Characterizing Energy Consumption in a Visual Sensor Network Testbed

Cintia B. Margi, Vladislav Petkov, Katia Obraczka and Roberto Manduchi
{cintia,vladi,katia,manduchi}@soe.ucsc.edu
Computer Engineering Department
University of California Santa Cruz
Santa Cruz, CA 95064

***Abstract*— In this work we characterize the energy consumption of a visual sensor network testbed. Each node in the testbed consists of a "single-board computer", namely Crossbow's Stargate, equipped with a wireless network card and a webcam. We assess energy consumption of activities representative of the target application (e.g., perimeter surveillance) using a benchmark that runs (individual and combinations of) "basic" tasks such as processing, flash memory access, image acquisition, and communication over the network. In our characterization, we consider the various hardware states the system switches through as it executes these benchmarks, e.g., different radio modes (sleep, idle, transmission, reception), and webcam modes (off, on, and acquiring image). We report both steady-state and transient energy consumption behavior obtained by direct measurements of current with a digital multimeter. We validate our measurements against results obtained using the Stargate's on-board energy consumption measuring capabilities.**

I. Introduction

Most wireless sensor networks to date employ nodes with limited power, processing, storage, and communication capabilities. Additionally, they typically include relatively simple sensors (e.g., light, temperature, pressure, magnetometer, etc.). In these deployments, energy consumed by sensing-related tasks is relatively low, which means that the communication subsystem (i.e., the radio) dominates energy consumption.

However, an increasing number of current and upcoming applications can benefit considerably from "media-rich" sensors such as cameras and microphones as they typically provide wider coverage and richer information. Such applications, which often have considerable scientific, social, and strategic relevance, typically require monitoring events in wide areas over long periods of time. Examples include: a geological survey documenting the time, duration and height of hot water eruption from all the geyser at Yellowstone National Park over a period of several months; estimating the number and roaming patterns of mountain lions in the Santa Cruz mountains in order to determine guidelines for human/predator co-existence; keeping a large airport under continuous surveillance by detecting movement of humans in open areas. This diverse set of applications require a possibly large number of networked sensors for continuous and pervasive monitoring. Wiring the sensor network for power and communication is, in most outdoor cases, too expensive and not practical. Hence, battery–operated, wireless deployments are required.

The use of high–level sensors (e.g., cameras) in battery–operated networks has received relatively little attention so far by the research community. Unlike simpler sensors, cameras produce large loads of data, which require considerable processing for analysis (in order to extract semantic information) and/or compression. In addition, sensing itself can be highly power consuming. In "traditional" camera-based networks (e.g., for surveillance), nodes are wired and plugged to continuous power sources; thus, power and bandwidth are not a concern. However, in wireless camera networks, both bandwidth and power are premium resources. They must be efficiently managed to maximize the system's operational lifetime. Therefore, one of the main challenges posed by wireless visual sensor networks is the constant tension between power conservation and performance (e.g., event detection probability in surveillance applications).

Addressing this tradeoff between power efficiency and performance is one of the main goals of the Meerkats project [3]. To study this tradeoff in a real testbed, we have been developing a wireless network of battery-operated camera-equipped nodes that can be used for monitoring and surveillance of arbitrarily large (indoor or outdoor) areas. Because efficient energy management is so critical, we have conducted a thorough energy consumption characterization of the Meerkats testbed which is based on the Crossbow Stargate platform [5]. In this paper, we present results from this study. Our approach was based on assessing energy consumption of activities representative of the target application (e.g., perimeter surveillance) using a benchmark that runs (individual and combinations of) "basic" tasks such as processing, flash memory access, image acquisition, and communication over the network. In our characterization, we consider the various hardware states the system switches through as it executes these benchmarks, e.g., different radio modes (sleep, idle, transmission, reception), and webcam modes (off, on, and acquiring image). We report both steady-state and transient (i.e., when switching states) energy consumption behavior. Besides results obtained from direct measurement using a digital multimeter, we also present results obtained using the Stargate's on-board energy consumption measuring capabilities.

The remainder of the paper is organized as follows. Section II describes the Meerkats testbed, while Section III describes the methodology used on the energy consumption characterization, including the benchmark and measurement setup

used. Results for steady state and transition costs (delay and additional charge) are presented in Section IV. Then we present the in-system energy consumption monitoring tool in Section V. We summarize related work in Section VI. Section VII concludes the paper and discusses future work.

II. The Meerkats Testbed

Currently, our Meerkats testbed consists of eight visual sensor nodes and one information sink. A Dell Inspiron 4000 laptop with PIII CPU, 512M memory, and 20G hard disk is used as the sink. It runs Linux (kernel 2.4.20) and uses an Orinoco Gold 802.11b wireless card for communication.

The Meerkats visual sensor nodes use Crossbow's Stargates [5] [1]. The Stargate model we employ is based on the XScale PXA255 CPU (400 MHz), has 32MB flash memory and 64MB SDRAM, and provides PCMCIA and Compact Flash connectors on the main board. It also has a daughter board with Ethernet, USB and serial connectors. As shown in Figure 1, we equipped each Stargate with an Orinoco Gold 802.11b PCMCIA wireless card and a Logitech QuickCam Pro 4000 webcam connected through the USB. The QuickCam can capture video with resolution of up to 640x480 pixels. The operating system is Stargate version 7.3 which is an embedded Linux system (kernel 2.4.19).

