UNIVERSITY OF CALIFORNIA
SANTA CRUZ

ENERGY CONSUMPTION TRADE-OFFS IN POWER CONSTRAINED NETWORKS

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER ENGINEERING

by

Cíntia Borges Margi

June 2006

The Dissertation of Cíntia Borges Margi
is approved:

Professor Katia Obraczka, Chair

Professor Roberto Manduchi

Professor Scott A. Brandt

Lisa C. Sloan
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Energy Consumption Trade-offs in Power Constrained Networks

by

Cíntia Borges Margi

Wireless Sensor Networks are a valuable technology to support many applications in different areas, such as: environmental and habitat monitoring, surveillance, indoor climate control, structural monitoring, mapping, disaster management, and so on. Participating nodes in these networks are inherently resource constrained, since they have limited processing capabilities, storage, communications speed and bandwidth, and mainly they have very limited power supply. Most of current research on sensor network protocols focus on applications that requires low sampling rate and low bandwidth, such as applications that make use of sensors like humidity or temperature. Also, the main assumption is that communications cost dominate in Sensor Networks. But this is not the case for all types of sensor networks. One of the most complex scenarios is a Visual Sensor Networks (VSN), which include cameras as sensing devices. This kind of sensor networks has different processing and network requirements. These requirements are associated with the application of the visual sensor network performance (e.g. image/video acquisition frequency, processing and transmission), but they have a huge impact on the node lifetime.

The main goal of this work is to understand the energy consumption trade-offs between computation and communication in power constrained networks in general, and in visual sensor networks, in particular. In order to do so, we need to evaluate and model: energy spent

to process data by sensors (e.g., how much energy vision algorithms require, etc.), as well as energy required for communications.

The first step is to develop an accurate model for energy consumption due to communications. Such energy model must be as close to reality as possible, taking into account all radio states, i.e., energy spent not only while transmitting and receiving a packet, but also while in idle, overhearing, or sleep modes. We have developed such a model and used it to instrument the QualNet and GloMoSim network simulators. We have used QualNet/GloMoSim instrumented with our model to compare energy consumption of power-aware MAC protocols, multi-hop ad-hoc wireless networks routing protocols, and to evaluate an analytical model of energy consumption in single-hop IEEE 802.11 ad-hoc networks.

Next, we look at a model that also includes processing and sensing tasks, besides communication. Since we were targeting power constrained networks, we had to look into a typical platform for wireless mobile applications: a laptop. This step provided a good opportunity to define our methodology, which determines the energy consumption of basic tasks.

From this point on, we focused on Visual Sensor Networks. We designed and implemented a wireless camera network testbed, *Meerkats*, which is based on the Stargate platform. We followed the same methodology developed for the laptops' testbed to characterize the energy consumption of the Meerkats' node. Along this work, we also extended and validate the on-board battery monitoring capability on the Stargate.

Following this step, we tailored the tasks to the ones representative of activities carried out by wireless camera networks targeting surveillance applications. Given this set of tasks (which we call elementary tasks), a quantitative power consumption and temporal analysis of

set of duty cycles that apply to the Meerkats testbed was done.

Then we used what we learned from the duty cycle analysis along with a set of experiments on the Meerkats testbed to understand how a framework for lifetime prediction can be build. We proposed a simple deterministic lifetime prediction model based on task composition, which we validate using the experiments executed on the Meerkats testbed. Finally, we consider a set of possible duty cycles given requirements for the Meerkats testbed and analyze how they affect the node lifetime.

Acknowledgments

First and foremost I would like to thank my advisor Katia Obrazcka, for her guidance, encouragement and support. I also would like to thank Roberto Manduchi for his efforts to develop the Meerkats testbed and guidance in the work we did together.

I would also like to thank Carol Mullane and Jodi Rieger for their support and invaluable advice throughout these five years at UCSC. Also I would like to acknowledge the contribution from David Meek and Dave Van Unen from BELS, and Cyrus Bazeghi and Steve Petersen. I would also like to thank my colleagues and friends at the Meerkats project, and the INRG and CCRG labs. A special thanks to Marco, Marcelo and Renato, and their wives, who became my family here.

I would like to further thank Wilson Ruggiero, from University of São Paulo in Brazil, for encouraging me to pursue my PhD in USA, and Glenn Langdon who encouraged me to come to Santa Cruz.

I cannot forget to thank my parents, Michel and Cidália, and my siblings, Flávia, Daniel and Raquel, who always supported me, even though their were thousands of miles from here. I also have to thank Erica, Ana Beatriz, Vivian, Regina and Leandro, in Brazil, and Aline, Linda and Ana Lucia, in Santa Cruz, for their friendship.

Last but not the least, this work would not have been possible without the support from CNPq/Brazil, from whom I received the fellowship that allowed me to come to UCSC and supported me here from August 2001 until August 2005. This work was also supported by NASA, under contract NNA04CK89A.

To my parents, Michel and Cidália.

Chapter 1

Introduction

1.1 Motivation and Background

Wireless Sensor Networks are a valuable technology to support countless applications in different areas, such as: environmental and habitat monitoring, surveillance, indoor climate control, structural monitoring, medical diagnostics, mapping, disaster management, and so on. They can be defined as special type of multi-hop ad-hoc networks (MANETs), since they share a number of common characteristics. MANETs are wireless networks also known as ”networks without a network” since they do not use any fixed infrastructure. Participating nodes in these networks are typically battery operated, and thus have access to a limited amount of energy. Besides being power constrained, network nodes often have exhibit additional constrains, e.g., they typically have a limited processing, storage and communications capabilities [19].

Another interesting consideration about sensor networks is that, in several applications, once the network is deployed, it is left unattended for their whole operational lifetime.

Thus designing energy-aware communication protocols is critical for MANETs in general, and for sensor networks in particular. Also, the sensing and processing tasks the node will execute must be accounted for in the energy consumption. Thus, understanding energy consumption due to processing and communications tasks provides knowledge to define the appropriate duty cycle for nodes. The use of duty cycles is a common power conservation approach in sensor network deployments [46, 80]. This approach allows the sensor network node to alternate between active, idle and low-power periods, thus saving energy.

Research on network protocols for sensor networks has been attracting considerable attention from the networking research community. However, most efforts to-date focus on developing energy-aware protocols. Notable examples of recently proposed MAC protocols include: S-MAC [84], TRAMA [61] and TMAC [81]. There are also protocols for data collection, like diffusion [34] and SPIN [40]; data aggregation [71]; topology control [14], clustering [28], and so on. In general, one of the main goals driving the design of these protocols is to optimize network communications, in order to save energy and thus extend network lifetime.

Besides the communications aspects, one must also look into the computation required by sensing and data processing. Most of current research on sensor network protocols focus on applications that require low sampling rate and low bandwidth, such as applications that make use of sensors like humidity or temperature. The main assumption in these scenarios is that communications cost dominate in sensor networks. But what happens when more sophisticated sensors are used, like accelerometers or magnetometers? Doherty et al. [20] present three different applications, namely environmental monitoring, earthquake detection and tracking, each of which has a different component that dominates energy cost. In the environmental

monitoring scenario, the goal is to monitor environmental conditions in an office building, and sensors include temperature, humidity and light control. Since sensor sampling rate is low and nodes are within one hop of sink, communications dominate the energy consumption. In the earthquake detection scenario, the frequency of oscillation at several locations throughout a structure is monitored. To do so, 3 axis accelerometers and 3 axis magnetometers are used, and a typical sensor sampling rate is 100 Hz. In this scenario energy costs of sensing and communication are equivalent. The last scenario presented is the tracking scenario, where vehicles are sensed and tracked in a desert in a military setting, using magnetometers, microphones and accelerometers. Several nodes will behave only as data-forwarders. For the sensor nodes in this scenario, sensing tasks consume more energy than communications.

Visual sensor networks (VSNs) are another example of systems employing more complex sensors. i.e., cameras. This kind of sensor networks have very different requirements when compared to sensor networks consisting of scalar sensors, such as humidity and temperature. VSNs' specific requirements include: (1) ability to handle flows with different priorities; (2) ability to manage resources such as bandwidth and power; (3) ensure guaranteed delay and delay jitter for inter-camera communication; (4) ability to route with high priority visual data streams from cameras detecting interesting events to the final user; (5) cameras may cooperate, exchanging information with specific QoS requirements; (6) knowledge of current state of network (bandwidth, energy available on neighbors, etc); (7) and self-organization and self-management.

Furthermore, using cameras as sensors implies that sensing (i.e., image acquisition) and processing (i.e., image compression and analysis) are CPU-intensive tasks. Additionally,

computer vision algorithms might be applied to minimize the amount of data to be transfered, or to add intelligence to the node. An intelligent visual sensor node would be able to make decisions about when to track an object, when to hand off the object to a neighbor camera node in a better position, and when and what type of data to send to the end user at sink nodes. Depending on the amount of energy remaining on the node itself and on the network resources available, that node might decide to send a raw video stream (or sequence of images), a compressed video stream (or sequence of image), or some high-level representation of the object been tracked (e.g., a byte with an object descriptor).

As we mentioned before, the use of duty cycles is a common power conservation approach in sensor network deployments. Often times, the node duty cycle is defined beforehand. For example, in [20] node tasks are determined based on available power. Similarly, in [46] the duty cycle of the nodes is defined based on the available energy and the expected sensor network lifetime. In this paper, Mainwaring et al. describe the system requirements for habitat monitoring and its deployment on the Great Duck Island. The expected sensor network lifetime is nine months, which would include the seasonal changes of plants and animals to be monitored. Then considering the energy available per day and the cost of all operations to be performed (sensors, data transmission, maintenance tasks, and OS), the daily set of operations was determined, and therefore its duty cycle.

Another approach is the one where the end user at the sink node will determine the tasks a given node will execute, and thus its duty cycle. In TinyDB [45], the end user determines the type and frequency of data that will be sensed through a SQL-like interface. It also provides protocols that allow filtering, aggregation and routing of the data to the sink. Levis et al [42]

make use of Application Specific Virtual Machines(ASVM) to achieve node re-tasking. This centralized approach is more flexible than a pre-determined duty cycle, but depends on the end user to optimize data collection, besides incurring in network traffic overhead.

If using intelligent sensor nodes, a third approach may be used: the node itself will determine its duty cycle, considering the application requirements (minimum and maximum sampling rate, type of data to collect, etc), remaining energy, network status, and neighbor information. In order to implement this approach, nodes would need to obtain more information about current battery usage, network state, and future tasks to be done. With all this information available, sensor nodes could make their decisions on when to send data, when to turn themselves off, or when to collect data, and so on. To address this problem, we understand that an energy model that takes into account communications, processing and sensing is necessary. Such model can be used by protocols designers and sensor network designers. It can also be used by a framework that would make decisions about sensor node activity.

Another aspect that needs to be addressed is the sensor node hardware platform. As mentioned before, in most of current work on sensor networks protocols and architectures, communication is considered the main source of energy consumption and Motes [82] are used as the sensing devices. Motes are a good platform for sensors that do not require much processing, such as temperature, humidity, light, or even accelerometers and magnetometers. Another advantage of the Motes platform is that its hardware is simple enough that the energy consumption can be calculated by considering each of its components, like Doherty et al. explains in [20]. But when the sensor network platform has a more complex hardware, such as a Stargate [17], a hand-held or a laptop, the energy consumption due to processing can not be easily calculated.

From our knowledge, work done on modeling energy consumption for processing is task specific. For example, Barr and Asanovic [5] used the Skiff platform, which is based on the iPAQ hardware, to perform measurements of energy consumption in wireless Ethernet card, StrongARM CPU, DRAM and Flash memory, under different compression algorithms. Also, Flinn and Satyanarayanan [24] measured energy consumption while profiling system activity, so that energy consumption sources could be identified, and if not in use, turned off. They also look at data compression to optimize energy consumption.

1.2 Problem Statement

The main focus of this work is to understand the trade-offs between computation and communication in power-constrained networks, and mainly in visual sensor networks. Since we are targeting applications running multiple nodes that collaborate, it is important to look at the energy consumption trade-offs not only from the perspective of an individual node, but also consider how the tasks being executed on a particular node will affect the overall network.

In the case of the Meerkats project, this work is the basis to develop a resource manager, which will use different parameters (such as event detection rate, amount of charge available on the battery, information received from neighboring nodes, etc.) to determine the sequence of tasks to be executed on a given node.

1.3 Contributions

Our main contributions which are further elaborated in the remainder of this thesis are as follows:

- An energy model for communications, which was implemented under GloMoSim and QualNet network simulators (code already contributed to the maintainers, and also the GloMoSim code was made available on the webpage `http://www.soe.ucsc.edu/~cintia/energy-glomo.html`).
- Characterization of the energy consumption of two different wireless platforms, namely a laptop and the Stargate [17].
- Extension and validation of the on-board battery monitoring capability on the Stargate, which was contributed to the Stargate community and is available at `http://www.soe.ucsc.edu/~cintia/batmon.html`.
- Duty cycle prediction based on elementary tasks composition for a visual sensor network.
- A simple lifetime prediction model based on elementary task composition and experiments on lifetime of a visual sensor network for different duty cycles.
- Analysis of the Meerkats node lifetime under different duty cycles and trade-offs.

1.4 Publications

- Cintia B. Margi and Katia Obraczka, "Instrumenting Network Simulators for Evaluating Energy Consumption in Power-Aware Ad-Hoc Network Protocols". In Proceedings 12th Annual Meeting of the IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), Volendam, Netherlands, October 5-7, 2004.

- Marcelo M. Carvalho, Cintia B. Margi, Katia Obraczka, and J.J. Garcia-Luna-Aceves, "Modeling Energy Consumption in Single-Hop IEEE 802.11 Ad Hoc Networks". In Proceedings IEEE ICCCN 2004, Chicago, IL, October 11-13, 2004. This paper is candidate for best paper award.

- Marcelo M. Carvalho, Cintia B. Margi, Katia Obraczka, and J.J. Garcia-Luna-Aceves, "On the Energy Efficiency of IEEE 802.11 DCF in Single-Hop Ad Hoc Networks". Under submission, January 2006.

- Cintia B. Margi, Katia Obraczka, Roberto Manduchi; "Characterizing System Level Energy Consumption in Mobile Computing Platforms". In Proceedings of the IEEE WirelessCom 2005, June 13-16, 2005.

- Cintia B. Margi, Vladislav Petkov, Katia Obraczka, Roberto Manduchi; "Characterizing Energy Consumption in a Visual Sensor Network Testbed". In Proceedings of the 2nd International IEEE/Create-Net Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom 2006), March 1-3, 2006.

- Cintia B. Margi, Roberto Manduchi, Katia Obraczka; ”Energy Consumption Trade-offs in Visual Sensor Networks”. In Proceedings of 24th Brazilian Symposium on Computer Networks (SBRC 2006), May 29 - June 02, 2006.

- X. Lu, C. Margi, G. Stanek, G. Zhang, R. Manduchi, K. Obraczka; ”Energy-Performance Tradeoffs in a Wireless Camera Network”. Under submission (EURASIP JES), May 2006.

- Cintia B. Margi, Katia Obraczka, Roberto Manduchi; ”Energy Consumption Trade-offs and Lifetime Prediction in Visual Sensor Networks”. Under preparation, June 2006.

1.5 Thesis Outline

The remainder of the thesis is organized as follows. Chapter 2 presents the work done on energy model for wireless communications. Chapter 3 presents the energy measurements for laptops and Stargate platform, as well as the extension and validation of the on-board battery monitoring capability on the Stargate. Chapter 4 describes the duty cycle modeling by task composition, as well as a discussion on such approach. Chapter 5 presents the experiments on the visual sensor node lifetime, as well as a prediction model based on elementary task composition. Chapter 6 concludes the thesis summarizing the main ideas presented, and pointing future directions.

Chapter 2

An Energy Model for Communications

This chapter presents the work done on network simulators instrumentation for evaluating energy consumption in power-aware ad-hoc network protocols [49], and its use to evaluate an analytical model to predict energy consumption in single-hop IEEE 802.11 ad hoc networks [13].

It is worth to point out that the code implemented for the network simulators GloMoSim and QualNet was already contributed to the maintainers, and also that the GloMoSim code was made available on the webpage `http://www.soe.ucsc.edu/~cintia/energy-glomo.html`.

2.1 Introduction

Wireless ad-hoc networks are also known as ”networks without a network” since they do not use any fixed infrastructure. Participating nodes in these networks are typically battery operated, and thus have access to a limited amount of energy. Frequently, once nodes

are deployed, their batteries cannot be easily re-charged. Sensor network nodes are a typical example as some of them have very limited battery life; moreover, once deployed, a sensor network may be left unattended for its entire operational lifetime. This is due to the fact that sensor networks may be deployed in wide, remote, unaccessible areas.

The energy-constrained nature of ad hoc networks in general, and sensor networks in particular, calls for protocols that have energy efficiency as a primary design goal. Research on power-aware protocols has been very active and spans multiple layers of the protocol stack. As a result, several energy-efficient medium-access control (MAC)- and routing protocols have been proposed.

In order to evaluate and compare power-aware protocols in terms of their energy efficiency as well as assess the effectiveness of cross-layer mechanisms to achieve energy savings, accurately accounting of the energy consumed by data communication activities is crucial. Such accounting must be as close to reality as possible, taking into consideration all radio states, i.e., energy spent not only while transmitting and receiving a packet, but also while in idle, overhearing, or sleep modes. Frequently, the evaluation of network protocols is carried out using network simulators such as `ns-2` [36], `GloMoSim` [87], and `QualNet` [66]. As explained in detail in Section 2.2, the models used by these simulators to account for energy consumption by data communication activities are not accurate. More specifically, the models employed either do not consider all radio states or do not take into account the different energy levels they consume. Furthermore, most current simulators do not automatically measure energy consumption, leaving it up to the protocol designer to explicitly write code to account for it. And, clearly, depending on the layer of the protocol stack, energy consumption accounting can become quite

cumbersome and inaccurate. Not to mention the duplication of effort as code to accomplish the same task is written several times for the same simulation platform.

The contribution of this work is two-fold. First, it describes our work on instrumenting network simulators to enable them to adequately and accurately account for the energy consumed by ad hoc network protocols' communication-related tasks. This is accomplished by explicitly accounting for low-power radio modes and considering the different energy costs associated with each possible radio state, i.e, transmitting, receiving, overhearing, idle, sensing, and sleeping. Second, our energy consumption instrumentation also allows the energy accounting to be done automatically by the simulator irrespective of what layer of the stack the protocol designer is working. For example, in [13], the analytical model presented for energy consumption in IEEE 802.11 single-hop wireless networks is compared to the accounting provided by QualNet with our energy consumption instrumentation. The results match quite closely differing by at most 15%.

We then validate our energy consumption model by comparing (1) simulation results using the GloMoSim/QualNet [66] simulation platform with and without our instrumentation for the IEEE 802.11 DCF [18], (2) analytical results for both 802.11 and S-MAC [84] (a power-aware MAC designed for sensor networks), and (3) simulation results reproducing testbed experiments obtained for the S-MAC protocol [84]. Finally, we showcase the ability of a network simulation platform instrumented with our energy consumption model to evaluate power-aware protocols by comparing S-MAC against 802.11. We also evaluate the energy consumption of AODV [56] and DSR [38]. Although we implement our energy consumption instrumentation on the GloMoSim/QualNet platform, it can easily be ported to any existing network simulators

(e.g., ns-2 [36]).

The remainder of this chapter is organized as follows. Section 2.2 reviews related work. In Section 2.3, we describe our energy consumption instrumentation. Our experimental methodology and model validation results are presented in Section 2.4. Results comparing MAC protocols, namely S-MAC and 802.11, and routing protocols, namely AODV and DSR, are presented in Section 2.5 and results from the evaluation of the analytical model for energy consumption in IEEE 802.11 single-hop wireless networks [13] is presented in 2.6. Section 2.7 concludes the chapter.

2.2 Related Work

This section addresses energy models available, related research done on energy consumption in Network Interfaces, energy models in current available network simulators, and MAC power-aware protocols.

2.2.1 Energy models and energy consumption in Network Interfaces

In [77] the power consumption of some network interface cards (NICs) was measured when used by different end-user devices. They also report on transport- and application-level strategies to reduce energy consumption by NICs. Later, [23] reported detailed energy consumption measurements of some commercially-available IEEE 802.11 NICs operating in ad hoc mode. Energy consumption models using linear equations were also introduced. For example, Equation (2.1) models data transmission and reception, where the coefficients m and b

depend on the type of communication, i.e., broadcast, unicast, or packet discarded, and were determined empirically.

$$E = m * size + b \tag{2.1}$$

Along the same lines, [22] assessed the impact of transmission rate, transmit power, and packet size on energy consumption in a typical wireless network interface. In most previous measurements, however, the focus was on the characterization of energy consumption during the many modes of operation of a NIC (transmit, receive, idle, etc.), under extremely simple scenarios, e.g., only two nodes operating in ad hoc mode, with one node acting as the sender and the other as the receiver.

In [7], a power consumption model for sensor networks consisting of three components (1)sensor-, (2)computation-, and (3)communication cores is proposed. The sensor core, which assumes that the energy used to sense a bit is constant, defines power consumption as presented in Equation (2.2), where r is the bit-rate and a typical value for α_3 is 50 nJ/bit. Power consumed in the computation core is attributed to data aggregation and is defined in Equation (2.3), where α_4 can vary from a few pJ/bit to 10 nJ/bit, and η_{agg} is the number of streams being aggregated. Finally, the communication model has two portions: transmission and reception. Power consumption due to reception is proportional to r, the bit rate, as shown in in Equation (2.4), where α_{12} is 135 nJ/bit. Power spent on transmission, shown in Equation (2.5), also depends on the bit rate r, as well as the distance between nodes ($d(n_1, n_2)$) and path loss index n. However, no validation of the energy model is presented in this work. Moreover, energy spent in data processing is attributed to data aggregation and does not account for local

processing (e.g., data compression).

$$P_{sense} = \alpha_3 * r \tag{2.2}$$

$$P_{comp} = \eta_{agg} * \alpha_4 * r \tag{2.3}$$

$$P_{rx} = \alpha_{12} * r \tag{2.4}$$

$$P_{tx}(n_1, n_2) = (\alpha_{11} + \alpha_2 * d(n_1, n_2)^n * r) \tag{2.5}$$

An energy-aware simulation model, which considers a network consisting of multiple nodes, where each node is composed by a local request queue, a microprocessor, an external request queue, another processor, a service queue and a service provider is describe in [78]. All components are random variables. The total energy consumption on a node is the sum of the energy spent by node components, energy consumption for transmitting a data packet, and energy consumption for receiving a data packet. Although the model considers energy spent with processing and buffering requests, the radio model is quite simple and does not include a low-power radio mode, which is crucial for development of power-aware protocols.

A simple energy model is introduced in [30] to evaluate power-aware protocols in the LEACH project [29]. LEACH (Low-energy Adaptive Clustering Hierarchy) is a clustering-based protocol that uses randomized rotation of cluster-heads to evenly distribute the energy load among the sensors in the network. In the energy model, the energy spent on transmission is given by the energy dissipated by the radio electronics and the power amplifier, while the energy spent by the receiver is given by the the energy dissipated by the radio electronics. Since the energy necessary to amplify the signal depends on its attenuation, and the attenuation depends on the distance, the energy dissipated by the radio electronics is proportional to d^2 for

short distances and to d^4 otherwise. Using this same energy model, [21] examine the energy consumption in a wireless sensor network with two distinguished organizations: single layer versus clustered.

In [43], energy consumption in ad-hoc mobile terminals is modeled using the Advanced Configuration Power Interface [31], or ACPI, an open standard that allows computer systems to implement motherboard configuration and power management functions. ACPI was used to measure energy consumption due to transmission/reception. The resulting energy consumption model includes two states: **high consumption state**, where the host receives and transmits, and **low consumption state**, where the node receives or is in idle. While this approach to model battery discharge empirically is based on values that laptop power management would see in real systems, it is platform-dependent.

A comparison of transmitter's energy consumption of TCP Reno, Newreno and SACK using a laptop testbed under different network conditions (e.g.,variable round trip times, random loss, bursty loss, packet reordering, etc.) is presented in [69]. The results shows that SACK consumes the least total energy, but if the energy cost for idle states is much lower than receiving state, it performs poorly.

A sensor networks testbed of PicoNodes [62] is used to evaluate energy consumption on different radio states and different traffic types.

Developers of power-aware protocols often implement their own energy models (e.g., [61], [39]) at the layer of the protocol stack they are working on. An alternative is to use what is available in current network simulators or add extensions to obtain the desired results. Thus results from different efforts cannot be compared directly. For example, [83] does an evaluation

of topology control protocols using `ns-2` and the parameters for the radio are from [77].

2.2.2 Network simulators

2.2.2.1 GloMoSim and QualNet

In QualNet 3.6 [66], the energy consumption model for communication is implemented in the physical layer. The simulator currently includes six different physical layer models: 802.11a, 802.11b, abstract, GSM, FCSC prototype, and link16.

The current radio model only defines four states: idle, sensing, receiving (RX) and transmitting (TX); there is no state corresponding to the low-power energy mode where the radio cannot transmit or receive (usually referred as "sleep" state). For energy consumption purposes, QualNet considers that the radio is either in TX or RX states (in ad hoc network mode). If the radio is in RX, it spends 900 mW. The power consumption for transmitting signals is calculated as shown in Equation (2.6).

$$(TxPowerCoef * txPower + TxPowerOffset) * txDuration \tag{2.6}$$

The values of $TxPowerCoeff$ and $TxPowerOffset$ are statically defined based on the WaveLAN specifications, and are assigned the values of 16/sec, and 900mW (the same value as consumed in RX mode). $txPower$ is proportional to the distance the signal is supposed to travel. For each frame transmitted, the energy spent is calculated and added to the energy consumption statistics variable. Once the simulation ends, total simulation time is multiplied by the cost of being in RX mode and added to the energy consumption statistics.

Clearly, QualNet's current energy model is not realistic enough as it does not distin-

guish between RX, overhearing and idle states. Furthermore, it does not include a low-power, sleep state. Another drawback of QualNet is that there is no energy consumption information during simulation time. The amount of energy consumed is only available at the end of a simulation run. For the purpose of evaluating energy consumption, this is not a real issue, but if the goal is to simulate nodes failing when running out of battery and/or have energy-aware protocols looking at energy information to make certain decisions, this model is not adequate.

GloMoSim [87], QualNet's precursor provides an energy model that is very similar to QualNet's.

2.2.2.2 ns-2

The energy model supported by `ns-2` [36] includes four states: idle, sleep, receiving (RX) and transmitting (TX). Every node starts with an initial energy level and consumes energy as it transmits and receives data. Periodically, nodes update the amount of energy spent in idle state.

Energy consumption for TX, RX, and idle states have default values of $Pt_{consume} = 0.660$, $Pr_{consume} = 0.395$ and $Pi_{consume} = 0.0$, respectively. However, to our knowledge, there is no mention of the energy consumption in sleep state. While in sleep state, `ns-2` keeps an accounting of the time spent in this state. There is a note in the documentation that says that "time in the sleep mode should be used as credit to idle time energy consumption", which implies that $Ps_{consume} = 0.0$.

Some of the weaknesses of the current energy model employed by `ns-2` include the lack of calculation of the energy consumed in sleep state which does not allow a fair comparison

between protocols that explicitly use this mode for power savings. A minor drawback is that, if the user does not set the value of $Pi_{consume}$, the total energy consumption will reflect only what was spent in RX and TX.

2.2.3 Power-aware MAC protocols

In this work, we use two MAC protocols, namely IEEE 802.11 DCF [18] and S-MAC [84], to validate and showcase the proposed instrumentation. We describe their main features below.

The IEEE 802.11 DCF [18] is a contention-based protocol based on carrier sensing with collision avoidance (CSMA/CA). IEEE 802.11 performs both physical and virtual carrier sensing. Virtual carrier sensing is achieved by sending information about duration of each frame in the headers which is used by stations as an indication of how long the channel will be busy. After this time is elapsed, stations can sense the channel again. In order to solve the "hidden terminal" problem and avoid data frame collisions, the RTS-CTS handshake is used. Two power management mechanisms are supported: *active* and *power-saving* (PS).

The Sensor MAC protocol [84], or S-MAC, was developed with power savings as one of its design goals. It also falls into the contention-based protocol category but achieves energy efficiency by making use of low-power radio mode. Nodes alternate between periodic sleep and listen periods. Listen periods are split into synchronization and data periods. During synchronization periods, nodes broadcast their sleeping schedule, and, based on the information received from neighbors, they adjust their schedule so that they all sleep at the same time. During data periods, a node with data to send will contend for the medium (RTS-CTS exchange).

If the node acquires the medium or if it has data to receive, it will not sleep in the next period and the data will be exchanged. After that, if there is still enough time in the sleep period, the node goes to sleep. If a node does not have data to transmit or receive, it will sleep.

There has been considerable research activity in power-aware MAC protocols. Besides S-MAC, another notable example is the T-MAC [81] protocol, a contention-based MAC for wireless sensor networks which uses an active/sleep duty cycle. TRAMA [61] is an example of a power-aware scheduled-based (time-slotted) MAC Protocol. It establishes transmission schedules in a way that it is self adaptive to changes in traffic, node state, or connectivity.

2.2.4 Multi-hop ad-hoc networks routing protocols

Two well-known multi-hop ad-hoc networks (MANETs) routing protocols are DSR and AODV. Next, we summarize their main features.

Dynamic Source Routing (or DSR) [38] is a source routing protocol, i.e. each packet must carry its route to the destination. In order to discover a route to a given destination, a source S floods a route request (RREQ). Then each node forwarding the route request will append its own identifier. When the destination D receives the first route request, it sends a route reply (RREP) to S, obtaining the route by reversing the route appended to the route request received. Note that the route reply contains the route to S. Node S will then receive the route reply and use the route to send data to D. Again, the data packets will contain in their headers the route to D, which will be used by forwarding nodes to make routing decisions. Since DSR needs to add the route in the packet header, data packets will have their sizes increased.

Ad-hoc On-Demand Distance Vector Routing (or AODV) [56] is an on demand rout-

ing protocol, which maintains routing tables at nodes and thus packets do not have to carry their route to the destination. When a node S needs a route to a node D, it will transmit a route request. Nodes forwarding the route requests add an entry in their routing tables pointing to the source of the route request. When the destination D receives the route request, it will answer with a route reply, using the path set up when the route request was forwarded. AODV does not increase the header size like DSR does, but AODV needs to time stamp entries in the routing table to avoid them becoming stale.

2.3 Energy consumption instrumentation

The main goal of our instrumentation is to provide a "common ground" through which the effectiveness of different power aware techniques, at a specific layer of the protocol stack or across different layers, can be evaluated. Our model is simple and uses an approach similar to [11]. However, our instrumentation does account for time and energy consumed in all radio states, including low-power sleep mode.

Due to the shortcomings of the energy models available in existing simulation platforms (see Section 2.2 for details), developers of power-aware protocols often implement their own energy models (e.g., [61], [39]), usually at the layer of the protocol stack they are developing at. Another benefit of the proposed instrumentation is that, since it is implemented at the physical layer, it can be used by any protocol layer.

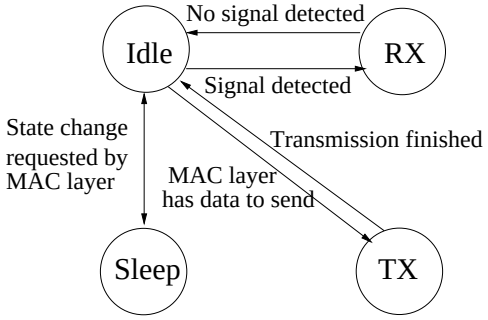


Figure 2.1: State diagram for radio modes

2.3.1 Description

The proposed energy model considers all possible radio operation modes, namely:

- **Transmitting**, i.e., radio is transmitting data;
- **Receiving**, i.e., radio is effectively receiving data;
- **Overhearing**, i.e., radio is receiving data that is not destined to the node[1];
- **Idle**, i.e., radio is ready to receive or transmit;
- **Sensing**, i.e., radio has detected some signal, but is not able to receive it;
- **Sleeping**, i.e., radio is in low power, and this is not able to receive or transmit.

Note that *sensing* and *overhearing* states are a special case of the *receiving* state. Figure 2.1 shows the state diagram, which depicts the main radio states and how state transitions occur.

[1]For deciding whether the radio is receiving or overhearing, the energy model needs information from the MAC layer regarding whether the node is the recipient of the frame.

