsystems.

Table 2 shows the percentage of the total energy consumption attributed to each subsystem, for the six different CPU speeds supported by the PXA270. Even though the networking subsystems dominate the energy consumption under any CPU speed, the power dissipation of the CPU itself becomes considerable in higher speeds.

Figure 4 and Table 2 indicate that considerable energy savings of up to 80% can be achieved by turning off the networking subsystems. In a deployed system, the Ethernet interface that is only used for debugging can be safely assumed to be shut down. The wireless interface however is required for node communication; therefore one needs to apply more sophisticated energy savings techniques such

Subsystem Energy (%)	104 MHz	208 MHz	312 MHz	416 MHz	520 MHz	624 MHz
Ethernet	45.43	42.04	41.25	40.03	38.62	37.21
CF Wireless	47.83	44.26	43.42	42.16	40.68	39.20
PXA	4.10	11.24	12.95	15.51	18.48	21.44
SDRAM	2.28	2.12	2.05	1.98	1.91	1.85
SRAM	0.15	0.14	0.13	0.13	0.12	0.12
USB	0.07	0.06	0.06	0.06	0.05	0.05
NOR	0.07	0.07	0.07	0.06	0.06	0.06
NAND	0.04	0.04	0.04	0.03	0.03	0.03

Table 2. Percentage of subsystem energy consumption for six CPU speeds, when the system is idling.

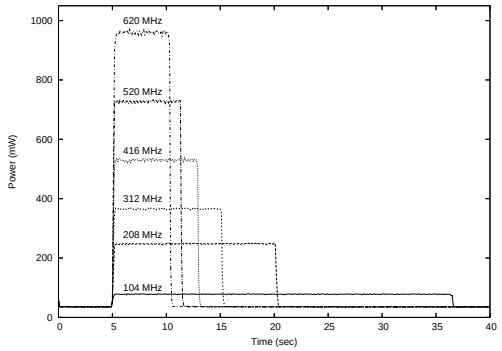

Figure 5. CPU power dissipation over time, when running *dcache_test*, for six CPU speeds.

as duty-cycling, scheduling or dual-radio networking. Further energy savings can be obtained by running the CPU at its lowest speed setting. In later sections (4.3, 5) we investigate whether this property also applies when the CPU is operating under load and together with other subsystems.

4.2 CPU Subsystem Energy Profile

The CPU is arguably the most important component of a platform, as every other subsystem depends on it, either directly or indirectly. It also is often the largest power consumer; therefore it needs to be energy efficient. We therefore focus our attention on investigating the power dissipation and energy consumption of the PXA270 CPU used in the LEAP2 node. Prior work [13, 2, 9] has demonstrated that a more capable CPU (such as the PXA255 or PXA270), while incurring significantly higher peak power dissipation than an MCU (e.g. MSP430) is in fact more energy efficient *while under load*, due to its architectural advantages—higher clock speed, L1 caches, hardware MMU, multi-stage pipeline etc. Therefore, our goal for this set of experiments was to verify whether a CPU running at a higher speed is still more energy efficient than *the same* CPU running at a lower speed.

Since we wanted to measure only the CPU power dissipation, we required a test application that would not considerably utilize any other system resources, such as storage or even RAM. We therefore used a simple test program called *dcache_test*, that computes the sum of an array of integers. The array's total memory size is 4Kbytes so that it fits into the processor's data cache. The computation is continuously repeated, up to a given number of consecutive runs. Therefore, the CPU utilization during our test program's execution is very close to 100%, while all other subsystems including SDRAM are not utilized.

We utilized the *ondemand* Linux CPU governor to dynamically change the CPU speed. This particular CPU governor automatically adjusts the CPU speed based on the observed CPU utilization. Under default settings, this governor increases the CPU speed to the maximum allowed value when the idle time is less than 20%. It then tries to find the lowest allowed CPU speed that can sustain the load while keeping the idle time over 30%. For each set of experiments, we gradually increased the maximum allowed CPU speed, from 104 MHz all the way to 624 MHz.