Fig. 1. Visual sensing node in the Meerkats testbed

The Stargate can be powered through a 5V DC adaptor or through a battery. Both the main- and daughter boards have battery input, but only the daughter board has a DC input. Since we use the USB connector on the daughter board, we need to power the Stargate through the daughter board with 5V. To achieve this, we use a customized 7.4 Volt, 1000mAh, 2 cell Lithium-Ion (Li-Ion) battery, manufactured by Energy Sales, Inc. and an external DC-DC (with efficiency of about 80%) switching regulator. The current battery has a small form factor ($67 \times 60 \times 9$ mm), which makes it ideal for encasing the system in a small container. A number of such batteries can be connected in parallel for increased capacity.

An important feature provided by the Stargate is its battery monitoring capability. This is achieved through a specialized chip (DS2438) on the main board. Two kernel modules, namely *onewire* and *batmon*, provide access to the battery monitor chip and retrieve information on the battery's current state. information.

[1]Other projects that use the Stargate as a sensing node include SensEye [8].

III. Methodology

The basis for our energy consumption characterization study of the Meerkats testbed is to define a benchmark representative of typical activities performed by wireless visual sensor networks. Such activities include acquiring, processing, and transmitting images. We then decompose these activities into "basic" tasks such as processing, input/output (flash memory, webcam), and communication (transmission and reception over the network). Running benchmarks consisting of these individual tasks and a combination thereof takes the system through different hardware states, i.e. different radio modes (sleep, idle, rx, tx) and webcam modes (off, on and acquiring images). To better characterize these different states, let us review the Meerkats node's major hardware subsystems, namely:

- **Processor core**: consists of the processor itself, memory (RAM and flash), and associated hardware;
- **Sensor core**: includes the sensing devices, i.e. the webcam, together with the USB interface;
- **Communication core**: consists of the wireless communication card, and associated PCMCIA modules.

These different components can be in different states. For instance, the processor can be sleeping, idle, active (processing), writing data or reading data; the sensor can be sleeping, idle, or active (i.e., acquiring image/video); while the radio can be sleeping, idle, receiving or transmitting. However, instead of exploring the whole state space, i.e., ALL state combinations, we only consider the physically possible ones. For example, it does not make sense to have the processor sleeping and radio and/or sensors idle or active, since the latter need to be controlled by the processor.

Switching the Meerkats node through its state space requires executing specific utilities made available by the operating system. More specifically, to put the processor in sleep mode, one must execute the utility *sys_suspend*, proving as parameter the sleep interval. *cardctl suspend* puts the wireless card in sleep mode, while *cardctl resume* switches the wireless card from sleep to idle. The mechanism we use to put the webcam in sleep mode is to remove the corresponding modules (*rmmod usb-ohci-sa1111*) from the kernel, while changing the webcam from sleep to idle requires inserting back the corresponding modules (*insmod usb-ohci-sa1111*).

It should be noted that, in the Stargate platform, when the timer for the *sys_suspend* command expires and the node "wakes up", the wireless card goes to idle no matter what its previous state was. This is not the case for the webcam because of the way the *sys_suspend* script is implemented. In our measurements, we unplugged the wireless card as well as the webcam to obtain the different combinations of hardware subsystems.

A. Measurement Setup

For our direct measurements, we use the HP E3631A power configured to provide 5.4V to power the Stargate. The HP34401A digital multimeter (DMM) is used to measure the current flow as the different hardware subsystems become active/inactive while the different benchmarks are executed.

Figure 2 shows a block diagram describing to our measurement setup.

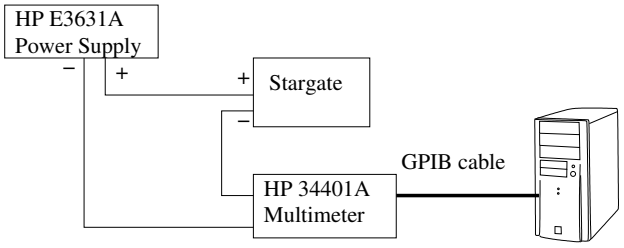

Fig. 2. Measurements setup

As shown in Figure 2, a GPIB cable is used to connect the HP 34401A DMM to a computer to collect and record measurement samples. The DMM was configured to provide a reading rate of 60 Hz. This setup allows us to measure steady state currents (via time integration) as well as transients.

B. Energy Consumption Characterization Benchmark

As previously discussed, we define an energy consumption characterization benchmark consisting of a set of basic operations that are representative of activities performed by visual sensor nodes. Our benchmark consists of five main task categories, namely: *idle*, *processing intensive*, *storage intensive*, *communication intensive* and *visual sensing*. The descriptions of these tasks follow.

a) Idle: The idle– or baseline benchmark captures the energy consumption behavior of the node when only basic operating system tasks are running. This benchmark characterizes energy consumption when the system is idle and also serves as baseline for all other tasks.

b) Processing-intensive: The characterization of processing-intensive tasks is performed using the FFT benchmark [2], which is part of SPEC's CPU2000 [14], an industry-standardized CPU-intensive benchmark suite.

c) Storage-intensive: The storage media available on the Stargates is flash memory. In order to understand its energy consumption behavior, we use a program that reads and writes files with random data.

d) Communication-intensive: To characterize the energy consumed by communication-related tasks, we use a set of UDP client/server programs. The client program transmits a certain amount of random bytes (provided as a argument) to the server. To obtain the energy cost of transmission, we run the client program on the Stargate being monitored. Then we monitor the Stargate running the server program to obtain the energy cost of reception.

e) Visual sensing: Power consumed by the webcam is characterized using the *videotime* program available on the Stargate 7.3, to acquire a sequence of frames.

