

RICE UNIVERSITY

**Efficient Duty Cycle MAC Protocols  
for Dynamic Traffic Loads  
in Wireless Sensor Networks**

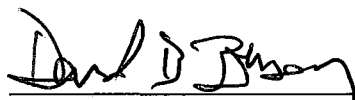
by

**Yanjun Sun**

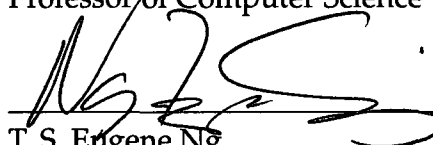
A THESIS SUBMITTED  
IN PARTIAL FULFILLMENT OF THE  
REQUIREMENTS FOR THE DEGREE

**Doctor of Philosophy**

APPROVED, THESIS COMMITTEE:



David B. Johnson, Chair  
Professor of Computer Science



T. S. Eugene Ng  
Assistant Professor of Computer Science



Edward W. Knightly  
Professor of Electrical and Computer  
Engineering

HOUSTON, TEXAS

APRIL, 2009

UMI Number: 3362416

### INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.



---

UMI Microform 3362416  
Copyright 2009 by ProQuest LLC  
All rights reserved. This microform edition is protected against  
unauthorized copying under Title 17, United States Code.

---

ProQuest LLC  
789 East Eisenhower Parkway  
P.O. Box 1346  
Ann Arbor, MI 48106-1346

# Efficient Duty Cycle MAC Protocols for Dynamic Traffic Loads in Wireless Sensor Networks

Yanjun Sun

## Abstract

*Idle listening* is one of the most significant causes of energy consumption in wireless sensor networks (WSNs), and many protocols have been proposed based on *duty cycling* to reduce this cost. These protocols, either *synchronous* or *asynchronous*, are mainly optimized for light traffic loads. A WSN, however, could often experience bursty and high traffic loads, as may happen for example with broadcast or convergecast traffic. In this thesis, I design and evaluate a new *synchronous* protocol, *DW-MAC* (Demand Wakeup MAC), and a new *asynchronous* protocol, *RI-MAC* (Receiver Initiated MAC), that are both efficient under dynamic traffic loads, including light or heavy loads. I also design and evaluate *ADB* (Asynchronous Duty-cycle Broadcasting), a new protocol for efficient multihop broadcasting in WSNs using asynchronous duty cycling.

*DW-MAC* introduces a new low-overhead scheduling algorithm that allows nodes to wake up on demand during the Sleep period of an operational cycle and ensures that data transmissions do not collide at their intended receivers; this demand wakeup adaptively increases effective channel capacity as traffic load

increases. RI-MAC, instead, uses *receiver-initiated* transmissions, in which each transmitter passively waits until its intended receiver wakes up and transmits a *beacon* frame; this technique minimizes the time a sender and its intended receiver occupy the wireless medium to find a rendezvous time for exchanging data. ADB is integrated with RI-MAC to exploit information only available at this layer; rather than treating the data transmission from a node to all of its neighbors as the basic unit of progress for the multihop broadcast. ADB dynamically optimizes the broadcast at the level of transmission to each individual neighbor of a node as the neighbors asynchronously wakeup, avoiding redundant transmissions and transmissions over poor links, and allowing a transmitter to go to sleep as early as possible. In detailed simulation of all three protocols using *ns-2*, they each substantially outperform earlier competing protocols in terms of reduced energy and latency and increased packet delivery ratio. I also implemented RI-MAC and ADB in a testbed of MICAz motes using TinyOS and further demonstrate the significant performance improvements made over prior protocols.

## Acknowledgments

I would like to express my deep and sincere gratitudes to my supervisor, David B. Johnson. His inspiring advice and guidance have absolutely been the most invaluable help I received to get this work done. Besides our weekly meetings, whenever I have questions, even including small questions about  $\text{\TeX}$  and Illustrator, Dave was always there to patiently listen and to give advice. Although most paper deadlines were around midnight, Dave always stayed up with me till the last minute to keep improving the quality of our submission. His enthusiasm and prudent attitude towards research projects not only have greatly impacted my thesis, but will impact my future career.

I am deeply grateful to Dr. Edward W. Knightly, Dr. T.S. Eugene Ng, and members in their group. Ed and Eugene were on the committees of both my Master's thesis and Ph.D. thesis. In the preliminary stage of my theses work, they had valuable discussions with me and recommended papers that were helpful for my work. During my thesis proposal and defense, Ed and Eugene continued to help me see the strengths and deficiencies in the theses with their valuable suggestions and insightful opinions. I would also like to express my sincere thanks to members in Ed and Eugene's group. Omer Gurewitz and Jingpu Shi had always discussed

with me for hours on my work. The discussion helped me to clearly define problems and solutions and to improve by ability of critical thinking. I would also like to thank Vincenzo Mancuso, Eugenio Magistretti, Stanislav Miskovic, Bo Zhang, Guohui Wang, Zheng Cai, Florin Dinu and Jie Zheng for their valuable discussion, feedback and support that helped to improve my work.

Furthermore, I owe sincere gratitude to members in our MONARCH group. Santa PalChaudhuri was the one who brought me to the field of wireless sensor networks. Amit Saha was always the first one I grabbed for questions and discussions as we were in the same office. Shu Du clearly showed me opportunities in MAC research and have been working side by side with me on several projects. New members in the group, Keyvan Amiri and Lei Tang, also had valuable discussions with me and supported me in my experiments, proposal and defense.

Throughout my Ph.D. years, I had the opportunity to work with many professors and their students and I am grateful to them as well. In the COMPASS project, Dr. Richard Baraniuk, Dr. T.S. Eugene Ng, and Raymond Wagner have given valuable feedback to DW-MAC and RI-MAC from different angles when these protocols were just a simple idea. Ryan Stinnett helped me to set up a testbed of motes, with which I turned RI-MAC into a protocol running in a real system. Collaboration with Dr. Lin Zhong and his students Mian Dong, Jiayang Liu helped me to get familiar with power saving schemes more from hardware side. The work experience with them is unforgettable. I also had memorable experience of working with

Deian Tabakov, Rui Zhang, Gua Hua and Yuan Zhao. They were always around when I needed help from them. A note of thanks goes to Johnny Ngan, Anwis Das, Scott Crosby and all my friends in the Computer Science Department for their support and sharing a good memory with me. I am also grateful to Darnell Price, Bel Martinez, Beth Rivera, Lena Sifuentes for their support during my Ph.D. years.

My deepest gratitude goes to my wife, Bing Yuan, my parents and parents-in-law, for their unconditional love and caring. Without their encouragement, help and understanding, it would have been impossible for me to finish this work in time. They care more about my work than their own work. My parents and in-laws even sacrificed their time and came to Houston to take care of our newborn before my defense, and my wife delayed her thesis work to take care of our baby when I was doing experiments. I would also like to thank my son, Richard, for being a good baby during my busiest time. Every small step of achievement during my Ph.D. process was supported by them and I am forever indebted to them.

My work was supported in part by NSF under grants CNS-0520280, CNS-0435425, CNS-0338856, CNS-0325971, and CNS-0209204; and by a gift from Schlumberger. The views and conclusions contained here are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of NSF, Schlumberger, Rice University, or the U.S. Government or any of its agencies.

# Contents

Abstract	ii
Acknowledgments	iv
List of Illustrations	xi
List of Tables	xvii
<b>1 Introduction</b>	<b>1</b>
1.1 Duty Cycling . . . . .	1
1.2 The Need for Handling Dynamic Traffic Loads . . . . .	2
1.3 DW-MAC: A New Synchronous Duty Cycle MAC Protocol . . . . .	4
1.4 RI-MAC: A New Asynchronous Duty Cycle MAC Protocol . . . . .	6
1.5 ADB: An Efficient Multihop Broadcast Protocol over Asynchronous Duty Cycling . . . . .	9
1.6 Thesis Outline . . . . .	12
<b>2 Related Work</b>	<b>13</b>
2.1 Synchronous Duty Cycle MAC Protocols . . . . .	13
2.2 Asynchronous Duty Cycle MAC Protocols . . . . .	18
2.3 Broadcast over Asynchronous Duty Cycling . . . . .	24



<b>3</b>	<b>DW-MAC Design</b>	<b>29</b>
3.1	Overview . . . . .	29
3.2	Mapping Function for Scheduling . . . . .	31
3.3	Scheduling Frame (SCH) . . . . .	34
3.4	Broadcast and Unicast in DW-MAC . . . . .	35
3.5	Optimized Multihop Forwarding . . . . .	36
3.6	Implementation Issues . . . . .	39
<b>4</b>	<b>Evaluation of DW-MAC</b>	<b>41</b>
4.1	Evaluation under Unicast Traffic . . . . .	43
4.2	Evaluation under Broadcast Traffic . . . . .	51
<b>5</b>	<b>RI-MAC Design</b>	<b>59</b>
5.1	Overview . . . . .	59
5.2	Beacon Frames . . . . .	61
5.3	Dwell Time for Queued Packets . . . . .	63
5.4	DATA Frame Transmissions from Contending Senders . . . . .	64
5.5	Collision Detection and Retransmissions . . . . .	66
5.6	Beacon-on-Request . . . . .	68
5.7	RI-MAC Implementation in TinyOS . . . . .	69

<b>6</b>	<b>Evaluation of RI-MAC</b>	<b>73</b>
6.1	Simulation Evaluation . . . . .	73
6.1.1	Results in Clique Networks . . . . .	77
6.1.2	Results in a 49-Node Grid Network . . . . .	82
6.1.3	Results in Random Networks . . . . .	86
6.2	Experimental TinyOS Evaluation . . . . .	89
6.2.1	Results in Clique Networks . . . . .	92
6.2.2	Results in a Network with Hidden Nodes . . . . .	93
6.2.3	Extra Ending Beacons for MICAz . . . . .	96
<b>7</b>	<b>ADB Design</b>	<b>101</b>
7.1	Design Motivation . . . . .	101
7.2	Overview of ADB Operation . . . . .	103
7.3	ADB Algorithm Details . . . . .	105
7.3.1	Neighbor Detection and Link Quality Estimation . . . . .	105
7.3.2	Coherent Encoding of ADB Control Information . . . . .	109
7.3.3	Delegation Procedure . . . . .	113
7.4	Analysis of End-to-End Delivery Latency . . . . .	116
7.5	ADB Implementation in TinyOS . . . . .	120
<b>8</b>	<b>Evaluation of ADB</b>	<b>124</b>
8.1	Simulation Evaluation . . . . .	126

8.1.1	Results with Default Channel Model in <i>ns-2</i> . . . . .	128
8.1.2	Results with Increased Packet Losses . . . . .	133
8.1.3	Comparison to Optimal Latency . . . . .	137
8.2	Experimental Evaluation on MICAz Motes . . . . .	139
8.2.1	Results in a Clique Network . . . . .	140
8.2.2	Results in a Random Network . . . . .	141
<b>9</b>	<b>Conclusions</b>	<b>145</b>
	<b>Bibliography</b>	<b>148</b>

## Illustrations

- 2.1 S-MAC with adaptive listening. Node C wakes up at the end of the transmission between node A and B based on the information in the overheard CTS, so that B can forward a packet to C immediately rather than waiting until the next operational cycle. . . . 14
- 2.2 Multihop forwarding of a unicast packet in RMAC. P indicates a PION frame that is used for scheduling. . . . . 17
- 2.3 Operation of X-MAC, including the *strobed preamble* and *early acknowledgment*. During a scheduled wakeup time, a node does a CCA (clear channel assessment) check that is longer than the gap between two short preambles. . . . . 20
- 2.4 The variation of X-MAC implemented in the UPMA package in TinyOS. The strobed preamble is replaced by a chain of DATA frame transmissions. . . . . 20

2.5	Broadcast support in X-MAC in the UPMA package of TinyOS. A transmitter $S$ repeatedly transmits copies of a broadcast packet (DATA frame) over a duty cycle interval, during which each neighbor (node $R1$ and $R2$ ) wakes up at least once and thus has an opportunity to receive the packet. . . . .	27
3.1	Overview of scheduling in DW-MAC. . . . .	31
3.2	Broadcast in DW-MAC. . . . .	36
3.3	Unicast in DW-MAC. . . . .	36
3.4	Optimized multihop forwarding of a unicast packet. Node B sends an SCH to wake up node C at the time indicated by $T_2^s$ and confirms the SCH received from node A. . . . .	37
3.5	Optimized multihop forwarding of a broadcast packet. Node B specifies node A as the immediate forwarder, which rebroadcasts an SCH SIFS after receiving that SCH from A. Node C rebroadcasts the SCH when its backoff counter expires. . . . .	39
4.1	Performance for unicast traffic in 49-node grid network scenarios. . .	45
4.2	Performance for unicast traffic in 49-node grid network scenarios, with 2 random events generated at a time. . . . .	50

4.3	Performance for random correlated-event traffic in 50-node networks with sensing range of 250 m. . . . .	52
4.4	Performance for broadcast traffic in grid networks. . . . .	55
4.5	Performance for broadcast traffic in 50-node networks. . . . .	56
5.1	Overview of RI-MAC. Each node periodically wakes up and broadcasts a beacon. When node <i>S</i> wants to send a DATA frame to node <i>R</i> , it stays active silently and starts DATA transmission upon receiving a beacon from <i>R</i> . Node <i>S</i> later wakes up but goes to sleep after transmitting a beacon frame since there is no incoming DATA frame. . . . .	60
5.2	The format of an RI-MAC beacon frame for an IEEE 802.15.4 radio. Dashed rectangles indicate optional fields. The Frame Length, Frame Control Field (FCF), and Frame Check Sequence (FCS) are fields from IEEE 802.15.4 standard. . . . .	62
5.3	The dual roles of a beacon in RI-MAC. A beacon serves both as an acknowledgment to previously received DATA and as a request for the initiation of the next DATA transmission to this node. . . . .	63

5.4	DATA frame transmission from contending senders in RI-MAC. For the first beacon, the receiver $R$ requests senders (here, $S_1$ and $S_2$ ) to start transmitting DATA immediately upon receiving the beacon. If a collision is detected, $R$ sends another beacon with increased BW value to request that senders do a backoff before their next transmission attempt. . . . .	65
5.5	RI-MAC <i>beacon-on-request</i> . When node $S$ wakes up for transmitting a pending DATA frame, it sends a beacon with the Dst field set to the destination of the pending DATA. If the destination node $R$ is already active, $R$ in response transmits a beacon to enable $S$ to begin DATA transmission immediately. . . . .	69
5.6	Composition of RI-MAC within the UPMA framework in TinyOS. . .	70
6.1	Performance comparison in clique networks with contending flows in simulation. The total number of nodes is 1 for 0 flows, and is twice the number of flows otherwise. . . . .	79
6.2	Performance for unicast traffic in 49-node ( $7 \times 7$ ) grid network scenarios in simulation. . . . .	84
6.3	Performance for random correlated-event traffic in 50-node networks with sensing range of 250 m in simulation. . . . .	88

6.4	Performance comparison in clique networks of MICAz motes with contending flows in TinyOS implementation. . . . .	90
6.5	Performance comparison when two sender are hidden to each other and when they are not in a 3-node network in TinyOS implementation. . . . .	94
6.6	Effectiveness of using an extra <i>ending beacon</i> in RI-MAC in TinyOS implementation. . . . .	97
7.1	Overview of ADB. Node $S$ broadcasts a DATA frame to node $R1$ and $R2$ via unicast transmission. The footer in DATA and ACK beacons helps $S$ and $R1$ to decide which node will deliver the DATA to $R2$ and helps $R2$ to learn that both $S$ and $R1$ have received the DATA. . . . .	104
7.2	Node $v$ analyzes an ADB footer that contains information about the progress of packet $i$ . This footer is <i>received</i> or <i>overheard</i> from $w$ , containing an array $S_w$ that lists the status of $w$ 's neighbors. The separate $S^i$ local array lists the status of $v$ 's neighbors with respect to packet $i$ . . . . .	115
7.3	ADB achieves optimal latency under simplified assumptions. . . . .	117
7.4	Interaction between ADB, RI-MAC, and the UPMA framework in TinyOS. . . . .	120



8.1	Performance comparison in 50-node networks with default channel model in <i>ns-2</i> . . . . .	130
8.2	Performance comparison in 50-node networks with increased packet losses. . . . .	135
8.3	Difference between optimal delivery latency to each node and that with ADB. . . . .	138
8.4	Topology of a 10-node random network deployed in an apartment. .	142

## Tables

4.1	Networking Parameters . . . . .	42
4.2	Duty Cycle Configuration . . . . .	43
4.3	Average number of packets generated for each event under different sensing ranges in the 49-node grid network . . . . .	44
6.1	Simulation Radio Parameters . . . . .	74
6.2	Simulation MAC Protocol Parameters . . . . .	75
6.3	Average Number of Packets Generated for Each Event under Different Sensing Ranges in the 49-Node Grid Network . . . . .	83
8.1	Performance comparison in a 5-node TinyOS clique network . . . . .	141
8.2	Performance comparison in the 10-node TinyOS network . . . . .	143
8.3	Average duty cycle % of each node in the 10-node TinyOS network .	144

# Chapter 1

## Introduction

Wireless sensor networks (WSNs) have a significant potential in applications interacting with the physical world, such as surveillance and environmental monitoring. In many of these applications, the use of battery-powered sensor nodes greatly eases deployment of the network, but the limited capacity of the batteries substantially limits the network lifetime. *Idle listening* is one of the most significant sources of energy consumption in sensor nodes. In idle listening, a node waits with its radio turned on, listening for a possible packet to be received even when none has been sent.

### 1.1 Duty Cycling

Many solutions to the problem of idle listening have been proposed utilizing the technique of *duty cycling* [48, 35]. In this technique, each sensor node turns its radio on only periodically, alternating between active and sleeping states. For example, with a 5% duty cycle, a node has its radio on only 5% of the time, resulting in substantial energy savings. When active, a node is able to transmit or receive data, whereas when sleeping, the node completely turns off its radio to save energy; duty cycles of 1–10% are typical in order to maximize energy savings.

