EcoDAQ: A Case Study of a Densely Distributed Real-Time System for High Data Rate Wireless Data Acquisition

Chong-Jing Chen[1]
[1]Center for Embedded Computer Systems
Univ. of California, Irivne, CA 92697, USA

Pai H. Chou[1,2]
[2]Department of Computer Science
National Tsing Hua Univ., Hsinchu, Taiwan

{chongji, phchou}@uci.edu

Abstract

Densely distributed wireless real-time system must perform communication scheduling and protocols in addition to task scheduling to achieve schedulability and reliable communication. While many wireless sensor networks are densely distributed, they assume low duty cycling and trivial data packet rates, and they cannot handle many real-world applications.

To highlight these design issues, this paper presents a case study with EcoDAQ, a wireless data acquisition system with 50 sensor nodes streaming real-time data over the same frequency channel. To achieve reliable communication, we define a pulling protocol that minimizes node complexity while guaranteeing collision freedom on a given frequency channel. Our case study illustrates the impact of communication requirements on the rest of the systems including bus and processing speeds. Experimental results show that our monitoring station can sustain gathering accelerometer data from up to 50 nodes in one square meter area at up to 15,000 samples per second with minimal latency and show very high expandability, flexibility and reliability.

1. Introduction

One of the most important trends in real-time and embedded systems is the combination of miniaturization and wireless connectivity. However, many researchers have focused on applications that require only occasional, very low-bandwidth communication. This assumption is necessary in order to support the relatively high overhead incurred by the network protocol, the medium access control (MAC) protocol, and limited battery life. Unfortunately, many real-world applications do not match some of these assumptions. The data streams might require much higher bandwidth, on the order of hundreds of samples per second per node, and low latency, on the order of hundreds of milliseconds. For instance, in biomedical applications, an electroencephalogram (EEG) or an electrocardiogram (ECG) system can easily require five to ten nodes for each patient, and each node typically requires over 200 samples per second [8, 7]. It is not uncommon for other vital sign monitors also to be attached to a patient, and several patients may share the same room. The density of deployment may be as high as 50 to 100 nodes in a small area.

A critical issue in such real-time systems is the wireless communication protocol. All nodes must utilize the available bandwidth effectively. Existing solutions are often divided into CSMA and TDMA styles. CSMA, as assumed by the popular IEEE 802.15.4, is efficient when the utilization is low, but the probability of collision increases rapidly as utilization increases [20, 13]. TDMA, popular with *wired* real-time communication, has been adopted by WSNs but primarily to save power by reducing *idle-listening* time of low duty-cycle communication. Unfortunately, TDMA incurs time synchronization cost, and lost packets still must be handled separately. Hybrid schemes attempt to combine the best features of both approaches, though they have mostly been limited to simulations or controlled environments with little or no loss packets.

To highlight design issues in these densely distributed, wireless-enabled real-time systems with non-trivial data rates, we propose EcoDAQ. Our protocol has the property of intra-network collision freedom, low complexity, and requiring very small code and RAM sizes. Furthermore, the low resource demand of our protocol makes it possible to build ultra-compact, ultra-wearable, ultra-low-power systems that can be deployed very densely.

The majority of WSN works choose the "push" style of communication, where data is actively transmitted by the sensor nodes back to the host, whether TDMA or CSMA is used. The alternative style as advocated by this paper is the "pulling" style, or the *thin-server, fat-client* organization, where the nodes are passive and the host actively pulls

data from them. By moving most of the complexity from the node to the host, it also enables interactive debugging over a wireless link [19]. The nodes only need to be implemented the necessary mechanisms, while the host can implement a variety of coordination policies. An example of pulling style communication over TDMA was developed for a dance ensemble application [6]. However, it also inherited the complexity of slot scheduling and time synchronization of TDMA style protocols.

One popular standard, Bluetooth, has higher specified bandwidth, but its high complexity, relatively high power consumption, and low scalability make it difficult to adapt for sensor networks [17, 1, 7]. DESYNC [9] targets high data rate applications with a self-organizing slotting protocol. They build a cooperative slot adjustment scheme on the top of a CSMA style MAC, where each node regularly adjusts its own slot time towards the most evenly distributed slot time based on the firing times of its *phase neighbors*. It is of low complexity and can achieve high utilization of bandwidth. Unfortunately, this protocol does not address the *hidden terminals* problem, and it becomes unstable when the packet loss rate is high. Directed Diffusion [10] is a request/reply model, but it works at the routing layer. Our approach solves issues above and has the property of collision freedom, very low complexity and high scalability without complicated time synchronization.

