Developing Disposable Wireless Sensor Networks

Zak Rubin
Computer Engineering
UC Santa Cruz
zrubin@soe.ucsc.edu

H. Blake Skinner
Computer Engineering
UC Santa Cruz
hskinner@soe.ucsc.edu

December 8, 2013

Abstract

Ad-hoc wireless sensor networks are predicted to increase in use significantly as advances in their core hardware components continue. They have already attracted considerable interest in academia and have been successfully deployed for a variety of applications. Despite advances in this area, cost and power limitations remain the major obstacles to their use.

In this paper we explore the general problem of transferring data around a network at minimal cost. We present several protocols to transfer data across an ad-hoc network while preserving the data's locality. We will explore the problem of exchanging throughput for power savings and will demonstrate what is possible in terms of through-put and power use with hardware currently available on the market. With this we hope to demonstrate the cost trade-offs associated with wireless ad-hoc networks today how throughput can be sacrificed for power savings.

Keywords: sensor networks, low power

1 Introduction

Wireless sensor networks generally consist of a large number of small computers, or nodes, often built using micro-controllers. There has been increasing interest in being able to construct such a network with and manage it with a fully distributed, ad-hock protocol. This has been enabled by advances in the size, power consumption, and overall efficiency of the core hardware components of the network, along with advances in development and simulation tools. Sensor networks have already generated a considerable amount of interest among academic researchers, numerous private companies including start-ups, and with various government, especially military, organizations.

Monetary cost and power consumption continue to be limiting factors, however, and although the quality of hardware expected to

continue to improve, applications involving wireless sensor networks often require, hundreds, or even thousands of nodes, even small differences in cost are multiplied.

Furthermore, the cookie-cutter manufacturing process of modern digital hardware, which has allowed the performance-to-price ratio to improve dramatically over the last few decades, also requires that engineers designing sensor networks pick from the chips currently available on the market. This can introduce considerable price-thresholds into the design process. For example, the difference in cost between a micro-controller with 8kB and 32kB of flash memory is often only a few dollars, but this could easily represent a thousands of dollars of cost difference between a network requiring less than 8kB of memory on each node and a network requiring more than 8kB. Furthermore, the power consumption of larger chips is greater as well, meaning a few extra kilo-bytes of memory could also add battery and logistical costs. Smaller micro-controllers also have less IO pins and a generally have a slower clock-speed so thresholds exist in around those parameters as well. Although the overall efficiency of hardware is expected to improve while cost continues to decrease, wireless sensor engineers will have to contend with price/feature thresholds for the foreseeable future.

In this paper, we would like to explore the cost of building a low power wireless network. We particularly wish to propose a distributed network built using the very lowest-end micro-controllers and wireless chips currently available on the market. We propose several protocols designed to implement a peer-to-peer, distributed system while minimizing the total cost of a network, in terms of hardware and power usage. We then demonstrate the throughput and other network characteristics possible with these schemes, which are not ideal, but are workable for many applications, and represents the minimum cost/functionality available with current technology.

We do this with platforms like Arduino [**?**] in mind. Arduino is a micro-controller development board built around open source hardware, designed to be easy to develop on and network together, with one of the stated goals of the project being to increase the accessibility of multi-disciplinary projects. Combined with the continued improvement of micro-kernels, simulators, and other platforms to facilitate the design of sensor networks, we believe the technology is only becoming more accessible.

The protocols we discuss could be implemented with minimal difficulty, without prior infrastructure, on even the lowest-end micro-controllers available. We do this because low-end chips such as the ATtiny [5] can come with as little as 2kB of memory, which is too small for most micro-kernels, and such chips are rarely given technical support. We also discuss implementing them on Contiki [1], an open-source operating system for small-scale networked devices which is accompanied by an open-source simulator, Cooja [**?**]. With this we hope to demonstrate the capabilities and cost of a minimally-effective distributed wireless sensor network and how it can be applied to a variety of applications. We believe this serves as a useful demonstration of

what can be expected from a functional, barebones, distributed, wireless sensor network and the cost and difficulty of implementing it.