Contention-based duty cycle MAC protocols in the literature can be roughly categorized into *synchronous* and *asynchronous* approaches, together with some hybrid approaches. Synchronous approaches [48, 8] synchronize neighboring nodes in order to align their active or sleeping periods. Neighbor nodes begin exchange of a packet only within the common active time, enabling a node to sleep for most of the time in an operational cycle without missing any incoming packet. This approach greatly reduces idle listening time, but the required synchronization introduces extra overhead and complexity, and a node may need to wake up multiple times if its neighbors are on different schedules. Existing *asynchronous* approaches [11, 35, 3], on the other hand, allow nodes to operate independently, each on its own duty cycle schedule, by employing low power listening (LPL). In LPL, prior to data transmission, a sender transmits a *preamble* lasting at least as long as the sleep period of the receiver. When the receiver wakes up and detects the preamble, it stays awake to receive the data.

## 1.2 The Need for Handling Dynamic Traffic Loads

Existing duty cycle MAC protocols, including synchronous and asynchronous ones, are mainly optimized for light traffic loads. A WSN, however, could often experience bursty and high traffic loads. For example, either broadcast [33] or convergecast [50] traffic could suddenly increase channel contention in a local neighborhood. In WSNs, broadcast is widely used for various network wide queries and

updates [39], and convergecast is often observed when multiple sensors that have detected the same event send their reports to the sink node or to a node that does data aggregation [13].

As existing approaches are mainly optimized for light traffic loads, I found that they become less efficient in latency, power efficiency, and packet delivery ratio as traffic load increases. As traffic in a WSN can be quite dynamic, depending on the events being sensed and the sensing application and protocols being used, *an ideal WSN MAC protocol should perform well under a wide range of traffic loads, including high loads and bursty traffic.*

Research on duty cycle MAC protocols has been active both in the synchronous approach and in the asynchronous one, as neither approach always outperforms the other. The target application and network configuration highly affect which approach is best to be used. For example, when most packets arrive at regular intervals and/or synchronization overhead is low (e.g., a GPS receiver is available), a synchronous duty cycle MAC is generally more energy efficient. On the other hand, if synchronization overhead is high, an asynchronous duty cycle MAC protocol might be best. Therefore, in this thesis, I present both a synchronous duty cycle MAC protocol, called DW-MAC (Demand Wakeup MAC), and an asynchronous duty cycle MAC protocol, called RI-MAC (Receiver Initiated MAC), in order to meet various needs from applications. Furthermore, with asynchronous duty cycling, multihop broadcast becomes especially challenging as neighbors of

a node wake up at different times. The general assumption that one transmission can reach multiple nodes no longer holds, and thus broadcast protocols based on this assumption become less efficient. Therefore, I also present a protocol called ADB (Asynchronous Duty-Cycle Broadcasting) to explore opportunities for efficient broadcast with asynchronous duty cycling.

### 1.3 DW-MAC: A New Synchronous Duty Cycle MAC Protocol

In order to transmit a packet from one node to another, the radios of both nodes must be on, motivating the use of synchronization between the operational cycles of different nodes. Examples of protocols using synchronized approaches include S-MAC [48, 47], T-MAC [8], and RMAC [9]. For example, in S-MAC [48] time at each sensor is divided into repeated operational cycles, each further divided into three periods: *Sync*, *Data*, and *Sleep*. Nodes in S-MAC wake up at the start of the Sync period to synchronize clocks with each other. During the Data period, all nodes remain active. If a node has a packet to send to a neighbor node, they exchange Request-to-Send (RTS) and Clear-to-Send (CTS) frames during the Data period, followed by the transmission of the data packet and the return of an Acknowledgment (ACK) frame. Nodes not involved in communication initiated during the Data period return to the sleep state at the start of the Sleep period; other nodes return to the sleep state only after completion of the ACK frame.

Although such approaches save energy, they can add significant latency in packet delivery, since transmission of a packet from one node to a neighbor node must wait until the next time the nodes are active, if the nodes are currently sleeping. Furthermore, forwarding a packet over multiple wireless hops, as is common in WSNs, often requires multiple operational cycles to complete. Several approaches have been proposed to mitigate the additional latency introduced by duty cycling [8, 47, 9], but they are mainly optimized for light traffic loads.

In the first part of this thesis, I present a new MAC protocol, called *Demand Wakeup MAC (DW-MAC)*, that introduces a new low-overhead scheduling algorithm that allows nodes to wake up on demand during the Sleep period of an operational cycle in order to transmit or receive a packet. This demand wakeup adaptively increases effective channel capacity during an operational cycle as traffic load increases, allowing DW-MAC to achieve low delivery latency under a wide range of traffic loads including both unicast and broadcast traffic.

DW-MAC differs from prior work in reducing the additional latency introduced by duty cycling. In DW-MAC, medium access control and scheduling are integrated, in that during a Data period of an operational cycle, the interval of time during which the transmission of an access control frame occupies the medium automatically reserves the proportional interval of time in the following Sleep period for transmitting and receiving a data packet. This integration minimizes scheduling overhead and collisions. Further, by avoiding transmission of data packets in a

Data period, DW-MAC maximizes the number of access control frames that can be exchanged in a Data period, thus increasing the number of data packets that can be exchanged in a complete operational cycle.

The contributions of this first part of my thesis include the following:

- DW-MAC introduces a new low overhead scheduling algorithm that ensures that data transmissions do not collide at their intended receivers.
- I present the design of DW-MAC that wakes up nodes on demand in order to efficiently handle a wide range of traffic load including both unicast and broadcast traffic.
- DW-MAC wakes up a node in a Sleep period only when the node needs to transmit or receive a packet, in order to minimize energy consumption.
- DW-MAC achieves lower latency, higher power efficiency, and higher packet delivery ratio compared to existing schemes.

## 1.4 RI-MAC: A New Asynchronous Duty Cycle MAC Protocol

*Asynchronous* duty cycling MAC protocols, such as B-MAC [35], X-MAC [3], and WiseMAC [10], allow nodes to operate independently, with each node on its own duty cycle schedule. Asynchronous duty cycling protocols typically employ low power listening (LPL), in which, prior to data transmission, a sender transmits a *preamble* lasting at least as long as the sleep period of the receiver. When the re-



ceiver wakes up and detects the preamble, it stays awake to receive the data. These protocols achieve high energy efficiency and remove the synchronization overhead required in synchronous duty cycle approaches. However, they are mainly optimized for light traffic loads, and I found that they become less efficient in latency, power efficiency, and packet delivery ratio as traffic load increases, due to their long preamble transmissions. WiseMAC attempts to improve efficiency by reducing the duration of preamble transmission, but this improvement requires nodes to maintain a fixed wakeup schedule and depends on frequent, regular communication to the same neighbors.

In asynchronous protocols, preamble transmission in LPL-based protocols may occupy the medium for much longer than actual data transmission. Such long preamble transmission from a sender could prevent all neighboring nodes with pending data from transmitting their data. As these nodes have to wait until the medium is not occupied, some of them could experience significant delay. This is often the case under bursty or high traffic load such as due to convergecast [50] and correlated-event workload traffic [17], where multiple sensors that have detected the same event send their reports to the sink node or to a node that does data aggregation [13]. As traffic in a WSN can be quite dynamic, depending on the events being sensed and the sensing application and protocols being used, an ideal WSN MAC protocol should perform well under a wide range of traffic loads, including high loads and bursty traffic.

In the second part of this thesis, I present a new asynchronous duty cycle MAC protocol, called *Receiver Initiated MAC (RI-MAC)*. RI-MAC attempts to minimize the time a sender and its intended receiver occupy the medium for them to find a rendezvous time for exchanging data, while still decoupling the sender and receiver's duty cycle schedules as B-MAC and X-MAC do.

RI-MAC differs from prior work in asynchronous duty cycle MAC protocols in how the sender and receiver reach a rendezvous time. In RI-MAC, the sender remains active and waits silently until the receiver explicitly signifies when to start data transmission by sending a short *beacon* frame. As only beacon and data transmissions occupy the medium in RI-MAC, with no preamble transmissions as in LPL-based protocols, occupancy of the medium is significantly decreased, making room for other nodes to exchange data.

I believe this is the first attempt to apply the idea of receiver-initiated transmission to duty cycle MAC protocols for ad hoc wireless sensor networks. By coordinating neighboring nodes using beacons in RI-MAC, a receiver adaptively increases channel utilization as traffic load increases, allowing RI-MAC to achieve high throughput, packet delivery ratio, and power efficiency under a wide range of traffic loads.

The contributions of this second part of my thesis include the following:

- I present a new asynchronous duty cycle MAC protocol, called *RI-MAC*, employing receiver-initiated transmissions, in order to efficiently and effectively operate over a wide range of traffic loads.
- Due to the receiver-initiated design, RI-MAC not only substantially reduces overhearing, but also achieves lower collision probability and recovery cost than do B-MAC and X-MAC.
- I have implemented RI-MAC in TinyOS and evaluate it in a small testbed network of sensor nodes. We also implemented RI-MAC in the *ns-2* network simulator for evaluations in larger networks.
- RI-MAC significantly improves throughput and packet delivery ratio, especially when there are contending flows such as bursty traffic or transmissions from hidden nodes.
- Even under light traffic loads for which X-MAC is optimized, RI-MAC achieves the same high performance in terms of packet delivery ratio and latency while maintaining comparable power efficiency.

## 1.5 ADB: An Efficient Multihop Broadcast Protocol over Asynchronous Duty Cycling

Existing systems using asynchronous duty cycling do not efficiently support multihop broadcast-based communication. Multihop broadcast is an important net-

work service in many sensor network applications and may be used, for example, in route discovery or in network-wide queries or information dissemination. Supporting a *single-hop* broadcast transmission using asynchronous duty cycling is difficult, due to the independent wakeup times of each of the neighbor nodes of the node originating the broadcast, generally requiring multiple transmissions of the single packet from the originating node [20]. The cost of such redundant transmissions is not well taken into account in existing broadcast protocols (e.g. [33, 46, 34]) designed for always-on networks such as ad hoc networks. With *multihop* broadcast, the problems of single-hop broadcast are amplified, as some neighbor nodes attempt to forward the broadcast while the original transmitting node still attempts to transmit the packet to others of its neighbors, increasing contention for the wireless channel and the possibility of collisions.

In the third part of this thesis, I present the design and evaluation of ADB (*Asynchronous Duty-cycle Broadcasting*), a new protocol for efficient multihop broadcast in wireless sensor networks using asynchronous duty cycling. ADB takes advantage of the fact that nodes wake up at different times to optimize the progress of a multihop broadcast at a finer granularity. Rather than treating the transmission from a node to all of its neighbors as a basic unit of progress for the broadcast, ADB optimizes at the level of transmission to each neighbor individually. As neighbors wake up at different times, a sender with ADB uses unicast to reach each neighbor, so that the sender accurately learns which neighbors have been reached by the

broadcast; this use of unicast also results in an improvement in reliability through the use of ARQ as part of the unicast transmission. At the same time, the sender updates each receiver with up-to-date information on the progress of the broadcast, helping a node to avoid redundant transmissions and to allow delegating transmission for some neighbor to another neighbor with better link quality to it. These optimizations allow a node to sleep as early as possible and avoid transmissions over poor links, leading to lower energy consumption and delivery latency. To achieve these goals, ADB is integrated with the MAC layer in order to exploit information only available at this layer.

The contributions of this third part of my thesis include the following:

- I present the first complete MAC protocol for efficient multihop broadcast in a wireless sensor network using asynchronous duty cycling, incorporating multihop broadcast transmission and MAC-layer details including collision avoidance and recovery and control of radio active state.
- ADB efficiently collects and distributes information on broadcast progress, substantially reducing redundant transmissions, collisions, and energy consumption, by allowing a node to transmit to only a subset of neighbors and to go to sleep as soon as possible.

- ADB substantially reduces delivery latency by avoiding collisions and transmissions over poor links. I prove that ADB achieves close-to-optimal delivery latency with error- and collision-free links.
- I evaluate ADB both through *ns-2* simulation and through implementation in a testbed of MICAz motes using TinyOS. This evaluation shows that ADB substantially outperform multihop broadcast based on X-MAC for power efficiency, packet delivery latency, and delivery ratio.

## 1.6 Thesis Outline

The rest of this thesis is organized as follows. Chapter 2 discusses related work on duty cycle MAC protocols. The next six chapters present the design and evaluation of DW-MAC, RI-MAC, and ADB. In particular, first, Chapter 3 describes the detailed design of DW-MAC, and Chapter 4 presents a comparative evaluation of DW-MAC with representative synchronous duty cycle MAC protocols. Following this, the detailed design of RI-MAC is presented in Chapter 5, and Chapter 6 presents an evaluation of RI-MAC and compares its performance with representative asynchronous duty cycle MAC protocols. Chapter 7 then describes the detailed design of ADB, and Chapter 8 presents the comparative evaluations of ADB with multihop broadcast with representative asynchronous duty cycle MAC protocols. Finally, Chapter 9 presents conclusions.

## Chapter 2

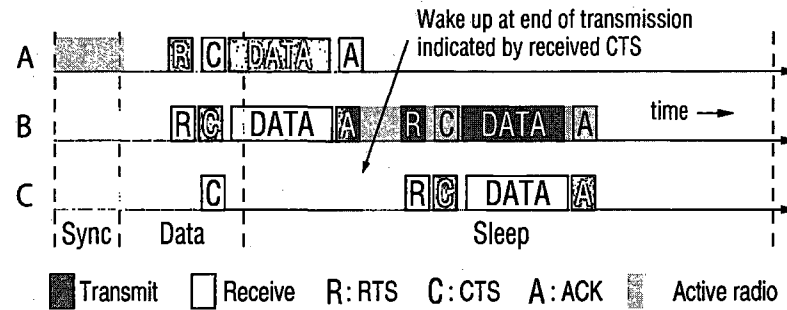
### Related Work

Many existing duty cycle MAC protocols are optimized for light traffic loads. This chapter discusses related work in the area of synchronous duty cycle protocols, compared with DW-MAC; in the area of asynchronous duty cycle protocols, compared with RI-MAC; and in the area of multihop broadcast in a wireless sensor network using asynchronous duty cycling, compared with ADB.

#### 2.1 Synchronous Duty Cycle MAC Protocols

A number of previous approaches to reduce latency in synchronous duty cycle MAC protocols for WSNs have been proposed, although none provides the generality or performance of DW-MAC. I discuss these previous approaches here.

S-MAC [48] was one of the original synchronized duty cycle MAC protocols for WSNs. However, as noted in Section 1.3, this approach can add significant latency in packet delivery, since if the nodes are currently sleeping, transmission of a packet from one node to a neighbor node must wait until the next time the nodes are active. The developers of S-MAC later introduced a modification to S-MAC known as *adaptive listening* [47] to improve its end-to-end delivery latency over multiple hops. With adaptive listening, if a node overhears another node's



**Figure 2.1.** S-MAC with adaptive listening. Node C wakes up at the end of the transmission between node A and B based on the information in the overheard CTS, so that B can forward a packet to C immediately rather than waiting until the next operational cycle.

communication (e.g., the RTS or CTS) during the Data period, it wakes up for a short time when the overheard communication finishes; if this node is the next-hop node along a multihop path, its neighbor can forward the packet immediately to this node rather than waiting for the Data period in the next operational cycle to initiate the forwarding. Figure 2.1 shows an example of the operation of S-MAC with adaptive listening. Node A here sends a data packet to node B, with a next-hop node of C. When node C overhears the CTS from B, it goes to sleep but wakes up again when the ACK from B should have been completed, based on the information in the overheard CTS. Node B can immediately forward the data packet to C at this time.

S-MAC with adaptive listening can deliver a packet up to 2 hops per operational cycle but generally cannot go beyond that within the cycle since the next hop after C (such as some node D) is unlikely to have been awake to overhear the communication from B to C; node C will transmit an RTS to D but will go back



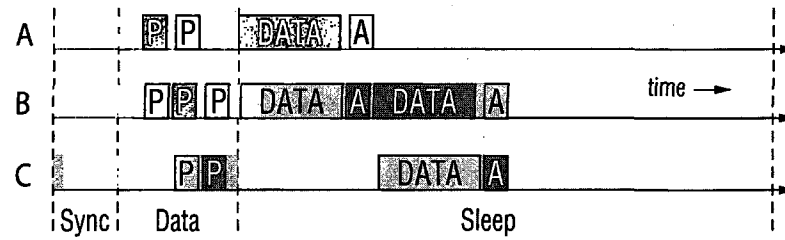
to sleep itself when it fails to receive a CTS in reply from D. The use of adaptive listening can also cause a significant increase in energy consumption, since many neighboring nodes may overhear the RTS or CTS and wake up, whereas only one of them is the next-hop node. Moreover, since a node does not wake up until an overheard communication ends, this node then may not have complete knowledge of the busy state of the wireless medium. For example, the node might have missed hearing an RTS or CTS of another data transmission in the neighborhood; if the node in this case starts transmitting any packet, the packet may cause collisions at other nodes.

Similarly, T-MAC [8] can reduce latency by adaptively changing the ending time of a Data period. Although T-MAC is primarily designed to shorten the Data period when no traffic is around the node, so that nodes can preserve more energy, T-MAC can also extend the Data period to allow multihop forwarding during a single Data period. However, as with S-MAC with adaptive listening, T-MAC can generally deliver a packet over only at most 2 hops within an operational cycle, since nodes further downstream will be unlikely to overhear the upstream communication 2-hops away and thus will not remain awake to receive a forwarded packet; T-MAC may also increase energy consumption, as many nodes other than an intended next-hop node will remain awake.