2. System Description

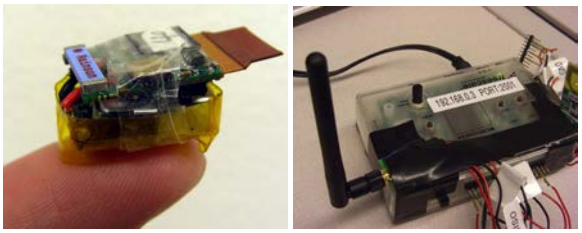
This section describes the hardware, software, and the wireless network of EcoDAQ in more details.

2.1 Hardware and Software

Sensor Node: We use Eco [14, 3], an ultra-compact, self-contained sensor node shown in Figure 1(a). It is only 1 cm^3 in volume including the MCU, RF, antenna, and sensor devices. We program Eco firmware to run on "bare metal," i.e., without an operating system or other runtime support. The code size for basic routines to drive the sensor node is around 1KB. Although TinyOS [16] is commonly used by other sensor platforms, the fundamental code space for TinyOS is around 3.5KB, which is too large for Eco.

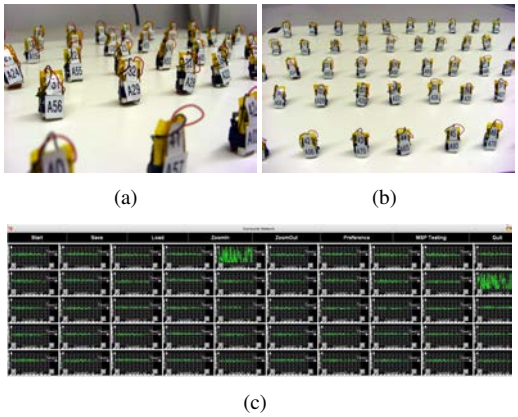
Base Station: We built a Fast Ethernet base station by connecting a Freescale DEMO9SNE64 evaluation board [4] to a transceiver module. The evaluation board is based on the Freescale 16-bit MC9S12NE64 MCU. The TCP/IP stack is provided by the Metrowerks CodeWarrior [2].

Radio: EcoDAQ's base station uses the Nordic nRF24L01 transceiver. It is compatible with the Nordic nRF24E1 integrated transceiver and MCU on Eco at 1Mbps speed mode. Its MAC includes hardware for CRC calculation and checking.



(a) (b)

Figure 1. EcoDAQ System Components. (a) Eco on an index finger (b) Freescale base station



(a) (b)

(c)

Figure 2. Experimental System (a)(b) 50 active Ecos on a poster board (c) Screenshot of real-time activities of 50 sensors

Host: A personal computer equipped with Windows XP and a 20-inch LCD monitor acts as the monitor station. The host computer controls wireless sensor nodes by a graphical user interface that displays the real-time data of all 50 sensor nodes. All Programs are written in Python [5] with the Tkinter tool kit.

2.2. Wireless Network

Fig. 2 shows the EcoDAQ wireless network. It consists of 50 Eco nodes in a star topology. Each node transmits a stream of data over the same frequency channel wirelessly to the base station connected to the host computer over Fast Ethernet. To run EcoDAQ, we first assign a unique software address to each node from 0 to 49. Next, we turn on the base station and the sensor nodes, and the host starts collecting

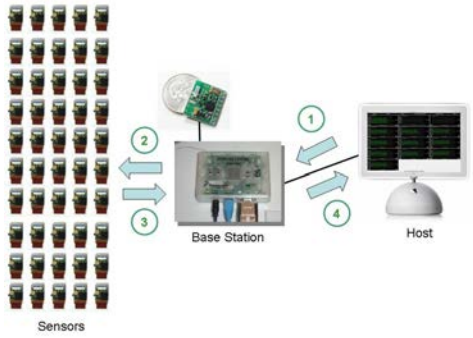


Figure 3. Steps of the EcoDAQ Protocol.