Section 3 further discusses the motivation and goals of the project. Section 4 discusses design of the hardware designed to evaluate the protocols and profile the networks' power usage. Section **??** discusses the minimal-effort distributed sensor network protocols we are proposing, and Section 6 discusses our evaluation of them. Finally, we discuss some general challenges in Section 7 and conclude in Section **??**.

2 Related Work

Wireless sensor networks have attracted considerable research interest, but much of the early groundwork in the field is still being laid. Min et. al. [7] discusses general technologies useful for designing low-power networks. They also discuss the general clustering strategies to reduce overall power dissipation, such as the clustering strategy of the Low Energy Adaptive Clustering Hierarchy (LEACH) protocol [4]. This dynamically partitions nodes into clusters to reduce the overall energy dissipation through network packets. LEACH also provides a mechanism to designate one node a "cluster-head", to handle the majority of the input and output for the cluster. The cluster-head designation is rotated all the nodes in the system to balance power usage and general wear-and-tear. Various improvements on LEACH have been proposed, many of which are covered by [9]. Schurgers et. al. [10] recognizes that radios mainly need to be on to forward data and manages their on/off cycles based on the network density and its forwarding needs. Finally, Mainland et. al. [6] proposes determining node behavior based on assigning value to certain data and actions and assign costs to power usage. All of these proposals are based around creating an ad-hoc topology for the wireless nodes. Though this could be useful for many applications, we wish to explore a more undirected solution which can be implemented on more modest hardware.

Other research has focused on improving the tools and paradigms for designing sensor networks. Telos [8] is a low-power wireless sensor module ("mote") developed to aid in wireless sensor research. TinyOS [2] is an open-source operating system for wireless sensor networks. It provides a custom C and Java programming environment and supports many different hardware platforms. Nano-RK [3] is an operating system designed to meet deadline criteria while scheduling multi-tasked applications on wireless sensor networks. EMERALDS [12] is a micro-kernel specifically designed for small systems. Contiki [1] is an open source operating system designed to provide modern protocol support to low-power micro-controllers, it is also the OS we use to evaluate our protocols in this paper. While there are many applications which could benefit from these systems, they cannot be implemented in the lowest-end hardware available and represent greater complexity than may be necessary given the relatively small datasets and throughput requirements of some sensor network applications.

3 Design Considerations

Cost remains a major limiting factor in the adoption of sensor networks. Though the performance-to-price ratio of hardware is expected to continue to improve, even small differences in price are multiplied across numerous nodes, and the cookie-cutter manufacturing process of modern digital hardware, which has delivered increasingly powerful chips for increasingly low costs, also presents cost thresholds to the design process. This is not likely to change anytime soon.

We present several protocols designed to create a minimal-effort, minimal-cost, low-power distributed sensor network which can be implemented with relatively little effort without using external infrastructure, such as micro-kernel or embedded OSs like EMERALD [12] or TinyOS [2]. We do this with low-end micro-controllers, such as the ATtiny [5], in mind.

The primary mechanism to save power on a sensor network is to turn the node's radio off. This presents an inherent challenge, especially for peer-to-peer systems because when the radio is off, the node is effectively removed from the network. We explore two different strategies for tackling this problem, the first, discussed in Section 5.1 is to dynamically put the nodes on a synchronized schedule, the other, discussed in Section 5.2 is to turn the radio on and off based on random duty cycles.

4 GumboNode

In order to accurately profile the network, we have created minimum-cost hardware nodes to test the protocols on, which we have named **GumboNodes**. GumboNodes are barebones sensor nodes consisting of a microcontroller, a wireless transceiver, a sensor, and a battery. In our specific hardware, the system already comes on-board with a temperature sensor. The goal of the network is to distribute updated sensor data to all the nodes in the network.