IV. Results

This section reports steady-state (Section IV-A) and transient (Section IV-B) energy consumption behavior for the Meerkats visual sensing node. Task execution (e.g. processing, acquiring image, communication, etc.) characterizes steady-state, while transients are captured when state changes occur, e.g., node goes to sleep and wakes up, or webcam is suspended.

A. Steady state

Using the setup described in Section III-A, we executed each of the benchmarks described in Section III-B activating different combinations of Meerkats node components (i.e., processor core, processor/sensor core, processor/radio core, and processor/radio/sensor core). Table I shows average current (over five runs) in milli-Amperes drawn from the Meerkats node when running the different benchmarks with different combinations of active hardware subsystems. Standard deviations are also presented.

Task	Processor Core	Proc./Sensor Core	Proc./Radio Core	Proc./Sensor/ Radio Core
idle	139 ± 1.57	324 ± 2.33	300 ± 2.92	487 ± 2.51
fft	291 ± 1.21	474 ± 9.70	457 ± 1.23	642 ± 9.20
read	248 ± 2.12	434 ± 5.58	414 ± 2.89	604 ± 1.94
write	248 ± 1.33	436 ± 6.90	413 ± 1.05	604 ± 2.16
image	-	341 ± 24.17	-	515 ± 2.04
TX	-	-	383 ± 0.92	559 ± 8.39
RX	-	-	361 ± 2.72	537 ± 9.10
sleep	3 ± 0.04	3 ± 0.02	10 ± 0.26	9 ± 0.11

TABLE I
AVERAGE CURRENT (OVER FIVE RUNS) IN MILLI-AMPERES AND STANDARD DEVIATION DRAWN BY THE MEERKATS' NODE.

These measurements highlight a number of interesting observations. For instance, the considerable difference in power consumption when comparing results from the "sleep" and "idle" benchmarks. It is also interesting to note that communication-related tasks (i.e., RX and TX) are less expensive than intensive processing and flash access when the radio modules are loaded. Additionally: the processing-intensive benchmark results in the highest current requirement; flash reads and writes cost about the same; and TX is only about 5% more expensive than RX.

The relatively small difference between RX and TX modes has been observed before, e.g., in [7] which reported the power consumption of a WaveLAN wireless card at 2 Mbps. The reported difference, however, considerably higher than the difference we see here. This is partially due to the fact that the difference reported in [7] was computed based on the current drained by the wireless card only. In our measurements, we consider the overall system, which also includes the processing associated with communication.

Figure 3 plots the results shown in Table I, highlighting other interesting results. For instance, we observe that the webcam adds about 185mA to *Processor Core*, while the wireless card adds about 165mA to *Processor Core*; i.e., the additional current consumed by activating a system component is constant over all tasks.

B. Transients

To characterize transient power consumption behavior when the Meerkats node switches between operational states, we

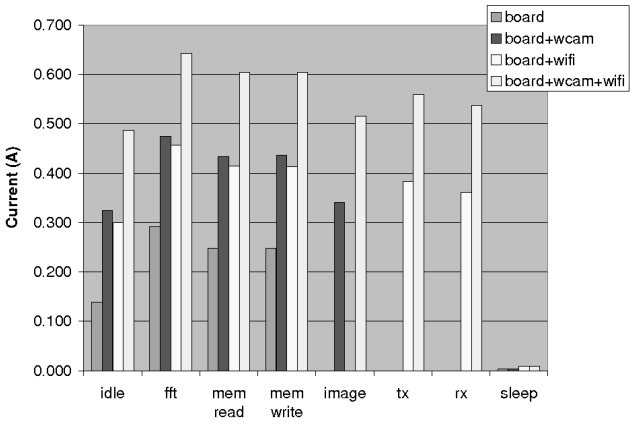

Fig. 3. Steady-state current draw in the Meerkats node

added and removed the corresponding operating system modules/drivers controlling each subsystem (e.g., webcam, network card) being activated/deactivated. In some cases, switching states involves powering on/off electronic components (USB/webcam, network card). This typically causes short transients with possibly high currents. The length of a transient depends both on the electrical characteristics of the subsystem being powered on/off and the processing/response time of the operating system when executing the task(s) required to switch states. The graph in Figure 4 gives an example of power consumption transients for the Meerkats node. This particular run starts with the node fully operational, i.e., with both the network card and webcam activated. The first transient corresponds to deactivating the network card followed by deactivating the webcam, and switching the board to *sleep* mode. Then, at time 6.5 sec, the board wakes up, and the webcam and network card are reactivated, respectively.