Figure 5 presents the CPU power dissipation over time when running *dcache_test*, for the six different PXA CPU speeds. To indicate the difference in power dissipation between the idle CPU state and the loaded CPU state, we started our test program five seconds after *endoscope*. As can be seen in Figure 5, the CPU power dissipation raises considerably at $t = 5sec$. Moreover, there is almost an order-of-magnitude difference between the power dissipation of the lowest CPU speed setting ($80mW$) and the highest one ($970mW$). This difference is even more apparent when comparing the idle CPU power dissipation at 104 MHz with the loaded CPU power dissipation at 624 MHz. Figure 5 also indicates that there is a *non-linear* increase in the CPU power dissipation as the CPU speed increases. This is due to dynamic voltage and frequency scaling (DVFS) where changes in CPU frequency are accompanied with changes in core voltage. Since power is the product of current and voltage, the increase in power dissipation for different frequencies is expected to be non-linear. As *endoscope* and EMAP2 can sample both the CPU current draw and the Vcore voltage at the same time, CPU power dissipation can be accurately estimated for each sample, thus capturing the non-linear DVFS effect.

Even though a lower CPU speed results in a considerable increase in completion time, the actual CPU *energy* consumption is lower, as Figure 6 shows. The increase in completion time at lower speeds is not significant enough to offset the drastic decrease in peak power draw, We therefore conclude that unlike the case where a faster 32-bit CPU is more energy efficient than an MCU despite the higher peak power dissipation [13], when comparing different speeds of

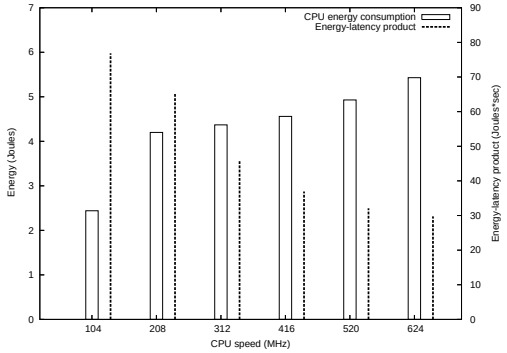

Figure 6. CPU energy (left) and energy-latency product (right) when running *dcache_test*, for six CPU speeds.

the same DVFS-enabled CPU, the lowest CPU speed requires the least amount of energy for the same workload. The above result however does not consider *latency*: even though the lowest CPU speed draws the least amount of energy, it requires the most amount of time to complete. If instead we use the energy-latency product, the highest CPU speed becomes the most energy-latency efficient operating point.

Our results above contain information about a *single subsystem*; the CPU. A valid question then is whether results still hold when taking into account *all* subsystems that have a considerable energy consumption, such as network interfaces and SDRAM. We can assume that for applications that perform computation without need for communication (e.g. signal processing), networking subsystems can be in a powerdown state. Therefore, a longer execution time would not affect the energy consumption of those subsystems. On the other hand, even though SDRAM may or may not be used in a CPU-intensive application, it cannot be turned off at runtime, without significantly modifying the operating system itself. SDRAM peak power draw is at least an order of magnitude lower than that of the CPU; consequently, even when taking SDRAM into account, the lowest CPU speed would still be the most energy efficient, even though the difference between it and the highest CPU speed would be slightly smaller. However, as will be shown in following sections, in applications that utilize several subsystems the lowest CPU speed is no longer the most energy efficient.

4.3 Storage Subsystem Energy Profile

Our next set of experiments focuses on the storage subsystem—an important component for both WSN nodes and embedded systems in general. Our goal for this set of experiments is two-fold. First, we want to investigate the differences in power dissipation between the two types of non-volatile storage included on LEAP2: NOR flash and

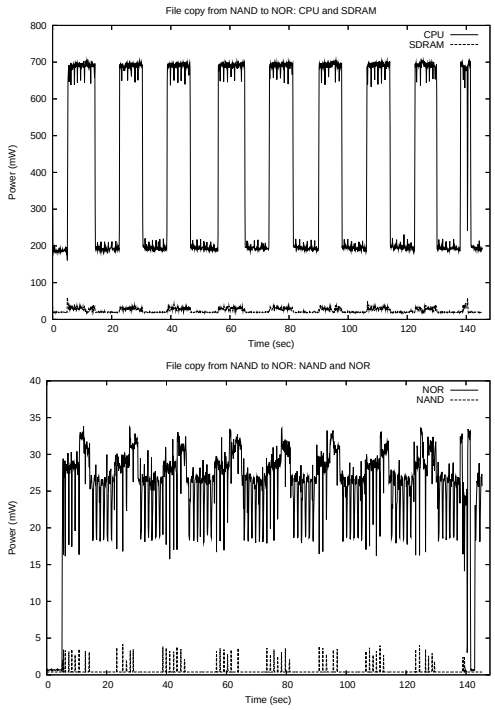

Figure 7. CPU, SDRAM, NAND and NOR power draw over time when copying a 7Mb file from NAND to NOR.