Several other approaches to reducing latency have been proposed, that make specific assumptions on the communication pattern among nodes or on the other

protocols used in the WSN. For example, DMAC [28] reduces latency only for data gathering communication in which multiple nodes try to send data to a sink node through a unidirectional tree of paths. Likewise, the streamlined wakeup optimization proposed by Cao et al. [4] address only the case in which each sensor node sends data to a sink node (although there may be more than one sink node for the network). For a network of tree topology or ring topology, Lu et al. [29] discuss how to minimize end-to-end latency. The work of Keshavarzian et al. [19] analyzes latency for specific communication and wakeup patterns for communication with the sink node and proposed the multi-parent technique to improve performance under the assumption that nodes at higher levels in the communication tree have more than a single neighbor and thus can have more than a single parent. In contrast to each of these protocols, DW-MAC supports arbitrary communication between any nodes, whether to a sink node or to the other peer nodes such as to facilitate in-network processing of sensor data. The fast path algorithm proposed by Li et al. [26] also supports arbitrary communication patterns but assumes that such “fast paths” are long-lived and are set up through the routing protocol; DW-MAC makes no such assumptions and supports arbitrary communication between nodes at any time without relying on other protocols for assistance.

RMAC [9] represents a different approach to reducing latency in multihop forwarding; an example of the operation of RMAC is illustrated in Figure 2.2. In RMAC, a control frame, called a Pioneer frame (PION), is forwarded over mul-



**Figure 2.2.** Multihop forwarding of a unicast packet in RMAC. P indicates a PION frame that is used for scheduling.

multiple hops (e.g.,  $A \rightarrow B \rightarrow C$ ) during a Data period in order to inform nodes B and C when to wake up during the Sleep period to receive or transmit the corresponding data packet. The number of hops over which RMAC can forward a data packet during an operational cycle is limited by the duration of the Data period but may be set to any value depending on the parameters used. However, as a source node always starts transmitting a data packet at the beginning of a Sleep period (e.g., node A in Figure 2.2), two hidden sources that have succeeded in scheduling through PIONs in a Data period always cause collisions at the beginning of the next Sleep period. In addition, a node woken up due to a previous PION will wake up unnecessarily if the expected data packet cannot arrive due to collisions at previous hops.

The scheduling mechanism in DW-MAC ensures that data transmissions do not collide at their intended receivers, and many other techniques for collision-free transmission in WSNs have been studied by others (e.g., [37, 22]). However, in contrast to these techniques, DW-MAC is a contention-based protocol that integrates medium access control and scheduling seamlessly. Furthermore,

DW-MAC supports not only unicast communication but also broadcast communication. Many other techniques for efficient broadcast communication in wireless sensor networks and in wireless ad hoc networks have been studied (e.g., [13, 46, 34, 36, 39, 34]). However, in contrast to these techniques, a node in DW-MAC wakes up on demand during a Sleep period; scheduling frames during the Data period explicitly coordinate nodes when to wake up during the Sleep period to transmit or receive a packet.

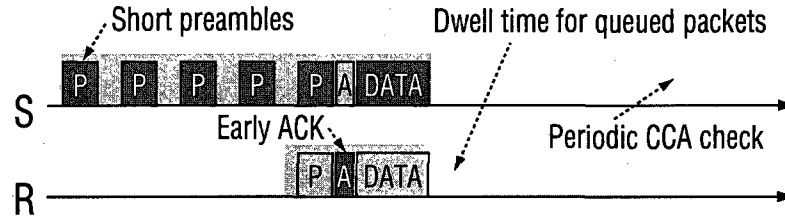
## 2.2 Asynchronous Duty Cycle MAC Protocols

In asynchronous duty cycle MAC protocols, some mechanism is needed in order for the sender and receiver to "rendezvous" in time, so that both are awake for the sender to transmit a packet and the receiver to receive it. B-MAC [35] and X-MAC [3] were among the first asynchronous duty cycle-based protocols and defined the basic structure of the mechanism for solving this problem commonly used in asynchronous duty cycle MAC protocols in sensor networks. In particular, in B-MAC, each node periodically wakes up to check if there is any activity currently on the wireless channel. If so, the node remains active to receive a possible incoming packet. Prior to DATA frame transmission, a sender transmits a long "wakeup signal," called a preamble, which lasts longer than the receiver's sleep interval. This policy ensures that the receiver will wake up at least once during the preamble, allowing each node to wake up or sleep based on its own schedule.

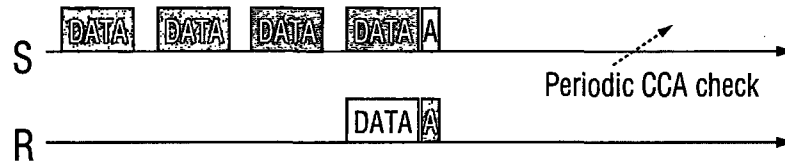
B-MAC is very energy efficient under light traffic because a node spends only a very short period of time in checking channel activity at each scheduled wakeup time. However, a node with B-MAC may wake up and remain awake due to channel activity, only to, in the end, receive one or more DATA frames actually destined for other nodes.

X-MAC solves this overhearing problem in B-MAC by using a *strobed* preamble that consists of sequence of *short preambles* prior to DATA transmission, as illustrated in Figure 2.3. In this and similar figures in this thesis, the period of time during which a node is active is indicated by a solid gray background, frame reception by a node is indicated by black text on the gray background, and frame transmission by a node is indicated by white text on a dark background. The target address is embedded in each short preamble, which not only helps irrelevant nodes to go to sleep immediately but also allows the intended receiver to send an *early ACK* to the sender so that the sender stops preamble transmission and starts transmitting the DATA frame immediately. In this way, X-MAC saves energy by avoiding overhearing while reducing latency almost by half on average. After receiving a DATA frame, a receiver in X-MAC stays awake for a duration equal to the maximum backoff window size to allow queued packets to be transmitted immediately. I refer to this duration as the *dwel time* in the rest of this thesis.

The UPMA (Unified Power Management Architecture for Wireless Sensor Networks) package [20] implemented a variation of X-MAC in TinyOS, in which the



**Figure 2.3.** Operation of X-MAC, including the *strobed preamble* and *early acknowledgment*. During a scheduled wakeup time, a node does a CCA (clear channel assessment) check that is longer than the gap between two short preambles.



**Figure 2.4.** The variation of X-MAC implemented in the UPMA package in TinyOS. The strobed preamble is replaced by a chain of DATA frame transmissions.

DATA frame itself is used as the short preamble, as illustrated in Figure 2.4. This strategy simplifies implementation and helps a sender to determine whether the DATA is successfully delivered from the ACK from the receiver. In the rest of this thesis, I refer to this variation of X-MAC as X-MAC-UPMA.

B-MAC and X-MAC achieve high power efficiency under light traffic load, but their preamble transmissions occupies the wireless medium for a long time until DATA is delivered, making them less efficient in case of contending traffic flows. In contrast, a sender in RI-MAC does not occupy the medium until the intended receiver is ready for receiving, by using *receiver-initiated* transmission. This property allows RI-MAC not only to achieve comparable performance to X-MAC under light traffic load, but to handle a wide range of traffic loads more efficiently.

In addition, the receiver-initiated transmission makes RI-MAC more efficient in detecting collisions and recovering lost DATA frames.

WiseMAC [10] is similar to B-MAC, but a sender in WiseMAC efficiently reduces the length of the wakeup preamble by exploiting the sampling of the schedules of its direct neighbors. In effect, although individual nodes are not synchronized in waking up at the same time as each other, a node does synchronize with its neighbors in learning the wakeup schedules of those neighbors to which it is sending data. To efficiently enable this learning, a node receiving a DATA frame includes in the following ACK frame the remaining time until its next sampling time. With this information, and taking possible clock drifts into account, the sender for its next DATA frame to this receiver estimates when the receiver will wake up next, and starts transmitting its preamble just before then. The resulting shortened preamble greatly helps to save energy and improve channel utilization. However, WiseMAC, as with B-MAC, suffers from the possibility of simultaneous transmissions from hidden nodes, due to the similar preamble sampling techniques they use. In addition, each node with WiseMAC must maintain the same regular wakeup schedule over time, allowing problems such as starvation due to repeated collisions between competing nodes that wake up at the same time over and over again.

The idea of *receiver-initiated* transmission in a MAC protocol is not new, but to the best of my knowledge, RI-MAC represents the first attempt to combine

this idea together with duty cycling in the context of MAC protocols for ad hoc wireless sensor networks, where power efficiency is a major concern. Garcia-Luna-Aceves et al. proposed a receiver-initiated collision-avoidance scheme [16] for general wireless networks, where collision is a major concern but power efficiency is of lesser importance.

Receiver-initiation has previously been applied to sensor networks in the PTIP (Periodic Terminal Initiated Polling) mechanism [11], but only for infrastructure WSNs, where each sensor node is in range of an access point, and access points are assumed to be energy unconstrained. With PTIP, a sensor node periodically wakes up and sends a poll packet to an access point with which the node is associated. If the access point has buffered any packets when the node was sleeping, the access point starts sending those packets to the node upon receiving the poll. The type of WSN assumed for PTIP is very different from a typical ad hoc WSN, where multihop packet delivery can be common and most sensor nodes have limited battery capacity. In addition, the PTIP mechanism was designed only for packets being sent *from* an access point to a sensor node.

Another receiver-initiated mechanism, known as Low Power Probing (LPP), was recently introduced in the Koala system [32]. Koala is designed for reliably downloading bulk data from all sensor nodes, for applications with no real-time requirements. All downloads in Koala are initiated by the gateway or gateways, allowing the nodes to sleep most of the time until the gateway's download initia-



tion. With LPP, each node periodically broadcasts a short probe packet requesting an acknowledgment. If an acknowledgment is received, the node remains active and starts waking up other nodes by acknowledging their probes; otherwise the node goes back to sleep. The LPP mechanism in Koala differs from RI-MAC in both objective and design. In particular, LPP is used in Koala only for waking up all sensor nodes for a download and is not involved in the actual data transfer during a download. As such, features of RI-MAC such as back-to-back data transmission and collision detection and recovery were not discussed in LPP.

*Synchronous* duty cycle MAC protocols (e.g., [48, 8, 9]) and *hybrid* approaches (e.g., [49]) also achieve great energy efficiency in WSNs. The major difference between RI-MAC and these MAC protocols is that RI-MAC does not require any synchronization, thus saving the overhead and complexity of clock synchronization. Even though no node occupies the medium for a long time in these synchronized duty cycle MAC protocols, it is still difficult for contending flows to finish their transmissions within a single cycle. Specifically, the time window during which transmission is allowed is usually very short in these protocols, as neighboring nodes' wakeup times are synchronized. Once one flow acquires the medium, other flows usually have to wait until next cycle, as their receivers might have gone to sleep when the medium becomes idle. Therefore, RI-MAC has the potential to handle contending flows, and thus bursty traffic, more efficiently and effectively.

### 2.3 Broadcast over Asynchronous Duty Cycling

The goal in *multihop* broadcast is for each node in a network to receive a copy of some broadcast packet. Multihop broadcast has been well studied in the context of mobile wireless ad hoc networks (e.g., [33, 46, 34]). For sensor networks, Trickle [25] and DIP [27] are two examples of efficient dissemination protocols that distribute program or data items to all nodes in a network based on gossiping; as long as the network is connected, these protocols achieve perfect reliability. Other protocols, such as RBP [40], target multihop broadcast for services such as routing and resource discovery, needing only propagation of small messages with high probability and low latency. RBP extends flooding-based approaches by allowing some nodes to adaptively rebroadcast a packet more than once based on the local density of the network, thus greatly improving end-to-end reliability without significantly increasing overhead.

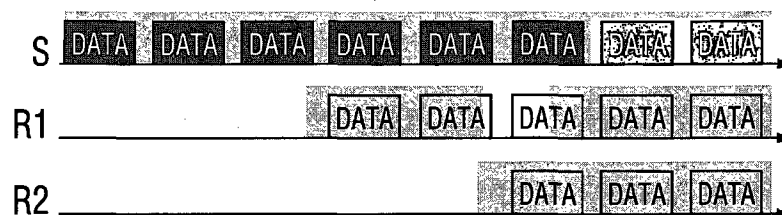
ADB differs from these protocols in that it is optimized for use with *asynchronous duty cycling* and is tightly integrated with the MAC protocol in order to exploit opportunities specific to asynchronous duty cycling. In the protocols above, the transmission of a broadcast from one node to all neighbors is treated as a single, basic unit of operation. Since the neighbors wake up at different times, this basic unit can extend over a long time. ADB, instead, optimizes the progress of the broadcast at the level of transmission from the node to each neighbor individually. Optimization at such a finer granularity avoids redundant transmissions,

allows following hops to quickly begin forwarding a multihop broadcast, and enables nodes to go to sleep again as soon as possible. In this way, ADB achieves near optimal latency, high energy efficiency, and high delivery ratio, making ADB efficient in distributing small messages for services such as routing and resource discovery when asynchronous duty cycling is used.

To my best knowledge, the only prior work that optimizes multihop broadcast over asynchronous duty-cycling in wireless sensor networks is that of Wang et al. [44, 45]. They present a centralized algorithm, transforming the problem into a shortest-path problem in a time-coverage graph, and also present two similar distributed algorithms that do not require this centralized coordination. However, they treat the problem as a transmission scheduling problem, not as a MAC problem, and also assume that the future wakeup schedules of 2-hop neighbors can be known in advance. Their work thus simplifies many aspects necessary for a complete MAC protocol. For example, they divide time into fixed slots, assuming that the active and sleeping periods of all nodes are integer multiples of these slots and that in each slot, an active node can either receive or forward one packet only. Their evaluations were based on simulations, but no information was given on details such as the mechanisms or overhead for learning the wakeup schedules of 2-hop neighbors or on how the wireless channel was simulated, making their results difficult to interpret. Moreover, none of their algorithms have been implemented and evaluated on real hardware.

In contrast, ADB does not depend on learning the future wakeup schedules of 2-hop neighbor nodes. ADB is also seamlessly integrated into a complete asynchronous duty-cycle MAC protocol, allowing it to process both unicast and broadcast traffic efficiently in the same network. In this thesis, I also evaluate ADB through detailed simulations using *ns-2* and through experiments in a real implementation in a testbed of MICAz motes using TinyOS.

Most work on asynchronous duty-cycling MAC protocols for sensor networks has focused on the *unicast* problem, and few of these protocols have clearly defined methods even for *single-hop* broadcast or studied broadcast performance. In single-hop broadcast, a node delivers a broadcast packet to all of its direct neighbors, which is then often used as a building block for multihop broadcast when needed. B-MAC [35] can support single-hop broadcast in the same way as unicast, since the preamble transmission, extended over an entire sleep period, gives all of the transmitting node's neighbors a chance to detect the preamble and remain awake for the DATA packet. X-MAC [3] substantially improves B-MAC's performance for unicast, but broadcast support is not clearly discussed in that paper. This gap is filled by the X-MAC implementation in the UPMA package [20, 43] of TinyOS, where a transmitter repeatedly transmits copies of a DATA packet over a duty cycle interval, as illustrated in Figure 2.5. In the rest of this thesis, I refer to this implementation of X-MAC as *X-MAC-UPMA*.



**Figure 2.5.** Broadcast support in X-MAC in the UPMA package of TinyOS. A transmitter *S* repeatedly transmits copies of a broadcast packet (DATA frame) over a duty cycle interval, during which each neighbor (node *R1* and *R2*) wakes up at least once and thus has an opportunity to receive the packet.

With X-MAC-UPMA, a transmitter must repeatedly transmit the packet over an entire duty cycle, even if all its neighbors have already received it. These repeated transmissions unnecessarily consume energy at the transmitter and delay forwarding from this node's neighbors for a multihop broadcast. In addition, the neighbors remain awake even after receiving the packet the first time, further wasting energy; a possible improvement would be to let a neighbor go to sleep once a broadcast packet is received, but this would require careful consideration as to when to turn a node on again later for forwarding the broadcast. In addition, if two transmitters, hidden to each other, transmit at the same time, their transmissions will produce repeated collisions at other receivers over a long period of time; after waking up, if a node cannot receive a valid packet after a short timeout (100 ms is the default value in X-MAC-UPMA), it will go to sleep and thus never receive the broadcast packet.

ADB avoids the problems faced by X-MAC-UPMA by efficiently distributing information on the progress of each broadcast, allowing a node to go to sleep im-

mediately if no more neighbors need to be reached. ADB also uses this progress information to coordinate neighbors of a node in transmitting a packet to the node, so that collisions are significantly reduced. ADB is designed to be integrated with a unicast MAC that does not occupy the medium for a long time, in order to minimize delays before forwarding a broadcast. The effort in delivering a broadcast packet to a neighbor is adjusted based on link quality, rather than the fixed number of transmissions in X-MAC-UPMA.

## Chapter 3

# DW-MAC Design

In this chapter, I present the design of DW-MAC, a synchronous duty cycle MAC protocol that efficiently handles a wide range of traffic load including both unicast and broadcast traffic.

### 3.1 Overview

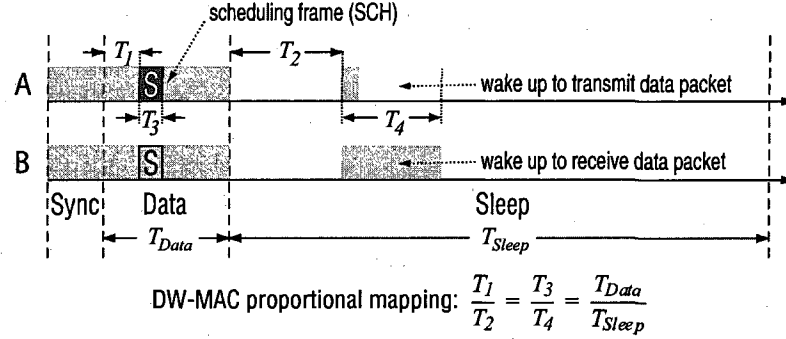
DW-MAC is a synchronized duty cycle MAC protocol, in which each cycle is divided into three periods: Sync, Data, and Sleep (Figure 3.1). I denote the duration of each period by  $T_{Sync}$ ,  $T_{Data}$ , and  $T_{Sleep}$ , respectively. Similar to prior work, DW-MAC assumes that a separate protocol (e.g., [12, 15]) is used to synchronize the clocks in sensor nodes with required precision. The basic concept of DW-MAC is to wake up nodes on demand during the Sleep period of a cycle in order to transmit or receive a packet. This demand wakeup adaptively increases effective channel capacity during a cycle as traffic load increases, allowing DW-MAC to achieve low delivery latency under a wide range of traffic loads including both unicast and broadcast traffic.