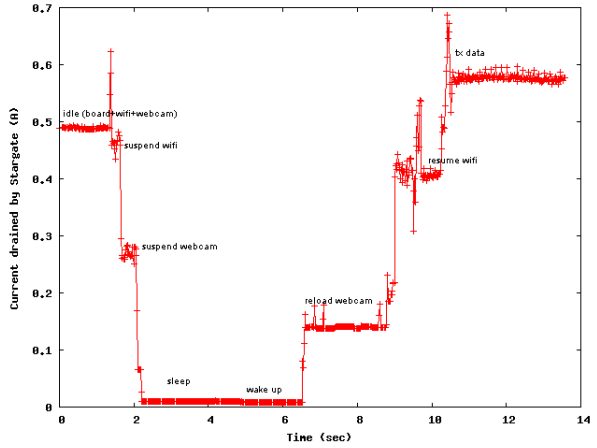

Fig. 4. Transients from complete system to suspend wifi, suspend webcam, and switch board to sleep state; then, wake up board, reload webcam, resume wifi and start tx

Another example of power consumption transients for the Meerkats node is shown in Figure 5. This particular example considers that the Meerkats node is in idle mode with only the board active, and that some application running on the node requests an image to be taken. Th is request will require turning the camera on, taking the picture, writing to flash, and then turning the camera off.

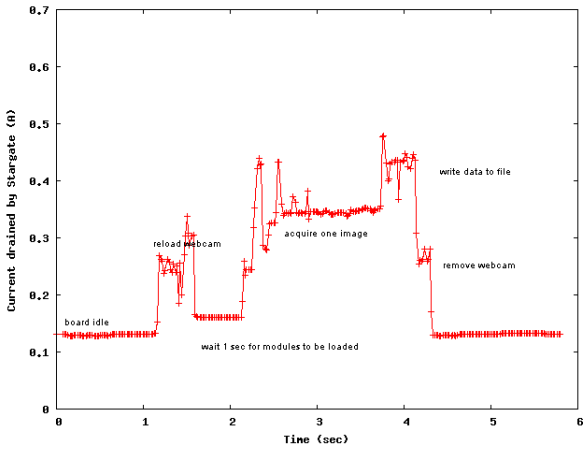

Fig. 5. Tasks involved when acquiring an image.

It is clear from Figures 4 and 5 that transients are not at all negligible in the Stargate (and likely in other sensor network platforms). For accuracy, energy characterization for these platforms must account for both the additional power consumed by transients as well as the associated delay.

Using our measurements, we are able to calculate the delay and the additional energy consumption due state transitions. The transitions considered are:

- **Suspend/resume webcam**: a script that removes/reloads related kernel modules is used;
- **Suspend/resume wireless card**: this is achieved using the command line utility *cardctl suspend/resume*;
- **Put node to sleep**: the script *sys_suspend* performs the necessary tasks to put the processor and the whole board in sleep mode for a given amount of time;
- **Wake up node**: the node wakes up to service an interrupt. The default for the Meerkats node is to wake up when the time interval given as a parameter to the *sys_suspend* script expires.

Transient delay is defined as the interval between the time the corresponding command to activate/deactivate a specific device is issued and the time current consumption becomes stationary, i.e., when all operations associated with the transient have been completed. Table II presents the delay associated with each transition considered.

As previously pointed out, transients are not at all negligible neither in terms of power consumed nor in terms of delay incurred. For instance, as shown in Table II, resuming webcam activities is the transition that takes the longest (about 1.2 seconds). Clearly, this has direct impact on node duty-cycling. Suspending the webcam takes about 300 milli-seconds, which is about the same for the wireless card. But resuming the wireless card takes about 500 milli-seconds, which is less than half the time necessary to resume the webcam. This happens because the operating system has support to suspend the PCMCIA modules, which makes this process more efficient.

Task	Processor Core	Proc./Sensor Core	Proc./Radio Core	Proc./Sensor /Radio Core
Resume cam	-	1272 ± 32	-	1294 ± 13
Suspend cam	-	303 ± 24	-	322 ± 14
Resume wifi	-	-	485 ± 19	487 ± 21
Suspend wifi	-	-	313 ± 11	294 ± 7
Go to sleep	372 ± 13	397 ± 29	444 ± 74	456 ± 43
Wake up	484 ± 114	1288 ± 24	2844 ± 210	3684 ± 48

TABLE II

TRANSITION DURATIONS IN MILLI-SECONDS (AVERAGE AND STANDARD DEVIATION CALCULATED OVER FIVE DIFFERENT RUNS).

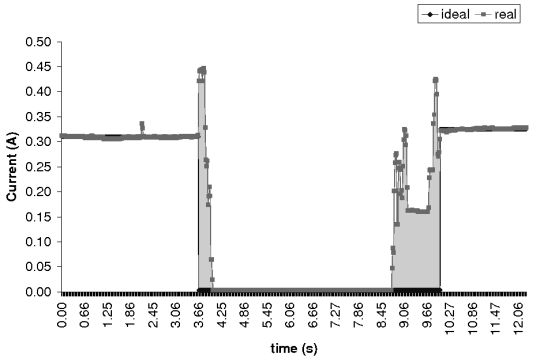

Fig. 6. Charge consumed by transitions from idle to sleep and back to idle of the *Processor/Sensor Core*.