NAND flash. Those two flash types differ considerably in their operational characteristics, so we expect significant differences in their power profiles as well. Second, we want to investigate the interdependencies between the storage subsystem and the computation subsystem (CPU and SDRAM). Our experimental setup consisted of copying a 7MB file from one storage subsystem to the other (i.e. from NAND to NOR or from NOR to NAND), with the CPU operating at a constant speed of 520 MHz. After each experiment we manually flushed the Linux VM file caches so as to avoid consecutive experiments completing faster due to caching. The subsystems measured with *endoscope* for this set of experiments were CPU, SDRAM, NAND and NOR.

Figure 7 shows the power dissipation of the four subsystems over time for a file copy from NAND to NOR, while Figure 8 depicts the results for the file copy from NOR to NAND. These figures indicate very significant differences between the two flashes. First, copying to NOR takes approximately 140 seconds, while copying to NAND only takes approximately 6 seconds. Second, NOR peak power when writing and erasing is more than twice that of the equivalent NAND peak power. On the other hand, NAND reads are more expensive, power-wise than NOR reads: in fact, NOR reads don't cause any discernable difference above NOR's idle power dissipation. These results are in accordance with the characteristics of NOR and

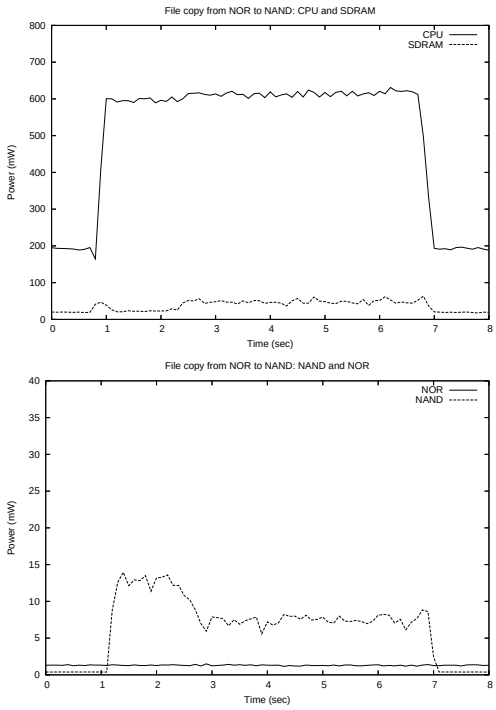

Figure 8. CPU, SDRAM, NAND and NOR power draw over time when copying a 7Mb file from NOR to NAND.

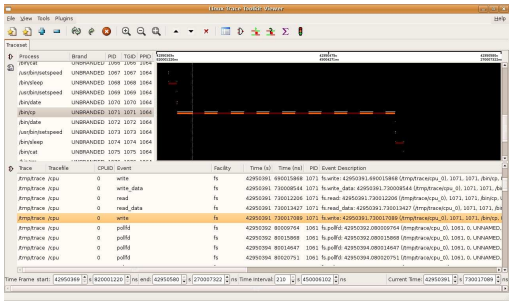

Figure 9. LTT screenshot highlighting the end of a write and the beginning of an erase operation in NOR flash.

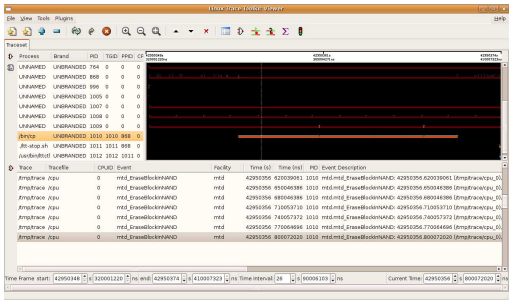

Figure 10. LTT screenshot highlighting a sequence of block erases in NAND.

NAND flashes. NAND flashes are optimized for sequential block access and have much faster write and erase speeds than NOR flashes, therefore the write operations on NAND execute much faster and with considerably less power than on NOR. Conversely, NOR flashes support random-access reads (unlike NAND flashes), therefore read operations on NORs execute faster and require significantly less power.