The power dissipated by the radio can be calculated using Equation 2.7, where V and i are the voltage and current specific to the radio being used and are typically available from the radio data sheet. The time the radio spends in a certain state depends on the packet size and the transmission rate and is given by Equation (2.8).

$$P = V * i \tag{2.7}$$

$$t = PacketSize/TransmissionRate \tag{2.8}$$

Thus, for each state, energy consumption is calculated as presented in Equation (2.9), where α_y represents the power dissipated by the radio while in state *y*, and t_y represents the time spent in state *y*. State *y* can be transmission, reception, idle, sleep or sensing. Note that the values of α_y are radio-specific.

$$E_y = \alpha_y * t_y \tag{2.9}$$

2.3.2 Implementation

The energy model was implemented at the radio/physical layer of both GloMoSim and QualNet. The implementation includes: (1) the necessary physical layer infrastructure to account for all possible radio modes (as specified above), and (2) an interface between the physical- and MAC layers to control the radio modes (e.g., switch radio on/off, overhearing versus reception, etc.).

The physical layer support for the energy consumption instrumentation includes: (1) the addition of the SLEEP state, (2) addition of a data structure for the energy model, (3) and implementation of energy consumption accounting functions. The radio/physical implementations

were modified such that these functions are called every time a radio state change occurs.

Since the radio layer now supports the ”sleep” state, it is necessary to provide functions for the MAC layer to set the radio state to and from sleep mode. Functions *GlomoEnergyRadioWakeUp* and *GlomoEnergyRadioGoToSleep* provide this functionality. Another aspect where the need of interaction between MAC- and physical layers become clear is in order to identify if a received packet was in fact received or overheard (i.e., the destination for the packet is not the current node). The energy model assumes that all packets received are overheard, and thus the function *GlomoEnergyUpdateEnergyRx* should be used every time a received packet is destined to the node. If this function is not used, all energy due to receiving will be accounted as overhearing.

Each time the radio changes state, energy consumption information is updated. In order to support power-aware protocols, current energy spent information in a node can be obtained by using the function $GlomoGetCurrentEnergySpent$. For example, a power-aware routing protocol may want to examine the current energy information at nodes to compute paths that only include nodes whose energy are above a certain threshold.

Through a configuration file, the user defines the energy consumption parameters (e.g., α_y in Equation (2.9)). Statistics provided by the energy model include: total energy consumption, energy consumption per state, time spent in each state (including or not a “warm up” period).

2.4 Validation

We validate the energy consumption instrumentation analytically and through simulations. For the latter, we compare QualNet/GloMoSim instrumented with our model against the original QualNet/GloMoSim. The protocol used is IEEE 802.11 DCF, which uses transmitting, receiving and idle modes. Since the idle state typically consumes the same energy as receiving, we can directly compare our model to GloMoSim/QualNet's which only considers transmitting and receiving states.

The other validation step is to compare analytically the results obtained when a simple topology is used. We performed this step for both IEEE 802.11 DCF and S-MAC. Finally, we validate the model by reproducing experiments conducted on a testbed implementation of S-MAC [84] and comparing the results from the testbed with our simulations.

All simulation results presented are averaged over 10 runs with different seeds.

2.4.1 Instrumented versus original GloMoSim/QualNet

In this experiment, the goal is to validate our model by comparing the energy consumed by IEEE 802.11 DCF using original GloMoSim/QualNet against results obtained using GloMoSim/QualNet with our instrumentation.

We used the default values for all parameters in the configuration file, i.e., the transmission rate is set at 11 Mbps, and the power consumption is 900 mW for both receiving/idle and transmitting states. The transmission range for each node is 100m (receiver threshold is -75dB).

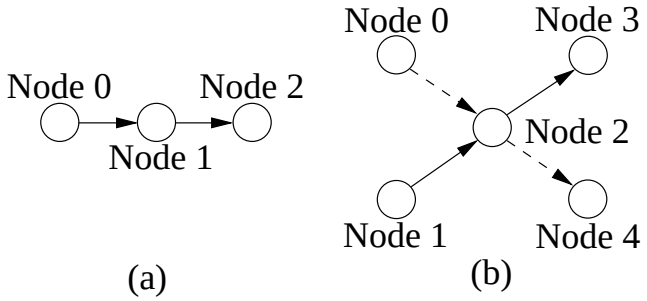


Figure 2.2: (a) Three-node topology and (b) Five-node topology used in validation.

Table 2.1: Total energy consumption in original versus instrumented GloMoSim/QualNet.

	Original	Instrumented
Node 0	224999.46	224999.46
Node 1	224998.74	224998.74
Node 2	224999.28	224999.28

The scenario used is a string topology with 3 nodes (two-hop topology), as shown in Figure 2.2(a). All nodes are stationary and routes are static. CBR traffic is generated from node 0 to 2 40 times with 5 second interval; the data size is 200 bytes. A simulation run lasts 250 seconds.

Table 2.1 shows the total energy consumed by each node for both original- and instrumented GloMoSim/QualNet. Note that IEEE 802.11 DCF only uses transmitting, receiving and idle modes, where consumption in idle state is typically the same as in receiving. Therefore, we can directly compare our model to GloMoSim/QualNet's, which only considers transmitting and receiving states. Thus, for this particular case, the values obtained are exactly the same.

2.4.2 Analytical versus simulation results

For this validation step, we use the specifications for the TR1000 [63] radio, which is designed for short-range wireless data communication, supports transmission rates of up to 115.2 Kbps, and has the sleep state built in. Power consumption is 13.5 mW, 24.75 mW and 15 μW in receiving/idle, transmitting, and sleeping states, respectively. The transmission range for each node is set to 100m (receiver threshold is -75dB). Data rate is 19.2 Kbps. Packet sizes are 20 bytes for RTS, 14 bytes for CTS and ACK, 380 bytes for DATA, and 24 bytes for SYNC.

The topology used is composed of five nodes as shown in Figure 2.2(b). Nodes 0 and 1 are sources, 3 and 4 are sinks, and 2 must route all the traffic in this two-hop network. The five-node topology was chosen in order to provide a scenario, which includes the possibility of collisions, overhearing and sensing. Static routes were set, so there is no influence from routing protocols. The experiment simulates 3 seconds of real time, which is the time needed to transmit one packet from sources to destinations.

2.4.2.1 802.11

Considering that each source is going to transmit one data packet, Table 2.2 shows the transmitted and received (and overheard) packets per node when 802.11 is used.

Based on the information on Table 2.2, data rate and packet size, we can calculate the time each node spent in each state, shown in Table 2.3. Note that the time corresponding to receiving state is split between receiving and overhearing. The time the radio is not transmitting or receiving is spent in idle state. Also IEEE802.11 doesn't support sleep state and thus no time should be spent in it, and thus it is omitted in the table.

Table 2.2: Packets transmitted and received per node for 802.11.

Node	Transmitted	Received	Overhears
0	RTS+DATA	CTS+ACK	(CTS+ACK) + 2*(RTS+DATA)
1	RTS+DATA	CTS + ACK	(CTS+ACK) + 2*(RTS+DATA)
2	2*(CTS+ACK) 2*(RTS+DATA)	2*(CTS+ACK) 2*(RTS+DATA)	
3	CTS+ACK	RTS+DATA	(RTS+DATA) + 2*(CTS + ACK)
4	CTS+ACK	RTS+DATA	(RTS+DATA) 2*(CTS + ACK)

Table 2.3: Analytical model: time spent in each relevant state for 802.11.

Node	TX	RX	Overhearing	Idle
0	0.166	0.011	0.344	2.479
1	0.166	0.011	0.344	2.479
2	0.356	0.369	0.000	2.275
3	0.011	0.166	0.189	2.634
4	0.011	0.166	0.189	2.634

Table 2.4: Simulation: time spent in each relevant state for 802.11.

Node	TX	RX	Over-hearing	Sensing	Idle	Sensing + Idle
0	0.1787	0.0121	0.3695	0.2028	2.2369	2.4397
1	0.1787	0.0121	0.3695	0.2028	2.2369	2.4397
2	0.3816	0.3815	0.0000	0.0000	2.2369	2.2369
3	0.0121	0.1787	0.2028	0.3695	2.2369	2.6064
4	0.0121	0.1787	0.2028	0.3695	2.2369	2.6064

To obtain the corresponding simulation results, we use QualNet and send one packet from each source node to the corresponding sink node according to the topology in Figure 2.2(b) with 802.11 as the underlying MAC protocol. Table 2.4 presents time spent in each state, i.e., TX, RX, overhearing, sensing, and idle.

The small difference between analytical and simulation results for TX, RX and overhearing is due radio synchronization and internal delays intrinsic to the simulation. For each packet to be transmitted, the simulator will add the synchronization time and delays to the transmission time, and thus it propagates to all results. The time spent in sensing is caused by nodes nearby, but not in range (radio is not able to lock the signal). For example, in the topology of Figure 2.2(b), node 0's sensing time is given by the sum of the transmission times of nodes 1, 3 and 4. Because the sensing range depends on the radio properties (e.g., transmission power of the neighbors, radio sensitivity for the receiving node, distance between nodes, etc), we decided not to consider this state in the analytical model. Thus when comparing simulation results to the analytical results, sensing time should be added to idle time, resulting in values presented in the last column of Table 2.4.

Table 2.5: Packets transmitted and received per node for S-MAC.

Node	Transmitted	Received	Overhears
0	RTS+DATA + SYNC	CTS+ACK + SYNC	CTS + 2*(RTS+DATA)
1	RTS+DATA + SYNC	CTS + ACK + SYNC	CTS + 2*(RTS+DATA)
2	2*(CTS+ACK) + 2*(RTS+DATA) + SYNC	2*(CTS+ACK) + 2*(RTS+DATA) + 4*SYNC	
3	CTS+ACK + SYNC	RTS+DATA + SYNC	(RTS+DATA) + 2*CTS
4	CTS+ACK + SYNC	RTS+DATA + SYNC	(RTS+DATA) + 2*CTS

2.4.2.2 S-MAC

Next, we repeat the same exercise using S-MAC as the underlying MAC protocol. By using S-MAC, other factors must be taken into account. Besides the time spent in transmitting and receiving data, it is necessary to account also for the transmission of SYNC frames[2]. S-MAC makes use of low-power sleep state by switching nodes to sleep if a CTS, DATA, or ACK from another node is received. Table 2.5 summarizes the packets exchange for S-MAC.

Considering packets exchange in Table 2.5, data rate and packet size, we can calculate the time each node spent in each state, shown in Table 2.6. Note that the time corresponding to receiving state is split between receiving and overhearing. In order to compute the time spent in idle state, we calculate how many listen periods fit within the 3-second simulation runs; from that, we subtract the time spent transmitting and receiving SYNCs, RTSs and CTSs. Similarly, we can estimate the time spent in sleep state by calculating how many sleep periods fit within

[2]Nodes periodically exchange SYNC frames in order to identify their one-hop neighbors and define their schedule.

Table 2.6: Analytical model: time spent in each relevant state for S-MAC.

Node	TX	RX	Overhearing	Idle	Sleep
0	0.1748	0.0217	0.3342	0.6335	1.8359
1	0.1748	0.0217	0.3342	0.6335	1.8359
2	0.3617	0.3917	0.0000	0.5685	1.6781
3	0.0217	0.1742	0.1758	0.6335	1.9948
4	0.0217	0.1742	0.1758	0.6335	1.9948

Table 2.7: Simulation: time spent in each relevant state for S-MAC.

Node	TX	RX	Overhearing	Sensing	Idle	Sleep
0	0.1842	0.0508	0.3381	0.0620	0.4592	1.9057
1	0.1811	0.0508	0.3381	0.0650	0.4592	1.9057
2	0.3974	0.4147	0.0000	0.0000	0.4592	1.7287
3	0.0262	0.2108	0.1695	0.0514	0.4592	2.0828
4	0.0242	0.2108	0.1685	0.0545	0.4592	2.0828

a simulation run, and from that subtract the time spent transmitting and receiving DATA and ACKs. Note that ideally no DATA should be overheard, because the data portion of the listen period is long enough to accommodate RTS and CTS packets.

To get simulation results, we again use QualNet to send one packet from each source node to the corresponding sink node in the topology of Figure 2.2(b) using S-MAC as the underlying MAC. Simulations run for 3 seconds after the warm-up period [3]. Table 2.7 presents time spent in each state as tracked by the energy model.

Similarly to the results obtained for 802.11, we observe a difference (10% on average) between analytical and simulation results due to radio synchronization and internal delays intrinsic to the simulator. Note that the calculation for sensing time in S-MAC is not as simple as for IEEE 802.11, since S-MAC has a sleep state. But still in order to compare analytical and

[3] S-MAC needs a 20 to 30-second warm-up period for initial neighbor synchronization

simulation results, sensing and idle time should be added: 0.5212 for node 0, 0.5242 for node 1, 0.4592 for node 2, 0.5106 for node 3 and 0.5137 for node 4.

2.4.3 S-MAC testbed versus simulation

We also validate the proposed energy model by comparing results obtained from an implementation of S-MAC on a real sensor network testbed [84] against simulations that tried to reproduce the testbed experiments. Since details of the testbed experiment were not available, an absolute comparison is not possible. Our goal is thus to obtain a qualitative evaluation of the model. Once again, we used the TR1000 [63] radio parameters as described above. The same five-node topology is used and is shown in Figure 2.2(b). Nodes 0 and 1 are sources, 3 and 4 are sinks, and 2 must route all the traffic in this two-hop network. CBR traffic with packet size of 380 bytes is sent throughout the whole simulation period. Simulation time varies with inter-arrival period in order to keep the number of packets transmitted constant for all scenarios.

The graph in Figure 2.4 shows the average energy consumed for source and intermediate nodes for each different packet inter-arrival time. When compared to the energy consumption graphs from S-MAC paper [84] (Figures 8 and 10), reproduced in Figures 2.3(a) and 2.3(b), we observe similar behavior for all nodes. The intermediate node spends more energy because it needs to forward data and thus cannot sleep as much as the other nodes. Also, the energy consumption increases because the simulation time (or the data collection time in [84]) increases with the inter-arrival period.

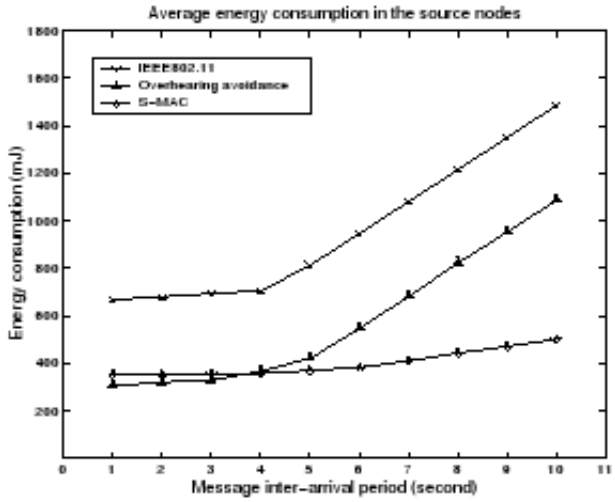


Fig. 8. Measured energy consumption in the source nodes.

(a) Energy consumption in the source node.

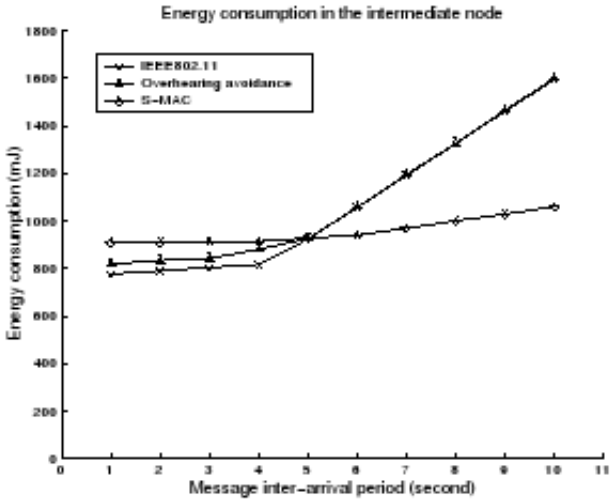


Fig. 10. Measured energy consumption in the intermediate node.

(b) Energy consumption in the intermediate node.

Figure 2.3: Measured energy consumption for S-MAC. Results reproduced from [84].

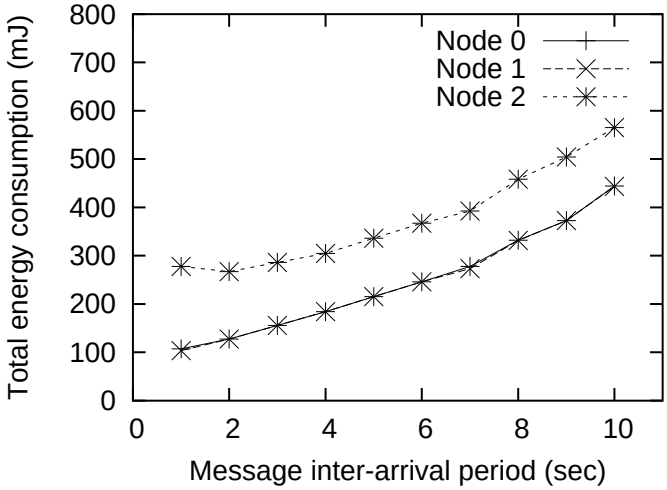


Figure 2.4: Simulations results for S-MAC energy consumption using instrumented GloMoSim / QualNet.

2.5 Protocol comparison

The goal of these experiments is to showcase how our instrumentation can help evaluate power-aware protocols, irrespective of their position in the protocol stack. In the first example, we evaluate power-aware MAC protocol, namely S-MAC. In the second example, we evaluate two well-known routing protocols for multi-hop ad-hoc networks (MANETs), namely AODV [56] and DSR [38]. We show that, through the use of an adequate energy instrumentation, it is possible to get insight into where energy is spent and how energy efficiency can be improved.

2.5.1 MAC Protocols

In this specific example, we evaluate power-aware MAC protocol, namely S-MAC. For these experiments, we employ a scenario including fifty nodes uniformly distributed over

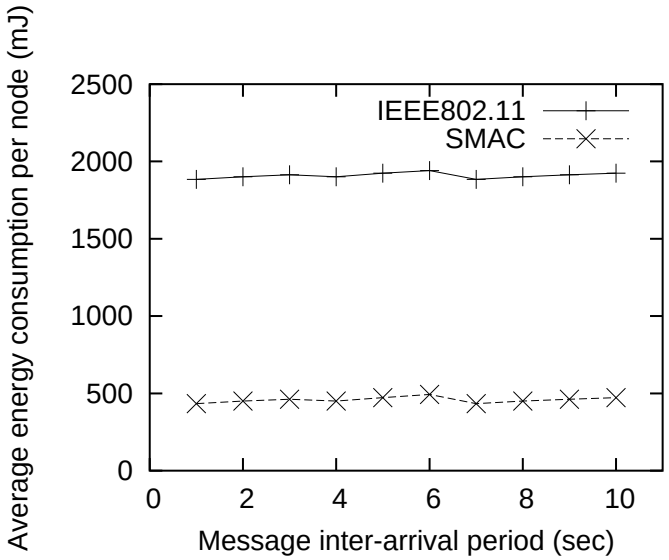


Figure 2.5: Simulation results for S-MAC and 802.11 using instrumented GloMoSim.

1000x1000m^2. The parameters in the configuration file were again set according to the TR1000 [63] radio specifications. A CBR application with 10 sources and 10 sinks sending 380-byte packets was used. AODV [56] was used as the routing protocol.

The graph in Figure 2.5 shows the average energy consumed per node for different packet inter-arrival times. Note that the different inter-arrival periods do not seem to affect the average time spent in each state because results are averaged over all nodes in the area (which might include nodes that do not transmit or receive at all).

Besides providing the overall energy consumption, our instrumentation allows a better understanding of which radio states are predominant and thus how energy savings can be achieved. Table 2.8 shows the time spent in each radio state (TX, RX, idle, sleep, sensing and overhearing) for message inter-arrival period of 1sec. These results demonstrate that the ability to use low-power sleep state yields considerable energy savings.

Table 2.8: Time spent in each state for 1sec message inter-arrival time.

Protocol	TX	RX	Overhearing	Sensing	Idle	Sleep
802.11	0.92	0.68	0.01	10.09	137.20	0.0
S-MAC	0.94	0.63	0.03	22.49	29.71	68.38

Since the instrumentation not only gives the time, but also the energy consumed in each radio state, it provides a better understanding of the protocols under consideration. For instance, in [61], the energy savings are attributed to the length of the sleep time.

2.5.2 Routing Protocols

In order to show the ability of the instrumentation to account for energy consumption irrespective of the stack layer we want to evaluate, we compare energy consumption for two MANET routing protocols: AODV [56] and DSR [38].

For these experiments, we employ a scenario including fifty nodes uniformly distributed over 500x500m^2. The parameters in the configuration file were set according to the WaveLAN radio [23], and thus the power consumption is 900mW for both receiving and idle mode, and 1400mW for transmitting mode. The transmission rate is 11Mpbs. A CBR application with 10 sources and 10 sinks sending 380-byte packets was used. Simulation runs for 150 seconds.

The graph in Figure 2.6 shows the average energy consumed per node for different packet inter-arrival times. Since the simulation had the same duration for all different inter-arrival times, the scenarios with smaller inter-arrival times were able to deliver more data packets, thus spending more time in transmission mode, and therefore had larger average energy

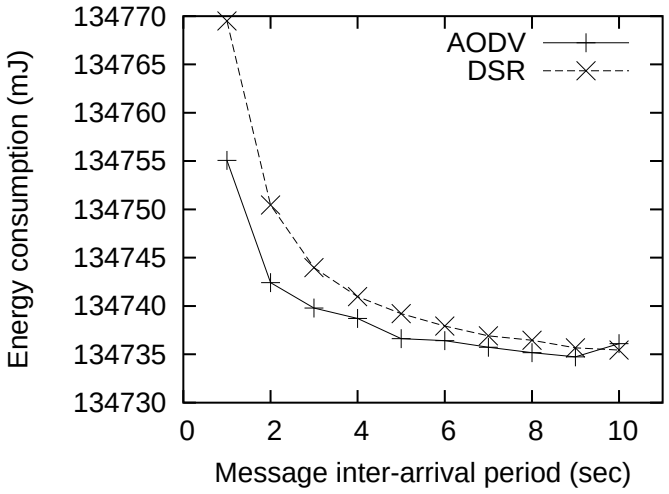


Figure 2.6: Simulation results for AODV and DSR using instrumented GloMoSim.

Table 2.9: Time spent in each state for 1sec message inter-arrival time.

Protocol	TX	RX	Overhearing	Sensing	Idle	Sleep
AODV	0.0491	0.0463	0.2214	1.8519	147.5320	0.0000
DSR	0.0779	0.0737	0.3643	2.9891	146.1956	0.0000

consumption.

In order to understand why DSR consumes slightly more energy per node than AODV, it is useful to look at the time spent in each state. Table 2.9 shows the time spent in each state for inter-arrival time of one second. In this case, DSR spends about 60% more time in transmission, receiving, overhearing and sensing states than AODV. This is due to the fact that DSR packets are longer, since they carry source routes.

2.6 Evaluation of an analytical model to predict energy consumption in single-hop IEEE 802.11 ad hoc networks

This work [13] done in collaboration with Marcelo Carvalho and J.J. Garcia-Luna-Aceves presents advances in two fundamental aspects of energy- aware protocols: it introduces an analytical model to predict energy consumption in single-hop IEEE 802.11 ad hoc networks (under ideal channel conditions), and it validates this model with discrete-event simulations using QualNet v3.6 [66] with our energy model instrumentation. In particular, we are interested in addressing the following questions: (1) How accurate is the analytical model compared to discrete-event simulations? (2) What is the relative energy consumption among the MAC operational modes (e.g., transmit, receive, idle, etc) when nodes are actively contending for channel access (under saturation)? (3) What is the efficiency (Joule/Bit) incurred at each node for a specific network size? (4) How does the efficiency behave as the network size increases? (5) What is the impact of payload size on energy consumption, as the number of nodes increases?

2.6.1 Energy-Aware Model

Carvalho *et. al.* [12] introduce an analytical model to characterize the service time of a node in a saturated IEEE 802.11 ad-hoc network under ideal channel conditions (e.g., ignoring capture effects, hidden terminals, modulation and encoding schemes, etc.). Then, in [13], authors extend the model to include a more realistic approach regarding the IEEE 802.11 binary exponential backoff algorithm, and present an energy consumption model, which is based on the service time characterization.

Next, we summarize the service time of a node in a saturated IEEE 802.11 ad-hoc network under ideal channel conditions considering a finite backoff operation, and then present the energy consumption model.

2.6.1.1 Service Time Model

According to Carvalho *et. al.* [13], the average service time can be decomposed in two parts: the time a node spends in backoff, and the time it takes to actually transmit the frame, as a result of a successful handshake with its intended receiver. The *average service time* ($\overline{T}$) is given by

$$\overline{T} = \overline{T}_B + \overline{T}_s, \tag{2.10}$$

where $\overline{T_s}$ is the average time to successfully transmit a packet at the end of the back-off operation (dependent on the packet size). Authors assume that nodes communicate through the four-way handshake mechanism supported by the standard (the so-called "RTS/CTS" hand-shake) [32]. In this case, the time intervals t_s and t_c are given by

$$t_s = \text{RTS} + \text{SIFS} + \delta + \text{CTS} + \text{SIFS} + \delta + \text{H} + E\{P\} + \text{SIFS} + \delta + \text{ACK} + \text{DIFS} + \delta \tag{2.11}$$

$$t_c = \text{RTS} + \text{DIFS} + \delta \tag{2.12}$$

where RTS, CTS, and ACK are the times to transmit each of the control frames, SIFS and DIFS are the standard-defined time intervals corresponding to the *short interframe space* and the *distributed interframe space*, δ is the propagation delay, H is the time to transmit the packet header, and $E\{P\}$ is the time to transmit the average payload size. The value of $\overline{T}_s$ in Equation (2.10) is simply $t_s - \text{DIFS}$.

The average backoff time $\overline{T}_B$ is given by:

$$\overline{T}_B = \frac{\alpha W_{\min}}{2}\beta_1 - \frac{\alpha}{2}\beta_2 + \beta_3\, t_c \tag{2.13}$$

where

$$\beta_1 = \tfrac{A_1+A_2+A_3}{1-(1-q)^M} \tag{2.14}$$

$$A_1 = \tfrac{2q\{1-[2(1-q)]^m\}}{2q-1} - 1 + (1-q)^m \tag{2.15}$$

$$A_2 = \left(2^{m+1}-1\right)(1-q)^m\left[1-(1-q)^{M-m}\right] \tag{2.16}$$

$$A_3 = \tfrac{2^m\left\{(1-q)^{m+1}-(1-q)^M[1+q(M-m-1)]\right\}}{q} \tag{2.17}$$

$$\beta_2 = \tfrac{1-(1-q)^M(1+qM)}{q[1-(1-q)^M]} \tag{2.18}$$

$$\beta_3 = \tfrac{(1-q)-(1-q)^M[1+q(M-1)]}{q[1-(1-q)^M]} \tag{2.19}$$

where $W_{\min}$ is the minimum contention window size specified for the backoff operation, m is the standard-defined maximum power used to set up the maximum contention window size, (i.e., $W_{\max} = 2^m W_{\min}$), $\alpha = \sigma p_i + t_c p_c + t_s p_s$, (where $p_i = P\{E_i\}$, $p_c = P\{E_c\}$, and t_c is given by Equation (2.12). Because we are dealing with a saturated network under ideal channel conditions, the computation of the channel state probabilities p_i, p_s, and p_c follow the derivations in [12].

2.6.1.2 Energy Consumption Model

To account for the energy consumption under saturation conditions, we need to consider the many events that take place while a node is trying to transmit its own data frame. For this purpose, let us first look at Equation (2.13), which describes the average time a node spends

in backoff. As described in [13], decrementing a node's backoff time counter depends on the node's perception of the *channel state*. As we have pointed out, there are three main channel states (or "events" during backoff): *successful transmission*, *collision*, and *idle channel* states. Notice that, these "states" do not correspond to the "modes of operation" of a network interface.

In the *successful transmission channel state*, the node in backoff experiences a successful transmission happening over the channel. This transmission, however, can either refer to a successful transmission between any two nodes in the network, or to a successful transmission having the node itself as the target receiver. In the former case, the node in backoff overhears an RTS and updates its network allocation vector (NAV) accordingly [32], freezing its backoff time counter for the duration of someone's else four-way handshake (or two-way handshake if the basic access mode is used). Notice that, in this case, the node in backoff first *overhears* the RTS and then stays *idle* for the duration of the advertised transfer, as recorded in the NAV. In the latter case, i.e., when the node itself is the recipient of the transfer, it has to *receive* the RTS and DATA frames from the sender, and *transmit* the corresponding CTS and ACK frames back to the sender. Meanwhile, power is also consumed during the time intervals corresponding to SIFS's, DIFS's, and propagation delays δ, during which the node stays *idle* or *senses* the channel[4].

In the *collision channel state*, similar events can take place from the standpoint of the node in backoff, i.e., the node is either overhearing or being the target of a transmission. Here, however, the node in backoff is overhearing an unsuccessful transmission or is being the target

[4]In reality, some power is also consumed when the network interface switches from one mode to another as, for instance, from the *receive* mode to the *transmit* mode (or *idle* mode). In this work, we disregard the power costs of switching from one mode to another.

of a failed handshake. As before, energy is consumed while in overhearing and receiving modes, respectively. Nodes also consume power during the DIFS interval as the node waits before resuming its own backoff operation, after overhearing/receiving a failed handshake. Finally, during the *idle channel state*, the node in backoff basically *senses* the channel and decrements its backoff time counter each time no activity is detected for the duration of a time slot [32], [12].

All these three states correspond to times when the node is in backoff, perceiving the channel activity, and before attempting its own handshake, i.e., during the course of a backoff stage, as mentioned in [13]. The node attempts to establish a handshake with its intended receiver only at the end of a backoff stage. Each time the handshake fails, the node backs off again and repeats the process, until it finally succeeds in establishing the handshake and before it reaches the maximum number of allowed retransmissions, in which case, it drops its current data frame. In each of its handshake attempts, the node waits for a `cts_timeout` period before deciding that its RTS was not successful. In our analytical model (Equation (2.13)), this collision resolution period is indicated by t_c, which, as described in Equation (2.12), includes the time to *transmit* the RTS frame. Therefore, from Equation (2.12), DIFS $+ \delta$ seconds are spent in *sensing* the channel, waiting for the CTS frame that is never received.

During a successful four-way handshake with the node's intended receiver, the NIC switches through a number of modes of operation. During the four-way handshake time interval $\overline{T}_s$ (i.e., $t_s -$ DIFS), the node *transmits* an RTS and a DATA frame (expressed in Equation (2.11) by the header H and payload P), *receives* the CTS and ACK back from the receiver, and *stays idle* during the time intervals corresponding to SIFSs and propagation delays δ. According to experimental results reported by Feeney et. al. [23], the costs of *overhearing* a frame, staying

idle, or *sensing* the channel are only marginally different from the cost of actually *receiving* a frame (in the IEEE 802.11 network interfaces evaluated). Note that these costs are not provided by manufacturers in data sheets. They were all measured in experiments. For this reason, many network simulators assume (implicitly or explicitly) the same power level for the *idle, overhear, sense*, and *receive* modes [66]. Consequently, because we want to compare our analytical model with simulation results, we follow this same assumption and consider two power levels only: *passive*, or P_{pas}, for the cases when the NIC is in any of the four aforementioned modes, and *active*, or P_{act}, for the mode in which the NIC is actually transmitting something. Given these considerations, all we need to do is to account for the time intervals in which the network interface stays either in the "passive" or "active" modes.