DW-MAC is unique in the way it schedules nodes to wake up during the Sleep period of a cycle. In DW-MAC, medium access control and scheduling are fully

integrated. In a Data period, a node with pending data contends for channel access using a CSMA/CA protocol as in IEEE 802.11. DW-MAC, however, replaces RTS/CTS with a special frame called a *scheduling frame* (SCH). The interval of time during which the transmission of a SCH occupies the wireless medium automatically and uniquely reserves the proportional interval of time in the following Sleep period for transmitting and receiving the pending data packet. Essentially, DW-MAC sets up a one-to-one *mapping* between a Data period and the following Sleep period. An SCH carries no timing information, and the transmission of an SCH simply replaces that of RTS/CTS for medium access control. In this way, DW-MAC minimizes scheduling overhead. As in an RTS, an SCH contains the destination address so this SCH wakes up only the intended receiver, minimizes energy consumption due to unnecessary wake-ups. Furthermore, this integration ensures that data transmissions do not collide at their intended receivers as discussed below.

Figure 3.1 shows an overview of scheduling in DW-MAC based on this one-to-one mapping between a Data period and the following Sleep period. In this example, node A wants to transmit a data packet to node B. Node A first contends for channel access and transmits an SCH during the Data period. Suppose transmission of the SCH starts  $T_1$  time units after the beginning of the Data period. Based on  $T_1$  and the duration of the SCH transmission,  $T_3$ , both nodes A and B will schedule their wakeup time to  $T_2$  from the beginning of the following Sleep period,





**Figure 3.1.** Overview of scheduling in DW-MAC.

and will agree on a maximum wakeup duration of  $T_4$ , based on the ratio between  $T_{Data}$  and  $T_{Sleep}$ , as shown in the figure. If the packet to be transmitted is a unicast packet, node B will return a confirmation SCH frame (not in the figure) SIFS delay after receiving the request SCH from A; if the packet is a broadcast packet, node B takes no further action. When nodes A and B both wake up at the agreed time, node A transmits the actual data packet, which can be either broadcast or unicast. In case of unicast packet, node B acknowledges the successful receipt of the packet with an ACK. Although I show the scheduling for only one pair of nodes in this example, DW-MAC allows multiple contending nodes to exchange SCH frames with their intended receivers during a Data period, so that multiple data transmissions can happen in the following Sleep period.

### 3.2 Mapping Function for Scheduling

As previously explained, DW-MAC exploits a contention based Data period in order to schedule actual data transmissions during the subsequent Sleep period.

To avoid collisions during the Sleep period, a sender must coordinate with its intended receiver to find a period of time in the Sleep period during which the neighboring nodes of both are idle. The challenge in designing such a protocol is twofold:

- minimize message exchanges between a sender, the intended receiver, and their respective neighbors for schedule negotiation; and
- minimize the size of a scheduling frame, e.g., avoid carrying timing information in a scheduling frame.

DW-MAC meets these goals by employing a one-to-one *proportional mapping* function between time during a Data period and time during the subsequent Sleep period. With this mapping function, DW-MAC schedules data transmissions without exchanging any timing information. Let  $T_i^D$  be the time difference between a specific time instance  $t_i$  in a Data period and the beginning of that Data period, and let  $T_i^S$  be the time difference between the start of the subsequent Sleep period and the corresponding mapped time instance during the Sleep period. Accordingly, DW-MAC defines the following mapping function:

$$T_i^S = T_i^D \cdot \frac{T_{Sleep}}{T_{Data}} \quad (3.1)$$

By mapping each time instant in a Data period into the subsequent Sleep period, the mapping function scales the time based on the ratio between  $T_{Sleep}$  and  $T_{Data}$ , and hence a time interval of  $T_1$  time units in the Data period will be mapped

into  $T_1 \cdot \frac{T_{Sleep}}{T_{Data}}$  time units during the Sleep period. With this mapping function, a sender and its intended receiver(s) can uniquely determine the starting point for data packet transmission in a Sleep period from the starting time of the corresponding SCH transmission during the previous Data period, without including even a single bit of timing information in the SCH. In addition, the difference between the mapped beginning and end of the SCH transmission determines the maximum data transmission time. Furthermore, this proportional mapping between the Data period and the Sleep period creates an important property of DW-MAC, defined by the following theorem:

**Theorem 1** *Any receiver that wakes up in a Sleep period is never in range of two simultaneous data packet transmissions, i.e., data transmissions by nodes that wake up during the Sleep period do not collide at their intended receivers.*

*Proof:* By contradiction. Assume that two data transmissions could collide. In order for data transmissions to collide at a node, they must overlap with each other. Therefore, the respective SCHs should also overlap at that node during the previous Data period. In this case, that node could not have decoded any SCH and thus would not wake up during the Sleep period, which contradicts the assumption. ■

This theorem only relates to collisions between data packets. A collision between a data packet and an ACK is still possible. This collision could be eas-

ily avoided by delaying the ACK to the mapped start time of the confirmation SCH sent from the node that transmits this ACK, but such data-ACK collision is a rare event that would require very specific topology and timing setup between the nodes involved in the collision. In my implementation, I require a receiver to immediately acknowledge a data packet, so that both the sender and the receiver can go to sleep immediately and avoid wasting energy waiting for the delayed ACK.

### 3.3 Scheduling Frame (SCH)

Besides the standard fields included in an RTS/CTS, such as sender and receiver addresses, and duration of the transmission, an SCH also includes some cross-layer information. For a broadcast packet, SCH includes the network layer address of its source and its sequence number. This information helps a node to decide whether the incoming broadcast packet has been received before or not, in order to avoid waking up to receive copies of the same packet multiple times. For a unicast packet, an SCH includes the network layer address of its final destination. This cross-layer information *enables a node to set up a schedule to the next hop neighbor before receiving the actual data packet*, as discussed in Section 3.5.

An SCH serves either as a scheduling request or a scheduling confirmation. For a multihop forwarding, an intermediate node sends a single SCH serving both purposes: first, it confirms the received SCH from the upstream node, and second, it schedules the forwarding of the packet to the next downstream node. In order to

distinguish between the two uses of SCH, an SCH includes two bits in the header to indicate which role(s) it is playing. Since an access control frame in S-MAC is 10 bytes [47] and the address of a node usually takes two bytes [24], I use 14 bytes as the size for an SCH to hold the additional cross-layer information in the DW-MAC simulations presented in Chapter 4.

### 3.4 Broadcast and Unicast in DW-MAC

DW-MAC supports two modes of operation: unicast traffic and broadcast traffic. An example of broadcasting of a data packet in DW-MAC is illustrated in Figure 3.2. After successfully transmitting an SCH, a sender (node A) starts broadcasting the packet at the time calculated based on the mapping function (Equation 3.1),  $T_1^S$  in this example. Based on the source address and the sequence number of the packet which are included in the SCH, each receiver decides whether it has received the packet before. In case the packet has already been received by this node, the SCH is ignored. Otherwise, the receiver registers a wakeup time for receiving the incoming packet. In this example, node B estimates  $T_1^D$  based on when the SCH is received and its transmission delay. Using the mapping function, node B sets up a timer to wake up at  $T_1^S$  after the beginning of the Sleep period. Note that node B can contend for another SCH transmission and schedule the rebroadcast of the incoming packet even though it does not yet have the packet.

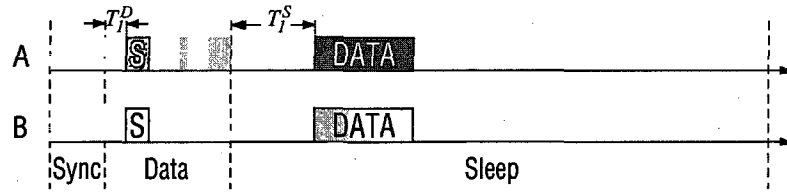


Figure 3.2. Broadcast in DW-MAC.

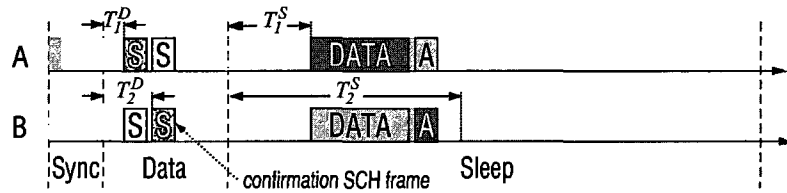
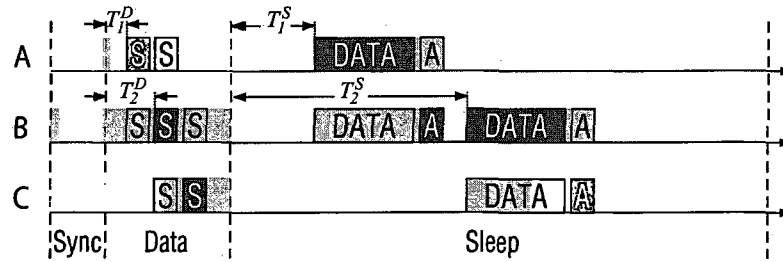


Figure 3.3. Unicast in DW-MAC.

For unicast traffic in DW-MAC, a sender still transmits an SCH prior to data transmission as it does for a broadcast packet. However, DW-MAC requires the intended receiver of the data packet to send back another SCH, SIFS after the receipt, to confirm the receipt of the first SCH. If the confirmation is received in time, the sender sets up a wakeup time for data transmission. Otherwise, the sender attempts to transmit another SCH later as the retransmission of an RTS. Figure 3.3 illustrates how node A transmits a unicast packet to node B.

### 3.5 Optimized Multihop Forwarding

DW-MAC optimizes the timing of transmitting SCH frames in order to maximize the number of hops either a unicast or a broadcast packet can traverse in a cycle. Figure 3.4 illustrates the optimized multihop forwarding of a unicast packet. In this example, node A first sends an SCH to node B in order to set up a schedule for

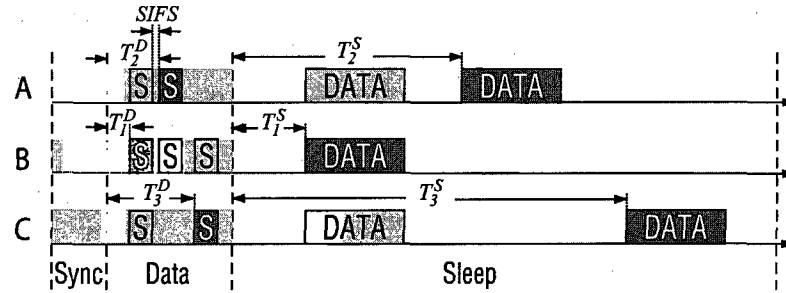


**Figure 3.4.** Optimized multihop forwarding of a unicast packet. Node B sends an SCH to wake up node C at the time indicated by  $T_2^S$  and confirms the SCH received from node A.

a pending packet with final destination of node C. The SCH contains the network layer address of the final destination C. Upon receiving this SCH, node B calculates the wakeup time  $T_1^S$  and checks the network layer destination in the SCH. Based on information from the routing layer (e.g., as is done in RMAC), node B will find that C is the next hop for the incoming packet. In this case, node B sends another SCH, SIFS after receiving the SCH from A. This SCH not only confirms the SCH just received from A but also wakes up C at the time indicated by  $T_2^S$  (both bits in the header of the SCH are set, indicating that this SCH is serving both roles). In this way, a unicast packet can traverse  $x$  hops by only using  $x + 1$  SCH frames in a cycle, and the gap between two consecutive SCHs is just SIFS, which suggests more SCH exchanges in a data period and more data transmissions in a cycle. Multihop forwarding in a similar manner is also supported by RMAC. However, DW-MAC dramatically reduce the collisions experienced by RMAC due to schedule conflicts, as DW-MAC ensures that two data frame transmissions will not collide with each other.

DW-MAC can also speed the propagation of a broadcast packet when some neighbor information is available. The main idea is to favor the rebroadcast of a broadcast packet along some path in order to shorten delays between rebroadcasts and to improve spatial reuse. In the SCH a node transmits, the node specifies an *immediate forwarder* that rebroadcasts the SCH SIFS after receiving the SCH. In the example illustrated in Figure 3.5, node A is specified as the immediate forwarder by node B. Any node other than the immediate forwarder (node C) backoffs before rebroadcasting the SCH. Node A and C will specify an immediate forwarder other than B in the SCH they rebroadcast respectively. This optimized forwarding makes it possible for an SCH and thus the corresponding data packet to reach further nodes in a single cycle than having all rebroadcasting nodes compete for the medium equally. Although this reduced randomness could increase collision probability, the improved spatial reuse usually offsets this increase or even lowers total collision probability as shown in the experiments in Section 4. Many criteria can be used for choosing an immediate forwarder, such as location, degree, or the number of children nodes of a neighbor. In DW-MAC implementation, this optimized forwarding is used when a broadcast tree of a WSN is available and a node knows its children nodes' height (the number of edges on the longest downward path to a leaf). For an SCH to be rebroadcast, if the SCH is received from the parent node, a node chooses the child with greatest height as the immediate forwarder. If





**Figure 3.5.** Optimized multihop forwarding of a broadcast packet. Node B specifies node A as the immediate forwarder, which rebroadcasts an SCH SIFS after receiving that SCH from A. Node C rebroadcasts the SCH when its backoff counter expires.

this SCH is received from one child node, the parent node of the SCH receiver is chosen as the immediate forwarder.

### 3.6 Implementation Issues

I chose to put a packet size limit in my implementation of DW-MAC, although DW-MAC can support larger packet sizes either by increasing the size of SCH frames or by using variable SCH frame sizes for variable packet sizes. This design choice was based on the fact that popular sensor radios usually have a packet size limit. For example, CC1000 in Mica2 [24] and CC2420 in MicaZ [31] have a packet size limit of 256 and 128 bytes, respectively. With a low duty cycle configuration such as is common (and as I used in my simulations), a small SCH can be mapped to a period long enough for these packet limits.

Wakeup times calculated at the sender and receiver(s) are not necessarily perfectly aligned due to propagation delay and processing time. However DW-MAC

does not require an accurate estimation of the start of a transmission. DW-MAC needs only to ensure that a receiver wakes up early enough during a Sleep period so that an incoming packet is not missed, which can be ensured by waking a receiver  $\epsilon$  seconds before an estimated arrival time.

## Chapter 4

### Evaluation of DW-MAC

I evaluate DW-MAC using version 2.29 of the *ns-2* simulator both under unicast and broadcast traffic. In the simulation configuration, each sensor node has a single omni-directional antenna, using the standard *ns-2* combined free space and two-ray ground reflection radio propagation model. Under unicast traffic, I compare DW-MAC against S-MAC, S-MAC with adaptive listening, and RMAC. Under broadcast traffic, because broadcast is not supported in S-MAC with adaptive listening or in RMAC, I compare DW-MAC only against S-MAC.

Table 4.1 summarizes the key parameters used in my simulations. Except for the parameters on radio power consumption that are typical values for Mica2 radios (CC1000) [49], I use the default settings in the standard S-MAC simulation module distributed with the *ns-2.29* package, also used for evaluations of S-MAC and RMAC in previous work [9]. The transition time of the CC1000 radio between sleep and active states is around 2.47 ms [6], but the state transition power is not available in the data sheet. Although the state transition power is normally much lower than Tx or Rx power, I set the state transition power to the same value as for Tx power in order not to favor DW-MAC, which requires more state transitions than S-MAC in this aspect; I observed similar trends in the results even if

**Table 4.1.** Networking Parameters

Bandwidth	20 Kbps	Channel Encoding Ratio	2
Tx Power	31.2 mW	Tx Range	250 m
Rx Power	22.2 mW	Carrier Sensing Range	550 m
Idle Power	22.2 mW	Contention Window ( <i>CW</i> )	64 ms
Sleep Power	3 $\mu$ W	Size of RTS/CTS/ACK	10 B
SIFS	5 ms	Size of PION/SCTL	14 B
DIFS	10 ms	State Transition Power	31.2 mW
Retry Limit	5	State Transition Time	2.47 ms

the state transition power is 0. In evaluating power efficiency, I focus on energy consumed by radios but ignore energy consumed by other components such as CPU and memory [38]. The transmission range and the carrier sensing range are modeled after the 914MHz Lucent WaveLAN DSSS radio interface, which is not typical for a sensor node, but I use these parameters to make the results comparable to those reported in previous work, and since measurements have shown that similar proportions of the carrier sensing range to the transmission range are also observed in some state-of-art sensor nodes [2].

In the simulations, the duty cycle is kept constant at 5% for S-MAC, RMAC, and DW-MAC. The durations for the Sync, Data, and Sleep periods are shown in Table 4.2. For generating comparable results with the earlier evaluation of RMAC [9], I use the same duty cycle-related parameters for DW-MAC as were used in that evaluation.

To simplify my evaluations, routing traffic is not included in the simulations; I assume that there is a routing protocol deployed to provide the shortest path

**Table 4.2.** Duty Cycle Configuration

	$T_{Sync}$ (ms)	$T_{Data}$ (ms)	$T_{Sleep}$ (ms)	$T_{cycle}$ (ms)
S-MAC	55.2	104.0	3025.8	3185.0
RMAC	55.2	168.0	4241.8	4465.0
DW-MAC	55.2	168.0	4241.8	4465.0

between any two nodes. I also ensure that every network used in the simulations is a connected network. In addition, I do not include any synchronization traffic and assume all the nodes in the network have already been synchronized to use a single wake-up and sleep schedule.