NOR energy profile: The most important difference however exists in the power variations over time for NOR versus NAND. For NOR, CPU power dissipation over time has a pulse-like shape, while for NAND, CPU power dissipation is almost constant. The differences are due to the way that each flash type (and its associated drivers) handles write and erase operations. The NOR driver (Memory Technology Device—MTD) and filesystem (Journaling Flash File System—JFFS2) implement *erase suspend on write*. Therefore, write and erase operations in NOR flash are *serialized*: during a write operation, an erase operation cannot proceed, and vice-versa. This behavior explains the pulse-like shape of Figure 7. Write operations to NOR happen when CPU power peaks: this is corroborated by the fact that peaks in the SDRAM and NAND subsystems are correlated with peaks in the CPU subsystem, indicating sequences of reads and writes. When CPU power drops—indicating that the CPU is idling—so does NAND and SDRAM power dissipation: however, NOR power dissipation remains high, since during that time, the NOR flash is erasing blocks that will be needed for the next write operation. This sequence of events was verified using the Linux Trace Toolkit [3] kernel tracer. Figure 9 shows a sequence of *read()* (from NAND) and *write()* (to NOR) system calls. The last *write()* system call does not complete however; instead, a sequence of *poll()* system calls on the write file descriptor follows, indicating that the */bin/cp* process is blocked on a *write()*.

NAND energy profile: When writing from NOR to NAND, we observe that during the first 1.3 seconds of the file copy, NAND power is at its peak. At the same time, CPU and SDRAM have not reached their peak power values; this is more evident in SDRAM. After this initial time period, CPU and SDRAM power dissipation reach peak values while NAND power dissipation decreases.

Using LTT (Figure 10) we observed that during this initial time period, the YAFFS2 filesystem (used for NAND) executes a series of *erase* operations. However, those erase operations do not continue throughout the entire file copy duration. During a file copy to NAND, YAFFS2 determines how many blocks (if any) need to be erased and proceeds to erase those blocks *in parallel* with normal write operations. Since the file size is considerably smaller than the total NAND capacity, and considering that YAFFS2 implements wear-leveling in software, the erase blocks are significantly

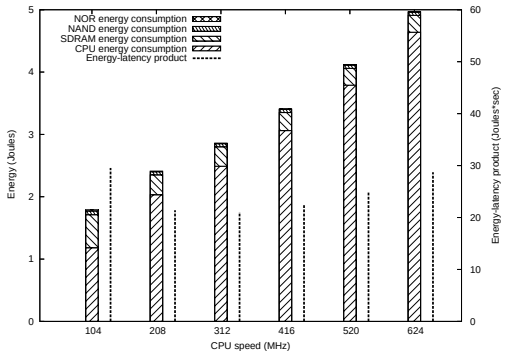

Figure 11. Average CPU, SDRAM, NOR and NAND energy (left) and energy-latency product (right) when copying a 7MB file from NOR to NAND, for six CPU speeds.

fewer than the total blocks that need to be written. Nevertheless, NAND power dissipation peaks as NAND blocks are being erased and written in very short succession. After the erase operations cease, writes can proceed at full speed, therefore the CPU and SDRAM subsystems are more heavily utilized and their power dissipation peaks.

SDRAM contribution: Regardless of whether we write to NOR or NAND, we notice that SDRAM power dissipation raises above idle during write operations. Even though both YAFFS2 and JFFS2 keep several data structures in SDRAM, their memory footprint is not large enough to explain the significant rise in SDRAM power dissipation. Instead, SDRAM is being used extensively because the Linux Virtual File System (VFS) implements *memory caching* in order to improve read and write access to files. Files are first written to SDRAM—or more accurately to the page cache—before they are committed to storage.

Effects of CPU speed: Caching is the first reason why CPU power dissipation is high during writes, as the CPU is also involved in this operation. The second one is *software error correction*. Neither the NOR or NAND modules in LEAP2 include hardware-supported error correction checks (ECC); therefore the filesystem needs to implement ECC in software. ECC is critical to the correct operation of NAND flashes, as those flashes always contain bad blocks.[1]