From our previous remarks, a node will be in "passive" mode during backoff except for the case when it is the target receiver of a handshake request, in which case it has to transmit CTS and ACK frames back to the sender. If we denote by T_{pas}^{back} the time a node is in *passive* mode during its backoff, we have, from Equation (2.13),

$$T_{pas}^{back} = \frac{\alpha(W_{\min}\beta_1 - \beta_2)}{2}. \tag{2.20}$$

At the end of a backoff stage, the node attempts to perform a handshake with its intended receiver. Before succeeding in doing that, however, the node will spend $\beta_3 t_c$ seconds, on average, in collision resolutions due to unsuccessful attempts (as shown in Equation (2.13)). In each collision resolution time interval t_c, the node spends DIFS $+ \delta$ seconds in "passive" mode. Hence, if $T_{pas}^{col\_res}$ denotes the average time spent in passive mode during collision resolutions and, likewise, $T_{act}^{col\_res}$ the average time spent on "active" mode during collision resolution, we

have that

$$T_{pas}^{col\_res} = \beta_3(\text{DIFS} + \delta) \quad \text{and} \quad T_{act}^{col\_res} = \beta_3 \text{RTS}. \tag{2.21}$$

When the node succeeds in performing a handshake, it will spend $T_{pas}^{4\_way}$ seconds in passive mode during the four-way handshake. From Equation (2.11), this time interval corresponds to

$$T_{pas}^{4\_way} = \text{CTS} + \text{ACK} + 3 \times \text{SIFS} + 4\delta, \tag{2.22}$$

whereas in transmission the node will spend

$$T_{act}^{4\_way} = \text{RTS} + \text{H} + E\{P\}. \tag{2.23}$$

Finally, we need to take into account the case when the node is the target receiver of a handshake request during its backoff, in which case it needs to transmit CTS and ACK frames back to the sender. In a single-hop ad-hoc network under ideal channel conditions, no capture or hidden terminal problems happen. Therefore, it is always assumed that all frame collisions are due to RTS collisions *at the intended receiver*. This means that, under such assumptions, *no CTS or ACK frame is ever transmitted unsuccessfully*. Therefore, the recipient of a handshake request only transmits a single CTS and a single ACK frame for each data transmission request, i.e., only those frames corresponding to the completion of a successful handshake. Furthermore, assuming a balanced and fair distribution of load in the network [5], if T_{total} denotes the total observation time, then, on average, $T_{total}/\overline{T}$ data frames will be received by any node during the time interval T_{total}. From the remarks, for each data frame transmitted successfully, there

[5] This assumption is realistic in ad hoc network scenarios such as n-way conferencing, and sensor network monitoring. Motivated by applications that may produce non-uniform traffic distributions, one of our directions of future work is to extend our model accordingly.

is one and only one CTS and ACK frame sent by the intended receiver. Therefore, the average time T_{act}^{back} a node spends transmitting CTS and ACK frames back to other nodes (while the node itself is in backoff) is given by

$$T_{act}^{back} = \overline{N}(\text{CTS} + \text{ACK}), \tag{2.24}$$

where $\overline{N} = T_{total}/\overline{T}$ is the average number of data frames transmitted over the interval T_{total}. Hence, if $\mathcal{E}_{passive}$ and $\mathcal{E}_{active}$ denote the energy consumptions in the *passive* and *active* modes, respectively, during the observation time T_{total}, then, from above,

$$\mathcal{E}_{passive} = \overline{N} P_{pas} \left(T_{pas}^{back} + T_{pas}^{col\_res} + T_{pas}^{4\_way} \right), \tag{2.25}$$

$$\mathcal{E}_{active} = \overline{N} P_{act} \left(T_{act}^{back} \overline{N}^{-1} + T_{act}^{col\_res} + T_{act}^{4\_way} \right), \tag{2.26}$$

where the $\overline{N}^{-1}$ accounts for the $\overline{N}$ already included in T_{act}^{back}. Finally, the total energy consumption $\mathcal{E}_{total}$ is simply

$$\mathcal{E}_{total} = \mathcal{E}_{passive} + \mathcal{E}_{active}. \tag{2.27}$$

2.6.2 Model Validation and Performance Analysis

We validate our energy-aware model through packet-level simulations using the QualNet v3.6 simulator [66] with improved energy consumption instrumentation [49]. We then use the model to account for energy consumption of saturated IEEE 802.11 single-hop ad hoc networks in a variety of ad hoc scenarios with different network- and payload sizes. Results correspond to the average of 10 runs with different seeds and different transmission start times (necessary to reduce IEEE802.11 unfairness). Table 2.10 summarizes the simulation parame-

Table 2.10: Simulation parameters

Parameter	Value
Area	50×50 m
Number of nodes	10, 20, 30, 40, 50
Node placement	random
Node mobility	none
Simulation time	300 sec
Bandwidth	1 Mbps
TX-POWER	10 dBm
RX-POWER	-82.045 dBm
Path loss model	free-space
Fading model	none
Phy model	phy-IEEE802.11
RX model	SNR-based
PHY-RX-SNR-THRESHOLD	10.0
Routing protocol	static
Traffic	CBR
Payload size	20 and 1472 bytes
Packet interval	0.024 sec
Power consumption in TX	1650 mW
Power consumption in RX	1400 mW

ters used. It is important to note that simulation parameters were chosen in order to provide a simulation environment as close as possible to the assumptions made in our model.

Figures 2.7 and 2.8 show the average energy consumption per node (in Joules) for the *active* and *passive* modes, as well as their sum (i.e., the total energy consumption) for different network sizes. Figure 2.7 plots results for the 1472-byte payload, whereas Figure 2.8 shows the results for a payload of 20-bytes. As we can observe, the analytical model predicts quite well the simulation results. Because the analytical model is more conservative in terms of throughput [12], it leads to slightly smaller energy consumption values for the *active* mode and, consequently, slightly bigger values for the *passive* mode, compared to simulations. It is worth

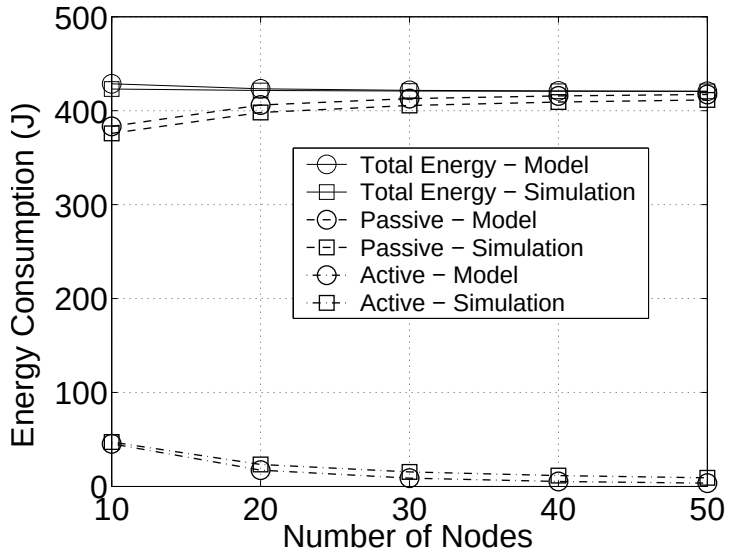


Figure 2.7: Per-node average energy consumption versus network size for a 1472-byte payload.

mentioning that, when compared to the model, QualNet simulations use more "realistic" PHY-layer parameters, as shown in Table 2.10. Despite this fact, the analytical model proved to be a good abstraction of the simulated scenarios, supporting our earlier assumptions of restricting collisions to RTS frames only (for fully-connected networks).

Regarding individual contributions of the operational modes to overall energy consumption, the *passive* mode is responsible for the largest fraction of the total energy consumed. For the network sizes investigated, the *passive* mode consumes more than 88% of the total energy drained (for the chosen parameters). This result is a direct consequence of the fact that, under saturation and high contention, nodes spend most of their time backing off and listening to the channel, instead of actually transmitting data. For all network sizes investigated, the average total energy consumption is about 420 J, leading to an average power consumption of 1.4 W for the 300 s period, i.e., equivalent to the nominal power setting for the passive modes (energy

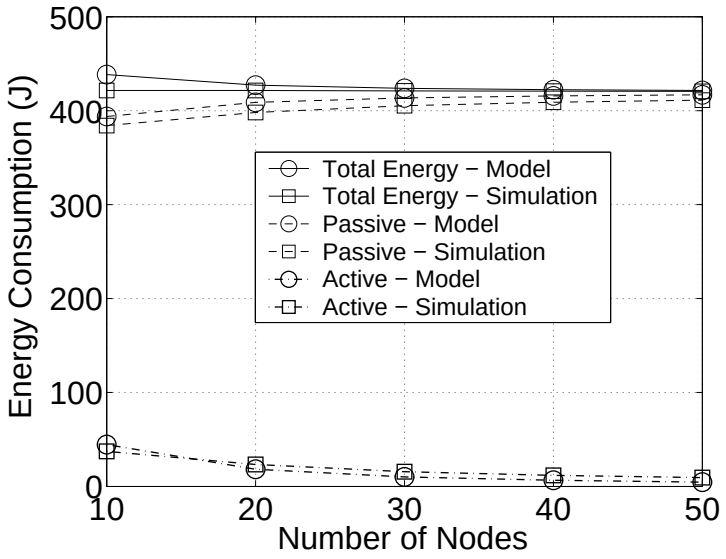


Figure 2.8: Per-node average energy consumption versus network size for a 20-byte payload.

consumption in RX) as shown in Table 2.10. This is consistent with the observation that passive modes are responsible for most of the energy dissipated. As the number of nodes increases, the power consumption in *passive* mode increases from 1.25 W up to 1.37 W. In other words, although the nominal value for the *transmit* (active) mode is 250 mW higher than the value for the *receive* (passive) mode, its impact is practically insignificant as far as the MAC operation in saturation conditions is concerned. This result opposes the findings in [23] and [22] which, under the perspective of a two-node scenario (sender/receiver) without contention, transmit mode is the largest overall energy consumer.

In cases where power-saving methods cannot be employed for some reason (like the IEEE 802.11 power saving (PS) mode [32], not available at some NICs), this result suggests that one can design energy-efficient WLAN devices by focusing on the optimization of circuits that are mainly active during *passive* modes of operation (see [58] for a simplified block diagram of

a WLAN device). In fact, in typical WLAN devices, the RF power amplifier is a key component that, alone, demands most of the nominal power consumption, and it is used only in the transmit mode [22], [58]. According to our results, this is exactly the component that will *affect performance the least*, as far as energy consumption under channel contention is concerned.

Another interesting observation from Figures 2.7 and 2.8 is that the energy spent on both 20- and 1472-byte transfers are equivalent in all modes of operation. In other words, from the standpoint of energy-efficiency, it is better to transmit bigger payloads than smaller payloads, because the net energy consumption is the same. This last result can be better illustrated by the *energy efficiency to transmit useful data*, E_{eff} given by

$$E_{\text{eff}} = \frac{\text{Total Power Consumption}}{\text{Goodput}} \frac{\text{J}/s}{\text{Bit}/s}. \tag{2.28}$$

Figure 2.9 shows the behavior of E_{eff} for the cases of 20- and 1472-byte data payloads as the number of nodes increases. Surprisingly, the energy cost appears to have an *almost linear* increase with network size. Moreover, the energy cost to transmit a 20-byte data payload grows at a rate that is about one order of magnitude higher than the cost to transmit a 1472-byte payload. For the 1472-byte scenario, the energy cost grows at a rate of approximately 0.002 mJ/Bit, whereas in the 20-byte scenario, the energy cost grows at 0.02 mJ/ Bit.

The usefulness of an analytical model such as the one we provide here is the ability to provide quick answers without resorting to simulations. As an example, we use it to analyze the energy consumption of commercially-available NICs, namely the Lucent WaveLAN card transmitting at both 1 Mbps (TX = 1650 mW and RX = 1400 mW) and 11 Mbps (TX = 1400 mW and RX = 900 mW), and the Cisco Aironet PC4800 at 1 Mbps (TX = 2200 mW and RX

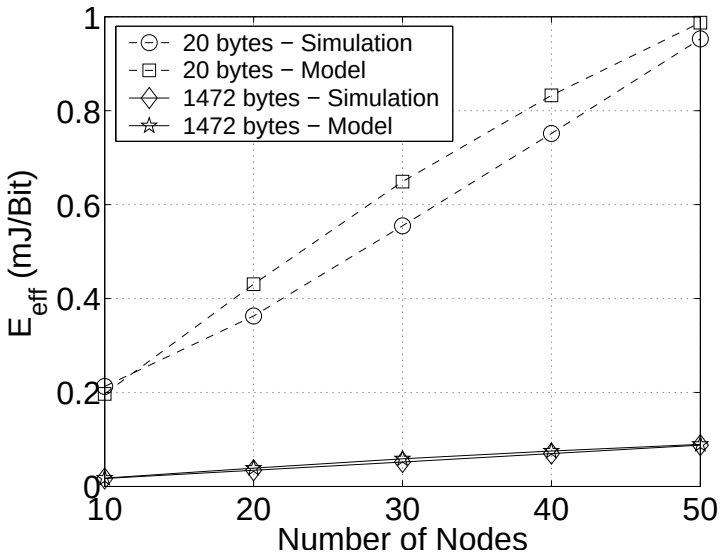


Figure 2.9: Energy efficiency per bit versus network size for 20-byte and 1472-byte payloads.

= 1350 mW) [22]. Figure 2.10 shows that, as expected, the power setting that provides the smallest energy consumption is exactly the one with the smallest power level in *passive* mode, i.e, the 11-Mbps WaveLAN. Another interesting result is the little impact that transmit power has on overall results. All three settings showed similar performance in active mode, despite their relative nominal power differences.

2.7 Conclusions

This chapter presented our work on instrumenting network simulators to enable them to adequately and accurately account for the energy consumed by ad hoc network protocols' communication-related tasks. This is accomplished by explicitly accounting for all possible radio states, i.e, transmitting, receiving, overhearing, idle, sensing, and sleeping, and considering

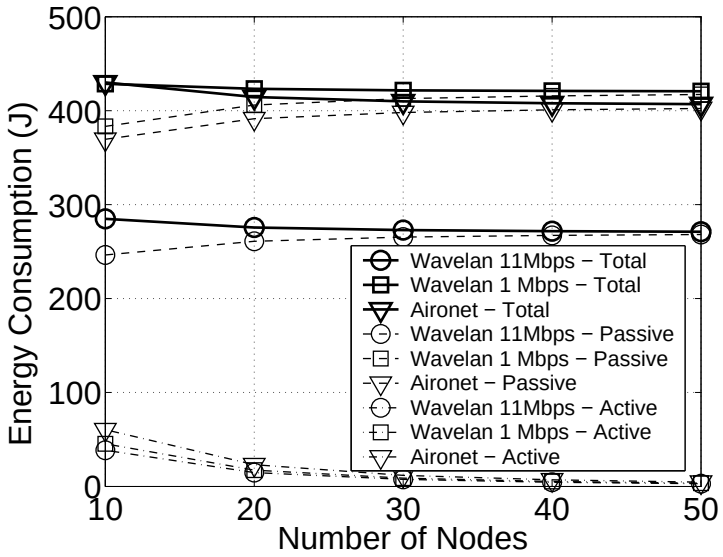


Figure 2.10: Energy consumption from the analytical model for different power settings.

the different energy costs associated with each of them. Another contribution of our energy consumption instrumentation is to allow the energy accounting to be done automatically by the simulator irrespective of what layer of the stack the protocol designer is working.

The instrumentation energy model is validated analytically and through simulations using two MAC protocols, i.e., IEEE 802.11 DCF and S-MAC. We also showcase our instrumentation's ability to evaluate energy consumption of protocols by comparing S-MAC against 802.11, and AODV against DSR. Then work simulation results obtained using instrumented QualNet with our energy consumption accounting are used to validate the analytical model proposed to evaluate energy consumption in IEEE 802.11 single-hop wireless networks.

Chapter 3

Energy Consumption Characterization of Wireless Platforms

This chapter presents the work done on characterization of the energy consumption of different wireless platforms, namely a laptop (a typical example of a mobile computing platform) [50], and the Meerkats node [51] (example of a visual sensor network node), as well as the work done to extend and validate the on-board battery monitoring on the Stargate.

3.1 Introduction

Due to a combination of technology advances in fields such as wireless communications and circuit integration, the last ten years have witnessed a proliferation of mobile computing platforms. Examples of such platforms, which vary widely in terms of capability and functionality, include laptops (e.g., notebooks, tablets, etc.), pocket PCs, personal digital assistants (PDAs), cell phones, wireless single-board computers, sensor nodes, etc. Following the

general trend in the consumer electronics market, the cost of these devices has been steadily decreasing while their capacity (i.e., processing, storage, communication) has been steadily increasing. However, the fact that they are typically powered by non-continuous energy sources imposes serious limitations to these devices' utility from the end user's point-of-view.

As a result, energy consumption in mobile computing platforms has been an area of intense research spanning many fields such as computer architecture, operating systems, computer networks, and application design. In the computer architecture community, energy characterization is usually performed at the instruction level. Proposed power savings mechanisms include shutting off parts of the processor not currently being used, designing machine-level instructions that trade generality for power efficiency, etc. At the operating systems level, example of power management strategies include disk spin-down, periodic system hibernation, etc. Power-efficient network protocols have also been attracting considerable attention from the networking research community and include a variety of techniques such as hibernation of idle nodes, controlling transmission power and/or direction, routing based on remaining energy in nodes, etc.

This work takes a different approach to energy consumption characterization and focuses on characterizing energy consumed by "basic" application-level tasks. Characterizing energy consumption at the task level allows us to (1) predict whether the energy currently available is sufficient to execute a given application, and (2) perform application-level power management. For example, in a distributed computation, given the task at hand and how much energy there is left, a machine's task manager decides whether the task can be executed locally or needs to be shipped elsewhere. In order to make that decision, the task manager, given the

machine's current energy budget, considers the amount of energy the execution of the task will consume versus the amount of energy consumed by sending the necessary information over the network to another machine.

Given the technologies advances on mobile platforms and wireless networks, the Sensor Network area has had a lot of attention from the research community. But the use of high–level sensors (e.g., cameras) in battery–operated networks has received relatively little attention so far. 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). These considerations make the energy consumption characterization of a visual sensor node a necessary step to achieve better power management schemes.

The remainder of this chapter is organized as follows. Sections 3.2 and 3.3 describe the energy consumption characterization in a mobile computing platform (laptop) and on a visual sensor network node, respectively. Then we present the effects of the in-system energy consumption monitoring tool available on the Stargate in Section 3.4. Related work is presented in Section 3.5, while Section 3.6 concludes the chapter.

3.2 Characterizing System Level Energy Consumption in Mobile Computing Platforms

In this Section, our focus is on mobile computing platforms, e.g. laptops. Applications executed by multi-purpose mobile devices like laptops can be as general as "fixed computing" applications and typically consist of basic tasks such as processing, input/output (disk, display, etc.), communication (transmission and reception over the network), or a combination thereof. Consider reading e-mail: it includes communication with the email server and thus network transmission and reception, processing information received, and storing it on disk.

We identify a set of basic tasks representative of mobile computing workloads. Based on these tasks, we define a set of benchmarks that consider each task in isolation or task combination. To observe the battery discharge behavior as a benchmark executes, we employ the Advanced Configuration and Power Interface (ACPI) [31] to monitor the battery's discharge rate. A brief description of the ACPI is presented in Section 3.2.1.

As case studies, we use the benchmark set to characterize the energy consumption of a mobile computing platform (the Dell Latitude C600) under different operating systems, namely Debian [75] and Mandrake [47] Linux. We used the Dell Latitude C600 because it was easily available to us (we have several of them in our lab).

In summary, the main contribution of this work is a task-level energy consumption characterization benchmark that accounts for basic tasks such as processing, disk access (including read and write access), terminal usage, and communication (transmission and reception). Such characterization is critical to power management decisions.

The remainder of this section is organized as follows. Section 3.2.1 provides an overview of the ACPI standard. Sections 3.2.2 and 3.2.3 describes the proposed energy consumption benchmark and our experimental methodology. Section 3.2.4 presents results from our case studies, while Section 3.2.5 discusses benchmark set results application.

3.2.1 Background

The Advanced Configuration and Power Interface (ACPI) is an open standard developed by a consortium involving Hewlett-Packard, Intel, Microsoft, Phoenix, and Toshiba [31]. It is a standard that defines power and configuration management interfaces between an operating system and the BIOS [8].

The ACPI provides both static information about the battery such as model number, serial number, design voltage, etc., as well as current battery status, e.g., whether the battery is charging or discharging, current voltage, discharge rate, estimate of the remaining battery capacity, etc. In Linux, the ACPI name space is mapped to the */proc* file-system. For example, in Debian, the ACPI name space is mapped to */proc/acpi/*; and battery related information is in */proc/acpi/battery*.

The ACPI updates battery information (both static and dynamic) every time the corresponding file (i.e., */proc/acpi/battery/info* or */proc/acpi/battery/state*) is read [1]. Therefore the frequency these information is updated depends on the application doing the battery monitoring.

[1] Simon Fowler, maintainer of $wmacpi$ [25], provided this information.

3.2.2 Energy Consumption Benchmark

As previously discussed, our goal is to characterize energy consumption macroscopically at the task level (instead of, for example, at the machine instruction level). To this end, we define a set of "basic" application-level tasks that are representative of typical mobile computing workloads. In the case of laptops, basic tasks include: processing, input/output (disk, display, etc.), and network communication (transmission and reception). Since the focus of this work is mobile nodes, we consider wireless network interfaces. Often, applications consist of a combination of such tasks.

For our benchmarks, we define four main task categories, namely *baseline*, *processing intensive*, *storage intensive*, and *communication intensive*. We also consider the effect of the display by turning it off and on when executing some of the benchmark tasks.

- **Baseline**: The baseline benchmark captures the energy consumption behavior of the mobile when no user activity is taking place, i.e., only basic operating system tasks are running. This benchmark characterizes energy consumption when the system is idle and also serves as a reference for all other benchmarks. We disable the wireless network interface to isolate the effects of this device on energy consumption.

- **Processing-intensive**: To characterize processing-intensive tasks, we use the FFT benchmark [1], which is part of SPEC's CPU2000 [74], an industry-standardized CPU-intensive benchmark suite. FFT, short for fast Fourier transform, is an efficient algorithm to compute the discrete Fourier transform (DFT) and its inverse.

- **Storage-intensive**: We chose the IOzone [55] file-system benchmark to characterize en-

ergy consumed by tasks that are disk intensive. IOzone can be configured to perform a variety of disk access operations.

We run IOzone in two different modes, one which performs only read accesses and another that only writes to disk. In both cases, it accesses a 3GB file. We use an option which purges the disk cache before each file operation. The write tests includes writes of new files and re-writes of existing files, and the read tests reads and re-reads a file.

- **Communication-intensive**: We characterize communication-intensive tasks by using Iperf [79], a tool designed to measure TCP available bandwidth.

  Network transmission is implemented using Iperf in client mode, while the reception task uses Iperf in server mode. In both cases, Iperf is configured to generate UDP traffic at 10Mbs for all the experiment duration. A 10Mbs rate was chosen because this is maximum nominal capacity of the wireless card.

- **Display**: When we needed to turn the display off, we used the $xset$ [65] utility. $xset$ can be run from the command line and allows the setting of several user preference options for the display, including an option that turns the video card and LCD display off.

3.2.3 Measurements

While the tasks run, we observe the battery discharge behavior through measurements provided by the ACPI. In particular, we monitor the battery discharge rate. As mentioned in Section 3.2.1, we can obtain this information by sampling */proc/acpi/battery/state*. We run the sampling script which periodic reads the current values of the battery discharge rate, while the

benchmark is executing.

When deciding which sampling rate to use, accuracy is an important consideration. However, making sure that the measurements do not interfere with the observations is also critical. In other words, we want to sample as frequently as possible without being intrusive. Preliminary experiments showed that a sampling rate of deci-seconds is not intrusive.

3.2.4 Case Study: Dell Latitude C600

We use our benchmark set to characterize energy consumption in the Dell Latitude C600 which has a 750 Mhz Pentium III (Coppermine) processor with 256K cache, 256 MB RAM, and 20Gb hard-disk. Its is powered by a Li-ion battery, with eight cells, design voltage of 14.8 VDC and nominal capacity of 59.0 Wh. As the network interface, we use the Cisco Aironet 350. As the operating system, we use two of the most widely used versions of the Linux operating system, namely Debian [75] (Debian kernel 2.6.1) and Mandrake [47] (Mandrake 10.1 kernel 2.6.8).

In the remainder of this section, we present energy consumption results for this platform when executing the basic benchmarks described in Section 3.2.2 as well as some combinations thereof.

3.2.4.1 Basic Tasks

We execute all six tasks described in Section 3.2.2: baseline, processing (FFT), disk writes (IOzone write), disk reads (IOzone reads), network transmission (Iperf client) and reception (Iperf server). We run both operating systems using their default configuration. The script

Table 3.1: Mean discharge rate for each task on Dell C600

Operating System	Debian	Mandrake
Task	Mean and St. Deviation	Mean and St. Deviation
Baseline	10.586 W, 4.285	10.525 W, 4.904
Processing	25.111 W, 1.155	24.836 W, 1.189
Disk Write	19.588 W, 5.218	22.429 W, 4.838
Disk Read	16.233 W, 5.124	21.849 W, 4.747
TX	18.315 W, 4.295	18.301 W, 4.549
RX	16.041 W, 4.236	16.350 W, 4.201

that monitors the battery discharge rate runs at a one second sampling rate.

The discharge rate time series are shown in Figure 3.1. We summarize these results in Table 3.1 which tabulates average discharge rate and the corresponding standard deviation for all tasks under both Debian and Mandrake [2].

From Table 3.1, we note that, for both operating systems, the most energy hungry task is FFT (i.e., intensive CPU activity), followed by disk writes. This is somewhat surprising as we expected disk write intensive tasks to be more expensive than processing intensive tasks in terms of energy consumption. For Debian, network transmission is the next task in energy consumption descending order, followed by disk reads and network reception, both of which have about the same energy cost. For Mandrake, the order is slightly different and has disk reads as the third most expensive task followed by network transmission and then network reception.

We should note that we used two laptops, one with Debian and one with Mandrake. They have exactly the same configuration except for their hard disks. Even though both hard disks are manufactured by Hitachi, they have different specifications (e.g., number of heads)

[2]Since the discharge rate curves for all six tasks on Mandrake exhibit very similar behavior to the corresponding curves obtained for Debian, we omit the Mandrake curves.

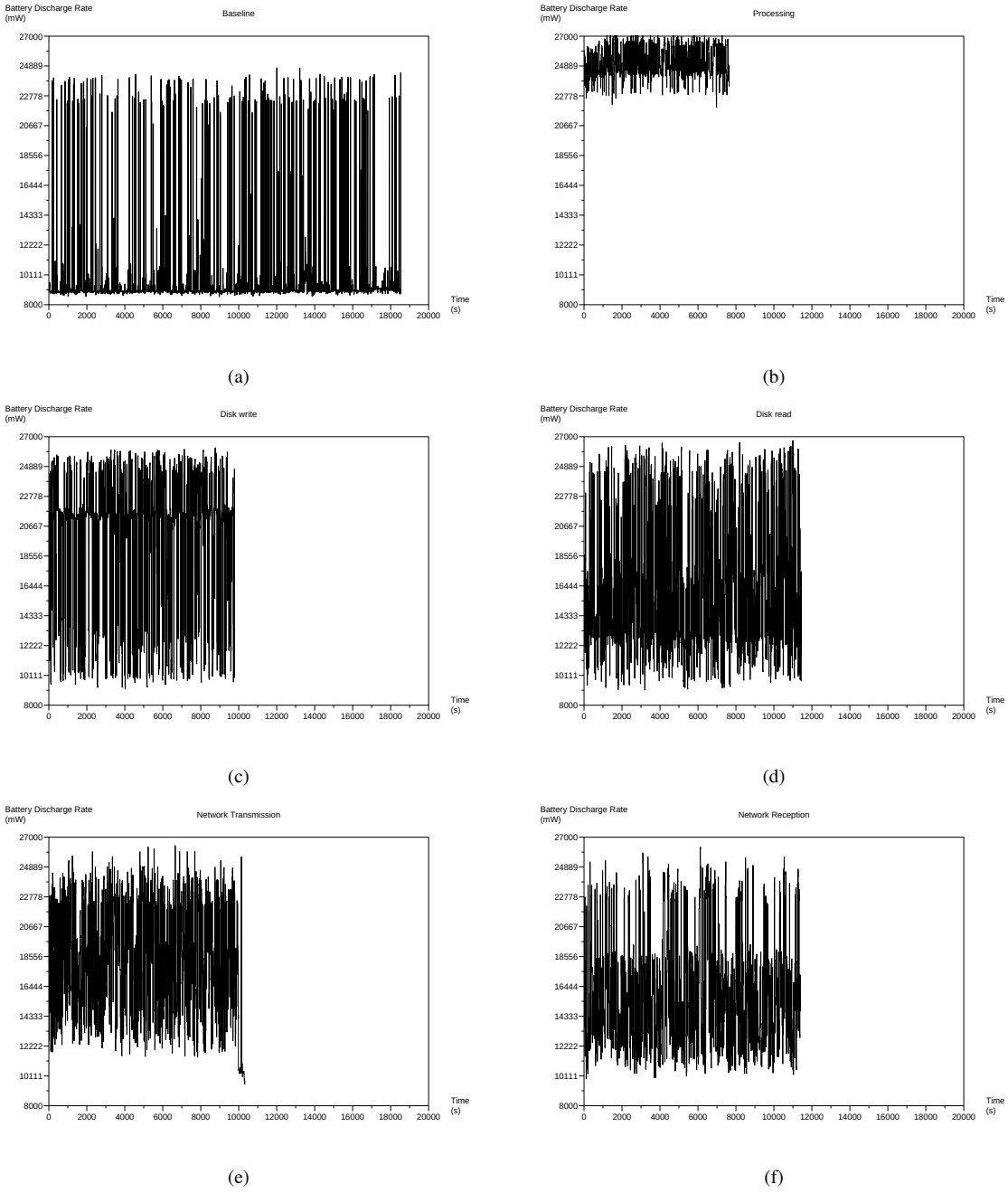


Figure 3.1: Discharge Rate for Dell C600 with Linux Debian: (a) baseline, (b) processing task (c) disk write, (d) disk read, (e) network transmission, (f) network reception

and firmware revisions. The configuration parameters are the same for both disks, and they use the same file system (EXT3). However, there is one considerable difference between the disks: the disk on the Mandrake laptop achieves much higher throughput than the one in the Debian laptop. We executed a simple test using IOzone in both platforms, and it took about seven minutes (real time) in the Mandrake laptop, while it took about 10 minutes (real time) in the Debian laptop to write 3GB. This indicates that the difference in the mean discharge rate for both experiments could be due the fact that the Mandrake laptop can transfer more data, i.e. do more writes and reads per experiment.

Next, we do some back-of-the-envelope calculations to validate the discharge rate results.

According to Intel's 750 Mhz Pentium III Coppermine data-sheet, the voltage and current for the processor core is $Icc_{core} = 15A$ and $Vcc_{core} = 1.65V$ and for the processor in sleep state is $Icc_{sleep} = 2.5A$ and $Vcc_{sleep} = 5V$. If we calculate the power from these two values, we obtain $Pcc_{core} = 24.75W$ and $Pcc_{sleep} = 12.5W$. If we compare Pcc_{sleep} and average discharge rate for baseline experiment, we notice that Pcc_{sleep} is about 20% higher. As for Pcc_{core} and average discharge rate for the FFT experiment, Pcc_{core} is about 4% lower.

During the network transmission experiment, 6.7GB were transmitted at an average bandwidth of 5.7Mb/s. In the network reception experiment, 12.1GB were received at an average bandwidth of 6.6Mb/s. Note that the network reception experiment runs for a longer time than network transmission one because it has lower discharge rate (and thus take longer to reach the cutoff voltage). Since we are using UDP as the transport layer protocol, packet size is 1472 bytes. Since the two laptops are the only two wireless nodes in the lab and they are close to

each other, we can assume that in the transmission experiment the time spent in the transmission state (e.g., when compared to the time spent sensing the medium, backing off, performing the 802.11 [18] handshake) dominates. Similarly, for the reception experiment, the time spent in receiving dominates.

A similar validation for the disk experiments is not as straightforward since other factors such as disk seek times, default power management techniques as well as the effect of disk caches need to be accounted for.

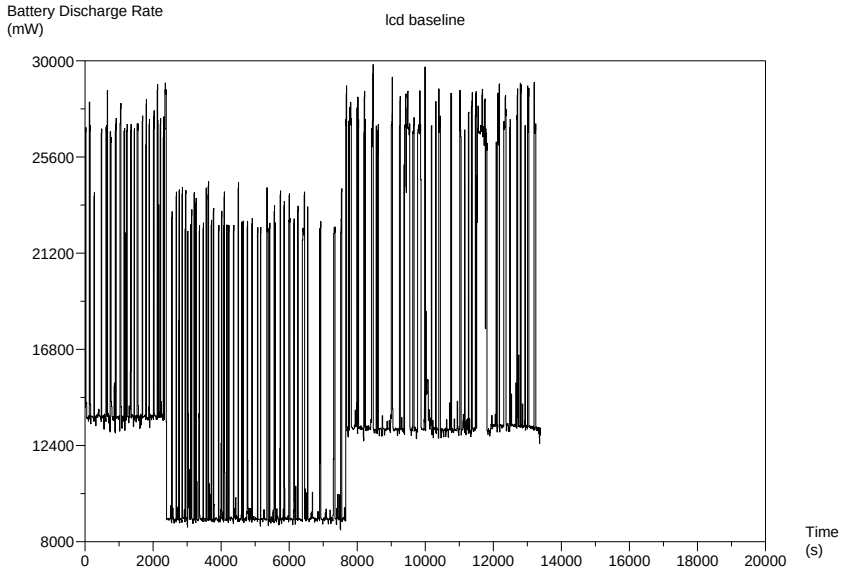
3.2.4.2 Effect of the Display

In all previous experiments, we had the display off. However, it is a well-known fact that the display is a major source of energy consumption. In this set of experiments, we monitor the energy consumption for some of the basic tasks while the display is on. In particular, we show results for baseline and processing. We ran these experiments on the Debian laptop.