For simulations under unicast traffic, each run contains unicast packets toward a sink node that are triggered by a series of 500 events, and each average value is calculated from the results of 10 random runs. For simulations under broadcast traffic, each run contains 500 broadcast packets generated by a sink node, and each average value is calculated from the results of 30 random runs. Confidence intervals of the average values are not shown because even 99% confidence intervals are so close to average values that they overlap with the data point markers.

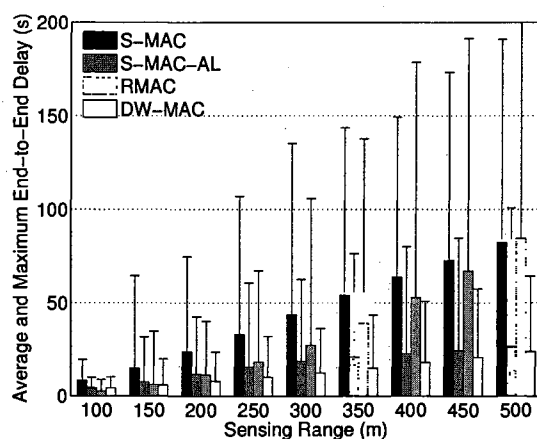
#### 4.1 Evaluation under Unicast Traffic

I compare DW-MAC with S-MAC, S-MAC with adaptive listening (shown as S-MAC-AL in all figures), and RMAC both in a 49-node ( $7 \times 7$ ) grid network and in random networks.

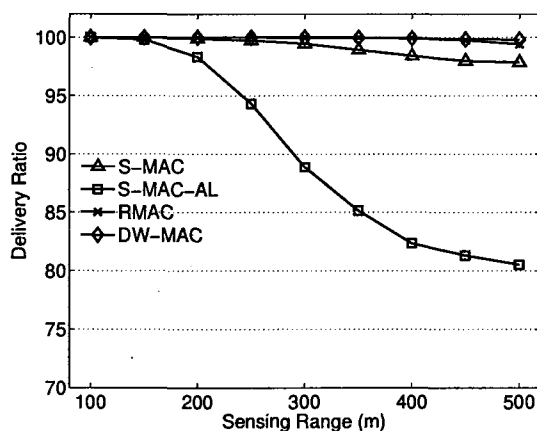
**Table 4.3.** Average number of packets generated for each event under different sensing ranges in the 49-node grid network

Range (m)	100	150	200	250	300	350	400	450	500
Packets	0.8	1.7	3.1	4.6	6.4	8.4	10.6	12.9	15.2

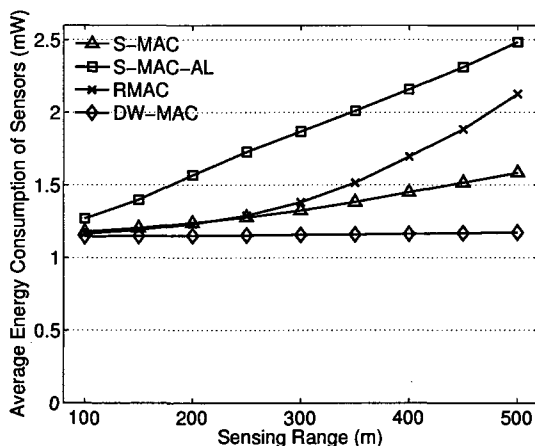
In the grid network, each node is 200 meters from its neighbors, and the sink node is at the center. Based on a correlated-event workload [17], I introduce a *Random Correlated-Event (RCE)* traffic model to simulate the impulse traffic triggered by spatially-correlated events commonly observed in detection and tracking applications. RCE picks a random  $(x, y)$  location for each event. If every node has a sensing range  $R$ , only nodes that are within the circle centered at  $(x, y)$  with radius  $R$  generate packets to report this event. By adjusting the sensing range  $R$ , different degrees of workload in a network can be simulated. In the experiments, a new event is generated once every 200 seconds, and each node having sensed the event sends one packet to the sink node. The value  $R$  is varied from 100 meters to 500 meters; the average number of packets generated per event is listed in Table 4.3. Note that an event triggers at most one packet when  $R$  is 100 meters. The lengths of paths traversed by these packets range from 1 to 6 hops, and the average is 3.05. In this way, the simulations explore how efficiently S-MAC, S-MAC with adaptive listening, RMAC, and DW-MAC handle different degrees of traffic load. The performance of these protocols for unicast traffic in the 49-node grid network scenarios is shown in Figure 4.1.



(a) Average and maximum end-to-end delay versus sensing range.



(b) Delivery ratio versus sensing range.



(c) Average energy consumption of sensors versus sensing range.

**Figure 4.1.** Performance for unicast traffic in 49-node grid network scenarios.

Figure 4.1(a) shows the average and maximum end-to-end latency of packets in the RCE model as the sensing range (and thus traffic load) increases. DW-MAC has a much smaller rate of increase than do S-MAC and RMAC. When there are around 15 packets generated for each event with the 500-meter sensing range, DW-MAC reduces average end-to-end delay by around 70% compared to S-MAC and RMAC. DW-MAC outperforms S-MAC because DW-MAC allows more transmissions in a cycle by using the Sleep period for actual data transmissions. RMAC experiences more delay than DW-MAC as workload increases, because of increased packet collisions caused by scheduling conflicts. It is the retransmission effort to recover these collided packets that results in larger end-to-end delay. When the sensing range is 500 meters, the maximum end-to-end delay with RMAC is 374.95 seconds, which is off the top of the graph. This extreme delay occurs when a packet generated for one event failed to reach the sink before the next event happened. Under the light traffic with the 100-meter sensing range, DW-MAC shows slightly larger delay than RMAC, due to the time that a received data packet is forwarded to the next hop in multihop forwarding. In RMAC, a data packet is forwarded immediately, whereas in DW-MAC, forwarding starts at a later time determined by the corresponding SCH frame. This extra delay experienced by DW-MAC, however, is less than the duration of a Sleep period. S-MAC with adaptive listening shows slightly larger delay compared to DW-MAC. This low delay achieved by



adaptive listening, however, comes at the cost of lower packet delivery ratio and increased energy consumption as shown next.

The packet delivery ratios corresponding to Figure 4.1(a) are shown in Figure 4.1(b). DW-MAC maintains close to 100% packet delivery ratio and outperforms the other protocols across all sensing ranges. The delivery ratio with S-MAC with adaptive listening drops quickly, since with larger the sensing ranges, more collisions are caused by transmissions from hidden nodes, as discussed in Section 2.1; in addition, a node may transmit a packet when its intended receiver is in sleep state, further decreasing packet delivery ratio. DW-MAC and RMAC outperform S-MAC mainly for two reasons. First, they only transmit short scheduling frames during a Data period, avoiding collisions between a control frame and a long data frame. Second, a node does more retransmission attempts for a data packet in DW-MAC and RMAC. Specifically, a scheduling frame sent by an intermediate node in multihop forwarding serves both as RTS and as CTS; even if this frame fails to reach the next-hop neighbor, the intermediate node does not increase its retry count, as the node has not received the corresponding data packet yet, although the node has attempted to reserve the medium to forward the incoming data packet once. Even with such extra retransmission attempts, the delivery ratio of RMAC drops more quickly than that of DW-MAC beyond a 400-meter sensing range, as retransmissions are not enough to recover the increased collisions due to RMAC's scheduling conflicts.

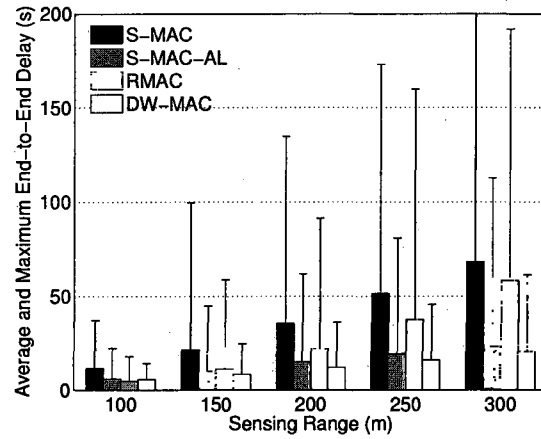
Figure 4.1(c) shows the average energy consumption of nodes versus sensing ranges in the 49-node grid network scenarios. Under light workload, when the sensing range is 100 meters, all four MAC protocols show almost the same power consumption, but when traffic load increases as the sensing range gets larger, average energy consumption in all protocols except DW-MAC increases quickly (energy consumption for DW-MAC does increase, but increases very slowly). When the sensing range is 500 meters, DW-MAC consumes less than 50% of the energy consumed by S-MAC with adaptive listening to achieve even lower packet delivery latency.

In order to understand how efficiently these protocols handle concurrent traffic, I use REC traffic model to generate 2 random events at a time in the grid network. As the two random events happen at the same time, it is likely that propagation of the packets triggered by them overlap in the network.

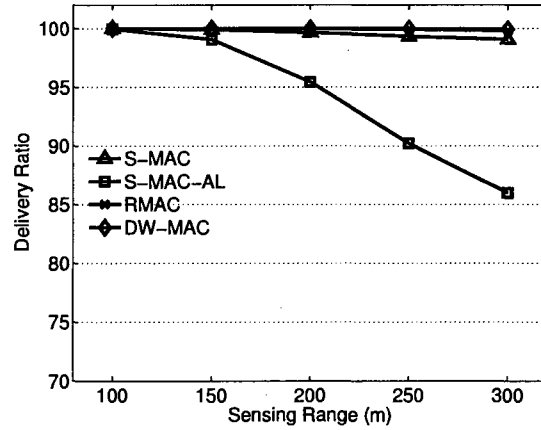
Figure 4.2 compares end-to-end delays, delivery ratios and energy consumption with S-MAC, S-MAC with adaptive listening, RMAC, and DW-MAC. In this set of simulation, I only vary the range of the REC traffic model from 100 to 300 meters, as the maximum end-to-end delay of S-MAC with 300-meter range is already greater than 200 seconds, suggesting that packets for an event are still in propagation when those for the events of next round are generated. Due to increased traffic loads, each protocol shows increased end-to-end delays, delivery ratios and energy consumption compared with the results in Figure 4.1. The trends, however,

agree well with those in Figure 4.1: DW-MAC still outperforms the rest as traffic load increases.

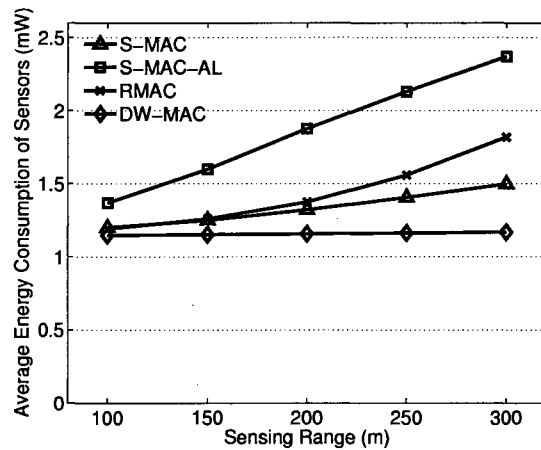
I also compare S-MAC, S-MAC with adaptive listening, RMAC, and DW-MAC in 100 random networks, each with 50 nodes randomly located in a  $1000\text{ m} \times 1000\text{ m}$  area. For each network, one random node is chosen as the sink, and the RCE model with 250-meter sensing range is used to generate 500 events, once every 200 seconds. One simulation run was conducted for each network, and 3845 packets were generated in each run on average. The results are plotted in Figure 4.3. For the same reasons discussed above, DW-MAC outperforms the other three protocols in delivery latency, delivery ratio, and energy consumption. Figure 4.3(a) shows the CDF of end-to-end latency for all packets in all 100 runs. Average end-to-end latency with S-MAC, S-MAC with adaptive listening, RMAC, and DW-MAC are 61.8%, 21.6%, 36.7%, and 15.7%, respectively. Although adaptive listening greatly reduces end-to-end latency for S-MAC, this gain is at the cost of lower delivery ratio and more energy consumption. Figure 4.3(b) shows the CDF of delivery ratios in these 100 runs. The average delivery ratios of S-MAC, S-MAC with adaptive listening, RMAC, and DW-MAC are 99.63%, 95.03%, 99.99%, and 99.99%, respectively. The average energy consumptions of the sensors are plotted in Figure 4.3(c), where the average values with S-MAC, S-MAC with adaptive listening, RMAC, and DW-MAC are 1.386, 2.666, 1.724, and 1.163 mW, respectively.



(a) Average and maximum end-to-end delay versus sensing range.



(b) Delivery ratio versus sensing range.



(c) Average energy consumption of sensors versus sensing range.

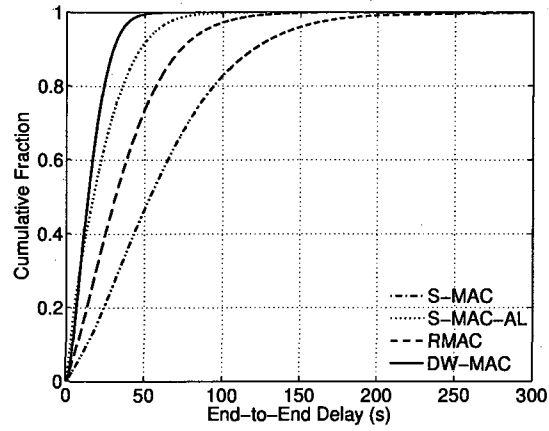
**Figure 4.2.** Performance for unicast traffic in 49-node grid network scenarios, with 2 random events generated at a time.

The trends observed in these random networks are consistent with those observed in the 49-node grid network.

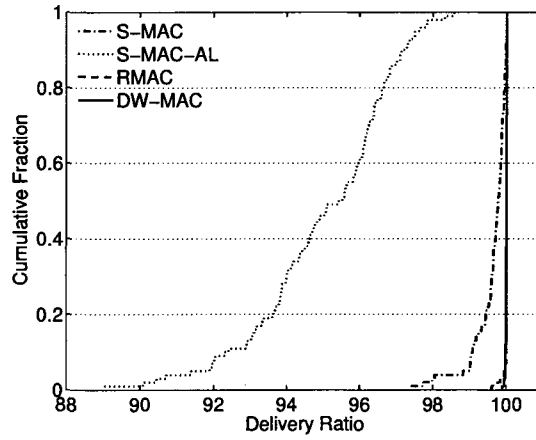
## 4.2 Evaluation under Broadcast Traffic

I compared DW-MAC with S-MAC, both in regular grid networks and in random networks, under broadcast traffic. For broadcast in S-MAC, a broadcast packet is transmitted during a Data period without using RTS/CTS [48].

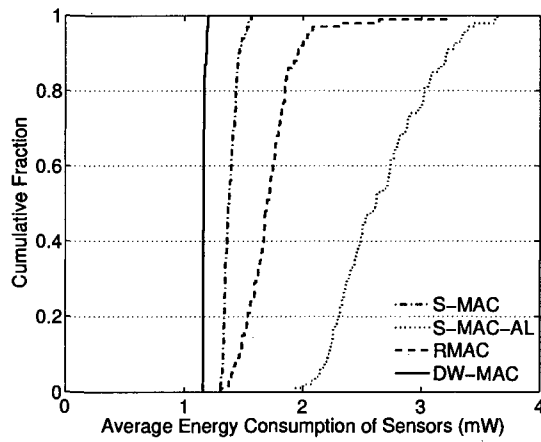
In the grid network, the sink node is at the center, and each node is 200 meters from its neighbors. The grid size is varied from  $3 \times 3$  (9 nodes) to  $11 \times 11$  (121 nodes). The sink node generates a broadcast packet once every 100 seconds so that transmissions for one packet complete before the next packet is generated. I evaluate DW-MAC under two categories of broadcast protocols: simple flooding (all nodes that have received a broadcast packet rebroadcast it exactly once, indicated by "ALL") and Connected Dominating Set (CDS) based flooding (only nodes in a CDS that have received a broadcast packet rebroadcast it exactly once, indicated by "CDS"). The CDS is formed by the algorithm by Gandhi et al. [14], with a slight modification to always include the sink node in the CDS; the results for the optimized multihop forwarding for broadcast traffic are indicated by "DW-MAC CDS-MH." Note that this CDS algorithm is designed to minimize broadcast latency, and the resulting CDS is not necessarily a minimum CDS.



(a) CDF of end-to-end delays



(b) CDF of delivery ratios



(c) CDF of average energy consumption

**Figure 4.3.** Performance for random correlated-event traffic in 50-node networks with sensing range of 250 m.

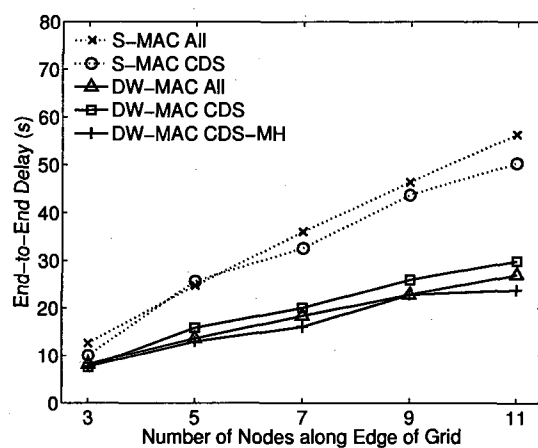
The simulation results in grid networks are shown in Figure 4.4. Figure 4.4(a) shows end-to-end latency (the time it takes for the last node to receive a given broadcast packet) with S-MAC and DW-MAC. DW-MAC reduces the end-to-end latency by around 50% over those with S-MAC, as DW-MAC allows more contending nodes to finish their transmissions in each cycle. When optimized multihop forwarding is enabled, DW-MAC further reduces end-to-end latency, as it increases spatial reuse and reduces delays before a rebroadcast. An interesting trend is that CDS-based flooding shows lower latencies than simple flooding with S-MAC but shows the reverse with DW-MAC. The reason lies in the combination of CDS formation, grid topologies, and duty cycle configuration in the simulation. First, a CDS formed is not necessarily an MCDS. Second, a CDS node may experience more latency before rebroadcasting a packet than does a non-CDS node with DW-MAC, due to defers caused by undecodable frames. When a node fails to decode a received packet, it defers for some time (such as EIFS in IEEE 802.11) to avoid interrupting ongoing transmission. Since this defer is much shorter than a Sleep period in the simulations, all neighboring nodes still compete for the medium fairly at the beginning of the next cycle with S-MAC. With DW-MAC, however, it is possible that a node is ready to rebroadcast a packet before its defer timer expires, as multiple SCHs can be transmitted during a Data period. A CDS node that defers could be slower in rebroadcasting a packet compared to a non-CDS node that does not defer, resulting in lower latency for DW-MAC. All

than for DW-MAC CDS. However, DW-MAC still reduces end-to-end latency by around 40% for CDS-based flooding compared to those with S-MAC.