The aforementioned reasons indicate a significant dependency between CPU speed and write performance. Figure 11 shows a per-component energy cost as well as the energy-latency product for the file copy from NOR to NAND, for six CPU speeds. Again, the lowest CPU speed results in the minimal energy consumption. The energy-latency product however is minimized when the CPU is operating at 312 MHz, with 208 MHz also being very close to the minimum. In Section 4.2, the test application was exclusively CPU-bound. Operating at the highest CPU speed resulted in a factor of five reduction in completion time, while energy consumption was increased by less than a factor of three; hence the energy-latency product was minimized at the highest CPU speed. The file-copy operation however is not exclusively CPU-bound as it involves I/O. As a result, in higher speeds, the CPU is not fully utilized. The peak power draw of the CPU operating at 520 MHz under 100% load is around $735mW$ (Figure 5), while the peak power dissipation in Figure 8 is around $600mW$—hence the CPU load is not at 100%. Therefore, lower CPU speeds yield similar performance benefits while requiring less energy due to lower idle and peak power dissipation. Deviating from the minimum energy-latency product thus leads to "diminishing returns", in terms of energy at speeds lower than the optimal and in latency in speeds higher than the optimal.

One final observation regarding the effect of CPU speed is its contribution to energy consumption versus that of the storage device. Figure 11 indicates that the CPU contribution ranges from 65% to 93% while the NAND contribution ranges from 3.3% to 1%. This result is in sharp contrast with mote-class systems, where the CPU energy consumption and the storage (NAND or NOR) energy consumption are within the same order of magnitude [12, 4].

5 Multi-Node Energy Profiling: Node-to-Node File Transfer

In Section 4, we focused on profiling the energy consumption of various subsystems on a single LEAP2 node. In this section we aim to profile the energy consumption of an application that exercises multiple subsystems in multiple nodes. A *network file transfer* application satisfies our requirements, as it involves multiple nodes by definition and it also exercises some of the most important subsystems on a node—CPU, SDRAM, networking and storage.

Our experimental setup consisted of two LEAP2 nodes—one sender and one receiver—with their wireless interfaces set to ad-hoc mode. In each experiment, the receiver would initiate a 20 MB file transfer, using TCP. The location of the file in the sender as well as the file's destination on the receiver was the NAND flash. *Endoscope* was used on both the sender and the receiver to collect experimental data. In order to synchronize the timelines of the sender and the receiver for the purposes of data processing, both nodes were running NTP.

From Section 4.3 we saw that the CPU speed has a significant impact in energy consumption when writing to stable storage. Therefore, we expect the receiver's energy consumption to depend on the CPU speed. Furthermore, on the sender side, a lower CPU speed can potentially result in increased latency, as the CPU might not be able to fill the network card's transmit buffers fast enough. Results from

[1] Manufacturers ship NAND flashes with some bad blocks to keep their yield high and the cost low.

	Energy Consumption (Joules)			
	Min		Max	
	Value	CPU Speed (Snd/Rcv)	Value	CPU Speed (Snd/Rcv)
Sender	26.12	208 524	58.87	104 624
Receiver	30.67	208 208	64.21	104 624
Combined	57.35	208 208	95.95	104 624

Table 3. Minimum and maximum sender, receiver and combined energy consumption and sender - receiver CPU speed during a 20 MB file transfer over 802.11b.

	Min E.L.P (kJ*sec)	
	Value	CPU Speed (Snd/Rcv)
Sender	1.15	208 624
Receiver	1.43	208 208
Combined	2.68	208 208

Table 4. Minimum sender, receiver and combined energy - latency product and sender - receiver CPU speed for a 20 MB file transfer over 802.11b.

Section 4.2 indicated that a lower CPU speed can be more energy efficient, even though latency is increased. However, in the network file transfer case, the considerable energy consumption of the wireless interface needs to be taken into account.. A reduction in latency could result in further energy savings as the networking subsystem could be placed into a low-power state (or even powered down) after the file has been transferred. In order to verify our expectations we therefore conducted experiments with various CPU speeds, for both the sender and the receiver.

During the course of our experiments, all available frequency values were explored other than at 416 MHz. It was discovered that the pcmcia driver for the wireless interface would not function correctly at this frequency.

5.1 Energy consumption

Table 3 shows the minimum and maximum energy consumption for the sender, the receiver and their combination. As expected, the energy consumption of the sender is always less than that of the receiver, since the receiver needs to write the file in NAND. The sender's minimum energy consumption occurs at different speed combinations (sender at 208 MHz, receiver at 520 MHz) than that of the receiver (sender at 208 MHz, receiver at 208 MHz). The minimum combined energy consumption however occurs when both the sender and the receiver operate at 208 MHz; the same speed combination that results in the minimum receiver energy consumption. In that operating point, the sender energy consumption increases by 2.1%. However, if the network were to operate at the sender's optimal point, the receiver's energy consumption would rise by 45.7% and the combined energy consumption would increase by 23.4%.