We selected the ATTiny85 as the microcontroller for the chip. The ATTiny85 is a low-cost, low-power 8-bit processor. It is Arduino-compatible and has a wide range of options to optimize the processor for power consumption. Table 1 provides power information from the datasheet [6]. In the lowest power mode, the processor uses less than 2 μA. There are only two ways to wake the chip from this deep sleep: an external interrupt caused by a logic level change on an input pin, or periodically by an internal oscillator known as the watchdog timer. The watchdog timer has maximum time limit of 8 seconds after which the microcontroller automatically wakes up and enters idle state. The system may immediately be put back to sleep, or act after it wakes up a certain number of times.

The Texas Instruments CC2500 was selected as the wireless chip. The CC2500 is a low-level 2.4GHz wireless transceiver. Communication occurs over a 4-wire serial perhiperal interface (SPI) [7]. The ATTiny85 we selected lacks an onboard SPI interface, but has a universal serial interface (USI) that

is capable of interfacing with the transceiver. The CC2500 also has a temperature sensor on-board, which served as our sensor.

5 Proposed Protocols

5.1 Heart Beat Protocol

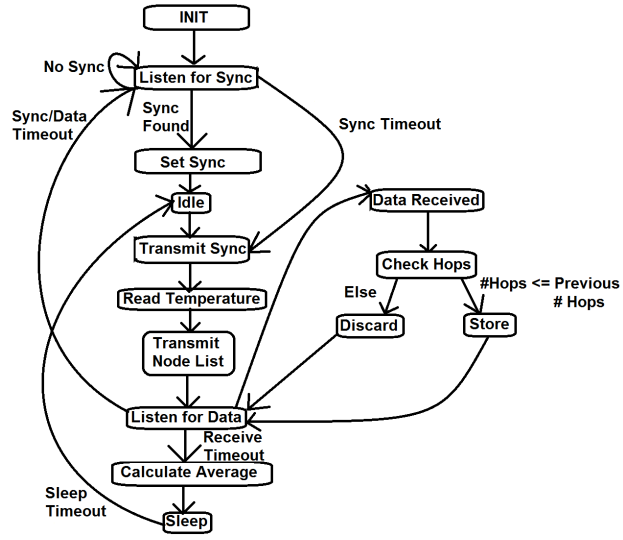

Figure 1: A flowchart of the communication protocol

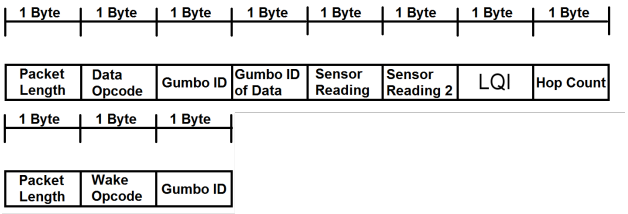

Figure 2: Data packet structure.

Our first proposed solution was to dynamically sync nodes to an on-off pattern, a flowchart of the protocol is shown in Figure 5. The structure of a packet is shown in Figure 2. After power-on, the system immediately goes into receive mode. If it does not hear a sync packet after a long enough period, the mote sends its own sync and .....

5.2 Random On/Off

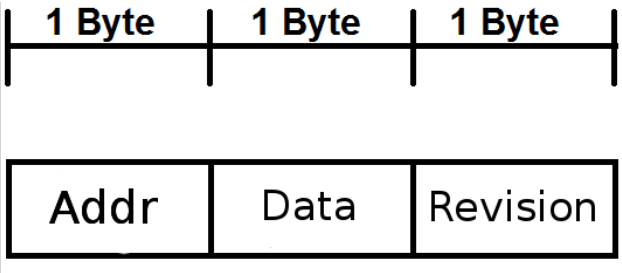

Figure 3: Data packet structure used for the "random" communication protocol.