On the other hand, to suspend the webcam we need to remove both the webcam and USB kernel modules, which takes longer. We also notice that when the node wakes up with the wireless card on, the process will take longer since the node must first have the board operational, and then power the PCMCIA hardware so that the wireless card can become operational.

In order to verify the delays reported in Table II, we use the operating system utility *time* to obtain the time necessary to execute the programs that trigger the transitions. Table III shows these results which are consistently smaller than the corresponding results in Table II. This confirms that the times reported in Table II correspond to the times it takes for the operating system to execute the corresponding software modules (as reported in Table III) plus the time for the corresponding hardware sub-system to become active/inactive.

Program	Time (s)
Remove wcam	0.21
Suspend wcam	0.259
Remove wifi	0.06
Suspend wifi	1.92

TABLE III

TIME TO EXECUTE SOFTWARE ASSOCIATED WITH TRANSITIONS.

The amount of charge consumed by a transition can be obtained by integrating over time the difference in current between the ideal and actual transient behavior. For example, Figure 6 shows a transition from idle to sleep and then again to idle for the *Processor/Sensor Core*. The gray area in the figure represents the amount of additional charge consumed due to transitory current fluctuations.

Table IV summarizes the charge needed for the state transitions mentioned above. The results presented are averaged over five different runs. Some of the results are noteworthy. For instance, for the *Proc/Sensor Core* sub-system, resuming the webcam consumes about the same charge as going to sleep, although the delay is about three times larger. Another interesting result is that there is no correlation between delay and charge, e.g., transitions with the same delay may need different charge. For instance, switching to sleep mode for the *Processor Core* takes 372 milli-seconds and requires 65 milli-Coulombs on average, while for the *Proc/Sensor Core* it takes 397 milli-seconds and requires 119 milli-Coulombs on average.

Task	Processor Core	Proc./Sensor Core	Proc./Radio Core	Proc./Sensor/ Radio Core
Resume cam	-	122 ± 4	-	126 ± 5
Suspend cam	-	74 ± 14	-	69 ± 4
Resume wifi	-	-	92 ± 2	95 ± 3
Suspend wifi	-	-	47 ± 4	47 ± 2
Go to sleep	65 ± 2	119 ± 9	127 ± 24	176 ± 19
Wake up	87 ± 14	269 ± 6	504 ± 91	835 ± 16

TABLE IV

CHARGE IN MILLI-COULOMBS (AVERAGE AND STANDARD DEVIATION CALCULATED OVER FIVE RUNS) REQUIRED FOR TRANSITIONS.

The results presented in Tables II and IV show that both the hardware and software involved in the transition play a significant role in determining the delay and charge of each transition.

We should again emphasize that delay and additional amount of charge due to transitions are an important consideration when scheduling duty cycles for visual sensing nodes. For example, in some cases, it may be more energy efficient to keep radio and webcam modules loaded rather than loading and unloading them very frequently.

V. IN-SYSTEM ENERGY CONSUMPTION MONITORING

As we mentioned before, an important feature provided by the Stargate is its built-in battery monitoring capability. This is achieved through a specialized chip, the DS2438, located on the main board. In this section, we describe how on-board battery monitoring works on the Stargate, how we extended it, and the power consumption measurements we obtain using it.

A. DS2438 Battery Monitor

Crossbow's Stargate smart battery monitoring capability, using the DS2438 chip, is normally found permanently mated to a rechargeable battery in a battery pack. The DS2438 8-pin chip has the capability to measure the voltage across the Stargate's power source and the current flowing out of the battery and into the Stargate. Measuring current is achieved by measuring the voltage at the ends of a low-value shunt resistor, as shown in Figure 7.

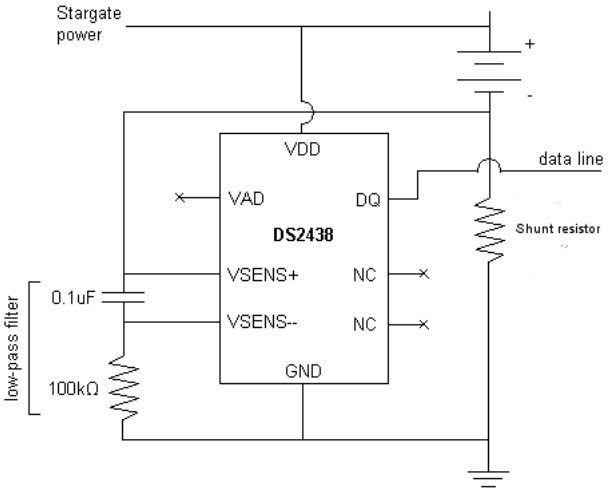

Fig. 7. Battery monitor connection diagram. Voltage across the shunt resistor is measured by V_{sens+} and V_{sens-}. Current can be determined using $V = I \times R$.

The DS2438 chip [12] takes current measurements at a rate of 36.41 times per second. These measurements are stored in the chip's RAM and can be acquired through the one-wire interface (DQ pin). The readings are stored in units of 0.2441mV ($1/4096$) and the maximum voltage difference between the V_{sens} pins is 300mV.

The Stargate is equipped with a $5m\Omega$ shunt resistor. Since voltage readings are stored in units of 0.2441mV, this would allow a granularity of 48.82mA (obtained using $V = R \times i$) for the current readings. Additionally, the DS2438 chip can introduce an error of ± 2 LSb (least significant bits), which could result in as much as ± 97.64mA of error.