When the sender operates at 104 MHz and the receiver operates at 624 MHz, the combined energy consumption is 67.3% higher than the minimum. When the sender operates at its lowest CPU speed, it cannot fill its transmit buffers fast enough, therefore the network bandwidth is not fully utilized, leading to an increase in latency. As a result, the wireless interface on both the sender and the receiver needs to stay on for longer, therefore increasing the total energy consumption. At the same time, the receiver operates at its highest speed but, since the sender cannot transmit data fast enough, the receiver's CPU is idling (with a high idle power dissipation) for a considerable amount of time. When the sender is operating at 624 MHz and therefore able to fully utilize the available network bandwidth, the combined energy consumption is 60.8% higher than the minimum. Similar to the results of Section 4.3, since the network transfer operation is not CPU-bound for all but the lowest sender/receiver CPU speeds, the CPU is not fully utilized past a certain point, leading to increased energy consumption. Therefore, selecting appropriate CPU speeds for *both* the sender and the receiver can drastically affect the energy consumption of a network transfer.

Table 4 shows the minimum energy-latency product for the sender, receiver and their combination. The combined minimum energy-latency product occurs at the same CPU speed combination as the minimum combined energy consumption, unlike Sections 4.2 and 4.3. This is a highly desirable result, as it indicates that when both the sender and the receiver operate at 208 MHz, the energy consumption is not only minimized but is also optimally utilized to produce the best energy-latency tradeoff.

5.2 Profiling Network Communication with Endoscope

The introduction of the new LEAP2 and EMAP2 architecture and its Energy Endoscope is intended to enable a WSN system designer to accurately profile energy dissipation not previously visible. A direct demonstration of this is described below, for the fundamental example of data transport between two nodes, where large energy reduction is obtained in the optimized system without performance loss.

Figure 12 shows the power dissipation over time of the wireless, CPU, SDRAM and NAND subsystems, for the sender and the receiver, with both nodes' CPUs at their optimal energy consumption operating point of 208 MHz. The wireless interface draws the largest amount of power in both the sender and the receiver. Moreover, the difference in power dissipation of the two interfaces is negligible, as it is around $1mW$. More importantly, however, the

wireless power dissipation over time is *constant*, regardless of whether the interface is transmitting, receiving or idle. This is a well-studied property of 801.11b and in sharp contrast with lower-power radios such as 802.15.4. In terms of energy-efficient system design, this result implies that significant energy gains can be achieved by keeping the wireless interface in powerdown state most of the time and only activate it when necessary [13, 2, 14].

In terms of CPU power dissipation, there are some observable differences between the sender and the receiver. First, the receiver's CPU power dissipation is higher than the sender's and close to the maximum power dissipation ($279mW$) for the 208 MHz frequency, indicating a higher CPU utilization on the receiver. As previously noted, this is due to the fact that the receiver has to store the file in NAND flash which requires a significant amount of CPU time as well as SDRAM usage (Section 4.3).

In the first 7 seconds of the file transfer, the receiver's CPU power reaches its maximum value, while the sender's CPU is at its minimum value (excluding the time when the CPU is idle). At the same time, SDRAM power draw on the receiver is low and NAND power draw is high. This indicates that the YAFFS2 filesystem is executing block erases on NAND during that time, as shown in Section 4.3. During that time, writes cannot proceed at full speed, so the network bandwidth is not fully utilized; therefore the sender's CPU power dissipation is kept fairly low. After the erase operations complete on the receiver, the writes can proceed at full speed, the network can be fully utilized and the sender's CPU and SDRAM power dissipation rises.

SDRAM is used at the receiver for network buffer storage but more importantly, for *write caching*. The combination of those two operations results in considerable SDRAM power dissipation at the receiver—up to $50mW$. NAND power dissipation over time at the receiver is similar to that of Figure 8. At the sender, NAND power draw is negligible.