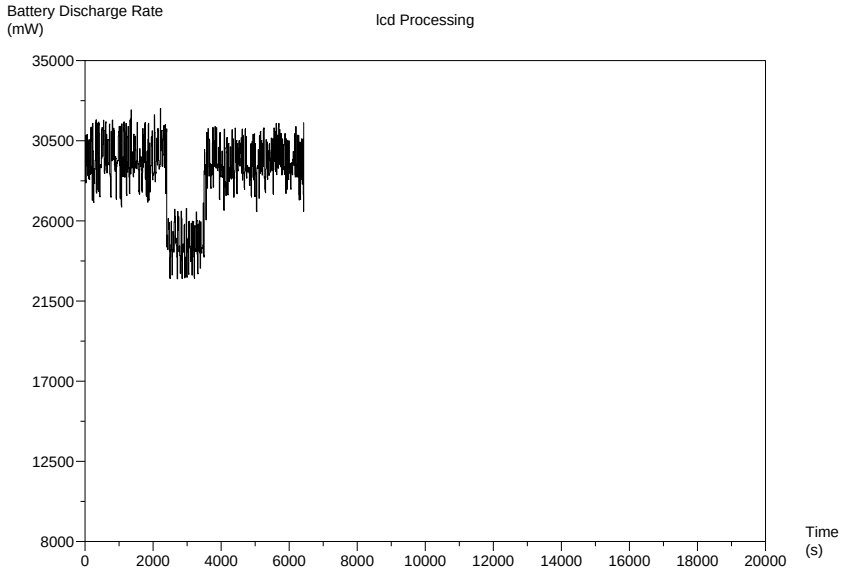
The discharge rate for the baseline experiment is shown in Figure 3.2 (a). We notice that during part of the experiment the bottom of the discharge rate decreases to about 9 W, and this is because the display was turned off and then the discharge rate is about the same we obtained for the baseline task. Thus we observe that the display consumes about as much energy as a disk read or a network reception task.

Figure 3.2 (b) shows the discharge rate for the processing experiment. Again we observe that the discharge rate decreases to about 24W, which is the mean discharge rate for the basic processing task. This happens because at this point the display was turned off.

Table 3.2 shows the mean discharge rate and standard deviation for the basic tasks



(a)



(b)

Figure 3.2: Battery discharge rate for Dell C600 with Linux Debian and display on: (a) baseline, (b) processing

Table 3.2: Mean discharge rate for basic tasks with display on on Dell C600 with Linux Debian

Task	Mean	St. Deviation
Baseline	14.516 W	6.129
Processing	28.648 W	2.147

with display on. When comparing this table with Table 3.1, we notice, as expected, an increase in the mean discharge rate, since the display increases the energy consumption.

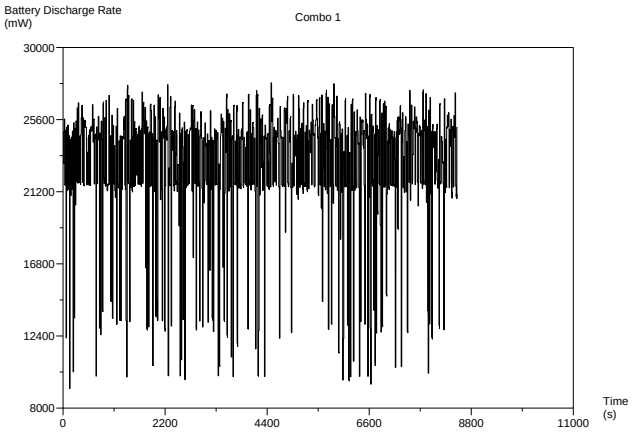
3.2.4.3 Combining Tasks

When we were in the process of choosing the tasks for our energy consumption benchmark, we conjectured that a generic user task consists of a combination of idle, processing, disk access and network communication. In this section, we aim at evaluating the energy consumption and the battery discharge behavior for different combinations of basic tasks. For instance, we choose the following combinations:

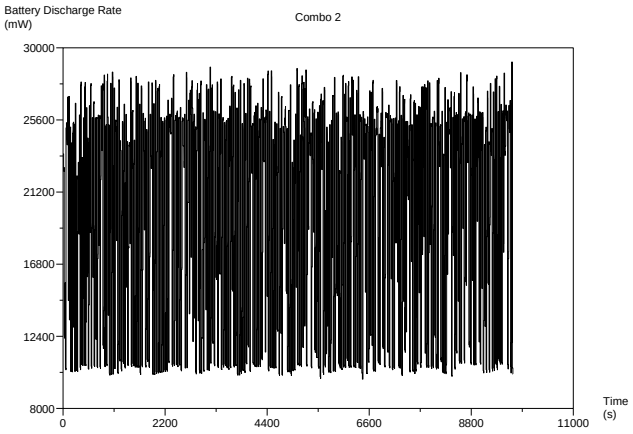
- Combo 1: processing (1.2s real time) and disk access: write (10MB of data, 2.8s real time) and read (10MB, 0.4s real time);
- Combo 2: processing (1.2s real time) and network transmission (10M o data, 2.3s real time);
- Combo 3: processing (1.2s real time), network transmission (10M o data, 2.3s real time) and disk reads (10MB, 0.4s real time).

We ran these benchmarks on the same platform, i.e. Dell C600 running Linux Debian.

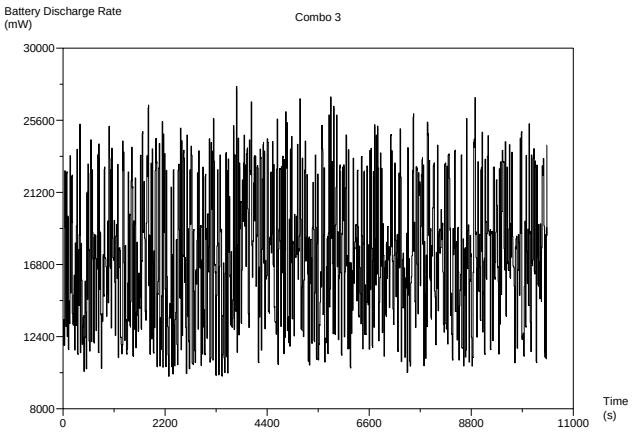
Figure 3.3 presents the battery discharge rate curves for each of the composite tasks, i.e. Combo 1, Combo 2 and Combo 3.



(a)



(b)



(c)

Figure 3.3: Battery Discharge Rate for (a) Combo 1, (b) Combo 2 and (c) Combo 3, all on Dell C600.

Table 3.3: Mean discharge rate for combination of tasks on Dell C600 running Linux Debian

Task	Mean	St. Deviation
Combo 1	22.455 W	4.343
Combo 2	19.511 W	6.754
Combo 3	17.486 W	4.392

The mean discharge rate and the standard deviation for each task are presented in Table 3.3. The mean discharge rate for Combo 1 is less than the basic processing task (Table 3.1, second row), but it is more that the disk access (Table 3.1, third and fourth rows), which is the dominating task, i.e. the task that takes longer to execute within the combo. For Combo 2, the mean discharge rate is higher than the discharge rate for network transmission (Table 3.1, fifth row), while for Combo 3 it is about the same as the discharge rate for disk writes (Table 3.1, third row). From this results we note that the dominating task influences the most the discharge rate.

3.2.5 Discussion

Our premise in this work is that task-level energy consumption information is key to achieving adequate power-aware task distribution in wireless distributed computing environments. Consider, for instance, the case of emergency rescue operations needed after a major (e.g., natural) disaster which destroyed (completely or partially) basic infrastructure such as the power grid and data communication network. Emergency rescue crews would then use their mobile wireless devices to perform all needed computation to assess damage, find survivors, etc. For example, in collapsed buildings, the rescue crew can use information from seismic

sensors embedded in the building to perform structural assessment in order to find portions of the building that have (or not) been affected, what is the probability they will collapse (if they haven't yet), and when that will occur. This computation should be distributed among all (or some) of the participating networked nodes to load balance the computational load and the energy spent. To do so effectively, information on the current energy budget of the nodes as well as the amount of energy consumed by "basic" tasks must be employed. By looking at the typical mix of "basic" tasks to be executed, the "task distribution manager" will be able to assess whether some node can take part on the computation and what is the operational lifetime of the network.

Next we apply a similar methodology and benchmark to another mobile platforms: the Meerkat's visual sensor node, which is based on the Crossbow Stargate.

3.3 Characterizing Energy Consumption in a Visual Sensor Network Testbed

We have conducted a thorough energy consumption characterization of the Meerkats testbed [9], which is based on the Crossbow Stargate platform [17]. We follow the same approach presented in Section 3.2, i.e. it is 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 section is organized as follows. Section 3.3.1 describes the Meerkats testbed, while Section 3.3.2 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 Sections 3.3.3 and 3.3.4 respectively.

3.3.1 The Meerkats Testbed

This section describes the current Meerkats visual sensor network testbed. It presents a detailed description of the Meerkats node's hardware and software organization.

The Meerkats project is developed in collaboration with Professor Roberto Manduchi, and Graduate Students Xiaoye Lu and Gefan Zhang.

3.3.1.1 Hardware Organization

The Meerkats node (shown in Figure 3.4) is based on the Crossbow's Stargate [17] platform, which has an XScale PXA255 CPU (400 MHz) with 32MB flash memory and 64MB SDRAM. PCMCIA and Compact Flash connectors are available on the main board. The Stargate also has a daughter board with Ethernet, USB and serial connectors. We equipped each

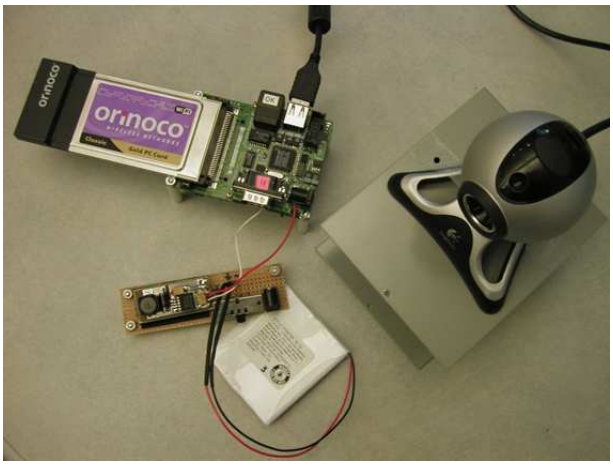


Figure 3.4: Visual sensing node in the Meerkats testbed

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 Stargate can be powered through a 5V DC adapter or through a battery. Both the main– and daughter boards have battery input, but only the daughter board has a DC input. Since we are using the USB connector located 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 (the white rectangle shown in Figure 3.4), manufactured by Energy Sales, Inc and an external DC-DC (with efficiency of about 80%) switching regulator. The operating system is Stargate version 7.3 which is an embedded Linux system (kernel 2.4.19).

The choice of Crossbow's Stargate [17] as the Meerkat's node main component was based on a number of considerations. First, given that our focus was to devise effective resource

management policies for visual sensor networks, it made sense to use of-the-shelf components. Choosing a platform that runs an open source operating system was also important in developing an open source system. And, since we selected a webcam as the visual sensor, we needed a board with a USB connector. As previously discussed, we also needed a platform that provided reasonable processing and storage capabilities.

Currently, the Meerkats testbed is composed of eight visual sensor nodes, and one laptop (Dell Inspiron 4000 with PIII CPU running Linux (kernel 2.4.20) and equipped with an Orinoco Gold 802.11b wireless card) acting as the information sink.

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.

3.3.1.2 Software Organization

The Meerkats' visual sensor node current software organization, shown in Figure 3.5, consists of three main components, namely the Resource Manager, Visual Processing, and Communication modules.

Resource Manager

The Resource Manager is the main thread of control running on the Meerkats node. It controls the activation of the webcam and wireless network card in order to perform image acquisition/processing and communication-related tasks (e.g., transmit an image), respectively, as needed.

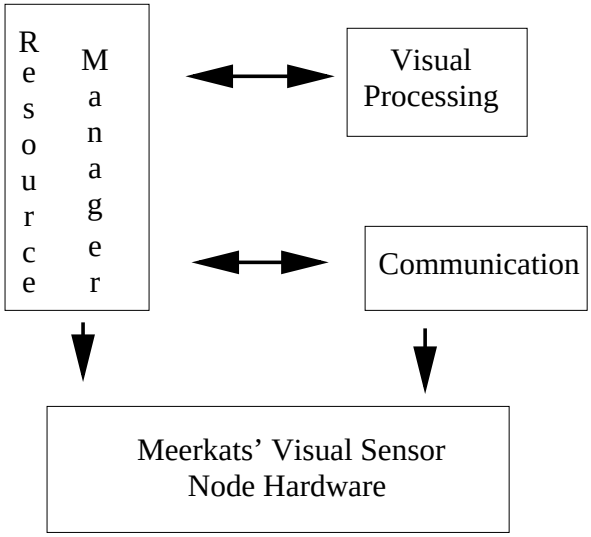


Figure 3.5: Meerkats software organization

For energy conservation, the Resource Manager has the Meerkats sensor node operating on a duty cycle basis, i.e., the node periodically wakes up, performs some task as needed, and goes back to either "idle" or "sleep" mode. The sleep mode has the lowest power consumption, but switching the node to and from sleep mode requires a non-negligible amount of time. This corresponds to the sum of the time it takes to physically switch the operating mode of the CPU, and to activate/deactivate the necessary peripherals. The node is in "idle" mode when it is not performing visual acquisition, processing, or transmission, and when it is not activate/deactivate the necessary peripherals. Note that the node "idle" mode may or may not correspond to the processor "idle" mode [35]: the operating system determines whether/when to perform this switch, in a way that is transparent to the user. A node's idle mode has in fact a number of variations, depending on which system components (webcam or wireless card) are activated (more details in Section 3.3.2).

Visual Processing

The Visual Processing module [44] performs all vision-related tasks, including image acquisition, compression, and processing. It is invoked by the Resource Manager after the webcam has been activated. Upon completion, Visual Processing returns control to the Resource Manager with a parameter flagging whether an event has been detected. The goal is to detect events, in the form of image areas that are different from the background. If an event is detected, the relevant portion of the image is compressed and transmitted to the sink.

The visual processing involved in this experiment is quite simple. A background color image is stored (in a RAM file) and compared to the newly acquired image. When the difference in any color channel at a certain pixel between a new image and the corresponding background is larger than a pre-set threshold, the pixel is labeled as "foreground". The choice of the threshold is rather critical; mechanisms that choose a threshold based on the local color channel variance (or more complex schemes) have been proposed [76], but are computationally demanding, so we opted for a fixed threshold instead. We do, however, update those pixels in the background image that are not labeled as "foreground". The update consists in computing a convex linear combination of the color values of the pixel in the background and in the newly acquire image. This applies only for the pixels that were not labeled as "foreground". After all pixels have been labeled, we run a "blob detector", that is, we identify the connected components of the foreground map. Components that are too small (less than 300 pixels) are rejected from the map. This phase is very important, as it efficiently removes false positive due to noise. We use a fast implementation of a classical blob filter [27]. Finally, if there are remaining blobs with area larger than the threshold, a sub-image is identified by selecting the smallest rectangle that

contains all of the remaining blobs. This sub-image is then JPEG–compressed for transmission.

Communication

The third component of the Meerkats software architecture is the Communication module [3] which is responsible for all communication-related tasks between a Meerkats node and the information sink, as well as communication among Meerkats nodes themselves.

The information sink runs a server program that listens for connection requests from sensor nodes, opens a connection, receives image files and renders images on the sink's console. This program is multi-threaded: one main thread listens for connection requests while individual requests are handled by separate threads. In other words, the server is always listening for requests and when a new request is received, the server will invoke a new thread to handle that request and establish the connection with the requesting sensor node. The corresponding image file will be transmitted over this connection. Once the server receives the whole image, it will be displayed on screen.

Currently, the server's screen is divided into four windows, which means that images from four different sensor nodes can be shown simultaneously. Images from a particular Meerkats node will use the same window, and the most recently received image will be displayed. Older images are saved on the server's hard disk.

The client program running on the sensor node initiates a connection with the server program running on the information sink whenever it needs to transmit an image. Note that images are sent one at a time. The current version of the client and server programs use TCP as the transport protocol for image transmission.

[3] The Communication module was developed by Meerkats project member Gefan Zhang.

Meerkats is a multi-hop wireless sensor network, and thus its nodes need to run a routing protocol in order to send data over the network. We picked the Dynamic Source Routing (DSR) protocol [38], an on-demand routing mechanism especially designed for multi-hop wireless ad-hoc networks. The version of DSR run on Meerkats nodes was ported from the DSR kernel module available for the PocketPC Linux [73] to run on Linux kernel 2.4.20.

3.3.2 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:

- **Processing core**: consists of the processor itself, memory (RAM and flash), and associated hardware;
- **Sensing 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 system calls which activate/deactivate the system components. More specifically, to put the processor in *sleep* mode, one must execute the utility *sys_suspend*, providing as parameter the sleep interval. The node automatically wakes-up when the timer set with the sleep interval expires. 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. *cardctl suspend* deactivates the wireless card, while *cardctl resume* activates it. The mechanism we use to deactivate the webcam is to remove the corresponding modules (*rmmod usb-ohci-sa1111*) from the kernel, while activating the webcam requires inserting back the corresponding modules (*insmod usb-ohci-sa1111*).

In our experiments, we observed that, after loading the webcam modules, we still have to wait for some time (one second turned out to be sufficient) until the camera is ready to acquire

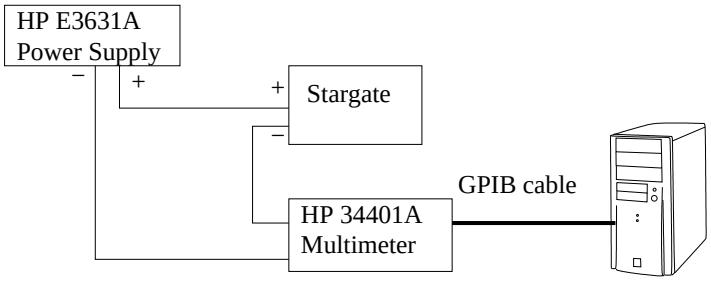


Figure 3.6: Measurements setup

images. We thus had to explicit add that wait time to the node's duty cycle when appropriate. Similarly, we also have to add a wait time of one second after activating the wireless card. Although the task to activated the component was finished, the hardware itself was not ready to be used.

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.

3.3.2.1 Measurement Setup

For our direct measurements, we use the HP E3631A power supply 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 3.6 shows a block diagram describing to our measurement setup.

As shown in Figure 3.6, a GPIB cable is used to connect the HP 34401A DMM to a computer to collect and record measurement samples. The computer is running a custom *vi* on LabVIEW [33], which will save the DMM readings with a time-stamp in a two column text file. 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.

3.3.2.2 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. Similarly to what we defined in Section 3.2.2, 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.

- **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.

- **Processing-intensive**: The characterization of processing-intensive tasks is performed using the FFT benchmark [1], which is part of SPEC's CPU2000 [74], an industry-standardized CPU-intensive benchmark suite.

- **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.

- **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.

- **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.

3.3.3 Steady-state Results

This section reports steady-state energy consumption behavior for the Meerkats visual sensing node. Steady-state is characterized by task execution: processing, acquiring image, communication, etc.

Using the setup described in Section 3.3.2.1, we executed each of the benchmarks described in Section 3.3.2.2 activating different combinations of Meerkats node components (i.e., processor core, processor/sensor core, processor/radio core, and processor/radio/sensor core). Table 3.4 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.

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

Table 3.4: Average current (over five runs) in milli-Amperes and standard deviation drawn by the Meerkats' node.

Task	Processing Core	Proc./Sensing Core	Proc./Communication Core	Proc./Sensing/ Communication 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

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 [23] 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 [23] 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 3.7 plots the results shown in Table 3.4, highlighting other interesting results. For instance, we observe that the webcam adds about 185mA to *Processing Core*, while the wireless card adds about 165mA to *Processing Core*; i.e., the additional current consumed by activating a system component is constant over all tasks.

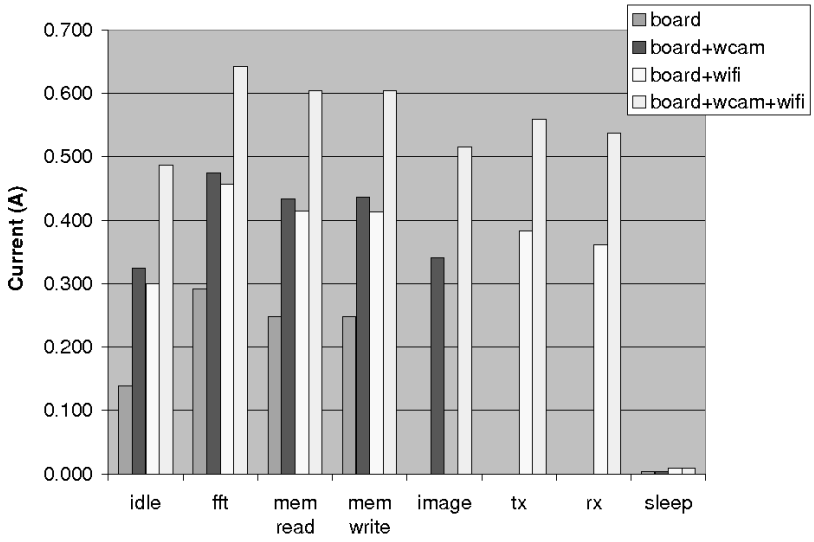


Figure 3.7: Steady-state current draw in the Meerkats node

3.3.4 Transients Results

This section reports transient energy consumption behavior for the Meerkats visual sensing node. Transients are captured when state changes occur, e.g., node goes to sleep and wakes up, or webcam is deactivated.

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

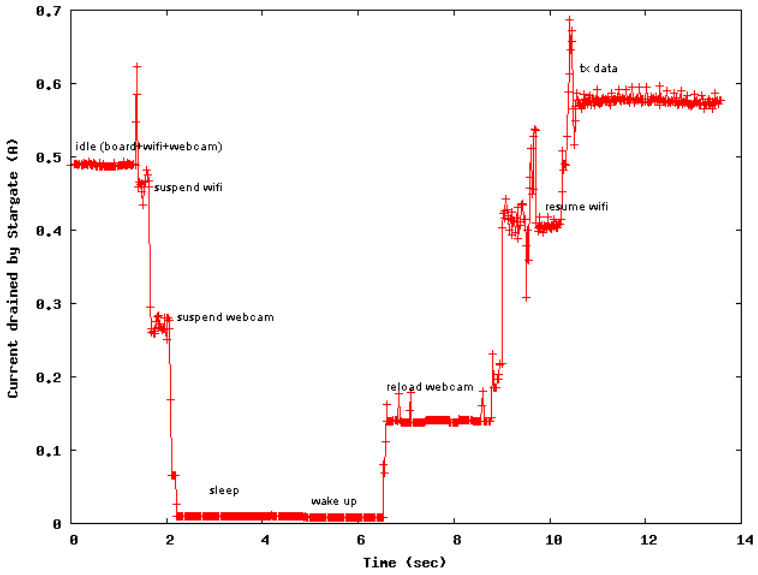


Figure 3.8: Transients from complete system to suspend wireless card, suspend webcam, and switch board to sleep state; then, wake up board, reload webcam, resume wireless card and start tx.

and the processing/response time of the operating system when executing the task(s) required to switch states. The graph in Figure 3.8 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.

Another example of power consumption transients for the Meerkats node is shown in Figure 3.9. 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. This request will require turning the camera on, taking the picture, writing to flash, and then turning the camera off.

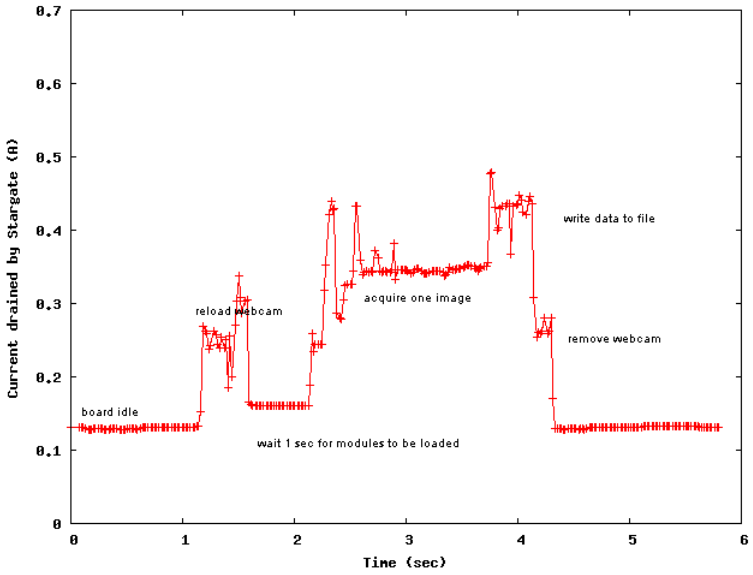


Figure 3.9: Tasks involved when acquiring an image.

It is clear from Figures 3.8 and 3.9 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:

- **Activate/deactivate webcam**: a script that removes/reloads related kernel modules is used;
- **Activate/deactivate wireless card**: this is achieved using the command line utility *card-ctl suspend/resume*;
- **Put node to sleep**: the script *sys_suspend* performs the necessary tasks to put the proces-

Table 3.5: Transition durations in milli-seconds (average and standard deviation calculated over twenty different runs).

Task	Processing Core	Proc./Sensing Core	Proc./Comm. Core	Proc./Sensing/ Comm. Core
Resume webcam	-	1301 ± 27	-	1308 ± 10
Suspend webcam	-	356 ± 11	-	363 ± 12
Resume wireless card	-	-	2316 ± 329	2251 ± 10
Suspend wireless card	-	-	374 ± 80	350 ± 11
Go to sleep	389 ± 33	398 ± 12	358 ± 9	491 ± 23
Wake up	455 ± 57	1292 ± 9	2798 ± 10	3739 ± 13

sor 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 3.5 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 3.5, resuming wireless card activities takes about 2.3 seconds, while resuming the webcam takes about 1.3 seconds. Suspending the webcam takes about 350 milli-seconds, which is about the same for the webcam. We also notice that when the node wakes up with the *radio core* 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. This is also true when the *sensor core* is on.

Table 3.6: Time to execute software associated with transitions.

Program	Time (s)
Resume webcam	0.34
Suspend webcam	0.16
Resume wireless card	1.97
Suspend wireless card	0.13

Clearly, this has direct impact on node duty-cycling.

In order to verify the delays reported in Table 3.5, we use the operating system utility *time* to obtain the time necessary to execute the programs that trigger the transitions. Table 3.6 shows these results which are consistently smaller than the corresponding results in Table 3.5. This confirms that the times reported in Table 3.5 correspond to the times it takes for the operating system to execute the corresponding software modules (as reported in Table 3.6) plus the time for the corresponding hardware sub-system to become active/inactive.

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 3.10 shows a transition from idle to sleep and then again to idle for the *Processing/Sensing Core*. The gray area in the figure represents the amount of additional charge consumed due to transitory current fluctuations.

Table 3.7 summarizes the charge needed for the state transitions mentioned above. The results presented are averaged over twenty different runs. Some of the results are noteworthy. For instance, for the *Processing/Sensing 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

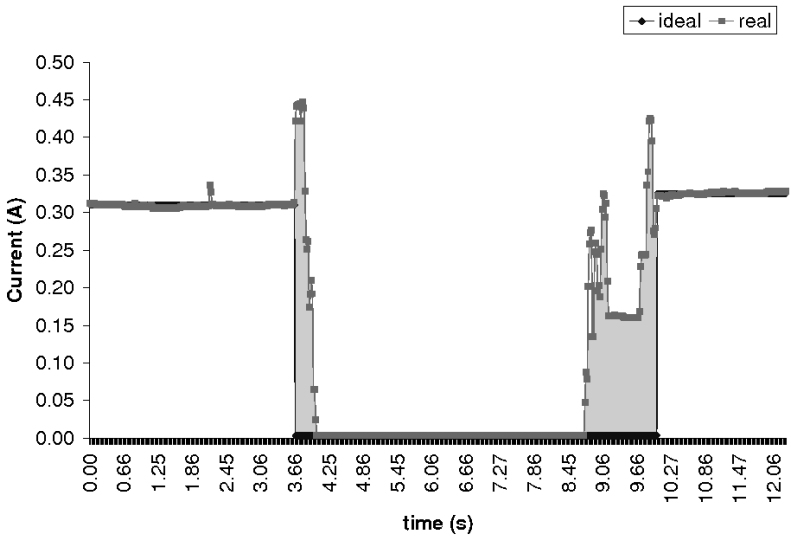


Figure 3.10: Charge consumed by transitions from idle to sleep and back to idle of the *Processing/Sensing Core*.

the same delay may need different charge. For instance, switching to sleep mode for the *Processing Core* takes 389 milli-seconds and requires 75 milli-Coulombs on average, while for the *Processing/Sensing Core* it takes 398 milli-seconds and requires 111 milli-Coulombs on average.

The results presented in Tables 3.5 and 3.7 show that both the hardware and software

Table 3.7: Charge in milli-Coulombs (average and standard deviation calculated over twenty runs) required for transitions.

Task	Processing Core	Proc./Sensing Core	Proc./Comm. Core	Proc./Sensing/ Comm. Core
Resume webcam	-	126 ± 5	-	127 ± 2
Suspend webcam	-	79 ± 6	-	73 ± 3
Resume wireless card	-	-	104 ± 18	110 ± 2
Suspend wireless card	-	-	54 ± 19	50 ± 3
Go to sleep	75 ± 12	111 ± 11	89 ± 3	175 ± 6
Wake up	86 ± 9	279 ± 3	496 ± 3	878 ± 6

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.

3.4 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.

This work done in collaboration with Vladislav Petkov [57], and was already contributed to the Stargate community. It is available at `http://www.soe.ucsc.edu/~cintia/batmon.html`.

3.4.1 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 features of the DS2438 used by the Stargate are the ability to measure battery voltage and current and its current accumulators. With this information, battery consumption and battery life can be

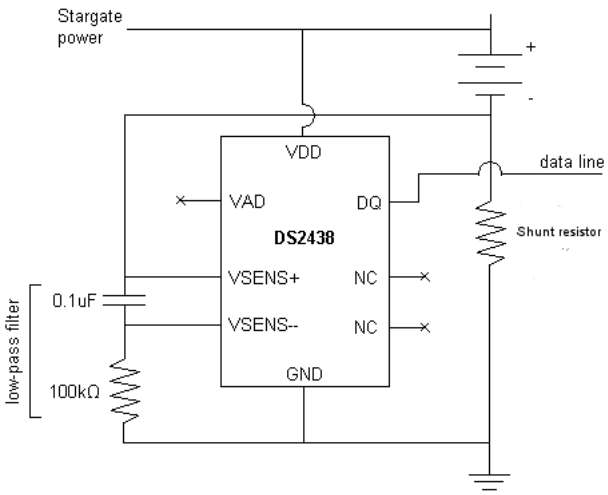


Figure 3.11: Battery monitor connection diagram. Voltage across the shunt resistor is measured by V_{sens+} and V_{sens-}. Current can be determined using $V = I \times R$.

monitored and system behavior adjusted accordingly.

Measuring current is achieved by measuring the voltage at the ends of a low-value shunt resistor, as shown in Figure 3.11. The DS2438 chip [67] 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 3.4), 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 gran-

ularity of 0.904mA with ± 1.808mA error. The equation necessary to obtain the instantaneous current reading is shown in Equation 3.1.

$$I = CurrentRegister/(4096 * R_{shunt}) \tag{3.1}$$

The DS2438 has three different current accumulators: the discharge current accumulator(DCA), the charge current accumulator(CCA) and the integrated current accumulator(ICA). When used in a battery pack, the DS2438 keeps track of charging and discharging current. Whereas the CCA and DCA registers are only affected by charging and discharging current respectively, the ICA is affected by current in both directions and therefore serves as a sort of fuel gauge. Since there is no support for charging batteries on the Stargate, the DS2438 does not see any charging current, which means that the ICA and CCA registers hold no useful information in this system, and only the DCA can be used. The equation for calculating the amount of charge that has flowed out of the battery is shown in Equation 3.2.

$$Charge(mAh) = \frac{DCA_{reg} \times 1000}{(64 \times R_{sens})} \tag{3.2}$$

Note that this *charge* value is relative to the very first moment that the DS2438 measured current flowing through the sense resistor. The DCA register never decrements, so in order to use it correctly, it's value at some defined *starting time* must be subtracted from its present value to figure out the charge that has flowed out since then. The granularity of the DCA register is $\frac{1000}{(64 \times 0.27\Omega)} = 57.87mAh$.