Considering the current range for the activities we define as representative of visual sensor network activities (see Table I), it is clear that the granularity of 48.82mA does not allow accurate readings. Thus we replaced the existing $5m\Omega$ shunt resistor with a $270m\Omega$ one. By doing this, we achieve a granularity of 0.904mA with ± 1.808mA error.

B. Kernel Modules and Interface

As described in Section II, the Meerkats node runs a Linux-based operating system, namely the Stargate 7.3 operating system which provides two kernel modules that interface with the battery monitoring hardware, namely: the *onewire* and *batmon* modules [2]. *onewire* implements the one-wire protocol used to communicate with the DS2438 chip, and provides wrapper functions for reading from and writing to the DS2348 through the one-wire data bus. *batmon* is a higher level module, that uses the functions made available by *onewire* to read the voltage from the chip and make it available for reading in user-space through a device (i.e.,/dev/platx/batmon).

The original *batmon* provides only voltage readings, therefore we had to modify it to obtain current readings as well. The function $get\_battery\_status()$ in the *batmon* module, which previously only returned the battery voltage, was modified to return the current across the shunt resistor as well, using the following equation: $I = CurrentRegister/(4096 * R_{shunt})$. Since floating point operations are not supported in the Stargate Linux kernel, the output from $get\_battery\_status()$ contains both the kernel space calculated current and the raw value of the current register, so that user space programs can calculate the value exactly.

Additionally, a new function $batmon\_sscanf(char * buf, int * value)$ was added to handle the conversion of a string representation of a hexadecimal number to an integer representation. The reason for this is that the kernel $sscanf()$ implementation presented a problem which prevented it from parsing the hexadecimal string correctly. Accordingly, the $get\_value()$ function was also modified.

Another issue with the current measurement is that the DS2438 chip has an offset register (intended to cancel small offset errors in the current ADC), which is added to the current register after every reading is taken. The offset register needs to be calibrated to avoid introducing errors [3]. In order to obtain the value currently stored in the offset register, we powered the Meerkats node through the DC jack, and then used *batmon* to measure the current. The reasoning for this is that when the board is powered straight from the DC jack no current flows through the shunt resistor and the reading should be 0mA. Instead the initial result was -14mA. We account for this offset in a user-level program (*batread*) that monitors *batmon* results.

batread is a simple program that polls */dev/platx/batmon* and records the current reading. As parameters it uses the number of samples, the interval (in seconds) between them, and the output filename. It is implemented as loop that sleeps in between readings (to minimize interference with the system's energy consumption).

C. Battery Monitoring Performance Validation

In order to validate the current readings provided by *batread*, we use the same setup described in Section III-A and followed the same methodology described in Section III to conduct direct current measurements, while using *batread* to record the current being drained by the system every second. Table V presents the results obtained by *batread*, while Table VI presents the current measured by the multimeter.

Task	Processor Core	Proc./Sensor Core	Proc./Radio Core	Proc./Sensor/ Radio Core
idle	143.5	331.4	304.4	495.7
fft	250.9	434.2	454.0	633.4
read	220.3	441.3	419.5	611.7
write	223.5	415.6	392.8	599.5
image	-	352.2	-	509.8
tx	-	-	421.3	592.8
rx	-	-	324.4	526.0

TABLE V
CURRENT (IN MILLI-AMPERES) OBTAINED WITH *batread*.

We do not present results for sleep in Table V since the

[2]Both modules were implemented by Trevor Pering

[3]According to the DS2438 data-sheet [12], a calibration is done before shipment, but the system should be calibrated altogether.

system will not be able to keep track of the current readings if the processor is sleeping.

Task	Processor Core	Proc./Sensor Core	Proc./Radio Core	Proc./Sensor/ Radio Core
idle	134 ± 3	317 ± 1	294 ± 1	479 ± 1
fft	287 ± 1	476 ± 1	455 ± 1	642 ± 1
read	249 ± 1	440 ± 1	417 ± 1	600 ± 5
write	246 ± 1	436 ± 1	413 ± 0	599 ± 1
image	-	358 ± 3	-	513 ± 3
tx	-	-	390 ± 1	564 ± 10
rx	-	-	369 ± 1	545 ± 11
sleep	3 ± 0	3 ± 0	9 ± 0.1	10 ± 0.1

TABLE VI

CURRENT, IN MILLI-AMPERES, MEASURED WITH DMM WHILE *batread* WAS RUNNING (AVERAGE AND STANDARD DEVIATION CALCULATED OVER FIVE RUNS).

Comparing the results in Tables VI and V, we can quantify the interference of on-board battery monitoring. For example, for tasks that are more power-demanding (e.g., fft), the average current readings obtained from *batread* are smaller than the corresponding DMM readings. This is expected since the processor needs to switch between tasks (e.g., fft and *batread*) to execute the in-system current measurements and because *batread* is less power-demanding, the average current consumed is lower that when just fft runs. For less power-consuming tasks, the interference may happen in the opposite direction, i.e., the current readings by *batread* may be higher.

Figure 8 summarizes the results from Tables VI, V and I for the *Proc/Sensor/Radio Core* subsystem.