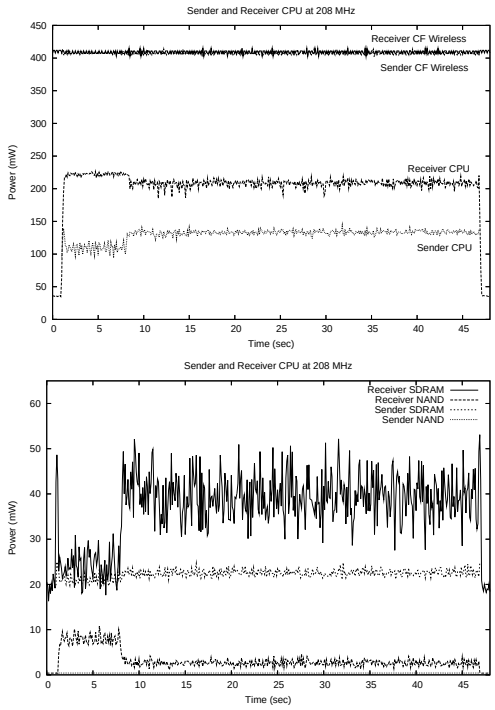

Figure 12. CPU, SDRAM, NAND and wireless power draw over time during a 20 MB file transfer over 802.11b, with the sender and the receiver operating at 208 MHz.

6 Related Work

This section focuses on prior work most closely related to the energy endoscope in terms of hardware and software.

PASTA [15] was one of the first WSN platforms that targeted modularity and utilized high performance components. It included power monitoring capability and device-level power management. MPlatform [10] emphasizes a modular design and includes a custom inter-module bus protocol designed for high efficiency data transfers. The previous LEAP [13] platform utilized a microcontroller architecture for power management functionality. With LEAP2, we augmented the measurement fidelity for detailed per-component energy usage data and lowered measurement overhead allowing for higher sampling rates over the previous LEAP design.

PowerTOSSIM [16] is a power-aware extension to the TOSSIM simulator. It uses an empirically-generated model of hardware behavior to simulate power dissipation on mote-class devices. The model is obtained by instrumenting and profiling a single node during actual operations. An oscilloscope was used to measure power dissipation of the entire mote. Since the oscilloscope does not provide per-subsystem resolution as EMAP2 does, the authors used a set of microbenchmarks that exercised each subsystem independently. Even though subsystem independence can apply to a highly integrated, low-power system such as the mote, our experiments indicate that in a complex 32-bit node such as LEAP2, subsystem independence cannot be assumed.

Jung et al. [9] introduce a model-based design tool that can identify the dominant energy-intensive hardware components over a set of operating patterns. The authors propose several operating states where the system components are operating in different power modes. Measured power values can be used to populate the model parameters. The proposed models assume that power dissipation of subsystems is constant among operations. However, our experimental data suggests that this is not always the case, especially when a DVFS-enabled CPU is present.

SPOT [8] is an integrated add-on board that enables energy accounting on mote platforms. Charge accumulation is

performed via a voltage to frequency converter circuit, similar to a sigma-delta ADC architecture, achieving high resolution output and large dynamic range. The SPOT module also includes a dedicated counter value to provide charge accumulation timing information. Similar to the first generation LEAP, SPOT utilizes the I2C serial bus for the readback channel. We found the I2C bandwidth limitation to hinder low overhead measurement at high sample rates and thus utilized a higher performance memory bus in LEAP2. Additionally, SPOT is a single-channel design, monitoring only the mote platform's input current and cannot easily support per-subsystem power information.

Triage [2] is a tiered hardware and software architecture for microservers. Similar to LEAP2, Triage uses a high-power platform with ample resources and a resource-constrained low-power platform. The low-power platform schedules tasks to optimize the use of the high-power platform. Scheduling is based on hardware-assisted profiling of execution time and energy usage. The energy profiler uses information about the remaining battery energy as well as energy consumption during a particular operation. However, the energy profiler does not provide per-subsystem information or power-tracing capabilities; instead, an external data acquisition board must be used.

Powerscope [5] combines hardware instrumentation and kernel software support to provide per-process energy usage information. Unlike *etop*, powerscope uses an external digital multimeter and a second computer for data collection; in addition, data processing does not happen in real-time. Furthermore, the power-measuring system itself requires significant energy and physical resources to operate, thereby limiting its application in large-scale systems.

ECOsystem [17] incorporates energy as a "first-class resource" in OS scheduling, through the *currentcy* abstraction. Per-process energy information is critical in ECOsystem as processes need to be charged appropriately, to enable energy-based scheduling. ECOsystem uses a combination of battery lifetime sampling (a system-wide energy measurement) and system modelling. Our work is complimentary to ECOsystem since we focus on accurate and detailed real-time energy measurements as opposed to energy-aware operating system abstractions and modifications.