3.4.2 Kernel Modules and Interface

As described in Section 3.3.1, 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 [4]. *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 as the three different current accumulators: the discharge current accumulator(DCA), the charge current accumulator(CCA) and the integrated current accumulator(ICA). The output from $get\_battery\_status()$ contains the raw value of the registers (hexadecimal notation).

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 [5].

[4] Both modules were implemented by Trevor Pering

[5] According to the DS2438 data-sheet [67], a calibration is done before shipment, but the system should be calibrated altogether.

Table 3.8: Current (in milli-Amperes) obtained with *batread*.

Task	Processing Core	Proc./Sensing Core	Proc./Communication Core	Proc./Sensing/ Communication 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

3.4.3 Battery Monitoring Performance Validation

In this section we show the results for the validation of the instantaneous current readings, as well as for the discharge current accumulator(DCA).

Instantaneous Current

In order to validate the current readings provided by *batread*, we use the same setup described in Section 3.3.2.1 and followed the same methodology described in Section 3.3.2 to conduct direct current measurements, while using *batread* to record the current being drained by the system every second. *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).

Table 3.8 presents the results obtained by *batread*, while Table 3.9 presents the current measured by the multimeter. We do not present results for sleep in Table 3.8 since the system will not be able to keep track of the current readings if the processor is sleeping.

Comparing the results in Tables 3.9 and 3.8, we can quantify the interference of on-

Table 3.9: Current, in milli-Amperes, measured with DMM while *batread* was running (average and standard deviation calculated over five runs).

Task	Processing Core	Proc./Sensing Core	Proc./Communication Core	Proc./Sensing/ Communication 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

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 3.12 summarizes the results from Tables 3.9, 3.8 and 3.4 for the *Processing/Sensing/ Communication Core* subsystem.

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.

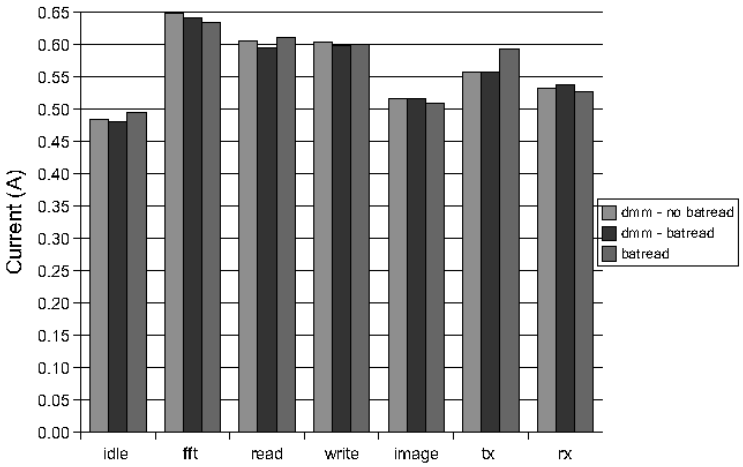


Figure 3.12: Current (in Amperes) measured with DMM with and without *batread* running, and *batread* results. The results apply to the *Processing/Sensing/Communication Core*).

Figures 3.13 and 3.14 illustrates the effect of Stargate's in-system energy consumption monitoring interference with respect to readings of current drawn.

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.

Discharge Current Accumulator

In order to validate the discharge current accumulator (DCA) readings provided by *batmon*, we use the setup described in Section 3.3.2.1. We use a program that reads the DCA register value from */dev/platx/batmon* and converts it to values in milli-Ampère-hour. This program is used first to determine the DCA reading associated with the battery fully charged.

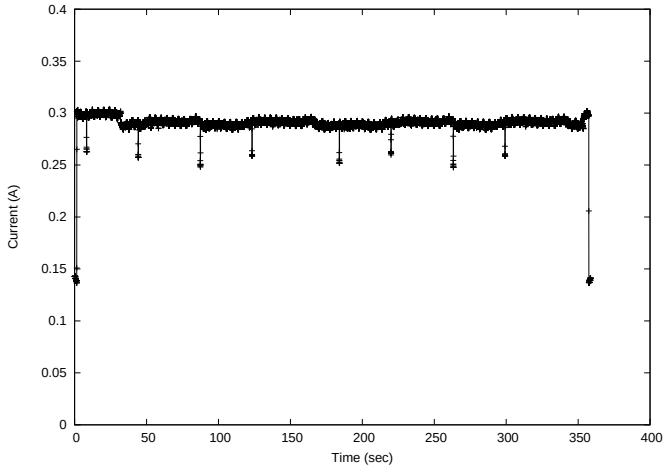


Figure 3.13: Current being drained by *Processing Core* when FFT is being executed.

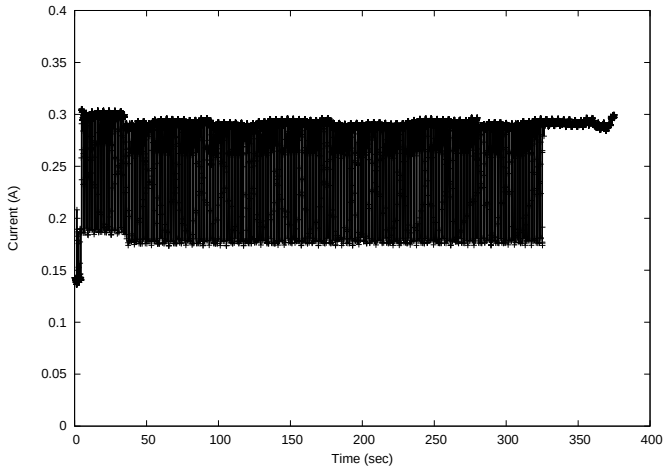


Figure 3.14: Current being drained by *Processing Core* when *batread* is monitoring the system at 1 Hertz and FFT is being executed.

Table 3.10: Charge (in milli-Coulombs) obtained with DMM and batmon.

Task	DMM	*batmon* DCA	Relative Error
duty cycle (a) - run 1	3679.7	3956.4	7.0%
duty cycle (a) - run 2	3695.1	3960.0	6.7%
duty cycle (a) - run 3	3723.3	3956.4	5.9%
duty cycle (a) - run 4	3434.7	3747.6	8.3%
duty cycle (a) - run 6	3729.5	3960.0	5.8%
duty cycle (a) - run 7	3419.4	3747.6	8.8%
duty cycle (a) - run 8	3457.4	3747.6	7.7%
duty cycle (a) - run 9	3441.4	3751.2	8.3%
duty cycle (a) - run 10	3682.7	3956.4	6.9%
duty cycle (a) - run 11	3411.5	3330.0	-2.4%
duty cycle (a) - run 12	3722.5	3330.0	-11.8%
duty cycle (a) - run 13	3708.8	3747.6	1.0%
duty cycle (a) - run 14	3706.2	3960.0	6.4%
duty cycle (a) - run 15	3702.1	3542.4	-4.5%
average:	3608.2	3763.8	4.1%
standard deviation:	136.7	224.2	

Then this program is used periodically to determine the amount of charge used, and once it reaches $1000mAh$ the system is shutdown. To mimic the use of the Stargate as a visual sensor node, we created a duty cycle [6] and executed it continuously.

Table 3.10 presents the results obtained by *batmon* as well as the results obtained by integrating the current readings from the DMM for fifteen runs of the same duty cycle. Note that the results in Table 3.10 are in milli-Coulombs, which is the direct result of the time-integrating the measurements obtained with the DMM. *batmon* results were converted from milli-Ampère-hour to milli-Coulombs for comparison purposes.

The overall relative error between DMM time-integration and *batmon* DCA is 4.1%, although it ranges from -11.8% up to 8.8%. As we already mentioned, in our testbed, the

[6] In Chapter 4 we will discuss the use of duty cycles in a visual sensor network and its relevant tasks. For this experiment, we used duty cycle (a), described in Section 4.4.

granularity of the DCA register is 57.87mAh. This coupled with the fact that we checked for the battery every 100 cycles accounts for the variation among the fifteen runs.

3.5 Related Work

In this section, we summarize related work in areas that are more relevant to our work, including energy consumption measurements for network interfaces and mobile devices, system's power management, storage energy consumption issues, etc.

Stem et al [77] measures the power consumption of some network interface cards (NICs) when used by different end-user devices. Authors also report on transport- and application-level strategies to reduce energy consumption by NICs. Later, Feeney et al [23] reported detailed energy consumption measurements of some commercially-available IEEE 802.11 NICs operating in ad hoc mode. Along the same lines, Erbert et al [22] assessed the impact of transmission rate, transmit power, and packet size on energy consumption in a typical wireless network interface.

In [43], energy consumption in ad hoc mobile terminals is modeled using the Advanced Configuration Power Interface [31], or ACPI. ACPI was used to measure energy consumption due to transmission/reception. The resulting energy consumption model includes two states: **high consumption state**, where the host receives and transmits, and **low consumption state**, where the node receives or is in idle. While this approach to model battery discharge empirically is based on values that laptop power management would see in real systems, it is platform-dependent.

In order to understand the issues on energy consumption of storage on mobile devices, Zheng et al [88] evaluated three different storage alternatives: a compact flash, a micro-drive and a wireless LAN card (which would be used to access a remote storage). By considering these different devices, their different power management schemes were studied, as well as the energy cost of their states were measured. Also the read/write latency and bandwidth was measured. Authors considered two types of files systems: update-in-place and log-structured. Results show that the energy consumption behavior depends on the device power management scheme, on the distribution of idleness in the workload, and on the file system strategies.

Dempsey [86] extends the Disk-Sim simulator to provide a simulation environment that also includes an energy consumption. Dempsey considers the power consumed for several different disk tasks, such as seek, rotation, read, write and stand-by.

Displays are a major consumer of energy. Iyer et al [37] discuss the use of energy-adaptive displays sub-systems. Authors use OLED displays and propose a software optimization called dark windows. Dark windows allows the windowing environment to change colors and brightness of areas that are not of interest to the user. According to a characterization of display usage done by authors, users use about 60% of the screen available. Thus, by changing the colors and brightness of areas not in the window of focus, energy can be saved.

Energy efficiency in mobile devices ranging from phones, laptops or hand-held devices is critical. Monticelli [54] presents an scheme using adaptive voltage scaling to control power management on 3G phones, and suggests that this same approach could be used in RF circuits. Yin et al [85] describe an power-aware prefetch scheme, which dynamically adjusts the number of prefetches based on the current energy level. Another approach to save energy

on mobile devices is the remote power control of wireless interface cards [2], where a remote server uses its knowledge of the workload to perform traffic reshaping. Another approach is μSleep [10], which puts the processor in sleep mode for short periods (less than a second), and reduces energy consumption by up to 60%.

Balakrishnan et al. [4] propose a power management software architecture developed at user level using a standard power interface (ACPI - Advanced Configuration and Power Interface [31]) that provides information about current hardware state (e.g. estimated battery lifetime, temperature, etc.). This architecture is implemented and tested for disk spin down and thermal management.

Anand et al [3] developed a middle-ware for Linux on iPAQs in order to allow better power management. The power management implemented consider what is the state of all devices to be used (in this platform, wireless card and micro-drive), and hints given by the application when it requested data access and the device was not available. It also provides a user interface that allows one to control the priorities in terms of performance and power conservation. By taking these factors into account, authors show that it is possible to increase performance and decrease energy consumption.

PowerScope [24] is a tool that combines hardware instrumentation to measure current level and software support to perform statistical sampling of system activity, allowing energy profiling of process and procedures, which can then be optimized to reduce energy consumption.

Barr and Asanovic [5] present an interesting scheme considering the trade-offs between data transmission and compression. They used the Skiff platform[7] to perform measure-

[7]The Skiff platform is based on the iPAQ hardware, but it has a differentiated circuitry to allow easy power consumption measurement.

ments of energy consumption in wireless Ethernet card, StrongARM CPU, DRAM and Flash memory, under different compression algorithms. Since the energy cost of compressing data in each part of the hardware is know, as well as the cost to transmit data, authors propose a scheme where they compress data to be transmitted in case this procedure will allow energy savings.

The Panoptes sensor hardware [15] 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 [82]. A detailed accounting of the energy consumed by the Motes is presented in [68]. A detailed power profile of the Consus platform, a personal data repository, is presented in [59]. 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 [60] is a low-power, small sensing node composed of a micro-controller, complex programmable logic device, external SRAM, external Flash and an CMOS imager. SensEye [41] is a multi-tier video surveillance network, which makes use of several different visual nodes including: Cyclops [60], CMUcam [16], Crossbow Stargate and webcam, and a high-end camera.

We should point out that our approach is completely orthogonal, and thus complementary, to efforts that focus on designing low power platforms. A noteworthy example is Cyclops [60], a low-power vision sensor node consisting of a CMOS camera, processor, and memory, all integrated on the same board. In the experiments reported in [60], the Cyclops

sensor board was connected to a MICA2 Mote [82]. However, as previously discussed, in their current instantiation, Cyclops would not have sufficient capability (in terms of processing power and storage capacity) to run the types of vision algorithms that we implement on the Meerkats node.

3.6 Conclusions

Application-level power management, which is critical when wireless computing platforms are employed, can only be performed based on energy consumption information. In this chapter, we presented a task-level energy consumption characterization benchmark that accounts for basic tasks being executed by a laptop (a typical example of a mobile computing platform), and the Meerkats node (example of a visual sensor network node).

We also presented the extensions we implemented to obtain current and discharge current accumulator 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.

In the next chapter, we use the characterization presented here to provide a quantitative power consumption and temporal analysis of a set of basic tasks as well as duty cycles representative of activities carried by the Meerkat's testbed, a wireless camera network targeting

surveillance applications.

Chapter 4

Duty Cycle Modeling by Task Composition

This chapter presents a thorough energy consumption characterization of a set of elementary tasks that characterize Visual Sensor Networks system's typical activities. Such elementary tasks include image acquisition and compression; image acquisition and processing (using a background subtraction algorithm), and possibly compression of sub-image; and network transmission of different image sizes. These tasks are defined at a coarse enough granularity to enable different resource management policies by simple task concatenation. We validate our hypothesis that the energy consumed in a duty cycle, as well as its execution time requirements, can be estimated by simply adding up the relevant quantities for each elementary task involved [48]. We also discuss the issues associated with the task concatenation approach to determine a duty cycle, and how its accuracy is affected.

4.1 Introduction

Continuous and pervasive monitoring of events over wide areas often necessitates a large number of networked, wireless sensors. There is widespread agreement within the sensor network community that multi-tier deployments, comprising both low– and high-level sensors, such as cameras, have great potential for a wide-range of current and upcoming applications. Visual sensors can cover larger fields of view and extract more substantial information about the scene than other simpler sensors such as temperature, humidity, and pressure [41].

Sensor networks are typically energy constrained and hard to access once deployed, hence, self–organization and energy management are two fundamental factors determining their performance and lifetime. Accordingly, new mechanisms for network discovery and information flow management have been proposed [29, 34, 72]. Since the vast majority of networks studied to date employ very simple sensors (providing few bits per measurement and consuming little power) and consequently, the bulk of energy consumption is due to communication-related tasks, most research in power conservation for sensor networks has addressed solely communication issues. Examples include power-aware protocols at the MAC layer, data aggregation mechanisms, and strategies for predictive activation/transmission, topology control, power-aware routing, etc. Another common power conservation approach in sensor network deployments is the use of duty cycles [46, 80], which alternate nodes between active and idle, low–power periods.

Most studies to date have typically considered the energy cost of sensing and data processing negligible when compared to communication–related tasks. This assumption, how-

ever, may not hold for higher-level sensors, such as cameras ([5]) which produce large loads of data, requiring considerable processing for analysis (in order to extract semantic information) and/or compression. In addition, sensing itself can be highly power consuming. Analyzing and compressing visual data reduces communication cost but requires more processing, and in some cases may increase net energy consumption. It is clear that traditional node and system-wide control strategies are not applicable to visual sensing networks.

We argue that a thorough energy consumption characterization is critical for designing effective resource management policies for wireless camera networks. In order to analyze and model power consumption over long periods of time, and therefore predict the lifetime of a node and change the control policy accordingly, it is useful to consider a number of *elementary tasks* whose scheduling and execution are controlled by a *resource manager* [6]. Each task has an associated power consumption cost and execution time, both of which could be random variables. Stochastic models can then be employed for predicting the statistics of the time series of tasks, and therefore the expected energy consumed over a period of time [53].

We conducted a thorough quantitative power consumption and temporal analysis of a set of basic tasks as well as duty cycles representative of activities carried out by wireless camera networks targeting surveillance applications. We focus on the wireless camera network testbed we have been building, dubbed *Meerkats* (described in Section 3.3.1).

Our hypothesis is that by defining and characterizing a set of elementary tasks representative of visual sensor network activities at a coarse enough granularity, we are able to model different resource management policies by simple task concatenation. For example, one of the tasks considered is image acquisition and compression. Another task is loading the software

modules that are necessary for the wireless card to operate, while yet another one is keeping the CPU in sleep mode for a certain period of time. Using per-task energy consumption information, the system would then be able to make decisions such as whether it can increase the image acquisition rate (and thus increase the probability of detecting events), or it should decrease the image acquisition rate.

Furthermore, given that the basic control policy of a Meerkats node is based on regular duty cycles synchronous across all nodes in the network, in this work we also validate the hypothesis that energy consumption for a given duty cycle can be obtained by the composition of the energy consumed by the corresponding individual tasks. An example of a duty cycle is as follows. The webcam in the node takes snapshots at regular times, and, if it detects something of interest, transmits the data using the node's wireless card to another node or to a sink. Clearly, the image acquisition period and the policy that is used to decide whether to transmit an image or not affect the *miss rate*, i.e., the probability that moving object in the scene went undetected, either because no snapshot was taken of it, or because the image was not transmitted. Power consumption depends on the same parameters: a higher frame and transmission rate drains more current, on average, from the battery.

In fact, the implementation of a duty cycle is a more complex problem than may look at first sight. A number of operational possibilities must be evaluated, each with different characteristics of power consumption and execution time. Consider, as a simple example, the state of the node between two subsequent snapshots. In order to reduce power usage, the CPU should be put to sleep for all the time it is not used. However, putting the system to sleep requires first unloading the software modules used for controlling the webcam and the wireless

card, saving the context, setting the interruption to wake up the processor and powering off hardware components. When the system wakes up, these modules need to be reloaded and context restored. If the period between two consecutive image acquisition is too short, there may not be enough time to perform all these operations, and the system would have to be kept in idle mode instead. As another example, consider the case of a camera node in a crowded environment. It will likely detect a new event (e.g., a moving person) in most of the images that it takes. Depending on the power cost of the computation that is necessary for event detection, it may be the case that the node is better off transmitting each image without first analyzing it, rather then spending energy by first processing it.

To validate our hypothesis that the energy consumed in a duty cycle, as well as its execution time requirements, can be estimated by simply adding up the relevant quantities for each elementary task involved, we compare direct measurements (given by a digital multimeter) against predictions based on the concatenation of the corresponding elementary tasks. We present results for a number of duty cycle variations.

This chapter is organized as follows. Section 4.2 summarizes the duty cycle model proposed. The elementary tasks a Meerkats node execute are described in Section 4.3, while the duty cycle analysis is presented in Section 4.4. Section 4.5 discusses the problems that affect duty cycle prediction accuracy. Related work is described in Section 4.6, while conclusions are presented in Section 4.7.

4.2 The Model

As tasks delimiters, we assume that at the beginning and at the end of a task, the node is in idle mode (with or without hardware components activated). We also assume that two tasks are never executed concurrently.

Our hypothesis states that it is possible to obtain the duty cycle charge (and duration) by simply adding up the charge (and the duration) of each single task in the sequence. So, given a set of tasks T_i with average charge $\overline{q}(T_i)$ and duration $\overline{d}(T_i)$, the charge consumed by a duty cycle j (Q_{dc-j}) composed by n tasks can be calculated as shown in Equation 4.1. Similarly, the duration of duty cycle j (D_{dc-j}) is obtained according to Equation 4.2.

$$Q_{dc-j} = \sum_{i=1}^{n} \overline{q}(T_i) \tag{4.1}$$

$$D_{dc-j} = \sum_{i=1}^{n} \overline{d}(T_i) \tag{4.2}$$

If we assume that tasks are independent, then a given duty cycle j can be described by a probability density function (pdf) with $\mu = C_{dc-j}$. The variance σ^2 of the pdf is given by the sum of the variances of the tasks ($\sigma^2(T_i)$) that compose the duty cycle.

4.3 Elementary Tasks

In this section we define a set of elementary tasks that a Meerkats node executes as part of a duty cycle. The goal is to characterize each task by the amount of energy it consumes, and by the amount of time it takes for its execution. The granularity of each task should be

coarse enough that task composition and scheduling is kept reasonably simple, yet fine-grained enough to allow for meaningful modeling.

In order to analyze the energy characteristics of each task, it is important to also consider the current *state* of the node. As discussed earlier, the Meerkats node consists of three basic hardware components, namely the *processor core*, the *sensing core* (i.e., the USB and the webcam), and the communication subsystem (or *radio core* comprised of the PCMCIA wireless card). Thus the possible states of a Meerkats node are:

- **Sleep** consumes the lowest power. The processor core is in sleep mode and all hardware components are deactivated. Note that this state is the equivalent of the sleep state for *processing/sensing /communication core* defined in Section 3.3.2. The current drawn by the node in this state is about 9 mA (from Table 3.4).

- In **idle** no specific task is performed, besides background OS (Operating System) processes. As pointed out earlier, the processing core may or may not be in its "idle" mode; this is controlled by the operating system. The current drawn is 139 mA when no additional hardware component is activated (i.e. corresponding to the *processing core* in Section 3.3.2), with the wireless card (*communication core*) and the webcam (*sensing core*) components drawing additional 161 mA and 185 mA respectively if active. We have verified that these measurements are additive, i.e., if both hardware components are active (*processing/sensing/communication core*), the node draws approximately $139\ mA + 161\ mA + 185\ mA = 485\ mA$ (refer to Table 3.4).

- When **active**, the node performs specific, finite–duration tasks involving the processor

core with or without cooperation of the other hardware components (e.g., compressing an image). Note that some of these tasks, when executed, simply change the state of the node, e.g. taking the node from sleep to idle mode. Our measurements highlight that these "state transition" processes take non-negligible time to execute and consume a non-negligible amount of energy. Therefore, they need to be accounted for, e.g., when planning a node's duty cycle. As previously pointed out, we assume that at the beginning and at the end of a task, the node is in idle mode (with or without hardware components activated). The only exceptions are: the task that brings the node from sleep to idle mode (wireless card module is automatically activated when the system is awaken, and thus the "wake up" task thus includes wireless card activation); and the task that brings the node from idle (with the webcam and wireless card deactivated) to sleep. We also assume that two tasks are never executed concurrently. This assumption is reasonable since, in most cases, the execution of a task depends on the completion of the task that precedes it in the duty cycle. For example, in order to acquire an image, one needs to load the webcam modules first, and then based on the results from image processing, it will decide whether to transmit or not the image (which would would require wireless card to be activated).

Table 4.1 shows the cost, in incremental charge and duration, of each elementary task executed by the Meerkats node. The top part of the table summarizes the state transition tasks (already presented in Table 3.4), while the bottom part of the table lists two other categories of tasks, namely *sensing/processing* and *transmission*. As discussed in Section 3.3.1.2, depending on the specific duty cycle organization, a node may take an image at regular intervals and JPEG

Table 4.1: Incremental charge drawn (in milli-Coulombs) and duration (in milli-seconds) for the elementary tasks considered. Average values and standard deviations were computed over twenty runs.

Task	Incremental Charge (mC)	Duration (ms)
Activate webcam	126 ± 5	1301 ± 27
Deactivate webcam	79 ± 6	356 ± 11
Activate wireless card	104 ± 18	2316 ± 329
Deactivate wireless card	54 ± 19	374 ± 80
Put to sleep	75 ± 12	389 ± 33
Wake up	496 ± 3	2798 ± 10
Acquire image	49 ± 2	585 ± 15
Acquire and save (raw image)	51 ± 2	590 ± 11
Acquire, compress and save image	81 ± 2	760 ± 9
Acquire and process image (no event detected)	88 ± 6	1129 ± 21
Acquire, process, compress and save sub-image	105 ± 12	1244 ± 56
Send 7KB (compressed image portion)	12 ± 1	123 ± 12
Send 20KB (compressed image portion)	20 ± 1	207 ± 11
Send 28KB (compressed image portion)	23 ± 2	248 ± 15
Send 48KB (full compressed image)	34 ± 2	356 ± 16
Send 200KB (raw image)	130 ± 3	1375 ± 21

compress it. Or the node may take an image, analyze it, and if an event is detected, compress only relevant portions of the image. The size of the compressed image (or portion thereof) varies and, for this reason, we considered three different transmission tasks, depending on the size of the transmitted data. These different image sizes were obtained through experiments, considering different distances between the object and the camera. We also include the three additional tasks as references, namely: image acquisition (without any processing or compression), acquire and save raw image, and transmit raw image.

Each task is characterized in Table 4.1 by its *duration* and by the *incremental charge*

drawn, that is, the total amount of charge drawn by the board when executing the task minus the charge the node would draw if it was idle (with the same hardware modules active at the beginning of task execution for image and communication related tasks, or idle for *processor core* for transitions). For example, the task of acquiring and compressing an image (with only the webcam activated) draws additional 49 mC with respect to when the node is in idle (with the webcam activated). To compute the total charge drawn by the node during the execution of this task, we need to add this value to the charge drawn by a node in idle mode (with the webcam activated) over the same period of time (585ms). Thus the total charge drawn in this case would be $49\ mC + (585\ ms * (139 + 185)\ mA) = 239\ mC$. Suppose now that the wireless card was active during the execution of this task. The total charge drawn in this case would be $49\ mC + (585\ ms * (139 + 185 + 161)\ mA) = 333\ mC$. Obviously, the charge drawn multiplied by the battery voltage gives the amount of energy dissipated. Since battery capacities are normally expressed in Ampère-hour, it seems more sensible to use the charge drawn to measure energy consumption (1 Ampère-hour = 3600 Coulombs).

A number of interesting observations can be drawn on inspection of Table 4.1. For example, we notice that the state transition tasks can take a long time to execute (1.3 seconds for webcam activation, 2.4 s to activate the wireless card and almost 3 seconds to bring up the board from sleep state). Likewise, the overall charge drawn can be relatively high. Also note that acquiring and compressing an image consumes over 2.5 times more charge than transmitting the compressed image, while transmitting a raw image (200KB) uses almost 4 times more charge than transmitting a full compressed image (48KB). All these considerations must be taken into account when specifying the system's duty cycle organization.

Table 4.2: Incremental charge drawn (in milli-Coulombs) and duration (in seconds) for different combinations of image acquisition and transmission related elementary tasks.

Combination of elementary tasks	Duration (s)	Incremental Charge (mC)
Acquire and save raw image, then transmit raw image	1.965	181
Acquire, compress and save jpg image, then transmit compressed image	1.116	115
Acquire and process image (no object detected)	1.129	88
Acquire and process image, then compress and save blob, transmit 30K jpg image	1.492	128

Another important observation is that, if no event is detected, acquiring and analyzing the image with our foreground detection algorithm draws about 7 mC more than acquiring and compressing the full image. In the case an event is detected, acquiring, analyzing and compressing an image portion draws about 24 mC more than acquiring and compressing the full image [1].

Given the different costs (both for incremental charge and duration) of the *sensing/processing* and *communication* elementary tasks, it is worth to compare different possible strategies, namely: image acquisition and transmission of raw image); image acquisition, compression and transmission; image acquisition and processing with background subtraction algorithm, followed by transmission of compressed portions of images when events are detected. Table 4.2 shows the incremental charge and duration of the combined elementary tasks. Acquiring and transmitting a raw image is the most expensive combination of tasks, lasting almost 2 seconds. Acquiring, compressing and saving a compressed image, followed by trans-

[1] Note that there is a large standard deviation in this case because the size of the compressed image may vary significantly.

mission lasts 1.1 s, while consuming 115 mC. If no event is detected, acquiring and processing the image lasts about the same, but consumes 27 mC less than the previous case. Consider now the case an event is detected. The cost of acquiring, analyzing, compressing and transmitting an image portion draws about 13 mC more than just acquiring, compressing and transmitting the full image, while lasting almost 400 ms longer. Intuitively, this means that if the rate of event detection is low, on-board event detection may lead to significant energy savings. On the other hand, if event detection is high, acquiring, compressing and transmitting the full image costs less to the node.

It is worth pointing out that although transmitting a larger image might be cheaper to the node, the transmission costs presented here take into account only one-hop transmission, and in a multi-hop network the overall cost will be much higher. Another aspect is that if the compression or processing is not done on the node, it will have to be done on the sink.

4.4 Duty Cycle Energy Consumption Analysis

The energy consumption characterization of the elementary tasks presented in the previous section makes it possible to compare different duty cycle organizations that yield the same functionality with different energy requirements. For example, one possibility is to never put the node to sleep between two consecutive image measurements, thereby avoiding the overhead associated with activating and deactivating hardware modules as well as transitions from idle to sleep mode. Would this be more energy efficient than a cycle that puts the system to sleep after each image acquisition and transmission? Clearly, the answer depends on the interval between

two consecutive image acquisitions: if the interval is larger than some critical value, the savings associated with being in sleep mode offsets the overheads due to state transitions. In fact, differences between duty cycles may be more nuanced than this. A duty cycle may branch out into different paths, e.g., depending on whether an event was detected or not. Additionally, the image acquisition periods need not be constant. For example, if a moving body was detected in the scene, it may be desirable to take a number of pictures of it while it is still in the field of view of the camera. Hence, the node may wait for a shorter period of time to take the next shot.

These simple considerations highlight the fact that the energy analysis of a duty cycle is not trivial and may require appropriate statistical modeling [52]. The key insight, however, is that a given duty cycle is a sequence of elementary tasks, possibly with conditional branching points. Hence, the energy consumed by the duty cycle (and its duration), can, in principle, be predicted by simply adding up the energy (and the duration) of each single task in the sequence. In this section we validate this important hypothesis by way of direct current measurements.

We consider six different possible duty cycles. Three of them do not include image analysis for event detection (Figures 4.1, 4.2 and 4.3). We call these deterministic duty cycles because their execution always follows the same sequence of tasks. The other three duty cycles include visual event detection (Figures 4.5, 4.6 and 4.7), and are thus called conditional duty cycles given that the output of the visual processing module will determine the sequence of tasks being executed.

4.4.1 Measurement Setup

For the duty cycle measurements, we use the same setup described in Section 3.3.2.1, i.e. we use the HP E3631A power supply configured to provide 5.4V to power the Stargate. The HP34401A digital multimeter (DMM) is used to measure the current flow as the duty cycles are executed. This setup allows us to measure the current and then, via time integration, obtain the charge used by each duty cycle.

4.4.2 Deterministic Duty Cycles

The first three duty cycles, which do not include image analysis for event detection, can be described as follows:

- **Duty cycle (a)**: the node simply acquires an image, compresses and transmits it. Then the node stays in idle for a period T_1 = 5 seconds. Both sensing and communication cores are always activated. Figure 4.1 summarizes the tasks executed.

- **Duty cycle (b)**: similar to the previous case, but the sensor and communication cores are activated only when needed, and deactivated as soon as possible. Figure 4.2 summarizes the tasks executed.

- **Duty cycle (c)**: different from the second case in that the system is put to sleep, rather than kept idle for the same period T_1. This duty cycle includes the additional task of wireless card deactivation after system wake-up. Figure 4.3 summarizes the tasks executed.

The temporal profile for the current drawn during duty cycle (b) is shown in Figure 4.4. As depicted in the figure, one can see the variations in current drawn to activate the

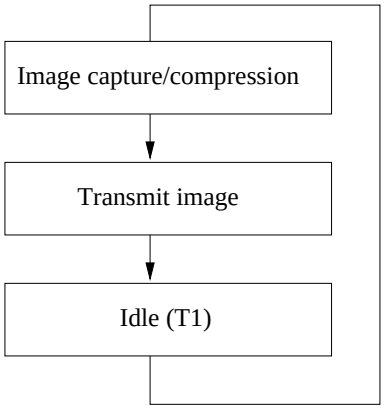


Figure 4.1: Elementary task sequences of duty cycle (a).