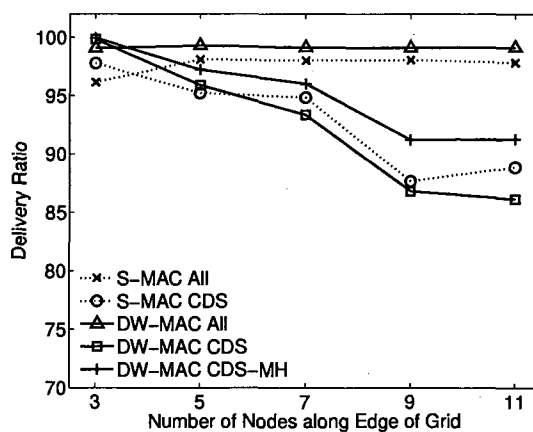
Figure 4.4(b) shows the delivery ratios (the percent of broadcast packets that are successfully received by *all* nodes in a network) of flooding in the grid networks. Because of the increased redundancy in simple flooding, S-MAC and DW-MAC achieve higher delivery ratio than CDS-based flooding. In simple flooding, DW-MAC outperforms S-MAC, since the use of (short) SCH frames instead of long data packets during contention helps to avoid collisions. However, when CDS-based flooding is used, DW-MAC sometimes shows lower delivery ratios than does S-MAC, mainly due to the special grid topology and selection of CDS as discussed before. Looking at the results in random networks (Figure 4.5(b)), on average, DW-MAC shows better delivery ratios than S-MAC when CDS-based flooding is used. With improved spatial reuse when optimized multihop forwarding is used, DW-MAC achieves higher delivery ratios than does S-MAC in CDS-based flooding.

Average energy consumption in the grid networks, calculated as I did in evaluations under unicast traffic, is shown in Figure 4.4(c). The interval between traffic bursts is changed from 200 seconds to 100 seconds to show the differences among protocols more clearly. DW-MAC reduces average energy consumption over S-MAC by about 26% under simple flooding and by about 18% under CDS-based flooding. DW-MAC achieves these savings by not overhearing data trans-

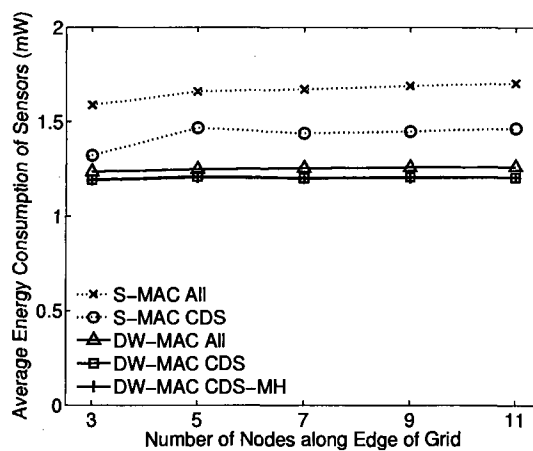




(a) End-to-end delay versus network size.

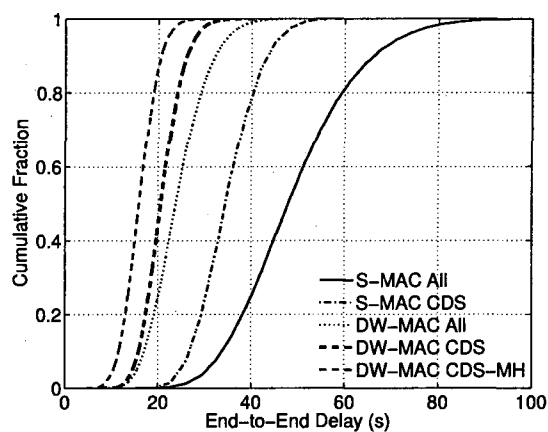


(b) Delivery ratio versus network size.

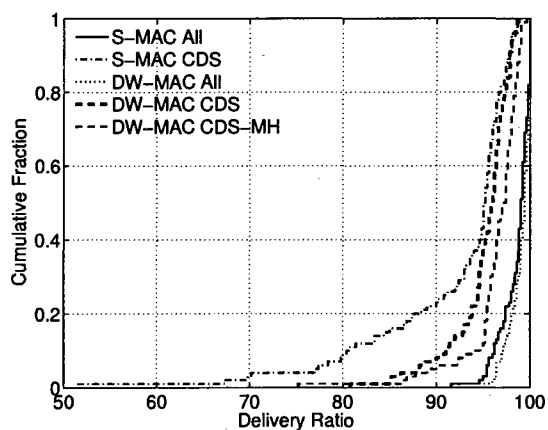


(c) Average energy consumption of sensors versus network size.

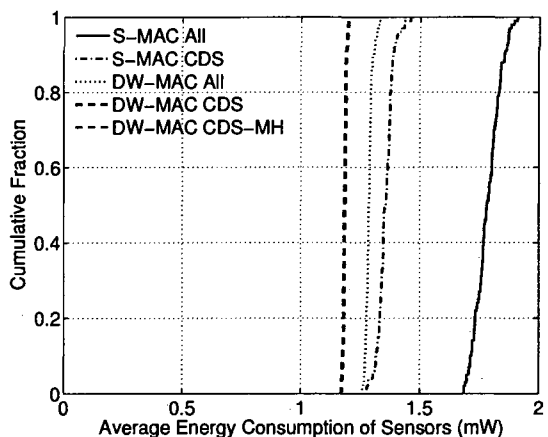
**Figure 4.4.** Performance for broadcast traffic in grid networks.



(a) CDF of end-to-end delays.



(b) CDF of delivery ratios.



(c) CDF of average energy consumption.

**Figure 4.5.** Performance for broadcast traffic in 50-node networks.

missions. In DW-MAC, a node only attempts to receive an incoming packet after receiving an SCH that indicates the packet has not been received. Simple flooding consumes more energy because of more rebroadcasts. Whether or not the optimized multihop forwarding is used, a flooding results in the same number of transmissions, so this optimization does not affect energy consumption much.

Finally, I compare these broadcast protocols in 100 random networks, the same networks used for evaluations under unicast traffic. The sink in each network generates 500 broadcast packets in each run, one packet every 100 seconds. Figure 4.5(a) shows the CDF of end-to-end latency for all packets in the 100 runs. All DW-MAC based broadcast protocols show much smaller end-to-end latency than those based on S-MAC. The average end-to-end latency for S-MAC ALL, S-MAC CDS, DW-MAC ALL, DW-MAC CDS and DW-MAC CDS-MH are 49.1, 34.8, 24.2, 20.8, and 16.0 seconds, respectively. On average, end-to-end latency is reduced by more than 50% both in simple flooding and in CDS-based flooding. Unlike the results in grid networks, DW-MAC shows lower average end-to-end latency in CDS-based flooding than those in simple flooding, because the speedup gained by fast propagation along CDS nodes is often greater than the slowdown caused by defers in these networks. For these 100 runs, the CDF of delivery ratios is shown in Figure 4.5(b), and the CDF of average energy consumption is shown in Figure 4.5(c). S-MAC ALL, S-MAC CDS, DW-MAC ALL, DW-MAC CDS, and DW-MAC CDS-MH show the average delivery ratios of

98.6%, 92.1%, 99.0%, 95.0% and 96.4%, and average energy consumption of 1.785, 1.355, 1.288, 1.185, and 1.183 mW, respectively. The difference in energy consumption between DW-MAC CDS and DW-MAC CDS-MH is almost invisible because the optimized multihop forwarding does not affect the number of data transmissions much. Overall, DW-MAC achieves lower end-to-end delays, higher delivery ratios, and more energy savings for broadcast traffic in these random networks.

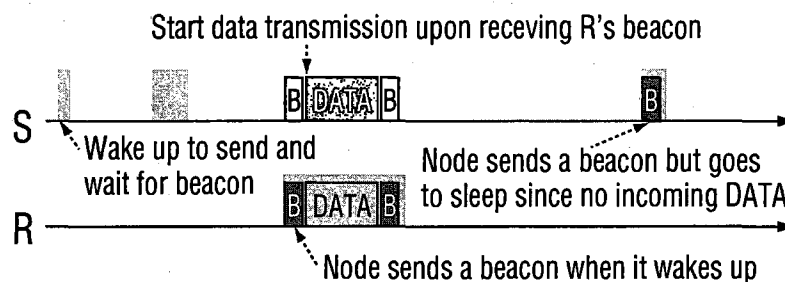
## Chapter 5

# RI-MAC Design

In this chapter, I describe the design of the RI-MAC protocol. After an overview of the protocol, I discuss details of RI-MAC's design and conclude with a discussion of how I implemented RI-MAC in TinyOS.

### 5.1 Overview

In RI-MAC, a DATA frame transmission is always initiated by the intended receiver node of the DATA. Figure 5.1 gives an overview of the operation of RI-MAC. Each node periodically wakes up based on its own schedule to check if there are any incoming DATA frames intended for this node. After turning on its radio, a node immediately broadcasts a beacon if the medium is idle, announcing that it is awake and ready to receive a DATA frame. A node with pending DATA to send, node *S* in this figure, stays active silently while waiting for the beacon from the intended receiver *R*. Upon receiving the beacon from *R*, node *S* starts its DATA transmission immediately, which will be acknowledged by *R* with another beacon. Note that this ACK beacon's role is twofold: first, it acknowledges the correct receipt of the sent DATA frame, and second, it invites a new DATA frame trans-



**Figure 5.1.** Overview of RI-MAC. Each node periodically wakes up and broadcasts a beacon. When node *S* wants to send a DATA frame to node *R*, it stays active silently and starts DATA transmission upon receiving a beacon from *R*. Node *S* later wakes up but goes to sleep after transmitting a beacon frame since there is no incoming DATA frame.

mission to the same receiver. If there is no incoming DATA after broadcasting a beacon, the node goes to sleep, as *S* does later in the figure.

RI-MAC significantly reduces the amount of time a pair of nodes occupy the medium before they reach a rendezvous time for data exchange, compared to the preamble transmission in B-MAC and X-MAC. This short occupation of the wireless medium enables more contending nodes to exchange DATA frames with their intended receivers, which helps to increase capacity of the network and thus potential throughput. More importantly, this increase is *adaptive*, by letting a beacon serve both as an acknowledgment to previously received DATA and as a request for the initiation of the next DATA transmission, as discussed in detail in Section 5.2.

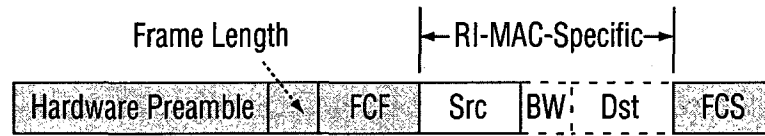
In RI-MAC, medium access control among senders that want to transmit DATA frames to the same receiver is mainly controlled by the receiver. This design choice makes RI-MAC more efficient in detecting collisions and recovering lost DATA

frames than B-MAC and X-MAC when the senders are hidden to each other, which can be common in ad hoc sensor networks. As discussed in Section 5.4, after transmitting a beacon, a receiver detects collisions within the duration of the backoff window specified in the beacon, which is much shorter than the delay of a sleep interval needed in B-MAC and X-MAC.

RI-MAC also reduces overhearing, as a receiver expects incoming data only within a small window after beacon transmission. Together with the lower cost for detecting collisions and recovering lost DATA frames, RI-MAC achieves higher power efficiency, especially when the network load increases. Even under light traffic load, which is the worst case for RI-MAC for power efficiency, RI-MAC still shows comparable performance to X-MAC in my simulation and experimental evaluation on MICAz motes. RI-MAC still decouples the sender's and receiver's duty cycle schedules as do B-MAC and X-MAC, which removes the overhead of synchronization compared to synchronous duty cycle MAC protocols.

## 5.2 Beacon Frames

A beacon frame in RI-MAC always contains a *Src* field, which is the address of the source transmitting node of the beacon. I call a beacon with *only* a *Src* field a *base* beacon. A beacon can also include two optional fields, depending on the roles the beacon serves: *Dst*, for destination address, and *BW*, for backoff window size. The

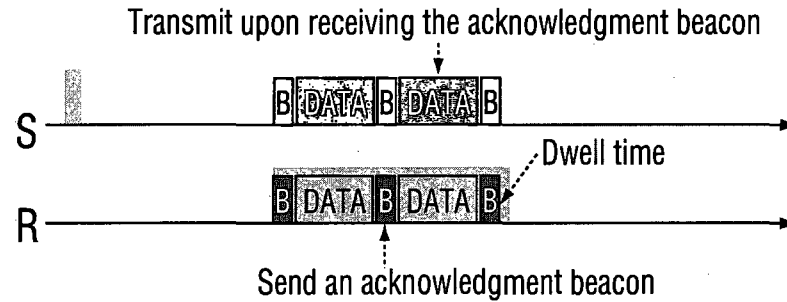


**Figure 5.2.** The format of an RI-MAC beacon frame for an IEEE 802.15.4 radio. Dashed rectangles indicate optional fields. The Frame Length, Frame Control Field (FCF), and Frame Check Sequence (FCS) are fields from IEEE 802.15.4 standard.

RI-MAC beacon frame format for an IEEE 802.15.4 radio is illustrated in Figure 5.2 as an example.

A node that receives a beacon can determine which fields are present in the beacon by looking at the size of the beacon; with an IEEE 802.15.4 radio, size of a beacon is saved in the Frame Length field. A beacon in RI-MAC can play two simultaneous roles: as an acknowledgment to previously received DATA, and as a request for the initiation of the next DATA transmission, as illustrated in Figure 5.3. After node *R* wakes up and senses clear medium, *R* transmits a base beacon. If the medium is busy, *R* does a backoff and attempts to transmit the beacon later. After receipt of the first DATA frame from *S* in the figure, in the following beacon transmission by *R*, the Dst field is set to *S* to indicate that this beacon also serves as the acknowledgment for the DATA received from *S*. Similar to ACK transmission in IEEE 802.11, transmission of this acknowledgment beacon starts after SIFS delay, regardless of medium status. Nodes other than *S* ignore the Dst field in the beacon and treat it as a request for the initiation of a new data transmission. The use of the BW field in a beacon is discussed in detail in Section 5.4.





**Figure 5.3.** The dual roles of a beacon in RI-MAC. A beacon serves both as an acknowledgment to previously received DATA and as a request for the initiation of the next DATA transmission to this node.

The duty cycle in RI-MAC is controlled by a parameter called the *sleep interval*, which determines how often a node wakes up and generates a beacon to poll for pending DATA frames. Suppose a sleep interval of  $L$  is used in some WSN. After a node generates a beacon, the interval before the next beacon generation is set to a random value between  $0.5 \times L$  and  $1.5 \times L$ . In this way, RI-MAC attempts to minimize the possibility that beacon transmissions from two nodes become coincidentally synchronized.

### 5.3 Dwell Time for Queued Packets

After successfully receiving a DATA frame, a node remains active for some extra time in order to allow queued packets to be sent to it immediately, as shown in Figure 5.3. I refer to this time as the *dwell time*. Unlike in X-MAC, where the dwell time is set to a fixed value of the maximum backoff window, the dwell time in RI-MAC adapts to the number of contending senders. The duration of the dwell time is

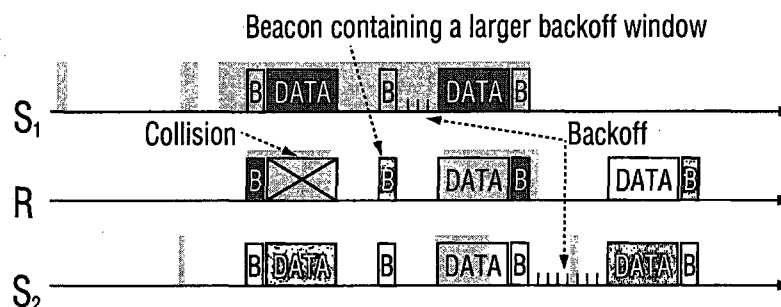
defined as the BW value from the last beacon plus SIFS and the maximum propagation delay. Since the BW in a beacon is automatically adjusted based on channel collisions observed by a node as discussed in detail next, so is the dwell time. The fewer contending senders and thus the fewer collisions, the shorter the dwell time. This self-adaptation helps RI-MAC using the shortest waiting time possible under light channel contention while avoiding collisions under heavy channel contention.

## 5.4 DATA Frame Transmissions from Contending Senders

The challenges in handling transmissions from an unpredictable number of contending senders are twofold:

- minimize the active time of a receiver for power efficiency; and
- minimize the cost for collision detection and recovery of lost data, whether or not senders are hidden to each other.

To meet these goals in RI-MAC, a receiver employs beacon frames to coordinate DATA frame transmissions from contending senders, as shown in Figure 5.4. The BW field in a beacon specifies the backoff window size senders should use when they contend for the medium. If a received beacon does not contain a BW field (i.e., a base beacon), senders for this receiver should start transmitting DATA without backing off. If a beacon contains a BW field, each sender does a random backoff



**Figure 5.4.** DATA frame transmission from contending senders in RI-MAC. For the first beacon, the receiver  $R$  requests senders (here,  $S_1$  and  $S_2$ ) to start transmitting DATA immediately upon receiving the beacon. If a collision is detected,  $R$  sends another beacon with increased BW value to request that senders do a backoff before their next transmission attempt.

using the BW as the backoff window size over which to choose the actual backoff.