7 Conclusion

In this paper we presented the Energy Endoscope, a new hardware and software architecture that provides unprecedented visibility into the energy consumption of 32-bit Wireless Sensor Nodes. We introduced the LEAP2 platform and its EMAP2 ASIC-based energy accounting unit that allows for low-power, integrated, real-time energy observation of individual platform components. In addition to the introduction of a new hardware platform, an associated software architecture has been developed, producing two software tools, *etop* and *endoscope* for real-time profiling of several important subsystems such as CPU, memory, storage and networking. The development of the new LEAP2 and EMAP2 ASIC and associated Energy Endoscope software systems have successfully met the challenge of providing real-time, high rate sampling at the hardware subsystem level and with very low overhead in terms of operating power, processor and memory utilization.

The Energy Endoscope system is now generally applicable to many WSN platforms. Experiments reported here demonstrate that the Endoscope can reveal the magnitude, temporal dependence and origin of energy dissipation in complex systems, where multiple forms of resource contention are present. As an example, our experiments revealed significant differences in temporal dependence of power dissipation that exist between two different storage technologies, as well as the considerable effect of CPU power dissipation on storage and networking operations. Using Energy Endoscope to profile multiple nodes, this new knowledge and measurement capability was exploited to optimize energy efficiency without reduction in communication performance. By profiling subsystem energy during a network file transfer, we discovered the optimal CPU operating points for both the sender and receiver that can induce more than 60% energy savings. These results have demonstrated that a wide range of important research opportunities appear ahead for optimization of WSN systems.

References

[1] Imote2. `http://www.xbow.com`.

[2] N. Banerjee, J. Sorber, M. D. Corner, S. Rollins, and D. Ganesan. Triage: Balancing Energy Consumption and Quality of Service in a Microserver. In *MobiSys '07*.

[3] M. Desnoyers and M. R. Dagenais. The lttng tracer: A low impact performance and behavior monitor for gnu/linux. In *Proceedings of Linux Symposium*, Ottawa, Canada, 2006.

[4] Y. Diao, D. Ganesan, G. Mathur, and P. J. Shenoy. Rethinking data management for storage-centric sensor networks. In *CIDR*, pages 22–31, 2007.

[5] J. Flinn and M. Satyanarayanan. Powerscope: a tool for profiling the energy usage of mobile applications. In *Second IEEE Workshop on Mobile Computing Systems and Applications*, Feb. 1999.

[6] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessors. In *IEEE Journal of Solid-State Circuits*, volume 31(9), pages 1277–1284, 1996.

[7] Intel. Intel stargate, http://platformx.sourceforge.net, 2002.

[8] X. Jiang, P. Dutta, D. Culler, and I. Stoica. Micro power meter for energy monitoring of wireless sensor networks at scale. In *IPSN '07*.

[9] D. Jung, T. Teixeira, A. Barton-Sweeney, and A. Savvides. Model-based design exploration of wireless sensor node lifetimes. In *EWSN 2007*.

[10] D. Lymberopoulos, N. B. Priyantha, and F. Zhao. mplatform: a reconfigurable architecture and efficient data sharing mechanism for modular sensor nodes. In *IPSN '07*.

[11] D. Lymberopoulos and A. Savvides. Xyz: a motion-enabled, power aware sensor node platform for distributed sensor network applications. In *IPSN '05*.

[12] G. Mathur, P. Desnoyers, D. Ganesan, and P. Shenoy. Ultra-low power data storage for sensor networks. In *IPSN '06*.

[13] D. McIntire, K. H. Hing, B. Yip, A. Singh, W. Wu, and W. Kaiser. The low power energy aware processing (leap) system. In *IPSN '06*.

[14] T. Pering, V. Raghunathan, and R. Want. Exploiting radio hierarchies for power-efficient wireless discovery and connection setup. In *18th Intl. Conf. on VLSI Design*, 2005.

[15] B. Schott, M. Bajura, J. Czarnaski, J. Flidr, T. Tho, and L. Wang. A modular power-aware microsensor with 1000x dynamic power range. In *IPSN '05*.

[16] V. Shnayder, M. Hempstead, B. Chen, G. W. Allen, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. In *SenSys '04*.

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