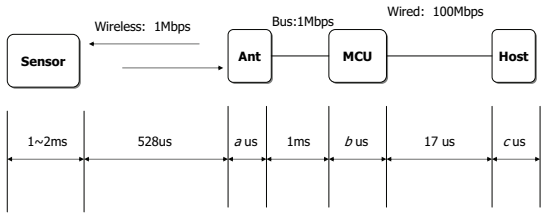


Figure 4. EcoDAQ Performance Estimation.

data from all 50 Eco nodes. The next section describes the communication protocol.

3. Communication Protocol

Our protocol follows pulling style. A light-weight scripting server inside each sensor node parses commands sent by the host. This may appear to generate extra pulling traffic, but it can be structured in a very efficient way while at the same time minimizing complexity on the sensor node. Figure 3 shows the steps of the communication protocol. The following paragraphs explains these steps.

1. The host issues a command packet when it wants data from a sensor node. The ID field in the command packet is X.

2. The base station parsers the command packet sent by the host and then broadcasts the command packet to all sensor nodes.

3. The sensor node whose ID field is equal to X will respond to the packet by transmitting sensing data back to the base station.

4. The base station will forward the response packet it receives from the wireless interface to the host.

If the host does not receive a reply from a sensor node, then it simply pulls again after a predefined timeout period, whether the previous pull or the reply message was lost. Note that the sensor nodes need not perform time synchronization or any wireless bus arbitration, since the host is assumed to have plenty of resources and processing power to perform communication scheduling, and the nodes are assumed to react within a known time bound. With a collision-free protocol, the system can save some energy for retransmission. The pulling overhead, which effectively includes acknowledgment, can be further amortized several ways. First is to have a single pulling message over a whole group of sensor nodes, each of which would reply with a different time offset. Second is to reply more data on each pull, though at the expense of longer latency. In either case, section 4.4 shows that the footprint for MAC and retransmission protocol in RAM and ROM can be truly minimized.

4. Analysis and Experimental Results

Our metrics include wireless data throughput and the code size. What the user is ultimately concerned with is the end-to-end throughput, but for us to identify the bottleneck, we also analyze the performance in each stage of the system, as shown in Fig. 4. This section first presents throughput results by varying (1) the bus on the base station and (2) the number of reply packets per pull. Then, we present the code size for the runtime support.

4.1. Base Station Performance

The base station is potentially the bottleneck, as it bridges between 50 sensor nodes on a wireless channel and the host via the Fast Ethernet interface. This subsection analyzes the effect of base station performance on the overall system. We measure and analyze the timing for the simplest protocol that entails a single pull by the host and a single reply from the node.

4.1.1 Fast Ethernet

The size of data payload at application layer is 27 bytes. Taking into account all the headers for a Fast Ethernet packet, the total packet length will be 105 bytes. So the total time spent on the Fast Ethernet cable is around 17 μs, without any MAC processing time.

4.1.2 Radio Interface

The transceiver is connected to the base station via a bus and controlled by commands sent by the base station. The size of data payload is 27 bytes and all the commands needed to

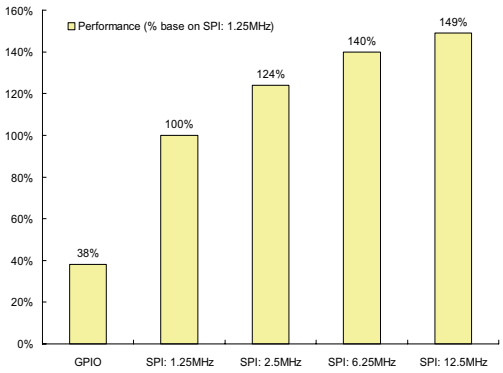


Figure 5. Bus Implementation vs. Performance.

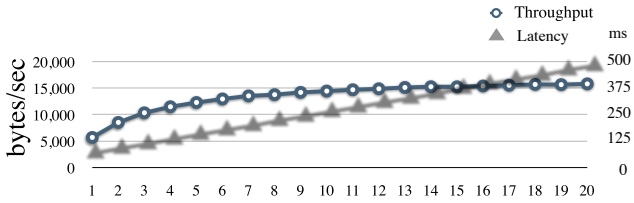


Figure 6. Performance Improvement via Multiple Replies. Circles: Aggregated payload bandwidth by multiple replies; Triangles: Response time for multiples replies.

control the transceiver are around 30 bytes. We consider the waiting time between commands and assume the bus speed is 1Mhz, then the estimated time spent on the radio interface is around 1ms.