We also wanted to explore the possibility of simply turning the radio on and off at random intervals, and syncing the data when possible. Though this is far less than ideal, it is significantly easier to implement, debug, and manage, and a relatively low throughput rate may be acceptable for some applications.

In this protocol, the node wakes up and sends a query packet asking for new data. If another node responds to the request, it will send the data it stores, starting with the most recently received. If the data is new, it replaces any old reading and the receiver of the data sends a packet asking for the next entry. If the data is old, the receiver asks the sender to stop. In order to determine the "newer" of two sets of data a simple counter

is used, the node can account for the counter looping around by considering a high number lower than a low number if the high number is close to the maximum possible value of the counter and the low number is close to zero. Although errors are possible with this system, its simplicity and redundancy may make it favorable for some applications. The results are shown in Figure **??**.

6 Evaluation

In this section we describe our investigation into sensor node communication. We simulated the networking behavior in software using Contiki OS's Cooja simulator while hardware shipped. As everything but the MCU came from China, shipping times varied considerably. Finally, we attempt to build a gateway node that would interface between Cooja nodes and the real nodes.

6.1 Hardware

As the primary purpose of this project is to produce inexpensive sensor networks, we built 20 nodes over a 2-week period. The total cost was $158.60. After our hardware arrived we proceeded to run two experiments: one investigating power consumption, and one investigating large network tests.

6.1.1 Hardware Costs

Table 3 shows the overall parts cost. CC2500 and ATTiny costs were taken from Mouser.com. CR2450 cost information came from Amazon for cost at 1, and AliExpress for cost at 10 and 1000. Costs not included are PCB fabrication, as the system has not left the breadboard phase at the time of this writing. Also not included are shipping costs as the lightweight nature of the components (The heaviest component being the battery at 6.8 grams) meant that shipping was not a significant cost at any magnitude. The breadboard prototypes cost a total of $7.59 per unit.

6.1.2 Network Performance

Three network tests were run: a small network test, a dense network test, and a sparse network test. The small network test had all nodes within 5 feet of each other, and all nodes could communicate with each other directly (no-hop). The dense network test had 17 nodes in the same configuration. The sparse network test had 17 nodes spread out in a 40'x50' area.

We elected to have each node calculate an average temperature using the mean of the data from received nodes. Nodes send averaged temperature out as the second sensor reading in network packets. However this scheme only works well if all nodes see very similar temperatures. Nodes need a method to determine how significant to treat data during averaging.

We used the Link Quality Indicator readout provided by the CC2500 combined with the hop count to develop a quality factor. During averaging, the system calculates the quaility factor based on the LQI and the number of hops taken, then the node's sensor

Mode	Minimum Consumption	Max Consumption
Active	0.8 mA	1.0 mA
Idle	0.5 mA	0.8 mA
Power-Down	2 μA	2 μA

Table 1: Power Consumption of ATTiny85 Modes at 1 MHz at 3V.

Mode	Minimum Consumption	Max Consumption
Power-Down	400 nA	160 μA
Idle	1.2 mA	2 mA
Temperature-Read	1.5 mA	1.5 mA
TX Mode (2.4 Kbaud)	11.1 mA	21.5 mA
RX Mode (2.4 Kbaud)	14.5 mA	17.0 mA

Table 2: Power Consumption of the CC2500 modes at 3V.

Equipment	Cost at 1	Cost at 10	Cost at 100
ATTiny85-20PU	\$1.29	\$1.08	\$0.74
CC2500	\$3.36	\$2.70	\$2.22
Battery (CR2450)	\$1.00	\$0.71	\$0.27
Total	\$5.65	\$4.49	\$3.23

Table 3: Bill of Materials and Per-Part Cost on Different Scales of Production (as of November 2013).