The receiver increases the value of the BW field upon detecting collisions.

If a node cannot start data transmission as soon as it receives a beacon, prior to actual DATA transmission, a sender should make sure that the medium has been idle for at least  $T_p$  time using CCA (clear channel assessment) checks. The CCA checks prevent a sender from starting DATA transmission while the intended receiver is generating an acknowledgment beacon to a DATA frame just received from another sender. The time  $T_p$  here is set to SIFS plus the maximum propagation delay. If a node needs more time to generate and send an acknowledgment beacon, such as a software ACK used in TinyOS,  $T_p$  should be increased correspondingly, as described in Section 5.7.

After waking up, a node always broadcasts a base beacon with no BW field. I made this design choice to optimize RI-MAC for the most common cases of a typi-

cal WSN where there is light or no traffic most of the time. By enforcing all senders with pending DATA frames to transmit immediately, the design attempts to minimize time for the node to determine whether or not there is incoming DATA. The shorter this duration, the less energy is used at each wakeup. In this way, I attempt to minimize energy consumption if the network is idle most of the time. The duration can be very short, as it is the maximum round trip propagation delay plus radio switch delay (SIFS in IEEE 802.15.4). If the receiver detects no channel activity within this duration, the receiver goes to sleep immediately. Although a base beacon could lead to concurrent DATA transmissions to a same receiver, I found that they do not necessarily lead to collisions in the experimental implementation on MICAz motes [7], due to the presence of capture effect in the CC2420 radio [5]. This feature makes it possible for one sender to successfully transmit a packet to the receiver even if the transmission overlaps with others, especially when senders have different distances to the receiver (and thus different received signal strengths) [21, 23].

## 5.5 Collision Detection and Retransmissions

By coordinating DATA frame transmissions at receivers, RI-MAC greatly reduces the cost for detecting collisions and recovering lost DATA frames compared to B-MAC and X-MAC. As a sender can transmit DATA frame only upon receiving a beacon, and since the backoff window size is explicitly controlled by the intended

receiver, the receiver knows the maximum delay before a DATA frame's arrival. This delay can be calculated from the BW value in the previous beacon. The receiver need only detect the Start of Frame Delimiter (SFD) to learn of an incoming frame. If no SFD is detected in time, while some channel activity is detected by the CCA (clear channel assessment) check, the receiver will decide that there was a collision and will generate another beacon with a larger BW value. In RI-MAC, this new beacon is transmitted after the longest possible DATA transmission has finished so that all senders' radios are already in receive mode. Prior to transmitting the beacon, a node does a random backoff to avoid possible repeated collisions with beacons from another node.

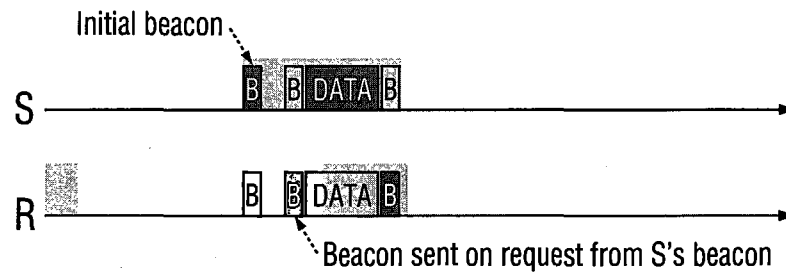
After detecting a collision, a receiver calculates the new BW value that will be used in the next beacon, by employing some backoff strategy such as binary exponential backoff (BEB) in IEEE 802.11 or Sift [42, 18], depending on the density of a network. BEB is used in my implementation in TinyOS, as I found it adapts to networks of different densities and resolves collisions efficiently in RI-MAC in the evaluations.

As RI-MAC initiates transmissions at the receiver, retransmission in RI-MAC is significantly different from that in sender-initiated approaches such as IEEE 802.11. In RI-MAC, a receiver plays the major role in retransmission control by managing the timing and number of beacon transmissions. If the BW value has reached the maximum backoff window size, or if the receiver keeps detecting collisions after a

number of consecutive beacon transmissions, the receiver goes to sleep without further attempts. The corresponding senders also become involved in retransmission control, because a sender could miss receiving a beacon either because of collisions or poor channel conditions. Thus, a sender maintains a retry count for each DATA frame. If no beacon has been received from the intended receiver within a time span 3 times as long as the sleep interval, the sender increases the current retry count by 1. In addition, the sender increases this retry count if no acknowledgment beacon is received within the maximum backoff window after the sender transmitted a DATA frame following receipt of a beacon. When the retry count reaches a pre-defined retry limit, the sender cancels the transmission of the DATA frame.

## 5.6 Beacon-on-Request

It is possible that the intended receiver node for some sender is already active when the sender wakes up to transmit a DATA frame to it. An optimization, called *beacon-on-request*, is for this sender, after waking up for DATA transmission, to broadcast a beacon following a CCA check, as illustrated in Figure 5.5. In this beacon, the sender  $S$  sets the Dst field to the receiver's address,  $R$ . If the receiver  $R$  happens to be active, it generates a beacon in response after some random delay longer than the BW announced in the received beacon from  $S$ . This beacon generated by the receiver on request of the sender allows the sender to transmit the



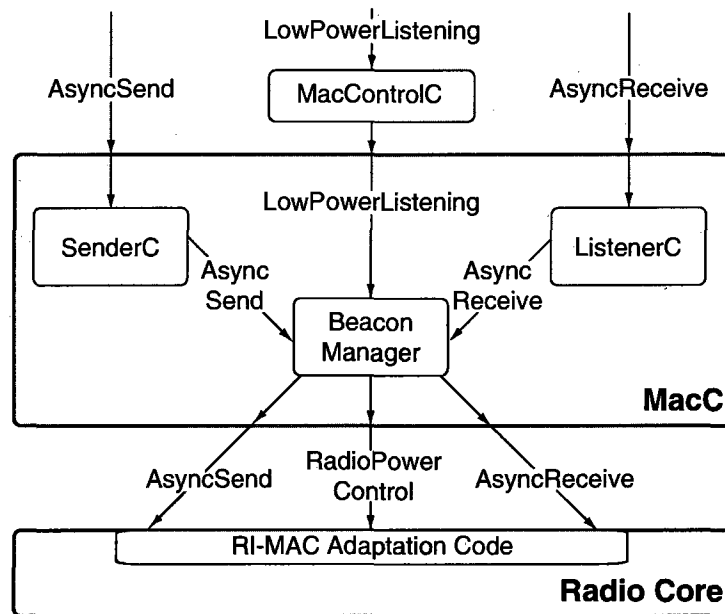
**Figure 5.5.** RI-MAC *beacon-on-request*. When node *S* wakes up for transmitting a pending DATA frame, it sends a beacon with the Dst field set to the destination of the pending DATA. If the destination node *R* is already active, *R* in response transmits a beacon to enable *S* to begin DATA transmission immediately.

pending DATA frame immediately, rather than waiting until the next scheduled beacon transmission by *R*.

## 5.7 RI-MAC Implementation in TinyOS

I implemented RI-MAC under the UPMA framework [20] in TinyOS on a network of MICAz sensor motes. The composition of RI-MAC under the UPMA framework in my implementation is shown in Figure 5.6. The implementation used the CC2420 radio, which is a packetizing radio used in popular MICAz and TelosB motes, although the code can be ported to motes with streaming radios such as the CC1000 [6] as well.

The *BeaconManager* module in Figure 5.6 performs most of the functionality of RI-MAC, including beacon generation, radio power control, wakeup/sleep scheduling, and retransmission control.



**Figure 5.6.** Composition of RI-MAC within the UPMA framework in TinyOS.

I also added some code to the radio core module of TinyOS, indicated by *RI-MAC Adaptation Code* in the figure. This adaptation code is introduced mainly for two purposes.

First, this adaptation code preloads a DATA frame into the CC2420 TX buffer. In this way, the DATA transmission can start immediately when a desired beacon arrives. This preloading helps to reduce the time a receiver node needs for detecting if there is incoming DATA after a beacon transmission. In the implementation on MICAz motes, after a node sends a beacon, the node needs to wait only 3.75 ms, listening to the medium, in order to detect whether or not there is an incoming packet. A beacon in the implementation is processed entirely in software, as the



beacon frame is not supported directly by the CC2420 hardware. With hardware support, this waiting time of 3.75 ms could be further reduced.

Second, the RI-MAC adaptation code starts contiguous CCA (clear channel assessment) checks immediately after a beacon transmission and counts the number of consecutive CCA checks that show busy medium. Suppose that after transmitting a beacon, a packet has not arrived within the expected arrival time that is proportional to the BW field in the beacon transmitted. The node will generate another beacon if the CCA checks indicate busy medium, or will go to sleep otherwise. In particular, on MICAz motes, if at least 20 consecutive CCA checks indicate busy medium during this time, the RI-MAC adaptation code notifies the BeaconManager of a collision; the BeaconManager then generates another beacon with a larger BW value, if necessary.

As the beacon frame is not part of the IEEE 802.15.4 standard and thus is not directly supported by the CC2420 radio, the implementation turns off hardware *address recognition* in the CC2420 and use a reserved frame type for beacon frames. To minimize the footprint of the RI-MAC implementation in the existing TinyOS code, I use a frame with only the CC2420 header (`cc2420_header_t` in TinyOS) as a beacon. Thus, a beacon is 12 bytes without the preceding hardware preamble, although the size of a base beacon could be implemented to be only 6 bytes, as discussed in Section 5.2.

To account for software processing delays on the MICAz motes, I also adjusted some parameters of RI-MAC in the implementation. A mote may experience some delays before transmitting consecutive packets in the queue, such as post-processing of a transmitted packet, moving a queued packet to the MAC layer, and loading the packet to the hardware buffer. Therefore, in the implementation, I added an extra 10 ms to the dwell time defined in RI-MAC to account for these delays. As an acknowledgment beacon is generated entirely by software in the implementation,  $T_p$ , defined in Section 5.4, is set to 2.5 ms, based on my measurements. If a beacon were processed in hardware, this time could be much shorter.

## Chapter 6

### Evaluation of RI-MAC

In this chapter, I evaluated RI-MAC both in the *ns-2* network simulator and in an implementation in TinyOS on MICAz motes. Simulations are used to explore RI-MAC's performance in a wide variety of networks, especially large network topologies which are hard to deploy and experiment with. As a protocol may not perform in the real world exactly as it does in simulation, for example due to the simplified physical layer models used in *ns-2* [1], I also evaluated RI-MAC in a small testbed network of MICAz motes running TinyOS; my experimental results from this testbed match the results obtained in simulation and further verify RI-MAC's performance advantages over existing protocols. Since Klues et al. [20] have implemented X-MAC-UPMA on real motes and shown that X-MAC-UPMA outperforms B-MAC and SCP, in this thesis, I compared RI-MAC only against X-MAC and X-MAC-UPMA.

#### 6.1 Simulation Evaluation

In the simulation evaluation of RI-MAC, I used version 2.29 of the *ns-2* network simulator, using the standard combined free space and two-ray ground reflection

**Table 6.1.** Simulation Radio Parameters

Bandwidth	250 Kbps	Size of Hardware Preamble	6 B
SIFS	192 $\mu$ s	Size of ACK	5 B
Slot time	320 $\mu$ s	CCA Check Delay	128 $\mu$ s
Tx Range	250 m	Carrier Sensing Range	550 m

radio propagation model commonly used with *ns-2*. Each sensor node is simulated with a single omni-directional antenna.

Table 6.1 summarizes the key parameters used to simulate the radio of each sensor node. Most of these parameters are from the data sheet of CC2420 radio [5], which is used in popular motes such as MICAz and TelosB. The RSSI sampling delay for CC2420 was reported by Ye et al. [49]; This delay is used as the time for a single CCA (clear channel assessment) check, i.e., the delay before actual transmission starts after a STXONCCA command is strobed [5]. The transmission range and carrier sensing range depend on many factors such as transmission power, antenna, and environment. In *ns-2*, the transmission range and the carrier sensing range are modeled after the 914MHz Lucent WaveLAN radio, which is not typical for a sensor node, but these *ns-2* default parameters are used since measurements have shown that similar proportions of the carrier sensing range to the transmission range are also observed in some state-of-art sensor nodes [2].

Table 6.2 summarizes the MAC protocol parameters used in the simulations. Backoff strategy and retransmission have not been explicitly discussed in prior work [3, 20], as X-MAC is optimized for light traffic load. 32 is used as the initial

**Table 6.2.** Simulation MAC Protocol Parameters

	X-MAC	X-MAC-UPMA	RI-MAC
Backoff Window	32	32	0–255
Retry Limit	0 or 5	0 or 5	5
Special Frame	Short Preamble	—	Beacon
Special Frame Size	6 B	—	6–9 B
Dwell Time	10.5 ms	100 ms	Variable

backoff window and 8 as the congestion backoff window, which are the default values used in the UPMA package distributed with TinyOS [43]. In the RI-MAC implementation, a receiver adjusts the BW value in each beacon using a binary exponential backoff (BEB) that takes values of 0, 31, 63, 127, and 255 in the evaluation. The backoff window size for beacon transmission is fixed at 32 slots in RI-MAC.

Although retransmission was not included in X-MAC's original design (none was specified in X-MAC's published design [3, 20, 43]), for fair comparison with RI-MAC in which retransmission is included, I evaluated X-MAC and X-MAC-UPMA both with and without retransmission in the simulations. When retransmission was enabled, 5 is used as the retry limit. The way in which an undecodable signal that is higher than the CCA threshold should be handled was also not explicitly discussed for X-MAC [3], but this occurrence could be common in a large network. Therefore, in my simulated X-MAC, a node turns off its radio if the medium has been idle for a time that is longer than the gap between short preambles. This is achieved by starting a timer that does CCA checks every 20 ms, and each CCA check lasts longer than the gap between short preambles.

The time 20 ms was used because that is the wake time used in X-MAC's evaluation [3]. In X-MAC-UPMA, a node that has detected busy medium turns off its radio if no packet is received within 100 ms, according to the code in the UPMA distribution. In the simulated X-MAC-UPMA, similar to the original X-MAC design, only the first preamble in a sequence of short preambles is subject to backoff before transmission (i.e., when the **RESEND\_WITHOUT\_CCA** option is used in the UPMA package).

In the simulations, a short preamble in X-MAC consists of a Frame Control Field (FCF), destination address, and Frame Check Sequence (FCS). Each of these fields is 2 bytes, resulting in a short preamble of 6 bytes plus the leading 6-byte hardware preamble. A base beacon has the same length and format, except that the address of the transmitting node is in the beacon instead of the destination address. If a beacon also serves as an acknowledgment, or if the BW field is included, a beacon can be 7, 8, or 9 bytes. Dwell time is defined as the maximum backoff window size in X-MAC; 10.5 ms, a slightly longer duration, is used to account for SIFS and propagation delays. The distributed UPMA code uses 100 ms as its default dwell time. Dwell time in RI-MAC is variable, as it is defined as the backoff window for senders (the BW field in a beacon) plus SIFS and propagation delays.

To simplify the evaluation, routing traffic is not included in the simulations and assume that there is a routing protocol deployed to provide the shortest path

between any two nodes. I also ensure that no network used in the simulations is partitioned.

As energy consumption of different radios varies significantly, even in the same radio state [49], effective duty cycle is reported in evaluating power efficiency, as done in prior work [3, 20]. The sleep interval for all three MAC protocols is 1 second, and the initial wakeup time of each node was randomized in the evaluation. Note that the sleep interval is an expected value in RI-MAC, as RI-MAC randomizes intervals of sleep time to avoid synchronized beacon transmissions from neighboring nodes. In the evaluation, data payload size was always 28 bytes, the default value in the UPMA package.

I compared X-MAC, X-MAC-UPMA, and RI-MAC in three types of networks: clique networks, a 49-node ( $7 \times 7$ ) grid network, and random networks. Beacon-on-request is not used in the clique networks, as no multihop communication takes place in these networks; in all other networks, beacon-on-request is used.

### 6.1.1 Results in Clique Networks

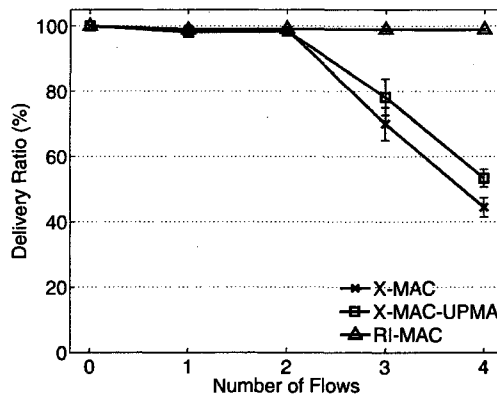
I discuss first the evaluation of X-MAC, X-MAC-UPMA, and RI-MAC in clique networks, such that all nodes in the network are within transmission range of each other. The traffic load is varied by varying the number of independent flows in the network, with no flow sharing source or destination node with any other flow. In each clique network, the total number of nodes in the network is twice the number of flows. For each flow, the source node starts to generate packets

10 seconds after the beginning of the simulation and generates new packets with an interval between two successive packet generations uniformly distributed between 0.5 and 1.5 seconds. At the beginning of the simulation, each node randomly chooses a time between 0 and 10 seconds as its next wakeup time. In this way, the wakeup/sleep schedule of each node is randomized. The recipient nodes count the number of packets received successfully over the course of 50 seconds. If a packet still resides in any queue or is still being transmitted at the end of the 50-second measurement, the packet is not counted as a delivered packet.