The radio interface can be implemented with either general purpose I/O (GPIO) in software or SPI in hardware. As shown in Fig. 5, the end-to-end data rate improvement is 2.5× faster while setting the SPI clock speed at 1.25Mhz.

Furthermore, the clock speed of the SPI bus on the base station's MCU can be configured to be the 25MHz core speed divided by 2, 4, 10, or 20, which translate into 12.5MHz, 6.25MHz, 2.5MHz, and 1.25MHz respectively. The SPI bus clock speed will affect the system performance. Fig. 5 shows the end-to-end throughput over these different SPI speeds. Note that doubling the SPI speed from 1.25MHz to 2.5MHz results in a 24% increase in throughput; 5× SPI speed to 6.25MHz results in 40% throughput increase. At the maximum SPI speed of 12.5MHz, the throughput is 49% over the baseline. This shows diminishing increase of throughput for SPI speed over 6.25MHz, and we can infer that there exists a different bottleneck elsewhere in the system.

4.1.3 Timing Breakdown

We use the following symbols for denoting the various estimated processing times:

symbol (μs)	component
a	wireless transceiver
b	MCU on base station
c	host

The total packet length for the wireless interface is 33 bytes. It takes 264 μs to finish one-way wireless transmission, and thus the total time spent on wireless transmission is 528 μs. In our experiment, it takes 1 to 2 ms for a sensor node to process data. So, the total waiting time for a host to receive a data packet after sending a command to a sensor node is between $(2545+a+b+c)$ μs and $(3545+a+b+c)$ μs. It is difficult to predict the processing times for the wireless transceiver, the MCU on the base station, and the host. Without consider those processing times, the minimal waiting time for one data packet is between 2545 μs and 3545 μs. The useful sensing data is only 25 bytes out of 27 bytes data payload. Assuming the scenario of one reply packet per pull command per node, the upper bound on the throughput of the data payload is between 9823 bytes/s and 7052 bytes/s. The throughput can be improved by having the node send multiple reply packets in response to each pull.

4.2. Multiple Replies

One downside with pulling style protocols is the overhead of the pulling message, which may be amortized by increasing the number of reply messages in response to each pull. However, this number is limited by several factors: the latency constraint, each node's buffer size in relation to the throughput, and the expected packet loss rate. Fig. 6 shows the 50-node aggregated payload bandwidth handled by one host in a highly controlled environment with no packet loss in the air. As the throughput increases over the number of reply packets, Fig. 6 shows that the packet round trip time increases linearly. The system archives 15.8 Kbytes/s throughput for 20 reply packets per pull, but at the same time it would take around 1.7 seconds to pull 20 sensing packets of 25 bytes of payload each from all of 50 sensor nodes. This latency may be too long for certain applications such as EEG, ECG, or interactive dance.

4.3. Performance Comparison

We measure the actual timing of each step in EcoDAQ using an oscilloscope. Fig. 7 shows the measured timing chart for EcoDAQ's round-trip communication. We use this

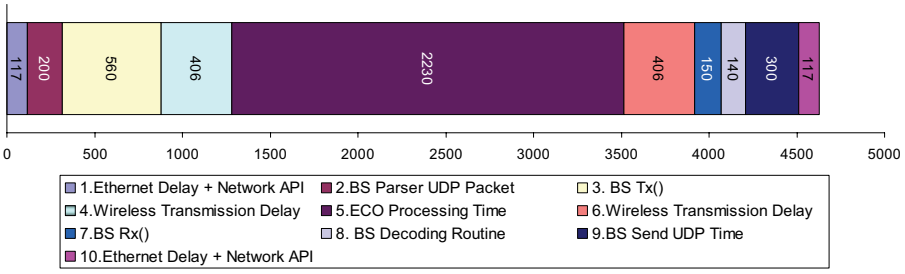


Figure 7. Measured timing of EcoDAQ's round-trip communication

timing chart to estimate the maximum system performance with ideal TDMA MAC. We also use nctuns [18] to simulate CSMA style MAC for comparison.