Table 4.3: Predicted and measured (averaged over twenty tests) charge drawn during the cycle are shown in the table below, along with the relative prediction errors.

	Charge (mC)		
Duty cycle	Predicted	Measured	Relative Error
(a)	3342	3038 ± 19	9.1%
(b)	2302	2433 ± 43	-5.7%
(c)	2284	2308 ± 20	-1.0%

webcam, followed by image acquisition, webcam deactivation, wireless card activation, image transmission, wireless card deactivation, and finally the node waits in idle mode until the next cycle starts.

Table 4.3 presents the average measured charge (with its standard deviation) drawn during each duty cycle, as well as the predicted charge according to Equation 4.1, based on the characterization of each elementary task given in Section 4.3.

Table 4.4 presents the average measured duty cycle duration (with its standard deviation) and the predicted duration according to Equation 4.2, based on the characterization of each elementary task given in Section 4.3.

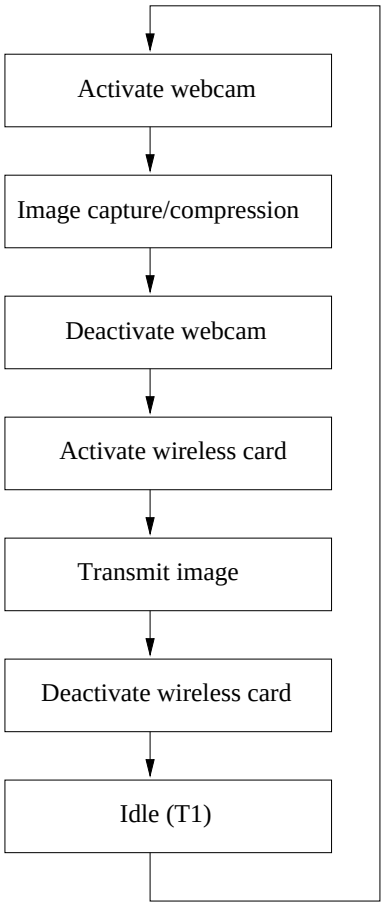


Figure 4.2: Elementary task sequences of duty cycle (b).

Table 4.4: Predicted and measured (averaged over twenty tests) cycle durations are shown in the table below, along with the relative prediction errors.

	Duration (ms)		
Duty cycle	Predicted	Measured	Relative Error
(a)	6584	6393 ± 38	2.9%
(b)	10931	11577 ± 78	-5.9%
(c)	14491	13999 ± 12	3.4%

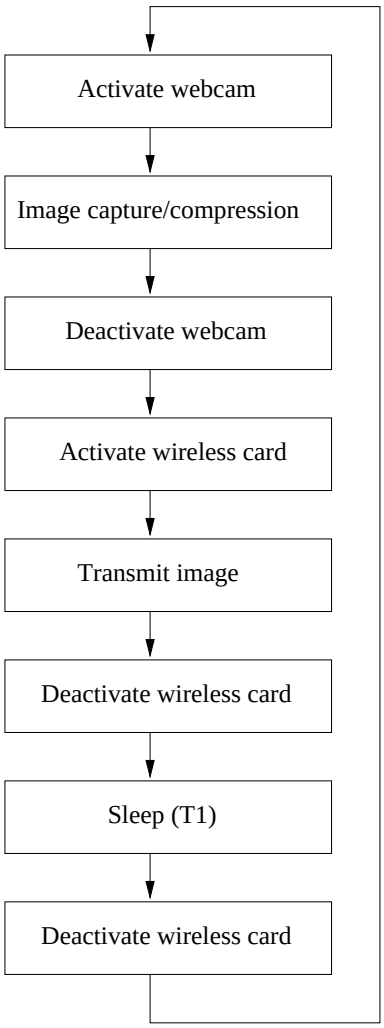


Figure 4.3: Elementary task sequences of duty cycle (c).

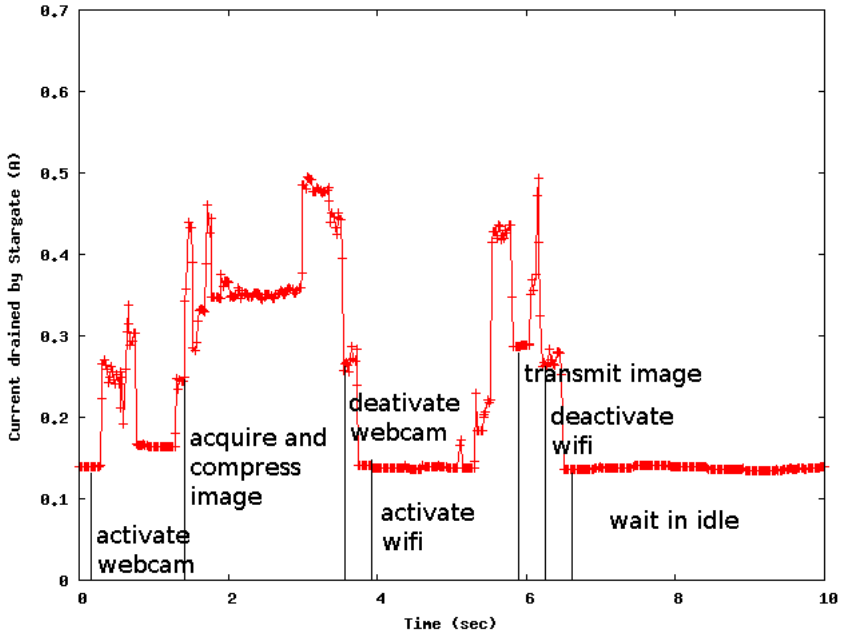


Figure 4.4: Temporal profile of current drawn in one cycle of duty cycle (b).

From Table 4.4, note that the cycles have very different durations. This can be explained by the fact that, while the amount of time (T_1) the system is idle or in sleep mode is constant, the different duty cycles include different transition tasks. The significant differences in the duration of the various duty cycles show that the overhead associated with state transitions contribute significantly to the overall cycle duration. In order to obtain the same image acquisition period, these duty cycles would have to have the same duty cycle duration, and thus the amount of idle/sleep time should be carefully tuned in each cycle.

The relative prediction error E_q for the charge drawn during one cycle is defined as 1 minus the ratio between the measured- and the predicted charge (the relative prediction error E_t for the cycle duration is similarly defined). It is seen from Tables 4.3 and 4.4 that E_q for

all the duty cycles of Figures 4.1, 4.2 and 4.3 is less than 10%. During our measurements for the duty cycles, we noticed some variation in the cycle duration (see the standard deviation for Table 4.4). By observing the temporal profiles of current drawn, we noticed that some of them presented a (variable) delay between the loading wireless card modules and actual transmission (node spend this delay mostly in idle mode). This delay is introduced by the program that transmits the image trying to connect to the server (it uses TCP) and most likely occurred due to wireless interference. The variation in the duty cycle duration reflects in the charge necessary to execute the duty cycle.

4.4.3 Conditional Duty Cycles

Figures 4.5, 4.6 and 4.7 show three different duty cycles that include visual event detection in which the task execution sequence depends on whether an event is detected or not. The basic idea is that if no event is detected, the system is put in sleep or idle mode for $T_1 = 5$ seconds. Otherwise, the system remains idle for $T_2 = 3$ seconds. The three duty cycles can be described as follows:

- **Duty cycle (d)**: the node starts in idle mode (only with the processing core active). Then it activates the webcam, acquires and process an image and then deactivates the webcam. If no event is detected, the system is put into sleep mode for $T_1 = 5$ seconds. Otherwise, the system activates the wireless card, transmits the image, deactivates the wireless card and then stays in idle for a period T_2 = 3 seconds. Figure 4.5 summarizes the tasks executed.

- **Duty cycle (e)**: this case, shown in Figure 4.6, is a variation of the previous one, where the node stays in idle mode instead of going to sleep when no event is detected.

- **Duty cycle (f)**: this is the simplest case (see Figure 4.7), where all cores (processing, sensing and communication) remain active all the time and the system stays in idle in between image acquisitions.

The temporal profile for the current drawn during Duty cycle (d) when no event is detected is shown in Figure 4.8. The figure depicts the current draw as the node engages in the following task sequences: the node starts sleeping (9mA), and then it *wakes up*, going to idle mode which activates the wireless card (as mentioned before, this is done automatically within the kernel), then deactivates the card, activates the webcam, acquires image and processes it. Since no object was detected, it deactivates the webcam, and finally goes into sleep mode.

The predicted and average measured charge drawn for each branch of each duty cycle, along with the relative prediction errors, are shown in Table 4.5. The predicted and average measured cycle durations, along with the relative prediction errors, are summarized in Table 4.6.

Note that the relative error for both the charge as well as for the duration presented in Tables 4.5 and 4.6 is less than 10%, for all cases. Again, similarly to what happened to our measurements for duty cycle (c), we noticed considerable variation in the duty cycle duration for (d) and (e) when an event is detected (see the standard deviation for the second row on Table 4.6). Two aspects contribute to this variation: first, there is a (variable) delay between the loading wireless card modules and actual transmission (node spend this delay mostly in

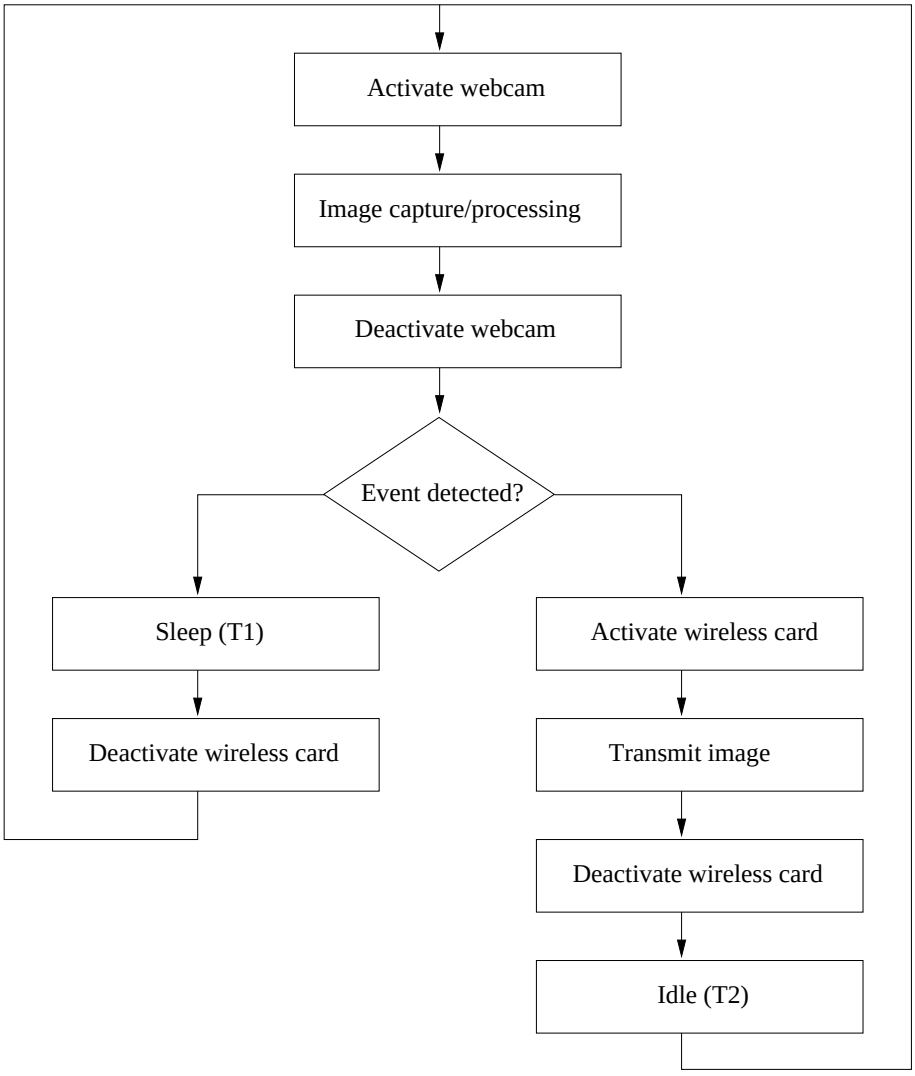


Figure 4.5: Elementary task sequences of duty cycle (d).

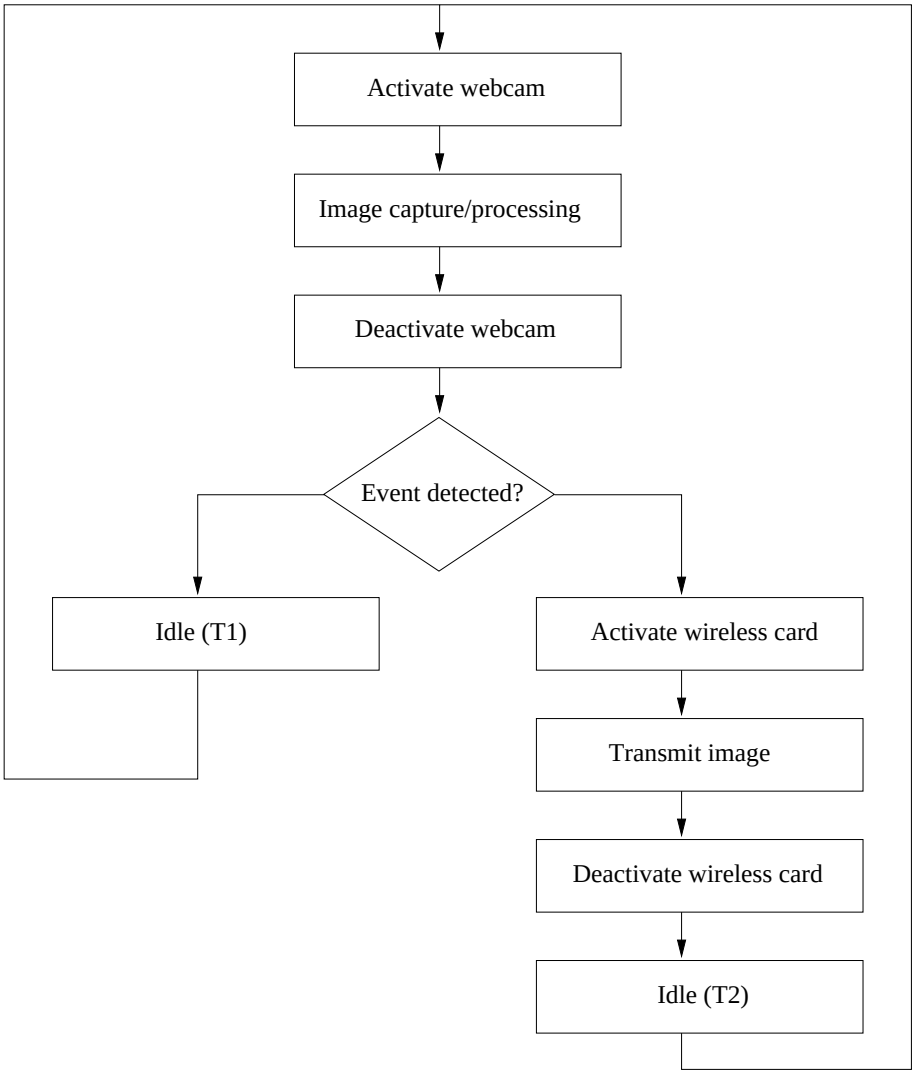


Figure 4.6: Elementary task sequences of duty cycle (e).

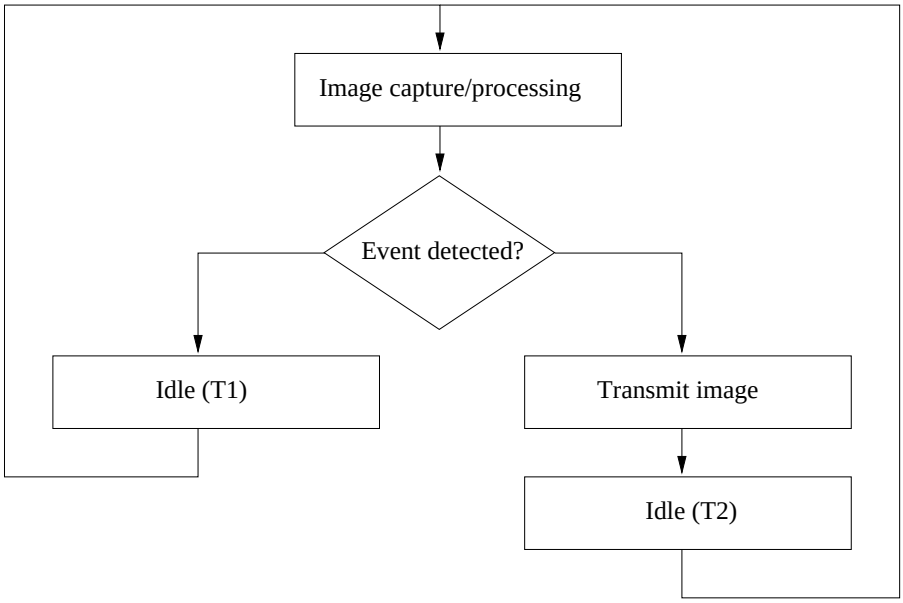


Figure 4.7: Elementary task sequences of duty cycle (f).

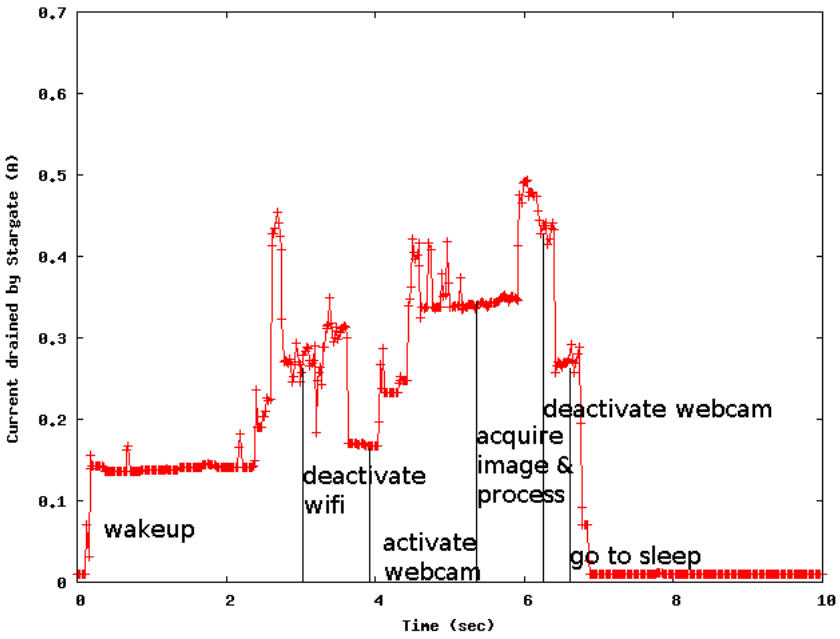


Figure 4.8: Temporal profile of current drawn during a cycle of duty cycle type (d) when no event is detected.

Table 4.5: Predicted and measured (averaged over twenty tests) charge drawn during the cycle are shown in the table below, along with the relative prediction errors. Note that the event detection branches in duty cycles (d) and (e) are the same, and thus values are omitted in this table.

	Charge (mC)		
Duty cycle	Predicted	Measured	Relative Error
(d) - no event	1565	1602 ± 15	-2.3%
(d) - event	1989	2176 ± 115	-9.4%
(e) - no event	1583	1688 ± 7	-6.7%
(f) - no event	3072	2915 ± 22	5.1%
(f) - event	2316	2127 ± 18	8.2%

Table 4.6: Predicted and measured (averaged over twenty tests) cycle durations are shown in the table below, along with the relative prediction errors. Note that the event detection branches in duty cycles (d) and (e) are the same, and thus values are omitted in this table.

	Duration (ms)		
Duty cycle	Predicted	Measured	Relative Error
(d) - no event	10973	11000 ± 13	-0.2%
(d) - event	8948	9536 ± 289	-6.6%
(e) - no event	7786	8300 ± 17	-6.6%
(f) - no event	6129	6199 ± 48	-1.1%
(f) - event	4452	4398 ± 34	1.2%

idle mode); and second, there is the variation in the camera perception of the detected object (due to the different positions the object could be relative to the camera), which affects the processing of the image and compression of the sub-image, i.e. the relevant portion of the image to be transmitted (note the standard deviation "*acquire, process and compress image portion*" when an event is detected shown in the tenth row of Table 4.1), and the actual duration of the transmission of the sub-image (note the different transmission duration for the different image sizes presented in Table 4.1).

4.5 Discussion

In Section 4.4, we described the results we obtained for the duty cycle prediction based on elementary task composition. In general, we noticed that the relative error was at most 10%. This discrepancy between our measurements and results from our model based on task composition can be explained by a number of factors, including variations in the image size obtained by image processing and also in the duration of the transmission task. In this section, we investigate these factors in more detail. More specifically, it is necessary to carefully evaluate the possible issues that affect the duty cycle prediction accuracy. The first step in this evaluation is to look at the definition of the duration of a task (and duty cycle) from the measurement perspective (Section 4.5.1) versus its duration from an OS (Operating System) perspective. Then we address how the model could be modified to decrease the relative error (Sections 4.5.2 and 4.5.3), and summarize what we have learned in Section 4.5.4.

4.5.1 Duration Issues

As we described in Sections 4.4.1 and 3.3.2.1, we obtained direct measurements of current drawn using a DMM. The DMM was connected to a computer, which through a LabVIEW [33] program recorded time-stamped DMM readings. Recall that the duration of a task was computed as the time interval between when the current level changes from idle and when the current level goes back to idle. To obtain the corresponding charge consumed by a duty cycle (or task), the current is integrated over the duration of the duty cycle (task), following the same methodology described above to determine the duty cycle (task) duration.

The question we pose in this section is how different is the measured task/duty cycle duration when compared to the duration reported by the operating system. This will provide us insight into how this difference affects the error between the measured duty cycle and the duty cycle composed by elementary tasks. Similarly to the results reported in Section 3.3.4, we use the operating system *time* utility to look into the time necessary to execute all tasks listed in Table 4.1, and summarize results in Table 4.7. Note that results obtained with the *time* utility for all elementary tasks are consistently smaller than the corresponding results for the measured durations.

We define the relative error E_d for the duration of the elementary task as one minus the ratio between the OS and measured durations. From Table 4.7 we observe that the relative error varies a lot depending on the elementary task.

The first elementary tasks to observe are the ones related to hardware activation and deactivation, i.e. activate and deactivate webcam and wireless card. It is clear that the time

Table 4.7: OS and measured duration (in milli-seconds) for the elementary tasks considered. Average values and standard deviations were computed over twenty runs.

Task	OS duration (ms)	Duration (ms)	Rel. Error
Activate webcam	335 ± 0	1301 ± 27	74%
Deactivate webcam	189 ± 18	356 ± 11	47%
Activate wireless card	1972 ± 0.4	2316 ± 329	15%
Deactivate wireless card	42 ± 22	374 ± 80	89%
Acquire and compress image	1028 ± 3	1227 ± 17	16%
Acquire and process image (no event)	1075 ± 25	1129 ± 21	5%
Acquire, process and compress sub-image	1207 ± 52	1244 ± 56	3%
Send 7KB (compressed image portion)	23 ± 2	123 ± 12	81%
Send 20KB (compressed image portion)	82 ± 11	207 ± 11	60%
Send 28KB (compressed image portion)	116 ± 18	248 ± 15	53%
Send 48KB (full compressed image)	176 ± 25	356 ± 16	51%

necessary for the operating system to execute the corresponding program to activate/deactivate the hardware is smaller than the actual time necessary for the corresponding hardware subsystem to become active/inactive. For both the cases of activating webcam and wireless card, we already introduce an additional one second delay in the duty cycle control script to allow the hardware to be ready to be used by the next elementary task. Therefore, the duration difference for activating the webcam, which is about one second, should not influence the duty cycle prediction. On the other hand, the delay introduced to wait for the wireless card to be ready will increase the duty cycles that use this task by 655 ms, which is the difference between the durations.

For the case of the image acquisition and compression elementary task, there is 16% difference between OS and model duration for the task (see Table 4.7), while for the image acquisition and processing elementary task, it is at most 5%. Since the former has to write a

full compressed image to a file (in RAM disk), the difference is likely due to I/O. This could be further verified by looking at the different image sizes.

Another group of elementary tasks that must be carefully observed is the one associated with image transmission. The relative error for these elementary tasks range from 51% to 81%, depending on the image size. The first aspect to consider is that the communication program we employ uses TCP as its transport protocol (and thus data might be stored in the TCP buffer when the program terminates, or data may still be in transit). If this happens, from the OS perspective the execution finished, but the hardware is still working (i.e., there is still I/O going on and thus electrical activity). Yet another aspect, is the fact that because we employ wireless communication, wireless channel impairments (e.g., interference, noise, etc.) may introduce additional, non-deterministic delays in data transfer.

We already looked into the duration issues related to the activation of the webcam and wireless card; image acquisition, compression and processing; and image transmission. We still need to look at the issues associated with system sleep/wakeup, and webcam and wireless card deactivation.

We use the command *sys_suspend (duration)* to put the system to sleep, and it wakes up when an interruption is generated after *duration* seconds. Since from the OS perspective this is only one task, we can not use the *time* utility to obtain the duration of the elementary tasks *go to sleep* and *wakeup*. Therefore, we will verify if the measured duration of the sleep period is the same given as a parameter to the *sys_suspend (duration)* utility. The measured duration of the sleep period was about 4.3 s long instead of the expected duration of 5 s, which is smaller than expected.

According to Table 4.7, the difference in the wireless card deactivation elementary task is significant (i.e., the measured duration of the elementary task deactivate wireless card is about 374 ms, while the OS duration is about 40 ms). One way to look at how this will affect the duty cycle duration is to look at the measured duration of the idle period following this task. In this case, the measured duration of the idle period was about 4.6 s instead of 5 s, or 2.6 s instead of 3 s. If the idle period is preceded by elementary tasks related to image acquisition and visual processing, or communication, there was no significant different in the duration of the measured and expected idle periods. Therefore, as previously pointed out, this difference in the duration of the idle period can be explained by the additional time necessary for the hardware to conclude the task of deactivating the wireless card.

4.5.2 Adjusting the Model

based on the previous discussion on potential error sources that cause discrepancy between the measured duty cycle duration and the one composed by elementary tasks, in this section we will explore different alternatives to adjust the proposed task composition model. The goal is to try to reduce the gap between the measurements and the model for the duty cycle.

The different adjustment alternatives include:

- Adjust duration of the elementary task associated with wireless card activation (should be 656 ms longer) and webcam activation (should be 34 ms longer).
- Adjust the sleep duration (it should be 4.3 s long from our measurements) for duty cycles (c) and (d) - no event detected.

- Adjust the idle period duration for duty cycles where wireless card deactivation precede it, i.e. duty cycles (b), (d) - event detected and (e) - event detected.

- Combine all previous adjustments to obtain the final correction.

Table 4.8 shows the predicted duty cycle durations adjusted using the first alternative described above, as well as the relative error E_d. The first alternative adjusts the duration of the elementary tasks associated with wireless card activation, increasing it by 656 ms, and webcam activation, increasing it by 34 ms, and in this way the additional one second delay in the duty cycle control script will be properly accounted for. Also, the charge associated with these elementary tasks were adjusted, considering that the system was in idle mode during the period increased in the duration. The adjusted duty cycles are the ones affected by the wireless card and webcam activation, i.e., duty cycles (b), (c), (d) and (e). As expected, the duty cycles affected by the wireless card activation are the ones that show larger variation in the relative error. Note that duty cycles (b), (d) and (e) presented negative relative prediction errors for the duration, which means that the prediction was smaller than the actual duty cycles, and thus confirms that the adjustment reduced the gap between the prediction and the measurements.

Table 4.9 shows adjusted predicted duty cycle duration, as well as the relative error E_d, of the duty cycles affected by the sleep period, i.e. duty cycles (c) and (d), when no event is detected. Since the sleep period decreased (recall that the measured duration of the sleep period was about 4.3 s long instead of the expected duration of 5 s), the relative prediction error increased and underestimated the duration.

Table 4.10 shows adjusted predicted duty cycle duration, as well as the relative error

Table 4.8: Duty cycle durations: measurements (averaged over twenty tests) and adjusted prediction with wireless card and webcam activation corrected times are shown in the table below, along with the relative errors (E_d) for the original and adjusted predictions. Note that the event detection branches in duty cycles (d) and (e) are the same, and thus values are omitted in this table.

	Duration (ms)			
Duty cycle	Measurement	E_d Orig. Pred.	Adjusted Prediction	E_d Adj. Pred.
(b)	11577 ± 78	-5.9%	11.621	0.4%
(c)	13999 ± 12	3.4%	15.181	7.8%
(d) - no event	11000 ± 13	-0.2%	11.007	0.1%
(d) - event	9536 ± 289	-6.6%	9.638	1.1%
(e) - no event	8300 ± 17	-6.6%	7.820	-6.1%

Table 4.9: Duty cycle durations: measurements (averaged over twenty tests) and adjusted prediction with sleep period corrected times are shown in the table below, along with the relative errors (E_d) for the original and adjusted predictions.

	Duration (ms)			
Duty cycle	Measurement	E_d Orig. Pred.	Adjusted Prediction	E_d Adj. Pred.
(c)	13999 ± 12	3.4%	13.775	-1.6%
(d) - no event	11000 ± 13	-0.2%	10.257	-7.2%

Table 4.10: Duty cycle durations: measurements (averaged over twenty tests) and adjusted prediction with idle period corrected times are shown in the table below, along with the relative errors (E_d) for the original and adjusted predictions. Note that the event detection branches in duty cycles (d) and (e) are the same, and thus values are omitted in this table.

	Duration (ms)			
Duty cycle	Measurement	E_d Orig. Pred.	Adjusted Prediction	E_d Adj. Pred.
(b)	11577 ± 78	-5.9%	10531	-9.9%
(d) - event	9536 ± 289	-6.6%	8548	-11.6%

E_d, of the duty cycles affected by the idle period preceded by wireless card deactivation, i.e. duty cycles (b), (d) - event detected and (e) - event detected. Since the idle period decreased (recall that the measured duration of the idle period was about 4.6 s instead of 5 s for duty cycle (b), and 2.6 s instead of 3 s for duty cycles (d) - event detected and (e) - event detected), the relative prediction error increased and underestimated the duration.

Table 4.11 shows predicted duty cycle duration adjusted using a combination of the three alternatives previously described, as well as the relative error E_d. The duty cycles affected by the wireless card and webcam activation, and the sleep and idle periods simultaneously are duty cycles (b), (c), (d) and (e). Note that for duty cycles (b), (d) - event detected and (e), the relative prediction error decreased significantly, while it decreased slightly for duty cycle (c). As for duty cycle (d) - no event detected, the relative prediction error actually increased, which is primarily due the adjustment in the sleep period.

Table 4.12 shows the predicted charge adjusted using a combination of the three alternatives previously described, as well as the relative error E_d. For all duty cycles adjusted, the relative prediction error decreased significantly.

Although all these adjustments can not be applied in practice, they are really helpful

Table 4.11: Duty cycle durations: measurements (averaged over twenty tests) and adjusted prediction affected by the wireless card and webcam activation as well as the sleep and idle periods simultaneously are shown in the table below, along with the relative errors (E_d) for the original and adjusted predictions. Note that the event detection branches in duty cycles (d) and (e) are the same, and thus values are omitted in this table.

	Duration (ms)			
Duty cycle	Measurement	E_d Orig. Pred.	Adjusted Prediction	E_d Adj. Pred.
(b)	11577 ± 78	-5.9%	11221	-3.2%
(c)	13999 ± 12	3.4%	14481	3.3%
(d) - no event	11000 ± 13	-0.2%	10307	-6.7%
(d) - event	9536 ± 289	-6.6%	9238	-3.2%
(e) - no event	8300 ± 17	-6.6%	7820	-6.1%

Table 4.12: Duty cycle charge: measurements (averaged over twenty tests) and adjusted prediction affected by the wireless card and webcam activation as well as the sleep and idle periods simultaneously are shown in the table below, along with the relative errors (E_d) for the original and adjusted predictions. Note that the event detection branches in duty cycles (d) and (e) are the same, and thus values are omitted in this table.

	Charge (mC)			
Duty cycle	Measurement	E_d Orig. Pred.	Adjusted Prediction	E_d Adj. Pred.
(b)	2433 ± 43	-5.7%	2398	-1.5%
(c)	2308 ± 20	-1.0%	2419	4.6%
(d) - no event	1602 ± 15	-2.3%	1608	0.4%
(d) - event	2176 ± 115	-9.4%	2085	-4.4%
(e) - no event	1688 ± 7	-6.7%	1588	-6.3%

in understanding and verifying where the prediction error comes from.