Mode	Calculated	Measured
Power-Down	2.4 μA	2.5 mA
Idle	2.5 mA	2.5 mA
Temp Read	2.5 mA	2.5 mA
TX Read	22.5 mA	16.0 mA
RX Read	18.0 mA	16.0 mA

Table 4: Power consumption.

Network Test	Syncs/Total	Data Used/Total	Nodes Seen/Total	Nodes in List
Small	950/7200 (13.2%)	337/1073 (31.4%)	4/4 (100%)	21
Dense	509/30600 (1.6%)	671/1848 (36.3%)	17/17 (100%)	85
Sparse	365/?	526/745 (70.6%)	14/17 (82.4%)	64

Table 5: Network Test Results.

reading is divided by that factor before being summed into the average. An LQI of 0 was found to be 7-12 feet away and was given a factor of 3. A node within 1 foot of the sensor had a minimum LQI of 30 and was assigned a factor of 1. The factor decreases by 1 for every 15 point increase in LQI, and increases in the same manner. In the given testing environment (a residential house), nodes had difficulty communicating after 15 feet.

We set the out of sync time at 5 wake cycles, or 5000ms. Nodes occasionally went out of sync in spread out networks where only nodes could only see 1 or 2 other nodes, and rarely went out of sync in denser networks. The system received data much less frequently - data was actually received once every 2 or 3 cycles. It is surprisingly difficult to line up transmitting and receiving periods without a master node and only 1 radio per node.

Bad data and fake nodes occurred frequently. We added in a 16-bit preamble check and a 2-byte checksum check, but still had issues. Over a 24 hour network test with 4 real nodes, the network reported in 21 nodes. Fortunately, the quality factor mitigated majority of the fake nodes as their LQI and hopcount numbers were randomized and nonzero.

6.1.3 Power Consumption

Battery life was estimated using the following equation

$$T_{Life} = \frac{C_{battery} * 3600}{T_{sleep} * A_{sleep} + A_{wake} * T_{wake}}$$

Where T is time measured in seconds, C is measured in milliamp hours, and A is measured in amperes. Our measured power consumption lined up almost exactly with the calculated values except during power-down which was an order of magnitude higher than expected at 2.5 mA. This was due to difficulties putting the transceiver into power-down mode. We measured the MCU current at 200 μA during sleep, leaving the analog to digital converters and the brown out detector active. Removing the transceiver and turning off those components resulted in consumption as low as 2 μA, but we elected to leave them on as it introduced complexity during startup and the transceiver's inability to sleep meant that further power optimization on the MCU would not significantly impact battery life.

The system spends 1 second in sleep followed by 1 second in wake for a 50% duty cycle. We tested the system with three nodes: one with our proposed CR2450 battery, one with a CR2032 battery, and an Arduino Uno attached to a PC to provide serial output.

The CR2450 had an estimated 60 hours of battery life, while the CR2032 had an estimated 22 hours of battery life. Our first CR2032 died after 9 hours of operation, but the second one died after 22 hours and 17 minutes. The first CR2450 died after 32 hours of operation, slightly over half of our expected battery life. The second died after 48 hours.

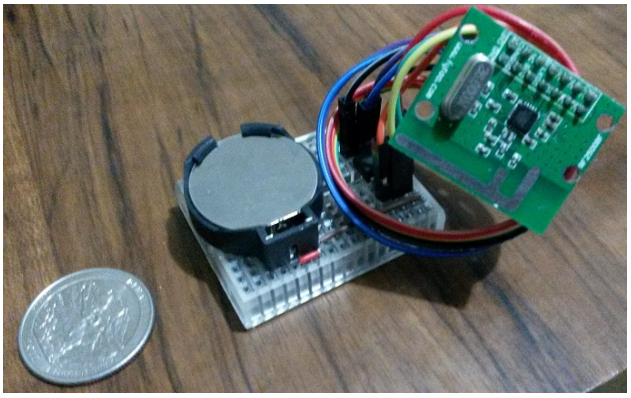

Figure 5: Node on a breadboard.