For the performance estimation of TDMA style system, we assume each TDMA frame has 101 fixed slot times. One slot time for global time synchronization, 50 slot times for 50 sensor nodes to transmit sensing data, and 50 slot times for the base station to send back an acknowledgment. Totally it takes around 2 ms for an Eco node to execute routines for sensing and sending data packet. So, we assume each TDMA slot time is 2 ms, and all 50 sensor nodes will use their own time slots to send data.

In CSMA scenario, we assume all 50 sensor nodes sending data with their best efforts as assumed by DESYNC [9], and we measure the data payload throughput on the host. Both TDMA and CSMA systems will use ACK for reliable wireless transmission, where the receiver acknowledges every packet. All sensor nodes transmit 27 bytes of data payload per packet at 1 Mbps wireless speed, and all of them can hear each other. As shown in Table 1, the CSMA style system has the worst performance under high data rate scenario due to wireless contention and collision. EcoDAQ's performance is 14% below the system with ideal TDMA MAC, which is also collision-free. However, the main drawback of TDMA is the dynamic joining and leaving of sensor nodes and the assignment of multiple time slots to one sensor node. EcoDAQ's protocol does not incur this overhead and thus the complexity on the sensor nodes can be kept very low.

Table 1. EcoDAQ vs. TDMA and CSMA .

Scheme	EcoDAQ	TDMA	CSMA
Data Payload Throughput (Bytes/s)	5837	6682	3707
Packet Collision & Dropping	No	No	Yes
Dynamic Nodes Joining & Leaving	Easy	Limited	Easy
Dynamic Multiple Slot Assignment	Yes, throughput goes up to 15.8 KB/s	None	Yes

Table 2. Firmware sizes for EcoDAQ, TDMA, and CSMA style MAC.

Scheme	Code Size
EcoDAQ with MAC Only	14 Bytes
Regular EcoDAQ	759 Bytes
Eco Basic Routines	1096 Bytes
CSMA Style B-MAC	3KB $\sim$ 4.5KB [15]
TDMA Style MAC	17.5 KB $\sim$ 21.3KB [11]

4.4. Footprint Comparison

A main advantage with EcoDAQ is that it moves bus arbitration and retransmission routines from the sensor nodes to the base station and the host. This mechanism enables the sensor nodes to be kept minimally simple with only the essential mechanisms. The firmware footprint in the EEPROM is also minimized.

Table 2 compares the size requirements for firmware between EcoDAQ, TDMA and CSMA style MAC. The binary size of the basic routines for an Eco is 1096 bytes. We implement a parser to enable execution of different types of commands from the host, e.g., to send back the values of certain parameters for debugging, to change wireless channel frequency on the fly, or to put an Eco in sleeping mode for a certain amount of time. The command dispatcher is written in C and occupies 759 bytes on top of the basic routines, for a total of 1885 bytes. If a user just wants the simplest MAC layer functionality, the code is only 14 bytes on top of the basic routines, and the total code size is down to 1100 bytes. Although the ideal TDMA protocol has 14% higher throughput, it would occupy 17–21KB of code, which unfortunately is an order of magnitude larger than our code and is five times the size of Eco's total EEPROM capacity [11].

5. Conclusion and Future Work

We describe a case study with EcoDAQ, a wireless sensor network with high expansibility and flexibility. With fifty sensor nodes deployed within a $1m^2$ area, the system can sustain the data payload bandwidth as high as 15.8 KB/s fully acknowledged without intra-network wireless collision. This translates into around 15,800 samples per second with either uncompressed single-byte samples or with simple compression. The EEPROM footprint of the MAC protocol is light weight and can be as small as 14 bytes only. Post-deployment changes can be done on the host side without complicated protocols for joining or leaving.

For future work, we plan to continue identifying and eliminating the remaining bottlenecks on the nodes and base station to achieve higher aggregate or single-stream bandwidth. One major enhancement is the use of multiple base stations to handle much larger scale networks after we saturate the bandwidth on one channel. Going to multiple base stations would require development of hand-off protocols and use of multiple frequencies. This time, the Fast Ethernet uplink may become the bottleneck, thereby necessitating integration of multiple hosts. Additional features such as hardware ACK and buffering are likely to enable new protocols to be designed. On the host side, display of real-time sensing data is also being scaled up to not only a single computer and a single screen but actually to giant tiled display systems driven by a cluster of workstations [12].