4.5.3 Another Approach

Given the fact that we had to adjust the proposed model because of the differences between the measured and the OS durations of the elementary tasks, one could wonder why not to use the OS duration as the basis for the prediction model. In this Section, we address this approach.

We still use Equation 4.2 to calculate the duration of the duty cycle, but we use the duration of elementary tasks from the OS perspective, presented in Table 4.7. To obtain the charge for a given elementary task in this approach, we divide the measured charge by the measured duration of each task, and then multiply it by the OS duration. This will give the incremental charge for each task (equivalent of Table 4.1). Using this and the each task duration (from Table 4.7), we obtain the duty cycle OS-time based prediction. Table 4.13 shows the resulting predicted duration, which has larger relative error than the original model, and also under-estimates the duration for all duty cycles presented.

Table 4.14 shows the resulting predicted charge. For duty cycles (a) and (f), which do not have wireless card and webcam activation and deactivation, the relative prediction error for the charge is smaller than the original model. But for duty cycles (b), (c), (d) and (e), the relative prediction error is larger than the the original model, besides under-estimating it.

Table 4.13: Duty cycle durations: measurements (averaged over twenty tests) and OS-time based prediction are shown in the table below, along with the relative errors (E_d) for the original and OS-time predictions. Note that the event detection branches in duty cycles (d) and (e) are the same, and thus values are omitted in this table.

	Duration (ms)			
Duty cycle	Measurement	E_d Orig. Pred.	OS-time Pred.	E_d OS-time Pred.
(a)	6393 ± 38	2.9%	6204	-3.1%
(b)	11577 ± 78	-5.9%	10741	-7.8%
(c)	13999 ± 12	3.4%	13970	-0.2%
(d) - no event	11000 ± 13	-0.2%	10786	-2.0%
(d) - event	9536 ± 289	-6.6%	8921	-6.9%
(e) - no event	8300 ± 17	-6.6%	7599	-9.2%
(f) - no event	6199 ± 48	-1.1%	6075	-2.0%
(f) - event	4398 ± 34	1.2%	4289	-2.5%

Table 4.14: Duty cycle charge: measurements (averaged over twenty tests) and OS-time based prediction are shown in the table below, along with the relative errors (E_d) for the original and OS-time predictions. Note that the event detection branches in duty cycles (d) and (e) are the same, and thus values are omitted in this table.

	Charge (mC)			
Duty cycle	Measurement	E_d Orig. Pred.	OS-time Pred.	E_d OS-time Pred.
(a)	3038 ± 19	9.1%	3124	2.7%
(b)	2433 ± 43	-5.7%	2124	-14.5%
(c)	2308 ± 20	-1.0%	2013	-14.7%
(d) - no event	1602 ± 15	-2.3%	1397	-14.6%
(d) - event	2176 ± 115	-9.4%	1897	-14.7%
(e) - no event	1688 ± 7	-6.7%	1509	-11.9%
(f) - no event	2915 ± 22	5.1%	3042	4.2%
(f) - event	2127 ± 18	8.2%	2218	4.1%

4.5.4 Lessons Learned

The first observation is that the time necessary for the operating system to execute the tasks associated hardware transitions is smaller than the actual time necessary for the corresponding hardware sub-system to become active/inactive. The duty cycle modeling needs to account for that in order to improve its accuracy. Also, for the elementary tasks associated with image transmission there is a considerable difference between the time the OS takes to execute it and the time the model uses (51% up to 81%). This is probably due interference in the wireless network, and data stored in the TCP buffer.

Given these differences in the duration, which in turn affect the relative prediction error for the duty cycle, one could consider using elementary tasks with a smaller granularity (but still not at instruction level). For example, we could divide the image acquisition and compression elementary task into two, but this introduce an overhead of saving the image data after acquisition and reading it before compression. Considering the set of elementary tasks we have (Table 4.1), the tasks are already ate their minimal granularity (transition tasks) or dividing them would introduce overhead and increase energy consumption, therefore the granularity defined for the set of elementary tasks is appropriate.

Tables 4.15 and 4.16 summarize the relative errors (E_d) for the original, adjusted and OS-time predictions for both duty cycle duration and charge, respectively. Overall, the original adjusted approach represents a better solution for duty cycle duration and charge prediction.

Table 4.15: Summary of relative errors (E_d) for the original, adjusted and OS-time predictions for duty cycle duration. Note that the event detection branches in duty cycles (d) and (e) are the same, and thus values are omitted in this table.

	Duration		
Duty cycle	E_d Original Prediction	E_d Adjusted Prediction	E_d OS-time Prediction
(a)	2.9%	-	-3.1%
(b)	-5.9%	-3.2%	-7.8%
(c)	3.4%	3.3%	-0.2%
(d) - no event	-0.2%	-6.7%	-2.0%
(d) - event	-6.6%	-3.2%	-6.9%
(e) - no event	-6.6%	-6.1%	-9.2%
(f) - no event	-1.1%	-	-2.0%
(f) - event	1.2%	-	-2.5%

Table 4.16: Summary of relative errors (E_d) for the original, adjusted and OS-time predictions for duty cycle charge. Note that the event detection branches in duty cycles (d) and (e) are the same, and thus values are omitted in this table.

	Charge		
Duty cycle	E_d Original Prediction	E_d Adjusted Prediction	E_d OS-time Prediction
(a)	9.1%	-	2.7%
(b)	-5.7%	-1.5%	-14.5%
(c)	-1.0%	4.6%	-14.7%
(d) - no event	-2.3%	0.4%	-14.6%
(d) - event	-9.4%	-4.4%	-14.7%
(e) - no event	-6.7%	-6.3%	-11.9%
(f) - no event	5.1%	-	4.2%
(f) - event	8.2%	-	4.1%

4.6 Related Work

Previous work has provided energy consumption values for other types of simpler sensor network platforms, such as Berkeley's motes [68, 82], or for small granularity tasks involving switching the state of a microprocessor [6]. A detailed low-level analysis of the power cost for lossless compression and transmission in a StrongARM processor was presented in [5].

Sanli et al [64] consider the transition costs (time and energy overhead) when evaluating the energy efficiency of the collaborative task scheduling mechanism proposed on a simulated environment using Mica motes as the sensor nodes. The drawback in their approach is that they do not have any measurements to support the model they use for the transition costs, and the energy cost (or overhead as authors say) of a transition is obtained by averaging the power cost of the high and low states, and then multiplying it by its duration.

Han et al. [26] discusses energy efficient data collections mechanisms for sensor networks, and present a series of state transition models to optimize energy consumption on the sensor node. The drawback in their work is the fact that the processor and sensor device is always on, and they only consider state transitions for the radio.

Sinha and Chandrakasan [70] prerent a workload prediction strategy based on adaptive filtering of the past workload profile. The power-aware sensor node model used by the authors is similar to the one we use, considering the different combination of sub-systens states and the transitions cost.

4.7 Conclusions

Wireless networks of embedded visual processing nodes are very attractive for surveillance and monitoring. We have presented an analysis of the power requirement and execution time of the elementary tasks that compose a typical duty cycle of a visual sensing node based on the Crossbow Stargate board. This analysis is necessary to predict the node's lifetime when it is battery-operated, and to choose the most appropriate design parameters. Our study has highlighted the fact that activation and deactivation of hardware components, as well as the transition between idle and sleep mode of the node's processor core, may require considerable overhead energy and may take substantial time. We also showed how elementary tasks can be combined together to form different duty cycles performing the same function but with rather different energy requirements.

In addition, we validated our hypothesis that the energy consumed in a duty cycle, as well as its execution time requirements, can be estimated by simply adding up the relevant quantities for each elementary task involved. Modeling duty cycles by task composition achieves a relative error in the range of -10% to 10% for both the average duration and charge. We also discussed the issues associated with the task concatenation approach to determine a duty cycle, and how its accuracy is affected.

In order to achieve our ultimate goal of conducting a comprehensive study of visual sensor network's lifetime, we divided the work in three main steps: (1) characterize energy consumption of the node in our testbed (work reported in Section 3.3); (2) determine the cost (in terms of charge and duration) of the main tasks in a visual sensor network and the possible duty

cycles, as well as validate the hypothesis that energy consumption for a given duty cycle can be obtained by the composition of the energy consumed by the individual tasks (work reported in this chapter); and finally (3) develop and validate an energy consumption model based on task composition to determine a visual sensing node's lifetime, which is the work presented in the next chapter (Chapter 5).

It is worth pointing out that, although our work was done considering a specific visual sensing platform, i.e., the Meerkats node, the methodology we used could be applied to other platforms and applications. Essentially, the methodology we proposed includes: (1) characterizing energy consumption of the platform; (2) determining the cost (in terms of charge and duration) of the tasks representative of the target application; (3) extend/adapt the energy model based on task composition to determine the node's lifetime.

Chapter 5

Visual Sensor Node Lifetime Experiments and Prediction

In this Chapter, we extend the work on duty cycle energy consumption prediction described in Chapter 3 to predict the lifetime of a Meerkats node.

5.1 Introduction

In order to achieve our ultimate goal of conducting a comprehensive study of visual sensor network's lifetime, we divided the work in three main steps: (1) characterize energy consumption of the node in our testbed (work reported in Section 3.3); (2) determine the cost (in terms of charge and duration) of the main tasks in a visual sensor network and the possible duty cycles, as well as validate the hypothesis that energy consumption for a given duty cycle can be obtained by the composition of the energy consumed by the individual tasks (work described in Chapter 4); and finally (3) develop and validate an energy consumption model based on task

composition to determine a visual sensing node's lifetime (work presented in this chapter).

To accomplish step (3) above, we first executed a series of experiments utilizing different duty cycles. Each experiment consisted of running a given duty cycle repetitively to obtain statistically representative data on the lifetime of the Meerkats node when executing that duty cycle.

The remainder of this chapter is organized as follows. Section 5.2 presents long term experiments with repetitive duty cycles for the visual sensor node. In Section 5.3, we present an energy consumption model based on task composition to determine a node lifetime, as well its validation. Next, in Section 5.5, we apply this results to the Meerkats' testbed. Section 5.6 concludes the chapter.

5.2 Node Lifetime Experiments

Each experiment consists of continuously running a given duty cycle. Using the same setup described in Section 3.3.2, we changed the scripts that implement the duty cycles to periodically check the DCA (Discharge Current Accumulator) available through *batmon*, until 1000 mAh is discharged. Once this threshold is reached, the system is shutdown. By following this approach, we intend to mimic a real-life situation, where the sensor node would shutdown itself once the battery discharges down to a certain threshold. A continuous monitoring of the DCA through *batmon* would interfere too much with the system energy consumption. In order to avoid this, *batmon* checks the DCA status every 100 cycles. Note that this approach might lead to experiments that use more than 1000 mAh.

For the sake of clarity, we revisit the duty cycles' main features here. The duty cycles employed in our experiments, which were described in Section 4.4, can be classified into deterministic (duty cycles (a), (b) and (c)) and conditional (duty cycles (d), (e) and (f)).

- **Duty cycle (a)**: the node simply takes an image, compresses it and transmits, and then stays in idle for a period T_1 = 5 seconds. Both sensor and communication cores are always activated.

- **Duty cycle (b)**: similar to the previous case, but the sensor and communication cores are activated only when needed, and deactivated as soon as possible. Figure 4.2 summarizes the tasks executed.

- **Duty cycle (c)**: different from the second case in that the system is put to sleep, rather than kept idle for the same period T_1 of time. This duty cycle includes the additional task of wireless card deactivation after system wake-up.

- **Duty cycle (d)**: the node starts in idle mode (only with the processor core active). Then it activates the webcam, acquires and process an image and then deactivates the webcam. If no event is detected, the system is put into sleep mode for $T_1 = 5$ seconds. Otherwise, the system activates the wireless card, transmits the image, deactivates the wireless card and then stays in idle for a period T_2 = 3 seconds.

- **Duty cycle (e)**: this case is a variation of the previous one, where the node stays in idle mode instead of going to sleep in case no event is detected.

- **Duty cycle (f)**: this is the simplest case, where all cores (processor, sensor and com-

Table 5.1: The average duration of the experiment for deterministic duty cycles. We executed 10 runs of each experiment, and results presented are the average and standard deviation.

Duty cycle	Measured Lifetime (s)
(a)	7425 ± 301
(b)	17251 ± 3426
(c)	20053 ± 740

munication) remain active all the time and the system stays in idle in between image acquisitions.

Next, we present the results for the experiments for each of the six duty cycles described.

5.2.1 Deterministic Duty Cycles

Table 5.1 shows the duration of experiments for each of the deterministic duty cycles. From these results, it is clear that the lifetime of a node can be considerably extended by making use of low-power states and by deactivating unnecessary hardware components. This is why duty cycle (c), which deactivates hardware components not in use and change to low-power mode while waiting, has the longest lifetime of all three deterministic duty cycles.

Table 5.2 shows the charge calculated by time-integration (i.e., the current drained by the node throughout the experiment is integrated over the duration of the experiment) and the charge obtained by converting the final difference in the DCA register (DCA readings are in mAh, thus to obtain the equivalent in mC, we do $DCA * 3600/1000$), as well as the relative error between them.

While running experiments with duty cycles (b) and (c), we noticed that the program

Table 5.2: The average charge used by the experiment for deterministic duty cycles, as well as the charge obtained through *batmon* DCA register and their relative error. We executed 10 runs of each experiment, and results presented are the average and standard deviation.

	Charge (C)		
Duty cycle	Measured	*batmon* DCA register	Rel. Error
(a)	3598 ± 143	3707 ± 237	2.9%
(b)	3835 ± 739	3843 ± 110	0.2%
(c)	3546 ± 70	3750 ± 97	5.4%

running on the sink node was crashing after several images were sent. We further investigated this problem and noticed that the TCP connections were not being properly closed. The sender program running on the visual sensor node displayed a message acknowledging that the image was fully sent, but the sink program did not fully receive it, and thus did not close the TCP connection. This happens because the wireless card is deactivated as soon as the sender program finishes its execution; however the fact that the program terminated does not mean that all data on the TCP queue was sent. In order to fix this problem, we ran a simple test: one second delay was added at the sender program before deactivating the wireless card. This change in the duty cycle allowed all images to be successfully transmitted and the TCP connection to be properly closed.

5.2.2 Conditional Duty Cycles

Table 5.3 shows the duration of experiments for each of the conditional duty cycles. Again, from these results, it is clear that the lifetime of a node can be considerably extended by making use of low-power states and by deactivating unnecessary hardware components (experiments employing duty cycles (d) and (e) lasted at least twice more than duty cycles (f)). But

Table 5.3: The average duration of the experiment for conditional duty cycles. We executed 10 runs of each experiment, and results presented are the average and standard deviation over all 10 trials.

Duty cycle	Measured Lifetime (s)
(d)	20044 ± 1489
(e)	15863 ± 506
(f)	7352 ± 296

Table 5.4: The average charge used by the experiment for conditional duty cycles, as well as the charge obtained through *batmon* DCA register and their relative error. We executed 10 runs of each experiment, and results presented are the average and standard deviation over all 10 trials.

	Charge (C)		
Duty cycle	Measured	*batmon* DCA register	Rel. Error
(d)	3478 ± 146	3605 ± 203	3.5%
(e)	3534 ± 120	3812 ± 101	7.3%
(f)	3549 ± 129	3812 ± 100	6.9%

comparing these three duty cycles is not as straightforward as comparing the deterministic duty cycles, since duty cycle behavior depends on the outcome of visual processing. Also, these duty cycles do not have the same duration. Duty cycle (d) lasts 11.4 s if no event is detected and 9.6 s if an event is detected (same as duty cycle (e) when an event is detected), while duty cycle (e) lasts 7.8 s if no event is detected, but duty cycle (f) lasts 4.6 s if an event was detected and 6.1 s if not.

Table 5.4 shows the charge calculated by time-integration and the charge obtained by converting the final difference in the DCA register (DCA readings are in mAh, thus to obtain the equivalent in mC, we do $DCA * 3600/1000$), as well as the relative error between them.

As mentioned before, the behavior of the conditional duty cycles depends on the outcome of the visual processing module. Therefore we report the number of events detected in

Table 5.5: The average number of total cycles during the experiment duration, as well as the number of cycles with and without events detected for conditional duty cycles. We executed 10 runs of each experiment, and results presented are the average and standard deviation over all 10 trials.

	Number of Cycles		
Duty cycle	Total	with Events Detected	without Events Detected
(d)	1870 ± 109	550 ± 347	1337 ± 446
(e)	1750 ± 178	625 ± 345	1126 ± 498
(f)	1340 ± 70	507 ± 256	834 ± 209

each experiment, which was recorded during the experiments. Table 5.5 shows the average and standard deviation of the number of cycles per experiment, as well as the number of cycles in which events where detected.

It is important to point out that these experiments were conducted with the visual sensor located in the lab, where there was no control over the event generation, i.e. events could be generated by people walking in the node's field of view, lights being turned off, etc. This fact is illustrated in Tables 5.6, 5.7 and 5.8, which shows the actual number of total cycles, as well as the number of cycles with and without events detected for all runs of duty cycles (d), (e) and (f), respectively.

We also encountered the same problem reported earlier for duty cycles (b) and (c), now happening for duty cycles (d) and (e). The problem happens because the wireless card is deactivated too soon, i.e. as soon as the sender program finishes its execution, but before the image had been fully sent. We also tested changing the duty cycles by adding a one second delay after the sender program and before deactivating the wireless card, which allowed all images to be successfully transmitted and the TCP connection to be properly closed.

Table 5.6: The number of total cycles during the experiment duration, as well as the number of cycles with and without events detected for all runs of duty cycle (d).

	Number of Cycles		
Run	Total	with Events Detected	without Events Detected
1	2001	359	1642
2	1801	841	960
3	1801	351	1490
4	1801	433	1391
5	1701	1188	513
6	1901	293	1672
7	1901	667	1234
8	2001	327	1674
9	2001	22	2006
10	1801	1014	787

Table 5.7: The number of total cycles during the experiment duration, as well as the number of cycles with and without events detected for all runs of duty cycle (e).

	Number of Cycles		
Run	Total	with Events Detected	without Events Detected
1	1800	839	962
2	1500	775	726
3	1900	378	1523
4	1700	901	800
5	1900	428	1473
6	1400	1113	288
7	1900	101	1800
8	1800	413	1388
9	1700	1017	684
10	1900	288	1613

Table 5.8: The number of total cycles during the experiment duration, as well as the number of cycles with and without events detected for all runs of duty cycle (f).

	Number of Cycles		
Run	Total	with Events Detected	without Events Detected
1	1400	865	536
2	1400	412	989
3	1400	653	748
4	1400	510	891
5	1400	890	511
6	1300	181	1120
7	1300	417	884
8	1300	584	717
9	1200	108	1093
10	1300	446	855

5.2.3 Battery Experiments

All previous experiments were done using the Agilent E3631A power supply to power the Meerkats node. In this section, we report the results obtained when we executed experiments with the node powered by a 2-cell Lithium-Ion battery with 1000 mAh capacity.

We ran duty cycle (d) continuously, following the same methodology reported earlier in this section. Again, there was no control over the event generation during experiments.

One consideration when using the battery to power the sensor node is that the battery should not be utilized until its charge is totally depleted, since that might damage the battery. Another consideration about Lithium-Ion batteries is that its voltage is kept almost constant, up to a point where it drops abruptly. For the battery used in the experiments, this drop point is when the voltage reaches about 6.5 V, and it corresponds to about 80% of the battery capacity. The voltage profile of a Lithium-Ion battery being discharged is shown in Figure 5.1.

Table 5.9 summarizes the results for the battery experiments. The first experiment

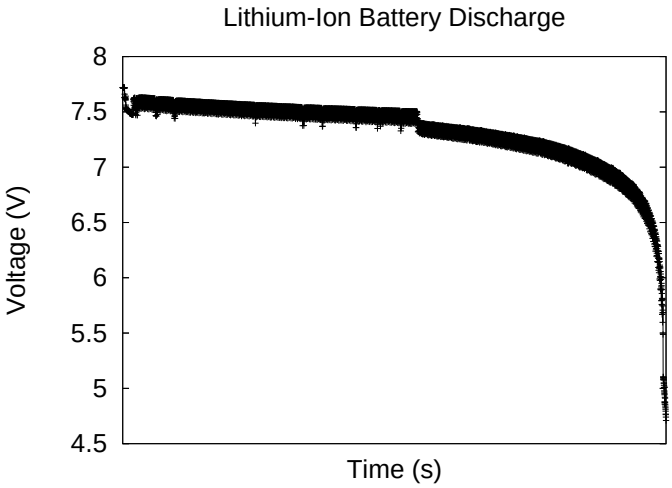


Figure 5.1: Voltage profile for discharge of a Lithium-Ion battery.

Table 5.9: The average measured duration and charge, as well as the charge obtained through *batmon* DCA register for the battery experiments for duty cycle (d).

		Charge (C)	
Run #	Measured Lifetime (s)	Measured	*batmon* DCA register
1	10577	3218	3751
2	15942	2311	2916

had its cut-off threshold set to 1000 mAh, while the second one had the value of 800 mAh.

We can not compare the charge obtained through the multimeter measurements with the *batmon* DCA readings for these experiments. This happens because the multimeter was monitoring the current provided to the switching regulator board and flowing out of the battery, which runs at 7.6 V. The *batmon* DCA readings are relative to the current flowing through the Stargate, which runs at 5 V (the voltage provided by the switching regulator board). Therefore the fair comparison would be in terms of power, i.e. charge times voltage. In computing power, we should account for the fact that the DC/DC switching regulator we use to convert the voltage

Table 5.10: The power calculated based on charge measurements and obtained through *batmon* DCA register for the battery experiments for duty cycle (d), as well as their relative error.

	Power (W)		
Run #	Measured	*batmon* DCA register	Rel. Error
1	19567	18756	-4.3%
2	14051	14580	3.6%

Table 5.11: The average number of total cycles during the experiment duration, as well as the number of cycles with and without events detected for duty cycle (d) running on a battery powered visual sensor node.

	Number of Cycles		
Run	Total	with Events Detected	without Events Detected
1	1300	1216	85
2	1500	206	1295

on the battery (7.6 V) to the voltage level expected by the Stargate (5 V) has an efficiency $\varepsilon_{regulator}$ of 80%. Equation 5.1 illustrates this.

$$
\begin{aligned}
P_{Battery} &= P_{Stargate} \\
Q_{Battery} * V_{Battery} &= \varepsilon_{regulator} * Q_{Stargate} * V_{Stargate} \\
Q_{Stargate} &= \varepsilon_{regulator} \frac{Q_{Battery} * V_{Battery}}{V_{Stargate}}
\end{aligned}
\tag{5.1}
$$

Table 5.10 presents the power calculated based on charge measurements and obtained through *batmon* DCA register for the battery experiments for duty cycle (d), as well as their relative error.

Table 5.11 shows the number of cycles per run, as well as the number of cycles when events were detected.

5.3 Lifetime Prediction for Deterministic Duty Cycles

We now continue to develop and validate our energy consumption model based on task composition to predict the lifetime of the Meerkats node. In this section, we consider deterministic duty cycles, i.e. the ones that follow a known sequence of tasks. Examples of such cycles are duty cycles (a) through (c) previously described in Section 4.4. In Section 5.4, we do the same exercise for conditional duty cycles.

The methodology follows the same hypothesis we used for duty cycle modeling by task composition (Section 4.2), i.e., that we can obtain duty cycle charge (and duration) by simply adding up the charge (and duration) of each single task in the sequence. Then, to obtain the lifetime of a node given its duty cycle (L_x, where x specifies the duty cycle type), we divide the total amount of charge available in the battery (Q_b) by the charge necessary for each duty cycle (Q_x) to obtain the number of cycles that can be executed (N_x), and then we multiply this by the duty cycle duration (D_x). Equation 5.2 summarizes this algorithm.

$$L_x = \frac{Q_b}{Q_d} * D_d \tag{5.2}$$

According to Equation 5.2, the lifetime L_x is a function of the charge consumed during a duty cycle, which can be obtained from direct measurements, or it can be obtained by task composition. In the latter case, we can expand Equation 5.2 by replacing Q_d with Equation 4.1 and D_d with Equation 4.2, thus obtaining Equation 5.3, where $\overline{q}(T_i)$ is the average charge of an elementary task T_i.

$$L_x = \frac{Q_b}{\sum_{i=1}^{n} \overline{q}(T_i)} * \sum_{i=1}^{n} \overline{d}(T_i) \tag{5.3}$$

For example, a duty cycle of type (a) (where the node takes an image, compresses and transmits it, and then stays in idle for a period T_1 = 5 seconds, with all hardware components active) uses about 3300 mC. A battery with 1000mAh has 3600 C. Assume all this charge is available to the node. Then we divide the charge available on the battery by the charge necessary per cycle to obtain the number of cycles ($3600/3.3 = 1091$). Finally we multiply the number of cycles (1091) by its duration (6584 ms), to obtain a lifetime of 7183 s (about 2 hours).

In Chapter 4.2, we described how to model duty cycle energy consumption through elementary task composition, as well as possible adjustments to the model, which we briefly revisit now. The adjustments we proposed to the model are related to hardware activation, as well as the actual duration of the sleep period and the duration of the idle period when preceded by wireless card activation.

In this section, we consider three possible schemes for lifetime prediction, namely:

- Lifetime prediction based on duty cycle measurements.
- Lifetime prediction based on the original duty cycle prediction model.
- Lifetime prediction based on the adjusted duty cycle prediction model.

In the first scheme, we use the measured charge consumed during a duty cycle as Q_d and the measured duration of a duty cycle as D_d in Equation 5.2 to obtain the expected lifetime. In the second and third schemes, Q_d and D_d are obtained by task composition, and therefore we

Table 5.12: The charge used by one duty cycle and its duration obtained through measurements, as well as the predicted number of cycles and lifetime for deterministic duty cycles based on duty cycle measurements, and the relative error comparing the prediction and the measurements.

	Duty Cycle Measurements		Lifetime Prediction		
Duty cycle	Charge (C)	Duration (s)	Number of Cycles	Lifetime (s)	Rel. Error
(a)	3.038	6.4	1185	7575	2.0%
(b)	2.433	11.6	1480	17130	-0.7%
(c)	2.308	13.9	1560	21840	8.2%

Table 5.13: The charge used by one duty cycle and its duration obtained by task composition, as well as the predicted number of cycles and lifetime for deterministic duty cycles based on duty cycle original prediction, and the relative error comparing the prediction and the measurements.

	Duty Cycle Orig. Prediction		Lifetime Prediction		
Duty cycle	Charge (C)	Duration (s)	Number of Cycles	Lifetime (s)	Rel. Error
(a)	3.342	6.6	1077	7091	-4.7%
(b)	2.302	10.9	1564	17092	-0.9%
(c)	2.284	14.5	1576	22837	12.2%

use Equation 5.3 to predict the lifetime. The difference between the second and third schemes is that in the third scheme we utilize the adjusted duty cycle model by task composition.

By comparing results obtained using these three approaches, we can understand how effectively we can predict a visual sensor node lifetime.

Tables 5.12, 5.13 and 5.14 summarize lifetime prediction results for the three deterministic duty cycles - (a), (b) and (c) - using the three different approaches described.

The best lifetime prediction results are obtained when using the charge and duration measured for the duty cycles, as expected. On the other hand, when comparing results obtained with the original and adjusted duty cycle prediction model based on elementary task composition, the lifetime prediction based on the adjusted model improves the results only for duty cycle (c). In summary, the overall results are encouraging. Our duty cycle energy consumption pre-

Table 5.14: The charge used by one duty cycle and its duration obtained through adjusted task composition, as well as the predicted number of cycles and lifetime for deterministic duty cycles based on duty cycle adjusted prediction, and the relative error comparing the prediction and the measurements.

	Duty Cycle Adj. Prediction		Lifetime Prediction		
Duty cycle	Charge (C)	Duration (s)	Number of Cycles	Lifetime (s)	Rel. Error
(a)	-	-	-	-	-
(b)	2.398	11.2	1501	16843	-2.4%
(c)	2.419	14.5	1488	21552	7.0%

diction model has a relative error of at most 10%, and this error does not increase significantly the relative lifetime prediction error.

Another aspect that contributes to the relative lifetime prediction error is the fact that this lifetime experiments measurements also include two tasks, which are considered as overhead and are not part of the prediction mechanism. The first task is a *camera stabilization* program, which is part of the visual processing module, and adjusts the parameters of the webcam according to the environment. This program is executed once, in the beginning of the experiment. The second task, is the *shutdown* procedure, which is the last task executed in the experiment. Besides these two tasks, there is control overhead for the experiments: monitoring the discharge current accumulator using *batmon* (already discussed), saving information on event detection and number of cycles executed.

5.4 Lifetime Prediction for Conditional Duty Cycles

We can extend the algorithm presented in Section 5.3 to consider a fixed probability of event detection, and then predict the lifetime of the node. Equation 5.4 gives the expression

Table 5.15: The predicted number of cycles and lifetime for conditional duty cycles based on duty cycle measurements, as well as the relative error comparing the prediction and the measurements.

Duty cycle	Number of Cycles	Lifetime (s)	Rel. Error
(d)	2033	21493	6.7%
(e)	1923	16895	6.1%
(f)	1375	7590	3.1%

Table 5.16: The predicted number of cycles and lifetime for conditional duty cycles based on duty cycle original prediction, as well as the relative error comparing the prediction and the measurements.

Duty cycle	Number of Cycles	Lifetime (s)	Rel. Error
(d)	2131	22110	9.3%
(e)	2083	17083	7.1%
(f)	1292	7099	-3.6%

for the lifetime in the case of the conditional duty cycles considering the probability of detecting an event P_{d-ev}.

$$L_x = \frac{Q_b}{Q_{d-ev} * P_{d-ev} + Q_{d-no} * (1 - P_{d-ev})} * D_d \tag{5.4}$$

Again, we will consider three possible schemes for lifetime prediction, namely:

- Lifetime prediction based on duty cycle measurements.
- Lifetime prediction based on the original duty cycle prediction model.
- Lifetime prediction based on the adjusted duty cycle prediction model.

Tables 5.15, 5.16 and 5.17 show the predicted number of cycles and lifetime (in seconds) for the three conditional duty cycles - (d), (e) and (f).

Table 5.17: The predicted number of cycles and lifetime for conditional duty cycles based on duty cycle adjusted prediction, as well as the relative error comparing the prediction and the measurements.

Duty cycle	Number of Cycles	Lifetime (s)	Rel. Error
(d)	2059	20573	2.6%
(e)	2039	16978	6.6%

For all the three different approaches, the relative lifetime prediction error for the conditional duty cycles is at most 10%. In the conditional duty cycles, we incur the same overhead as in the deterministic ones; i.e., monitoring the discharge current accumulator and saving control information.

5.5 Applying Duty Cycle Prediction to the Meerkats Testbed

The Meerkats testbed is a wireless network of battery-operated camera nodes, that can be used for monitoring and surveillance of wide areas. The goal of the system is detect and track moving bodies within the covered area. Ideally, any time a body enters the field of view (FOV) of a node, the node's camera would take one or more snapshots.

In this section, we will look into different resource management policies the Meerkats node could use to extend its lifetime, while still attending the application requirements. The Meerkats node should take snapshots at a given frequency, which depends on the expected event rate and acceptable event miss rate (i.e., the application requirements).

The duty cycles we described in Section 4.4 have in common the fact that the idle/sleep period is the same, and this results in different duty cycle durations. The duty cycles we will use to implement the different resource management policies the Meerkats node could use must

have the same duration, and therefore we need to adjust the idle/sleep period such that the duty cycle duration is the same, and the interval between image acquisitions is constant. For example, if the interval between images is 5 s and the node will run duty cycle (a), then the sum of the duration of all elementary tasks plus the idle period should be 5 s. Therefore, the duration of the idle period T_{idle} will be given by $T_{idle} = 5 - T_{acq\&comp} - T_{send}$, where $T_{acq\&comp} = 1.2$ and $T_{send} = 0.35$ (from Table 4.1). Thus $T_{idle} = 3.45$.

Note that not all duty cycles described in Section 4.4 might be feasible for a 5 s duration. For instance, consider duty cycle (c), where webcam and wireless card are deactivated when not in use, and the node is put in sleep mode. From Table 4.1, the duration of the elementary tasks (in seconds) are as follows: $T_{act-wcam} = 1.3$, $T_{acq\&comp} = 1.2$, $T_{deact-wcam} = 0.35$, $T_{act-wifi} = 2.3$, $T_{send} = 0.35$, $T_{deact-wifi} = 0.37$, $T_{go_sleep} = 0.39$, $T_{wakeup} = 2.8$. If we add these durations, we obtain a total of 9.06 s, which does not include any actual sleep period. Therefore, duty cycle (c) is not feasible for an interval between image acquisitions smaller than 10 s.