6.2 Intercommunication

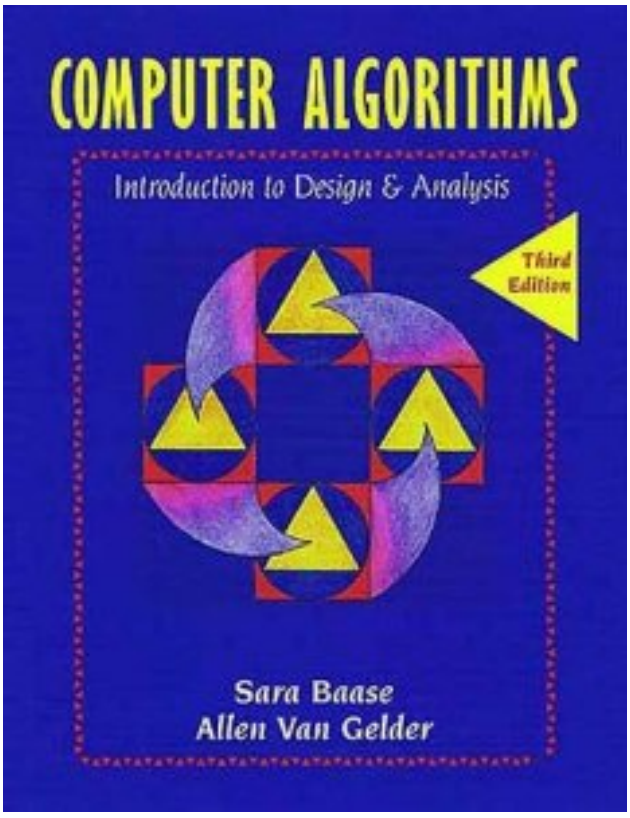

Figure 4: Project cost measured in textbooks.

While 20 nodes are fine for hardware simulation, we wanted to investigate a more realistic number of sensor nodes. However, it is not possible to install ContikiOS onto our MCU due to memory limitations. We attempted to enable intercommunication by acquiring an ATMega128RFA1. This microcontroller is directly supported by Contiki. Though it has a Zigbee-compatible transceiver on board, we would still need to provide it with a CC2500 transceiver as the Zigbee-compatible onboard transceiver is incompatible with ours. This plan ran into an early critical problem: the ATMega does not have on-board support for USB. This makes getting output from them extremely difficult. A solution could not be found in time, so this part remains unfinished.

6.3 Throughput

The throughput was observed by simulating in Contiki to observe the throughput possible given a duty cycle. The heart-beat pro-

Number of Nodes	Success rate	Latency (s)
20	26.5%	160
200	4.5%	114

Table 6: Bill of Materials and Per-Part Cost on Different Scales of Production (as of November 2013).

tocol produced consistent throughput during our energy simulations, although there were still instances of corrupted packets propagating despite the use of checksums. Our initial tests attempted to simulate these protocols with 200 randomly placed nodes using the Cooja [**?**] network simulator using the standard radio broadcast model, for 30 minutes. The nodes received simulated temperature data every 8 minutes from their sensor and attempt to update the network about it, while keeping track other nodes' data as well. The heartbeat protocol was unable to establish a global pattern. The random sleep cycle model was able to connect and send data, but the majority of the data did not reach the entire network. Of the data that remained persistent on the network, it was only found on 22% of the nodes, on average. For the successfully transferred data, the average throughput was observed to be 144 ms, however. On a smaller network with 10 randomly placed nodes the results improve a little, and could probably be improved further by manually placing nodes and managing the network set-up.

7 Challenges

The ATTiny85 is extremely limited on General Purpose Input Output (GPIO) pins. The entire chip only has 8 pins; after reset, power, and ground the user is left with only five pins to work with. 1 There are concerns of networks splitting and forming two separately synced networks, even more so in hardware where the watchdog is an imprecise timer.