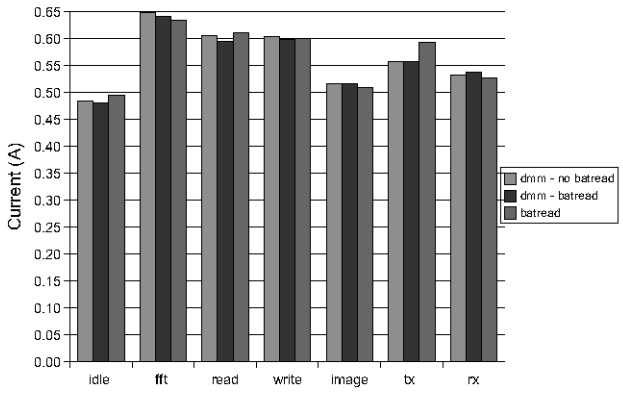

Fig. 8. Current (in Amperes) measured with DMM with and without *batread* running, and *batread* results. The results apply to the *Proc/Sensor/Radio Core*).

We also have to keep in mind that the DS2438 chip has an error of ± 2 LSb (± 1.808mA in the current Meerkats node configuration). Another issue is the rate at which *batread* is recording current readings, which is one reading every second. When compared to the multimeter reading frequency (60 Hz), this is very low and therefore may cause loss of information. On the other hand, the higher the reading rate for *batread*, the more interference it will cause.

Figures 9 and 10 illustrates the effect of Stargate's in-system energy consumption monitoring interference with respect to readings of current drawn.

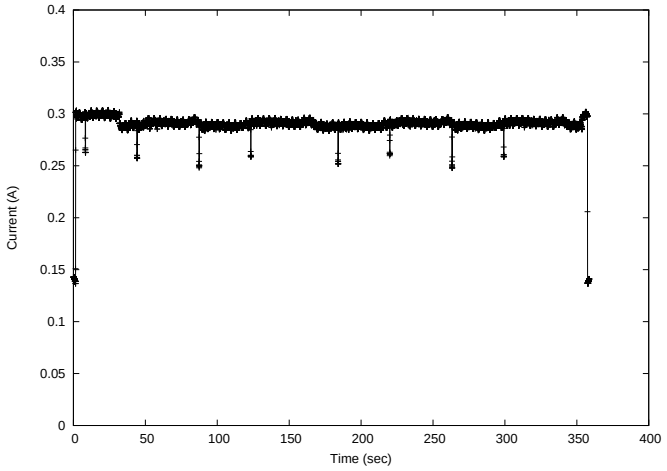

Fig. 9. Current being drained by *Processor Core* when fft is being executed.

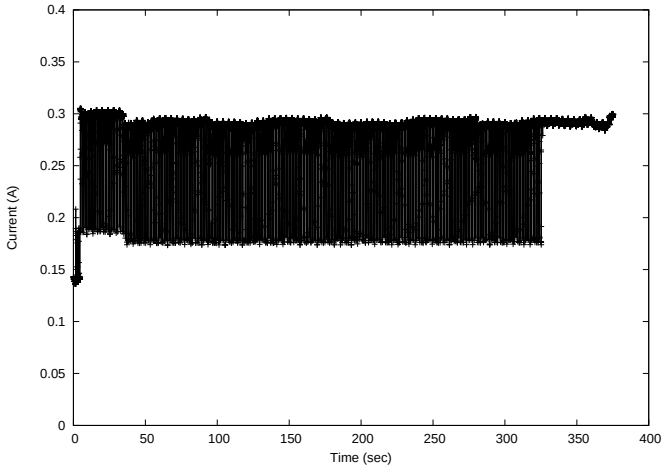

Fig. 10. Current being drained by *Processor Core* when *batread* is monitoring the system at 1 Hertz and fft is being executed.

In-system energy consumption monitoring is important in order to track battery discharge and be able to make decisions influencing the trade-off between power conservation (and thus operation lifetime of the system) and performance. Although in-system monitoring interferes slightly with energy consumption, we were still able to cross-validate on-board monitoring measurements against DMM readings.

VI. RELATED WORK

In this section, we summarize related work in areas relevant to our work, including energy consumption measurements for network interfaces, sensor networks platforms and mobile devices.

Stem et al. [15] measures the power consumption of some network interface cards (NICs) when used by different end-user devices. The work also report on transport– and application–level strategies to reduce energy consumption by NICs. Later, in Feeney et al. [7], detailed energy consumption

measurements of some commercially-available IEEE 802.11 NICs operating in ad hoc mode are provided. Along the same lines, Erbert et al. [6] assesses the impact of transmission rate, transmit power, and packet size on energy consumption in a typical wireless network interface. Pering et al. [9] presents a mechanism to exploit radio hierarchies for wireless devices, and reports energy consumption on Bluetooth and IEEE 802.11 NICs.

The Panoptes sensor hardware [4] is similar to the Stargate hardware, but has higher power requirements. When we compare the power requirements of both platforms, we observe that the Stargate provides energy savings of up to 25%.

Another widely used platform in sensor networks are the Berkeley Motes [16]. A detailed accounting of the energy consumed by the Motes is presented in [13]. A detailed power profile of the Consus platform, a personal data repository, is presented in [10]. The work reports energy consumption for basic tasks such as sleep, idle, processing, and communication over IEEE 802.11 and Bluetooth radios.