Another aspect to be considered is the charge necessary to execute a given duty cycle. From Tables 4.3 and 4.5, we observe that duty cycle (b) uses about 100 mC more than duty cycle (c), but lasts about 2.4 s less. Therefore, determining the appropriate duty cycles for the Meerkats node is a trade-off between the time requirements and the energy consumption.

Next we will consider four different sampling periods (or interval between image acquisitions) for the *end node*: 5, 10, 15 and 20 s. For each of these sampling periods, we verify if the six duty cycles previously described are feasible, and how the idle/sleep periods need to be adjusted. Then we will apply the lifetime prediction methodology described in Sections 5.3

and 5.4 to compare which duty cycle would be more energy efficient given different image acquisition periods.

5.5.1 Duty Cycle Feasibility

First, we need to adjust the six duty cycles previously described considering the one second delay after sender program for duty cycles (b), (c), (d) and (e)), as well as adjustments related to webcam and wireless card activation (see Section 4.5.1). The duty cycles are adjusted as follows:

- **Duty cycle (a)**: the node simply takes an image, compresses it and transmits, and then stays in idle for a period T_i. Both sensor and communication cores are always activated.

- **Duty cycle (b)**: similar to the previous case, but the sensor and communication cores are activated only when needed, and deactivated as soon as possible. Note that a one second wait is necessary after issuing webcam and wireless card activation tasks, as well as after the sender program.

- **Duty cycle (c)**: different from the second case in that the system is put to sleep, rather than kept idle for the same period T_i of time. This duty cycle includes the additional task of wireless card deactivation after system wake-up. Note that a one second wait is necessary after issuing webcam and wireless card activation tasks, as well as after the sender program.

- **Duty cycle (d)**: the node starts in idle mode (only with the processor core active). Then it activates the webcam (one second delay added here), acquires and process an image and

Table 5.18: The sum of the duration of the elementary tasks involved in a given duty cycle, and the idle/sleep period allowed for a given duty cycle duration. Note that duty cycle (d) and (e) when an event is detected are the same, and thus only one is shown here.

		T_i (s) for different duty cycle durations			
Duty cycle	Duration of tasks (s)	5 s	10 s	15 s	20 s
(a)	2.58	2.42	7.42	12.42	17.42
(b)	7.62	-	2.38	7.38	12.38
(c)	11.18	-	-	3.82	8.82
(d) - no event	6.38	-	3.62	8.62	13.62
(d) - event	7.53	-	2.47	7.47	12.47
(e) - no event	2.82	2.18	7.18	12.18	17.18
(f) - no event	1.13	3.87	8.87	13.87	18.87
(f) - event	2.49	2.51	7.51	12.51	17.51

then deactivates the webcam. If no event is detected, the system is put into sleep mode for T_s. Otherwise, the system activates the wireless card (one second delay added here), transmits the image (one second delay added here), deactivates the wireless card and then stays in idle for a period T_i.

- **Duty cycle (e)**: this case is a variation of the previous one, where the node stays in idle mode instead of going to sleep in case no event is detected.

- **Duty cycle (f)**: this is the simplest case, where all cores (processor, sensor and communication) remain active all the time and the system stays in idle in between image acquisitions.

Given the four different sampling periods (or interval between image acquisitions): 5, 10, 15 and 20 s we want to evaluate, the duty cycles idle/sleep period would be as depicted in Table 5.18. Note that the sampling periods will in fact be the duty cycle duration.

Table 5.19: The charge associated with the duty cycles for given duration. Note that duty cycle (d) and (e) when an event is detected are the same, and thus only one is shown here.

	Charge (mC) for different duty cycle durations			
Duty cycle	5 s	10 s	15 s	20 s
(a)	2.57	5.01	7.44	9.88
(b)	-	2.33	3.03	3.72
(c)	-	-	2.71	2.76
(d) - no event	-	1.60	1.65	1.69
(d) - event	-	2.31	3.01	3.70
(e) - no event	1.20	1.89	2.59	3.28
(f) - no event	2.52	4.96	7.39	9.83
(f) - event	2.56	5.00	7.43	9.87

From Table 5.18, we observe that duty cycles (b), (c) and (d), as well as duty cycle (e) when an event is detected, are not feasible for a 5 s duration duty cycle. Also, duty cycle (c) is not feasible for a 10 s duration duty cycle.

Next step is to calculate the charge associated with each feasible duty cycle. Table 5.19 presents these results.

Duty cycles that make use of hardware activation/deactivation and low-power mode to optimize energy consumption (e.g., duty cycles ((b), (c), (d) and (e)) have a larger minimum duration because of the transition costs. On the other hand, they consume less charge when compared to duty cycles (a) and (f), which always have all power subsystems active.

5.5.2 Meerkats node lifetime prediction

In this section, we use the charge and duration obtained for the Meerkats node feasible duty cycles to predict the node's lifetime, and thus evaluate simple resource management policies. We use the prediction methodology described in Sections 5.3 and 5.4 to compare which

Table 5.20: The predicted lifetime for a Meerkats' node running deterministic duty cycles for a given duration.

	Lifetime (s) for different duty cycle durations			
Duty cycle	5 s	10 s	15 s	20 s
(a)	7001	7191	7257	7290
(b)	-	15424	17828	19334
(c)	-	-	19892	26090

duty cycle would be more energy efficient given a sampling period for image acquisition.

Table 5.20 shows the predicted lifetime for a Meerkats' node running the deterministic duty cycles (i.e., duty cycles (a), (b) and (c)) for the four different sampling periods (5, 10, 15 and 20 s).

The prediction for the conditional duty cycles is more complex, since it must consider the probability of an event being detected by the visual processing module. Therefore we calculate the predicted lifetime for different probabilities of event being detected (it ranges from zero until one, in increments of 0.1). Tables 5.21, 5.22 and 5.23 present the results for the feasible durations for duty cycles (d), (e) and (f), respectively.

Since duty cycle (d) makes use of the low-power mode while waiting to complete the expected duty cycle duration, we observe a significant increase in node lifetime as duty cycle duration increases. Also, as expected, the predicted lifetime decreases as the probability of event detection increases, since event detection incurs in image transmission followed by a wait period in idle mode. Duty cycle (e) follows the same trend exhibited by duty cycle (d), but with less significant differences between duty cycle durations. Duty cycle (f) shows almost no difference in the lifetime for different probabilities of event detection, because since the

Table 5.21: The predicted lifetime (in seconds) for a Meerkats' node running conditional duty cycle (d) for different durations and probabilities of detecting an event.

P(event)	(d) - 10s	(d) - 15s	(d) - 20s
0.0	22468	33702	44936
0.1	21515	32273	43031
0.2	20640	30960	41281
0.3	19833	29750	39667
0.4	19087	28631	38175
0.5	18395	27593	36791
0.6	17752	26628	35504
0.7	17152	25728	34304
0.8	16591	24887	33182
0.9	16066	24099	32131
1.0	15573	23359	31145

Table 5.22: The predicted lifetime (in seconds) for a Meerkats' node running conditional duty cycle (e) for different durations and probabilities of detecting an event.

P(event)	(e) - 10s	(e) - 15s	(e) - 20s
0.0	19040	28560	38080
0.1	18625	27938	37251
0.2	18228	27343	36457
0.3	17848	26772	35696
0.4	17483	26224	34966
0.5	17133	25699	34265
0.6	16796	25194	33592
0.7	16473	24709	32945
0.8	16161	24242	32323
0.9	15862	23792	31723
1.0	15573	23359	31145

Table 5.23: The predicted lifetime (in seconds) for a Meerkats' node running conditional duty cycle (f) for different durations and probabilities of detecting an event.

P(event)	(f) - 5s	(f) - 10s	(f) - 15s	(f) - 20s
0.0	3631	7262	10893	14523
0.1	3628	7256	10884	14511
0.2	3625	7250	10875	14500
0.3	3622	7244	10866	14488
0.4	3619	7238	10857	14476
0.5	3616	7232	10848	14464
0.6	3613	7226	10839	14452
0.7	3610	7220	10830	14440
0.8	3607	7214	10822	14429
0.9	3604	7208	10813	14417
1.0	3601	7203	10804	14405

duration of the idle period if no event is detected is larger than the duration of the idle period when an event is detected, the cost of transmitting the image is evens out the overall charge drained by the system. Increasing the duty cycle duration still increases the node lifetime, since less activities are carried out.

Figure 5.2 depicts the predicted lifetime for all conditional duty cycles under different event detection probabilities, summarizing the results presented in Tables 5.21, 5.22 and 5.23. It is clear that duty cycle (d) is the one that allows better lifetime overall. But the choice of the duty cycle also depends on the application requirements in terms of image acquisition period, and if the requirements are for a sampling period of 5 s, then the best choice is duty cycle (a).

Another observation we can draw from Figure 5.2 is the fact that the gap between the predicted lifetime for duty cycles (d) and (e) is smaller for shorter duty cycle durations. Given this, we should look into a possible break-even point for duty cycles (d) and (e). If an event is detected, duty cycles (d) and (e) follow the same sequence of tasks, and therefore the predicted

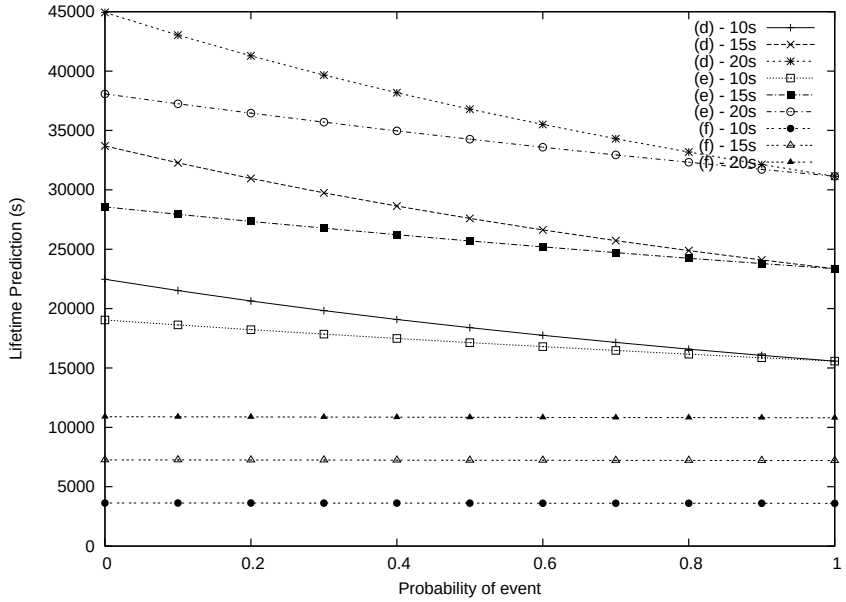


Figure 5.2: The predicted lifetime (in seconds) for a Meerkats' node running conditional duty cycles.

lifetime for (d) and (e) is the same when the probability of detecting an event equals one. When no event is detected, duty cycle (d) will make use of the low-power state, while duty cycle (e) will remain in idle mode. So a possible break-even point for these duty cycles will happen if there is a duty cycle duration where the charge incurred in transitioning to and remaining in low-power mode is equivalent to remaining in idle mode.

Figure 5.3 shows the break-even point for conditional duty cycles (d) and (e). The time cost associated with transitioning to low-power mode is 3.5 seconds. If the node is supposed to sleep for 1 s, the overall duration is 4.5 s, and the charge associated is 634 mC. If the node remains in idle mode (with only the Processing Core active) for 4.5 s, it draws 634 mC too. Therefore, if the waiting period is to be smaller than 4.5 s, the node should remain in idle mode instead of going to sleep.

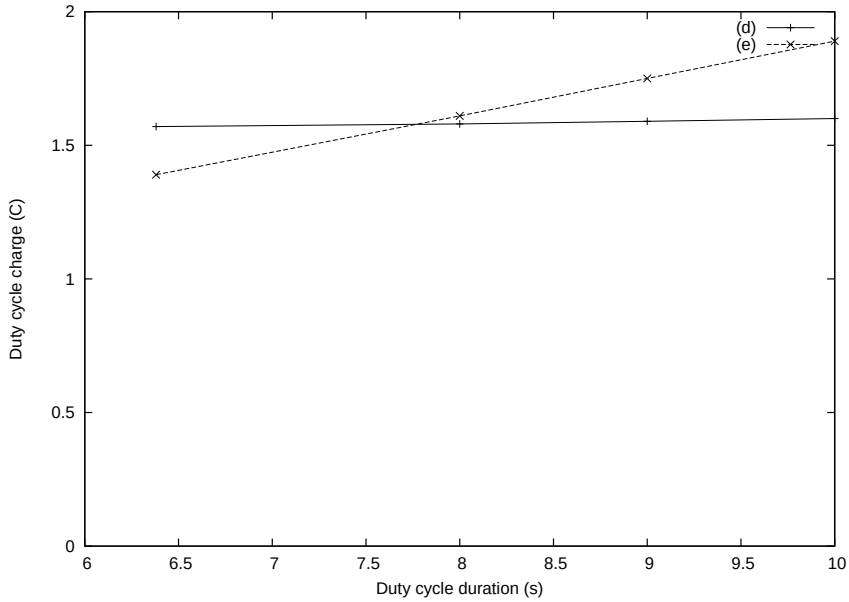


Figure 5.3: The break-even point for conditional duty cycles (d) and (e).

5.6 Conclusions

In this Chapter, we completed the third step to achieve our goal of conducting a comprehensive study of visual sensor network's lifetime: develop and validate an energy consumption model based on task composition to determine a visual sensing node's lifetime.

We first executed a series of experiments running duty cycles repetitive to understand the lifetime we could achieve with the visual sensor node being used. Then, considering the lifetime experiments and the duty cycle prediction model presented in Chapter 4, we extended the model to predict the lifetime of visual sensor node under a given duty cycle. Finally, we applied all we learned and developed to the Meerkats' testbed, being able to define a set of feasible duty cycles under different requirements, and to compare how the choice of different

resource management policies affect the Meerkats' node lifetime. We also showed that there is a break-even point for putting the system into low-power state.

Chapter 6

Conclusions

The main goal of this work is to understand the energy consumption trade-offs between computation and communication in power-constrained networks in general, and, in particular, in visual sensor networks.

In Chapter 2, we presented our work on instrumenting network simulators to enable them to adequately and accurately account for the energy consumed by ad hoc network protocols' communication-related tasks. This is accomplished by explicitly accounting for all possible radio states, i.e, transmitting, receiving, overhearing, idle, sensing, and sleeping, and considering the different energy costs associated with each of them. Another contribution of our energy consumption instrumentation is to allow the energy accounting to be done automatically by the simulator irrespective of what layer of the stack the protocol designer is working. The instrumentation energy model is validated analytically and through simulations using two MAC protocols, i.e., IEEE 802.11 DCF and S-MAC. We also showcase our instrumentation's ability to evaluate energy consumption of protocols by comparing S-MAC against 802.11, and AODV

against DSR. Then work simulation results obtained using instrumented QualNet with our energy consumption accounting are used to validate the analytical model proposed to evaluate energy consumption in IEEE 802.11 single-hop wireless networks.

In Chapter 3, we presented a task-level energy consumption characterization benchmark that accounts for basic tasks being executed by a laptop (a typical example of a mobile computing platform and node in a MANET), and the Meerkats node (example of a visual sensor network node). We also presented the extensions we implemented to obtain current and discharge current accumulator 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.

In Chapter 4, we have presented an analysis of the power requirement and execution time of the elementary tasks that compose a typical duty cycle of a visual sensing node based on the Crossbow Stargate board. This analysis is necessary to predict the node's lifetime when it is battery-operated, and to choose the most appropriate design parameters. Our study has highlighted the fact that activation and deactivation of hardware components, as well as the transition between idle and sleep mode of the node's processor core, may require considerable overhead energy and may take substantial time. We also showed how elementary tasks can be combined together to form different duty cycles performing the same function but with rather different energy requirements. In addition, we validated our hypothesis that the energy con-

sumed in a duty cycle, as well as its execution time requirements, can be estimated by simply adding up the relevant quantities for each elementary task involved. Modeling duty cycles by task composition achieves a relative error in the range of -10% to 10% for both the average duration and charge. We also discussed the issues associated with the task concatenation approach to determine a duty cycle, and how its accuracy is affected.

In Chapter 5, we completed the third step to achieve our goal of conducting a comprehensive study of visual sensor network's lifetime: develop and validate an energy consumption model based on task composition to determine a visual sensing node's lifetime. We first executed a series of experiments running duty cycles repetitive to understand the lifetime we could achieve with the visual sensor node being used. Then, considering the lifetime experiments and the duty cycle prediction model presented in Chapter 4, we extended the model to predict the lifetime of visual sensor node under a given duty cycle. Finally, we applied all we learned and developed to the Meerkats' testbed, being able to define a set of feasible duty cycles under different requirements, and to compare how the choice of different resource management policies affect the Meerkats' node lifetime. We also showed that there is a break-even point for putting the system into low-power state.

In summary, the contributions from this work are as follows:

- An energy model for communications, which was implemented under GloMoSim and QualNet network simulators (code already contributed to the maintainers, and also the GloMoSim code was made available on the webpage `http://www.soe.ucsc.edu/~cintia/energy-glomo.html`).

- Characterization of the energy consumption of two different wireless platforms, namely a laptop (typical example of a MANET node) and the Stargate [17] (used as visual sensor node).

- Extension and validation of the on-board battery monitoring capability on the Stargate, which was contributed to the Stargate community and is available at `http://www.soe.ucsc.edu/~cintia/batmon.html`.

- Duty cycle prediction based on elementary tasks composition.

- A simple lifetime prediction model based on elementary task composition and experiments on lifetime of a visual sensor network for different duty cycles.

- Analysis of the Meerkats node lifetime under different duty cycles and trade-offs.

6.1 Future Directions

There are two main paths to continue the work presented here.

The first path is to formalize the lifetime prediction model presented in Chapter 5 to include non-deterministic sequence of tasks. Similarly to what we have done, the set of elementary tasks would be know. But the order this tasks would be executed would depend on different parameters, such as event detection rate, amount of charge available on the battery, information received from neighboring nodes, etc.

The second path is to apply the *framework* we developed in this thesis to a different sensor network platform and application. As we already point out, although our work was done

considering a specific visual sensing platform, i.e., the Meerkats node, the methodology we used to analyze the energy consumption trade-offs of the different tasks and duty cycles could be applied to other platforms and applications. Essentially, the methodology we proposed includes: (1) characterizing energy consumption of the platform; (2) determining the cost (in terms of charge and duration) of the tasks representative of the target application; (3) extend/adapt the energy model based on task composition to determine the node's lifetime. Given the discussion presented in Section 4.5, applying the methodology a platform with simpler hardware and operating system should allow better results. An example of such a platform would be the mote [82] as part of a environmental sensor network.

Bibliography

[1] A. Aburto. FFT double precision benchmarks. ftp://ftp.nosc.mil/pub/aburto/, 2001.

[2] A. Acquaviva, T. Simunic, V. Deolalikar, and S. Roy. Remote power control of wireless network interfaces. In *Proc. of PATMOS in Lecture Notes in Computer Science*, Turin, September 2003. Springer-Verlag.

[3] M. Anand, E. B. Nightingale, and J. Flinn. Ghosts in the machine: Interfaces for better power management. In *The Second International Conference on Mobile Systems, Applications, and Services (MobiSys 2004)*, Boston, USA, June 2004.

[4] S. Balakrishnan and J. Ramanan. Power-aware operating system using acpi. CS736 Project - Fall 2001, 2001.

[5] K. Barr and K. Asanovic. Energy aware lossless data compression. In *The First International Conference on Mobile Systems, Applications, and Services*, San Francisco, CA, May 2003.

[6] L. Benini, A. Bogliolo, and G. D. Micheli. A survey of design techniques for system-level

dynamic power management. *IEEE Transactions on VLSI Systems*, 8(3):299–316, June 2000.

[7] M. Bhardwaj and A. P. Chandrakasan. Bounding the lifetime of sensor networks via optimal role assignments. In *Proceedings of the Twenty First International Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002)*, New York, NY, USA, June 2002.

[8] BIOS central. `http://www.bioscentral.com`, 2004.

[9] 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.

[10] L. S. Brakmo, D. A. Wallach, and M. A. Viredaz. usleep: A technique for reducing energy consumption in handheld devices. In *The Second International Conference on Mobile Systems, Applications, and Services (MobiSys 2004)*, Boston, USA, June 2004.

[11] J.-C. Cano and P. Manzoni. A performance comparison of energy consumption for mobile ad hoc network routing protocols. In *8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, San Francisco, CA, September 2000. IEEE Computer Society.

[12] M. M. Carvalho and J. J. Garcia-Luna-Aceves. Delay analysis of IEEE 802.11 in single-hop networks. In *Proc. of 11th IEEE International Conference on Network Protocols (ICNP)*, Atlanta, USA, November 2003.

[13] M. M. Carvalho, C. B. Margi, K. Obraczka, and J. J. Garcia-Luna-Aceves. Modeling energy consumption in single-hop IEEE 802.11 ad hoc networks. In *Thirteenth International Conference on Computer Communications and Networks (ICCCN'04)*, Chicago, USA, October 2004.

[14] A. Cerpa and D. Estrin. Ascent: Adaptive self-configuring sensor networks topologies. In *Proceedings of the Twenty First International Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002)*, New York, NY, USA, June 2002.

[15] 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.

[16] The CMUcam Vision Sensors. `http://www.cs.cmu.edu/~cmucam/`, 2005.

[17] Crossbow. Stargate. `http://www.xbow.com/`, 2004.

[18] B. Crow, I. Widjaja, J. Kim, and P. Sakai. IEEE 802.11 wireless local area networks. *IEEE Communications Magazine*, 35(9):116–26, September 1997.

[19] D. Culler, D. Estrin, and M. Srivastava. Overview of sensor networks. *Computer Magazine*, 37(8):41–49, 2004.

[20] L. Doherty, B. A. Warneke, B. Boser, and K. S. J. Pister. Energy and performance con-

siderations for smart dust. *International Journal of Parallel and Distributed Systems and Networks*, 4(3):121–133, 2001.

[21] E. J. Duarte-Melo and M. Liu. Analysis of energy consumption and lifetime of heterogeneous wireless sensor networks. In *IEEE Globecom*, Taipei, Taiwan, November 2002.

[22] 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.

[23] 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.

[24] J. Flinn and M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *2nd IEEE Workshop on Mobile Computing Systems and Applications*, New Orleans, Louisiana, February 1999.

[25] S. Fowler. wmacpi: A battery monitor dockapp for ACPI based systems. `http://himi.org/wmacpi-ng/`, 2004.

[26] Q. Han, S. Mehrotra, and N. Venkatasubramanian. Energy efficient data collection in distributed sensor environments. In *ICDCS*, pages 590–597, 2004.

[27] R. Haralick and L. Shapiro, editors. *Computers and Robot Vision*. Addison-Wesley, 1992.

[28] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the Hawaii International Conference on System Sciences*, pages 3005–14. IEEE, January 2000.

[29] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *33rd Hawaii International Conference on System Sciences (HICSS '00)*, Hawaii, January 2000.

[30] W. B. Heinzelman. *Application-Specific Protocol Architectures for Wireless Networks*. PhD thesis, Massachusetts Institute of Technology, 2000.

[31] Hewlett-Packard, Intel, Microsoft, Phoenix, and Toshiba. ACPI: Advanced configuration and power interface. `http://www.acpi.info/`, 2004.

[32] *IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, Nov 1997. P802.11.

[33] N. Instruments. LabVIEW 7.1. `http://www.rt.com/man/hdparm.8.html`, 2004.

[34] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Sixth Annual International Conference on Mobile Computing and Networking, (MobiCom 2000)*, pages 56–67. ACM, August 2000.

[35] Intel. Intel PXA255 applications processors developer's manual. `http:`

//www.intel.com/design/pca/applicationsprocessors/manuals/278693.htm, Janeiro 2004.

[36] ISI/USC. ns-2. http://www.isi.edu/nsnam/ns/, 2003.

[37] S. Iyer, L. Luo, R. Mayo, and P. Ranganathan. Energy-adaptive display system designs for future mobile environments. In *The First International Conference on Mobile Systems, Applications, and Services (MobiSys 2003)*, San Francisco, USA, June 2003.

[38] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[39] D. Kim, J. Garcia-Luna-Aceves, K. Obraczka, J.-C. Cano, and P. Manzoni. Performance analysis of power-aware route selection protocols in mobile ad hoc networks. In *IEEE Networks 2002*, Atlanta, GA, August 2002.

[40] J. Kulik, W. Heinzelman, and H. Balakrishnan. Negotiation-based protocols for dissemination information in wireless sensor networks. In *Submited to ACM Wireless Networks*, 2001.

[41] 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.

[42] P. Levis, D. Gay, and D. Culler. Active sensor networks. In *2nd USENIX/ACM Symposium on Network Systems Design and Implementation (NSDI)*, 2005.

[43] E. Lochin, A. Fladenmuller, J.-Y. Moulin, and S. Fdida. Energy consumption models for ad-hoc mobile terminals. In *Med-Hoc Net*, 2003.

[44] X. Lu and R. Manduchi. Fast image motion computation on an embedded computer. In *2nd IEEE Workshop on Embedded Computer Vision*, New York, June 2006.

[45] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TinyDB: An acqusitional query processing system for sensor networks. *ACM Transactions on Database Systems*, 2005.

[46] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. In *First ACM Workshop on Wireless Sensor Networks and Applications*, Atlanta, GA, September 2002.

[47] MandrakeSoft. Mandrakelinux. `http://www.mandrakelinux.com`, 2004.

[48] C. B. Margi, R. Manduchi, and K. Obraczka. Energy consumption tradeoffs in visual sensor networks. In *24th Brazilian Symposium on Computer Networks (SBRC 2006)*, Curitiba, Brazil, June 2006.

[49] C. B. Margi and K. Obraczka. Instrumenting network simulators for evaluating energy consumption in power-aware ad-hoc network protocols. In *12th Annual Meeting of the IEEE / ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS2004)*, Volendam, The Netherlands, October 2004.

[50] C. B. Margi, K. Obraczka, and R. Manduchi. Characterizing system level energy consumption in mobile computing platforms. In *IEEE WirelessCom 2005 - Symposium on Mobile Computing*, Maui, HI, USA, June 2005.

[51] C. B. Margi, V. Petkov, K. Obraczka, and R. Manduchi. Characterizing energy consumption in a visual sensor network testbed. In *2nd International IEEE/Create-Net Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom 2006)*, Barcelona, Spain, March 2006.

[52] R. A. Mini, M. do Val Machado, A. A. Loureiro, and B. Nath. Prediction-based energy map for wireless sensor networks. *Ad Hoc Networks Journal (Special Issue on Ad Hoc Networking for Pervasive Systems)*, 3(2):235–253, 2005.

[53] R. A. Mini, A. A. Loureiro, and B. Nath. Prediction-based energy map for wireless sensor networks. In *Proceedings of IFIP-TC6 8th International on Conference Personal Wireless Communications (PWC 2003)*, pages 12–26, 2003.

[54] D. Monticelli. System approaches to power management. In *Applied Power Electronics Conference and Exposition, 2002. APEC 2002. Seventeenth Annual IEEE*, Dallas, TX, USA, March 2002.

[55] W. D. Norcott and D. Capps. IOzone filesystem benchmar. `http://www.iozone.org/`, 2004.

[56] C. Perkins, E. Belding-Royer, and S. Das. RFC 3561 - Ad hoc On-Demand Distance Vector (AODV) Routing. `http://www.faqs.org/rfcs/rfc3561.html`, July 2003.

[57] V. Petkov. Using the DS2438 Battery Monitor on Crossbow's Stargate. Technical report, University of California Santa Cruz, 2006.

[58] D. Qiao, S. Choi, A. Jain, and K. G. Shin. Miser: An optimal low-energy transmission strategy for IEEE 802.11a/h. In *Proc. of the 9th Annual International Conference on Mobile Computing and Networking (MOBICOM)*. ACM Press, September 2003.

[59] V. Raghunathan, T. Pering, R. Want, A. Nguyen, and P. Jensen. Experience with a low power wireless mobile computing platform. In *ISLPED 2004*, 2004.

[60] 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.

[61] V. Rajendran, K. Obraczka, and J. Garcia-Luna-Aceves. Energy-efficient, collision-free medium access control for wireless sensor networks. In *ACM SenSys 03*, Los Angeles, CA, November 2003.

[62] J. M. Reason and J. M. Rabaey. A study of energy consumption and reliability in a multi-hop sensor network. *Mobile Computing and Communications Review*, 8(1):84–97, 2004.

[63] RFM. TR1000 datasheet. `http://www.rfm.com/products/data/tr1000.pdf`, 2003.

[64] H. O. Sanli, R. Poornachandran, and H. Çam. Collaborative two-level task scheduling for wireless sensor nodes with multiple sensing units. In *SECON*, 2005.

[65] B. Scheifler and D. Krikorian. Unix man pages: xset (1). http://www.mcsr.olemiss.edu/cgi-bin/man-cgi?xset+1, 2004.

[66] SCT. QualNet user manual. http://www.scalable-networks.com/, 2003.

[67] D. Semiconductor. DS2438: Smart battery monitor datasheet. http://www.maxim-ic.com, 2004.

[68] 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.

[69] H. Singh and S. Singh. Energy consumption of tcp reno, newreno, and sack in multi-hop wireless networks. In *SIGMETRICS02*, 2002.

[70] A. Sinha and A. Chandrakasan. Dynamic power management in wireless sensor networks. *IEEE Design & Test of Computers*, 18(2):62–74, 2001.

[71] I. Solis and K. Obraczka. The impact of timing in data aggregation for sensor networks. In *The 2004 International Conference on Communications (ICC 2004)*, June 2004.

[72] I. Solis and K. Obraczka. Efficient continuous mapping in sensor networks using isolines. In *Mobiquitous 2005*, July 2005.

[73] A. Song. Piconet ii - a wireless ad hoc network for mobile handheld devices. http://piconet.sourceforge.net/, 2001.

[74] Standard performance evaluation corporation. http://www.spec.org, 2004.

[75] SPI. Debian – the universal operating system. http://www.debian.org/, 2004.

[76] C. Stauffer and W. Grimson. Adaptive background mixture models for real-time tracking". In *IEEE Int'l Conf. on Computer Vision and Pattern Recognition*, 1999.

[77] 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.

[78] F. Tari, P. Rong, and M. Pedram. An energy-aware simulation model and a transaction protocol for dynamic workload distribution in mobile ad hoc networks. In *Proceedings of the 21st International Conference on Computer Design: VLSI in Computers and Processors*, San Jose, CA, October 2003.

[79] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs. Iperf. http://dast.nlanr.net/Projects/Iperf/, 2003.

[80] G. Tolle, J. Polastre, R. Szewczyk, N. Turner, K. Tu, P. Buonadonna, S. Burgess, D. Gay, W. Hong, T. Dawson, and D. Culler. A macroscope in the redwoods. In *Third ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2005.

[81] T. van Dam and K. Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. In *ACM SenSys 03*, Los Angeles, CA, November 2003.

[82] A. Woo. Mote documentation and development information. http://www.eecs.berkeley.edu/~awoo/smartdust/, 2000.

[83] Y. Xu, S. Bien, Y. Mori, J. Heidemann, and D. Estrin. Topology control protocols to conserve energy in wireless ad hoc networks. Technical Report CENS Technical Report 6, Center for Embedded Networked Sensing, USA, January 2003.

[84] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In *Proceedings of the 21st International Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002)*, New York, NY, USA, June 2002.

[85] L. Yin, G. Cao, C. Das, and A. Ashraf. Power-aware prefetch in mobile environments. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 571–578, 2002.

[86] J. Zedlewski, S. Sobti, N. Garg, F. Zheng, A. Krishnamurthy, and R. Wang. Modeling hard-disk power consumption. In *Second Conference on File and Storage Technologies*, San Francisco, USA, March 2003.

[87] X. Zeng, R. Bagrodia, and M. Gerla. GloMoSim: a library for parallel simulation of large-scale wireless networks. In *12th Workshop on Parallel and Distributed Simulations – PADS '98*, Banff, Alberta, Canada, May 1998.

[88] F. Zheng, N. Garg, S. Sobti, C. Zhang, R. Joseph, A. Krishnamurthy, and R. Wang. Considering the energy consumption of mobile storage alternatives. In *MASCOTS'2003*, Orlando, USA, October 2003.