The CC2500 is a very low-level radio transceiver. As such a significant amount of 'bit-banging'

It was surprising how easily bad data passed preamble, checksum, packet length, and opcode checks. Four forms of validation still was insufficient. Werner-Allen et. al. mention similar problems with their flooding time syncronization protocol despite using the more sophisticated Zigbee-compatible CC2420 provided on the Telos mote they used [11].

Ensuring a transmission is guaranteed to be received by at least 1 node was surprisingly difficult in smaller networks. The larger network test showed no sync loss, but on the 2-3 node network tests nodes fell out of sync around once per minute.

Lengths of cables caused problems in this low power environment. The jumper wires

between the nodes and the transceiver were made by the lowest bidder, and had a tendency to lose parts of the signal. While not difficult to measure, we are confident that this significantly contributed to the packet errors. Along with this, the USB cable used for serial communication was too long, eventually resulting in serial print errors that would lock the Uno and force a reset. During the long-term network test the Uno required frequent restarting. Fortunately, it regained the node list quite quickly (including any fake nodes).

8 Future Work

Due to the short time period given on this project, the greatest limiting factor was time. The system merits much deeper investigation, and there is plenty to explore.

Despite multiple forms of error checking, bad data still arrived and propagated through the network quite quickly. LQI this does lend itself to interesting potential applications. RSSI and LQI numbers are subject to interference from Wi-Fi router, cell phones, microwaves, and other sources of 2.4 GHz interference.

The system's sleep current consumption was significantly higher than planned. Though the wake/sleep duty cycle chosen was not significantly impacted by our limitations. As the duty cycle decreases and the system spends more time in sleep, the need to achieve the proposed power consumption are more critical. With a 1 second wakeup every 10 minutes the system lasts a little over 9 days with our current power consumption, but over 2 years if had the system hit the ideal consumption numbers.

9 Acknowledgements

Our thanks to ACM SIGCHI for allowing us to modify templates they had developed.

References

[1] Contiki 2.6 documentation, 2013.

[2] Tinyos documentation wiki, 2013.

[3] A. Eswaran, A. Rowe, and R. Rajkumar. Nano-RK: An Energy-Aware Resource-Centric Operating System for Sensor Networks. *IEEE Real-Time Systems Symposium*, 2005.

[4] W.R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. *System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on*, 2000.

[5] Atmel Inc. Attiny25/45/85 complete datasheet.

[6] G. Mainland, D. C. Parkes, and M. Welsh. Decentralized, Adaptive Resource Allocation for Sensor Networks. *Proceedings of the 2nd conference on Symposium on Networked Systems Design Implementation*, 2:315–328, 2005.

[7] R. Min, M. Bhardwaj, S. Cho, and E. Shih. Low-power wireless sensor networks. *VLSI Design, 2001. Fourteenth International Conference on*, pages 205 – 210, 2001.

[8] J. Polastre, R. Szewczyk, and D. Culler. Telos: enabling ultra-low power wireless research. *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 364 – 369, 2005.

[9] K. Ramesh and K. Somasundaram. A comparitive study of clusterhead selection algorithms in wireless sensor networks. *International Journal of Computer Science Engineering Survey*, 69(4), 2011.

[10] Curt Schurgers, Vlasios Tsiatsis, Saurabh Ganeriwal, and Mani Srivastava. Topology management for sensor networks: exploiting latency and density. *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking computing*, pages 135 – 145, 2002.

[11] G. et Al Werner-Allen. Fidelity and Yield in a Volcano Monitoring Sensor Network. *Proceedings of the 7th symposium on Operating systems design and implementation*, 2006.

[12] K. M. Zuberi, P. Pillai, and K. G. Sin. Emeralds: A small-memory real-time microkernel. *In Proceedings of the 17th ACM Symposium on Operating System Principles ACM Press*, pages 277–291, 1999.