More recently, some research efforts have focused on developing visual sensing nodes. For example, Cyclops [11] is a low-power, small sensing node composed of a microcontroller, complex programmable logic device, external SRAM, external Flash and an CMOS imager. SensEye [8] is a multi-tier video surveillance network, which makes use of several different visual nodes including: Cyclops [11], CMUcam [1], Crossbow Stargate and webcam, and a high-end camera.

VII. Conclusions

In this paper we presented a thorough energy consumption characterization of a visual sensor network testbed. Each sensing node in the testbed consists of a "single-board computer", namely Crossbow's Stargate equipped with a wireless network card and a webcam. We assessed energy consumption of activities representative of the target application (e.g., perimeter surveillance) using a benchmark that runs (individual and combinations of) "basic" tasks such as processing, flash memory access, image acquisition, and communication over the network. In our characterization, we considered the various hardware states the system switches through as it executes these benchmarks, e.g., different radio modes (sleep, idle, transmission, reception), and webcam modes (off, on, and acquiring image). We reported both steady-state and transient energy consumption behavior obtained by direct measurements of current with a digital multimeter (DMM). Our results show that delay and additional amount of charge due to transitions are not at all negligible and must be accounted for when determining important system parameters such as the duty cycle.

We also presented the extensions we implemented to obtain current readings using the Stargate's on-board energy consumption measuring capabilities. On-board energy consumption monitoring is important in order to track battery discharge and be able to make decisions influencing the trade-off between power conservation (and thus operation lifetime of the system) and performance. We showed that on-board monitoring interferes slightly with energy consumption but were still able to cross-validate on-board monitoring measurements against DMM readings.

As future work, we intend to use the characterization presented here to develop a probabilistic energy consumption model to predict energy consumption behavior of the Meerkats node. This energy prediction model will then be used to implement a resource manager that will be able to predict the node energy consumption, verify its current state (using the on-board energy consumption measuring capabilities), and decide the node's actions optimizing the energy consumption–performance trade-off.

VIII. Acknowledgements

The Meerkats project is supported by NASA under contract NNA04CK89A. Cintia Margi was supported by a scholarship from CNPq/Brazil until 31/Aug/2005.

References

[1] The CMUcam Vision Sensors. http://www.cs.cmu.edu/ cmucam/, 2005.
[2] A. Aburto. FFT double precision benchmarks. ftp://ftp.nosc.mil/pub/aburto/, 2001.
[3] J. Boice, X. Lu, C. B. Margi, G. Stanek, G. Zhang, R. Manduchi, and K. Obraczka. Meerkats: A Power-Aware, Self-Managing Wireless Camera Network for Wide Area Monitoring. Technical Report ucsc-crl-05-04, University of California Santa Cruz, 2005.
[4] W. chi Feng, B. Code, E. Kaiser, M. Shea, W. chang Feng, and L. Bavoil. Panoptes: scalable low-power video sensor networking technologies. In *MULTIMEDIA '03: Proceedings of the eleventh ACM international conference on Multimedia*, pages 562–571, New York, NY, USA, 2003. ACM Press.
[5] Crossbow. Stargate. http://www.xbow.com/, 2004.
[6] J. Ebert, S. Aier, G. Kofahl, A. Becker, B. Burns, and A. Wolisz. Measurement and simulation of the energy consumption of an WLAN interface. Technical Report TKN-02-010, Technical University Berlin, Telecommunication Networks Group, Germany, June 2002.
[7] L. M. Feeney and M. Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies.INFOCOM 2001*, volume 3, pages 1548–1557. IEEE, April 2001.
[8] P. Kulkarni, D. Ganesan, and P. Shenoy. The case for multi-tier camera sensor networks. In *International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2005)*, 2005.
[9] T. Pering, V. Raghunathan, and R. Want. Exploiting radio hierarchies for power-efficient wireless device discovery and connection setup. In *Proceedings of the 18th International Conference on VLSI Design held jointly with 4th International Conference on Embedded Systems Design (VLSID'05)*, 2005.
[10] E. W. A. L. P. W. M. C. Platform. Vijay raghunathan and trevor pering and roy want and alex nguyen and peter jensen. In *ISLPED 2004*, 2004.
[11] M. Rahimi, R. Baer, O. I. Iroezi, J. C. Garcia, J. Warrior, D. Estrin, and M. Srivastava. Cyclops: In situ image sensing and interpretation in wireless sensor networks. In *SenSys 2005)*, 2005.
[12] D. Semiconductor. DS2438: Smart battery monitor datasheet. http://www.maxim-ic.com, 2004.
[13] V. Shnayder, M. Hempstead, B. Chen, G. Werner-Allen, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. In *ACM SenSys 04*, Baltimore, MA, November 2004.
[14] Standard performance evaluation corporation. http://www.spec.org, 2004.
[15] M. Stemm and R. H. Katz. Measuring and reducing energy consumption of network interfaces in hand-held devices. *IEICE Trans. on Communications*, 8(E80-B):1125–1131, 1997.
[16] A. Woo. Mote documentation and development information. http://www.eecs.berkeley.edu/ awoo/smartdust/, 2000.