6. Acknowledgment

The authors would like to thank Seung-Mok Yoo, Jinsik Kim, Qiang Xie, Prof. Stephen Jenks, and Dr. Sungjin Kim for their assistance with this work. This research project is sponsored in part by the National Science Foundation CAREER Grant CNS-0448668, UC Discovery Grant itl-com05-10154, the National Science Council (Taiwan) Grant NSC 96-2218-E-007-009, and Ministry of Economy (Taiwan) Grant 96-EC-17-A-04-S1-044.

References

[1] Bluetooth technology. `http://www.bluetooth.com/bluetooth/`.

[2] Codewarrior development tools. `http://www.freescale.com/webapp/sps/site/homepage.jsp?nodeId=012726`.

[3] Ecomote. http://www.ecomote.net/.

[4] Freescale semiconductor. http://www.freescale.com/.

[5] Python programming language. http://www.python.org/.

[6] R. Aylward, S. D. Lovell, and J. A. Paradiso. A compact, wireless, wearable sensor network for interactive dance ensembles. In *BSN '06: Proceedings of the International Workshop on Wearable and Implantable Body Sensor Networks*, pages 65–70, Washington, DC, USA, 2006. IEEE Computer Society.

[7] N. Chevrollier and N. Golmie. On the use of wireless network technologies in healthcare environments. *Proceedings of the fifth IEEE workshop on Applications and Services in Wireless Networks*, pages 147–152, 2005.

[8] D. Cypher, N. Chevrollier, N. Montavont, and N. Golmie. Prevailing over wires in healthcare environments: benefits and challenges. *Communications Magazine, IEEE*, 44(4):56–63, April 2006.

[9] J. Degesys, I. Rose, A. Patel, and R. Nagpal. Desync: self-organizing desynchronization and tdma on wireless sensor networks. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 11–20, New York, NY, USA, 2007. ACM.

[10] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *MOBICOM*, pages 56–67, 2000.

[11] K. Klues, G. Hackmann, O. Chipara, and C. Lu. A component-based architecture for power-efficient media access control in wireless sensor networks. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 59–72, New York, NY, USA, 2007. ACM.

[12] F. Kuester, J.-L. Gaudiot, T. C. Hutchinson, B. Imam, S. Jenks, S. G. Potkin, S. A. Ross, S. Sorooshian, D. Tobias, B. Tromberg, F. J. Wessel, and C. Zender. Hiperwall: A high-performance visualization system for collaborative earth system sciences. 2004.

[13] J. V. Misic, S. Shafi, and V. B. Misic. The impact of MAC parameters on the performance of 802.15.4 PAN. *Ad Hoc Networks*, 3(5):509–528, 2005.

[14] C. Park and P. H. Chou. Eco: Ultra-wearable and expandable wireless sensor platform. In *Third International Workshop on Body Sensor Networks (BSN'06)*, April 2006.

[15] J. Polastre, J. L. Hill, and D. E. Culler. Versatile low power media access for wireless sensor networks. In J. A. Stankovic, A. Arora, and R. Govindan, editors, *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, SenSys 2004, Baltimore, MD, USA, November 3-5, 2004*, pages 95–107. ACM, 2004.

[16] TinyOS: An open-source OS for the networked sensor regime. http://www.tinyos.net.

[17] J. Vlimki. Bluetooth and ad hoc networking, May 2002.

[18] S. Y. Wang, C. L. Chou, C. H. Huang, C. C. Hwang, Z. M. Yang, C. C. Chiou, and C. C. Lin. The design and implementation of the NCTUns 1.0 network simulator. *Computer Networks*, 42(2):175–197, 2003.

[19] K. Whitehouse, G. Tolle, J. Taneja, C. Sharp, S. Kim, J. Jeong, J. Hui, P. Dutta, and D. Culler. Marionette: using rpc for interactive development and debugging of wireless embedded networks. In *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*, pages 416–423, New York, NY, USA, 2006. ACM.

[20] J. Zheng and M. J. Lee. *A Comprehensive Performance Study of IEEE 802.15.4*. IEEE Press Book, 2004.