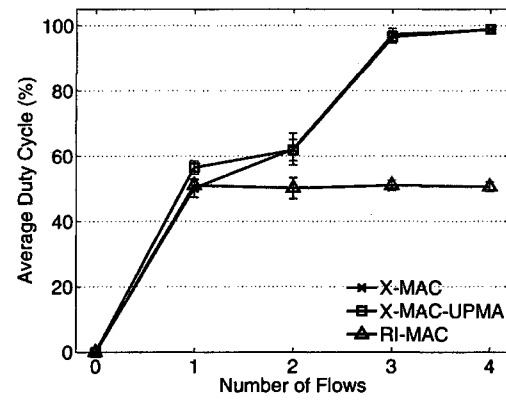
The results for the clique network simulations are shown in Figure 6.1, where each average value is calculated from the results of 10 random runs. Error bars show the 95% confidence interval. In Figure 6.1, a value of 0 for number of flows indicates the case in which there is no traffic and just a single node in a network, and thus all energy consumption is due to periodic wakeups of this single node.

Figure 6.1(a) shows the packet delivery ratios achieved by X-MAC, X-MAC-UPMA, and RI-MAC with increasing number of contending flows in the clique networks. Delivery ratios with RI-MAC are always close to 100%, indicating that total throughput achieved with RI-MAC increases linearly with the increasing traffic load. X-MAC and X-MAC-UPMA deliver most of the given load when there are no more than 2 flows, but their delivery ratios drop quickly beyond 2 flows. This sharp decline is not due to collisions, as all nodes can hear each other. Rather, it is because preamble transmissions in X-MAC and X-MAC-UPMA saturate the

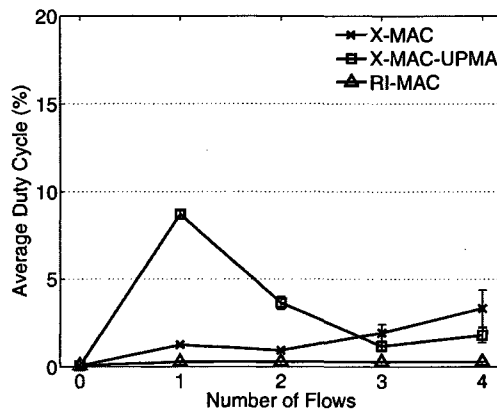




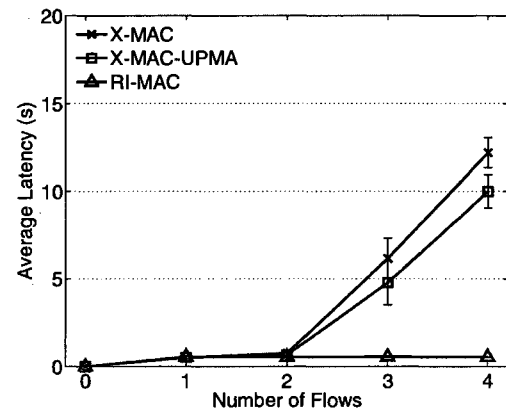
(a) Average Packet Delivery Ratio



(b) Average Duty Cycle of Senders



(c) Average Duty Cycle of Receivers



(d) Average Latency

**Figure 6.1.** Performance comparison in clique networks with contending flows in simulation. The total number of nodes is 1 for 0 flows, and is twice the number of flows otherwise.

network, resulting in a large number of queued packets. When there are 4 flows in a clique network, RI-MAC improves delivery ratio and thus throughput by about 100% compared to X-MAC and X-MAC-UPMA.

The average duty cycles of senders and receivers corresponding to Figure 6.1(a) are shown in Figure 6.1(b) and Figure 6.1(c), respectively. In addition to the improved delivery ratios, RI-MAC saves more energy when there are multiple flows in a clique network, compared to X-MAC and X-MAC-UPMA. With 1 flow, senders (Figure 6.1(b)) show around 50% duty cycle with all protocols, as it takes a sender half a sleep interval to reach its intended receiver, on average. The duty cycles with RI-MAC remain at around 50% with increasing flows, but those with X-MAC and X-MAC-UPMA increase quickly to almost 100% when there are 4 flows. This increase in X-MAC and X-MAC-UPMA is because a sender with pending DATA must do congestion backoff when the medium is occupied by a preamble transmission from another flow. If the corresponding receiver wakes up before the medium becomes idle, the sender must wait until the receiver's next wakeup. If the medium is sensed busy, the sender could go to sleep and to attempt transmission later, but in this approach, latency could be significantly increased without necessarily saving energy.

X-MAC and X-MAC-UPMA each result in a much higher duty cycle than does RI-MAC when there is 1 flow, as shown in Figure 6.1(c). This higher duty cycle is because of the longer dwell time used in X-MAC and X-MAC-UPMA. In

X-MAC, this dwell time is 10.5 ms, roughly a backoff windows of 32 slots, and in X-MAC-UPMA, this dwell time is 100 ms by default. The dwell time in RI-MAC is much smaller with 1 flow. As there is no collision and thus backoff window for senders is always 0, dwell time in RI-MAC is just SIFS plus propagation delay. The duty cycles of receiving nodes decrease with more contending flows in X-MAC and X-MAC-UPMA, as a receiver goes to sleep immediately after receiving packets from other flows.

Despite the high duty cycle at sending nodes, X-MAC and X-MAC-UPMA experience longer latency than does RI-MAC, as shown in Figure 6.1(d). This latency is mainly because transmission of preambles saturates the medium when there are more than 2 flows. The queuing delay in X-MAC and X-MAC-UPMA results in an average latency that is more than 10 times longer than that with RI-MAC when there are 4 flows.

When the number of flows is 0 in Figure 6.1, all three protocols show very similar performance, although this is the worst case for RI-MAC compared to X-MAC and X-MAC-UPMA. In this case, a node with RI-MAC has to stay awake each time slightly longer than it does with X-MAC and X-MAC-UPMA. In X-MAC and X-MAC-UPMA, a node needs to listen to the medium for at least SIFS plus the delay for ACK transmission at each wakeup. RI-MAC incurs some extra cost only for the CCA check before a beacon transmission and for detecting incoming signal after the beacon transmission. The difference caused by such extra cost, however, is

too small to show clearly in the figure, as all three protocols already show very low duty cycles under very light traffic. As RI-MAC substantially improves throughput and energy efficiency and reduces latency under higher traffic loads, RI-MAC is suitable for a wide range of traffic loads.

### 6.1.2 Results in a 49-Node Grid Network

In the comparison of X-MAC, X-MAC-UPMA, and RI-MAC in a 49-node ( $7 \times 7$ ) grid network, each node is 200 meters from its neighbors, and the sink node is at the center.

In the simulations, the RCE traffic model defined in Section 4 is used. RCE picks a random  $(x, y)$  location for each event. If every node has a sensing range  $R$ , only nodes that are within the circle centered at  $(x, y)$  with radius  $R$  generate packets to report this event. The sensing range  $R$  is adjusted to simulate different degrees of workload in the network. A new event is generated once every 60 seconds, and each node having sensed the event sends one packet to the sink node.  $R$  is varied from 100 meters to 500 meters; Table 6.3 shows the average number of packets generated per event. Note that an event triggers at most one packet when  $R$  is 100 meters. The lengths of paths traversed by these packets to the sink node range from 1 to 6 hops, with an average of 3.05 hops. In this way, the simulations explore how efficiently X-MAC, X-MAC-UPMA, and RI-MAC handle different degrees of traffic load.

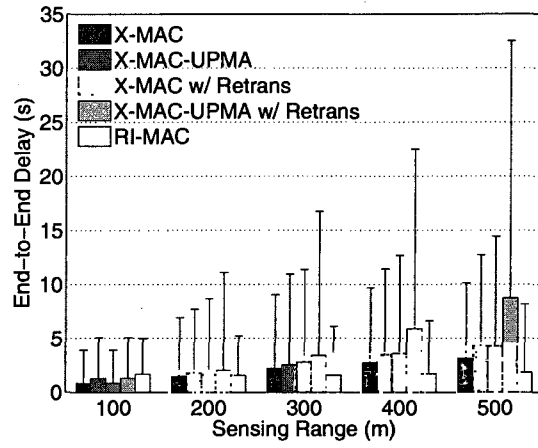
**Table 6.3.** Average Number of Packets Generated for Each Event under Different Sensing Ranges in the 49-Node Grid Network

Range (m)	100	200	300	400	500
Packets	0.8	3.1	6.4	10.6	15.2

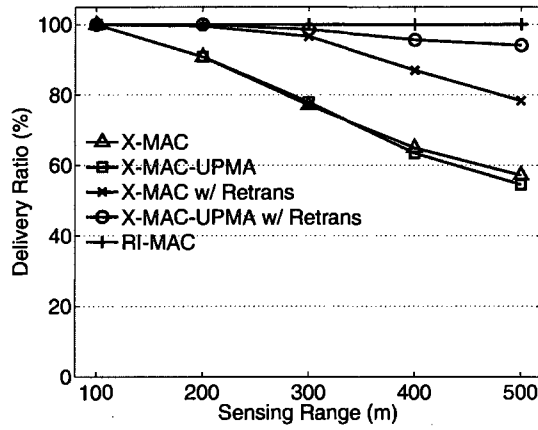
Each simulation run contains unicast packets sent toward a sink node that are triggered by a series of 100 events, and each average value is calculated from the results of 30 random runs. Confidence intervals of the average values are not shown because even 99% confidence intervals are so close to average values that they overlap with the data point markers. The curves labeled *X-MAC w/ Retrans* and *X-MAC-UPMA w/ Retrans* show the results when the original X-MAC and X-MAC-UPMA protocols, respectively, are augmented with retransmission.

The performance comparison in these grid network scenarios is shown in Figure 6.2. Figure 6.2(a) shows the average and maximum end-to-end latency of packets in the RCE model as the sensing range (and thus traffic load) increases. RI-MAC has a much smaller rate of increase than do X-MAC and X-MAC-UPMA, regardless of whether or not retransmission is used. When there are about 15 packets generated for each event (a 500-meter sensing range), RI-MAC reduces average end-to-end delay by 85% compared to X-MAC-UPMA with retransmission, and by around 50% compared to the other protocols.

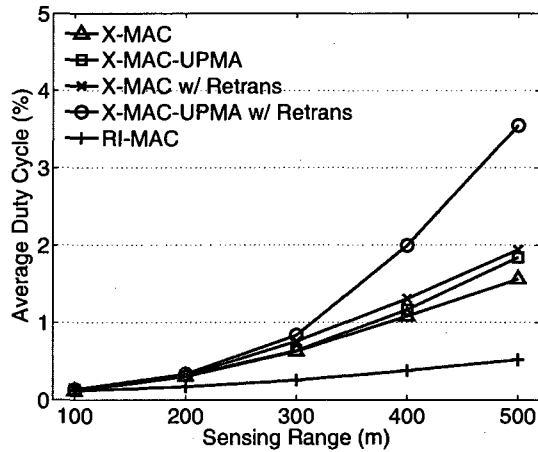
RI-MAC outperforms X-MAC and X-MAC-UPMA because it greatly increases idle medium time, allowing more competing flows to transmit in single a cycle. End-to-end latency increases when X-MAC and X-MAC-UPMA are augmented



(a) Average and maximum end-to-end delay versus sensing range.



(b) Delivery ratio versus sensing range.



(c) Average duty cycle of sensors versus sensing range.

**Figure 6.2.** Performance for unicast traffic in 49-node ( $7 \times 7$ ) grid network scenarios in simulation.

with retransmission, due to the added effort to recover packets that would otherwise be lost in collisions. Under the very light traffic load when sensing range is 100 m, X-MAC shows lower latency due to how it handles undecodable signals. For example, consider a chain consisting of nodes  $A$ ,  $B$  and  $C$ , where node  $A$  can reach  $B$ , and  $B$  can reach  $C$ . Nodes  $A$  and  $C$  cannot reach each other but can sense each other's transmission. When  $A$  sends short preambles followed by a DATA frame to  $B$ , node  $C$  will remain active after sensing the medium busy, even though no incoming packet can be decoded. If  $C$  still has its radio on when  $B$  immediately starts forwarding the just-received packet to  $C$ , the forwarding will experience less delay. Because  $C$  turns off its radio if no packet is successfully received for 100 ms in X-MAC-UPMA, even though the medium is still busy, node  $C$  can be either active or sleep when  $B$  starts forwarding, depending on when  $C$  starts the 100 ms timer. This is why X-MAC-UPMA shows lower latency than does RI-MAC but higher latency than X-MAC under very light traffic load. However, as traffic load increases when sensing range is greater than 100 m, RI-MAC achieves the lowest latency on average due to increased idle medium time.

The packet delivery ratios corresponding to Figure 6.2(a) are shown in Figure 6.2(b). RI-MAC maintains 100% packet delivery ratio and outperforms X-MAC and X-MAC-UPMA across all sensing ranges. RI-MAC achieves these high delivery ratios mainly by efficient collision detection and retransmission control. The delivery ratios with X-MAC and X-MAC-UPMA drop quickly, since the larger the

sensing range, the more collisions caused by transmissions from hidden nodes. When X-MAC and X-MAC-UPMA are augmented with retransmission, packet delivery ratio increases. X-MAC with retransmission shows lower delivery ratios than does X-MAC-UPMA due to the lack of an ACK after DATA transmission. If a DATA frame is lost due to collision at a receiver, the corresponding sender has no way to detect the collision and thus the DATA will not be retransmitted.

RI-MAC, in addition to achieving 100% packet delivery ratios, at the same time achieves lower duty cycles. The improved packet delivery ratios by retransmission in X-MAC and X-MAC-UPMA, however, come at the cost of higher energy consumption, as shown in Figure 6.2(c). All protocols show larger duty cycles as sensing range, and thus traffic load, increases. However, RI-MAC has a much smaller rate of increase than do the other protocols. For example, when sensing range is 500 m, RI-MAC's duty cycle is only 15% that of X-MAC-UPMA with retransmission and 27% that of X-MAC with retransmission. At the same time, RI-MAC achieves much lower latency and higher packet delivery ratio, as discussed above. With retransmission, X-MAC shows lower duty cycle than does X-MAC-UPMA, mainly due to less retransmission effort because of undetectable DATA collisions.

### 6.1.3 Results in Random Networks

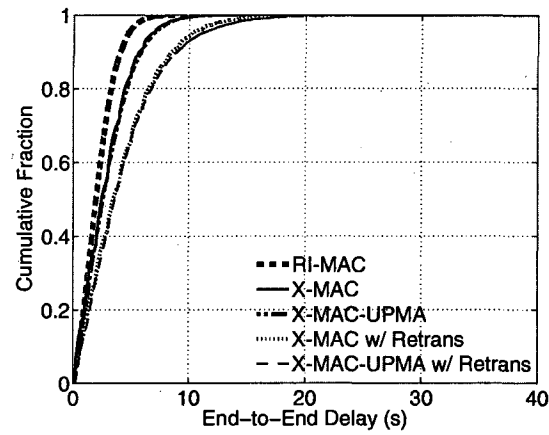
This set of simulations compares RI-MAC, X-MAC, and X-MAC-UPMA in 100 random networks, each with 50 nodes randomly located in a 1000 m×1000 m area. For each network, one of these nodes is randomly selected as the sink, and the RCE



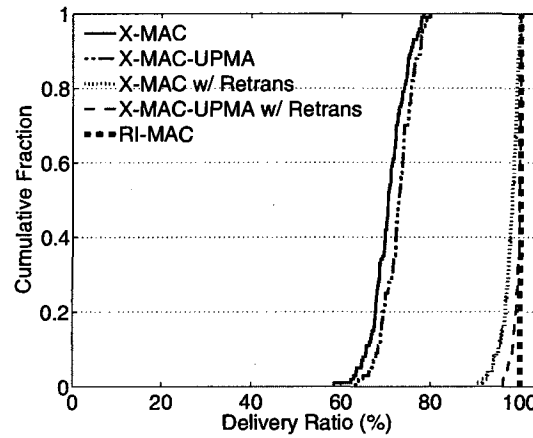
model with 250-meter sensing range is used to generate 100 events, one every 60 seconds. one simulation run is conducted for each of these 100 networks, with 763 packets on average generated in each run.

The results for these simulations are shown in Figure 6.3. Figure 6.3(a) shows the CDF of end-to-end latency for all packets in all 100 runs, Figure 6.3(b) shows the CDF of packet delivery ratios in these 100 runs, and Figure 6.3(c) shows the average duty cycles of the sensors. To improve clarity in these graphs, the protocols are listed in each graph's legend, from top to bottom, in the same order as the curves appear in the graph, from left to right. The X-MAC and X-MAC-UPMA curves in Figure 6.3(a) are almost indistinguishable from each other in the graph, and the curves for these two protocols with retransmissions are likewise almost indistinguishable from each other in this same graph.

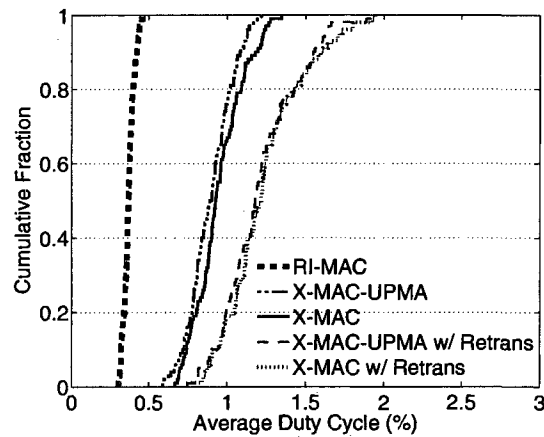
For the same reasons as discusses above, RI-MAC outperforms the other protocols in each of these metrics. For end-to-end latency (Figure 6.3(a)), the average values for RI-MAC, X-MAC, X-MAC-UPMA, X-MAC with retransmission, and X-MAC-UPMA with retransmission, are 2.21, 2.88, 3.02, 4.19, and 4.40 seconds, respectively. Although the addition of retransmissions in X-MAC and X-MAC-UPMA improves packet delivery ratios by helping to recover packets that would otherwise be lost due to collisions (Figure 6.3(b)), these retransmitted packets have higher delivery latency than other packets, producing higher average end-to-end latency for these protocol versions. The average packet delivery ratios



(a) CDF of end-to-end delays



(b) CDF of delivery ratios



(c) CDF of average duty cycles

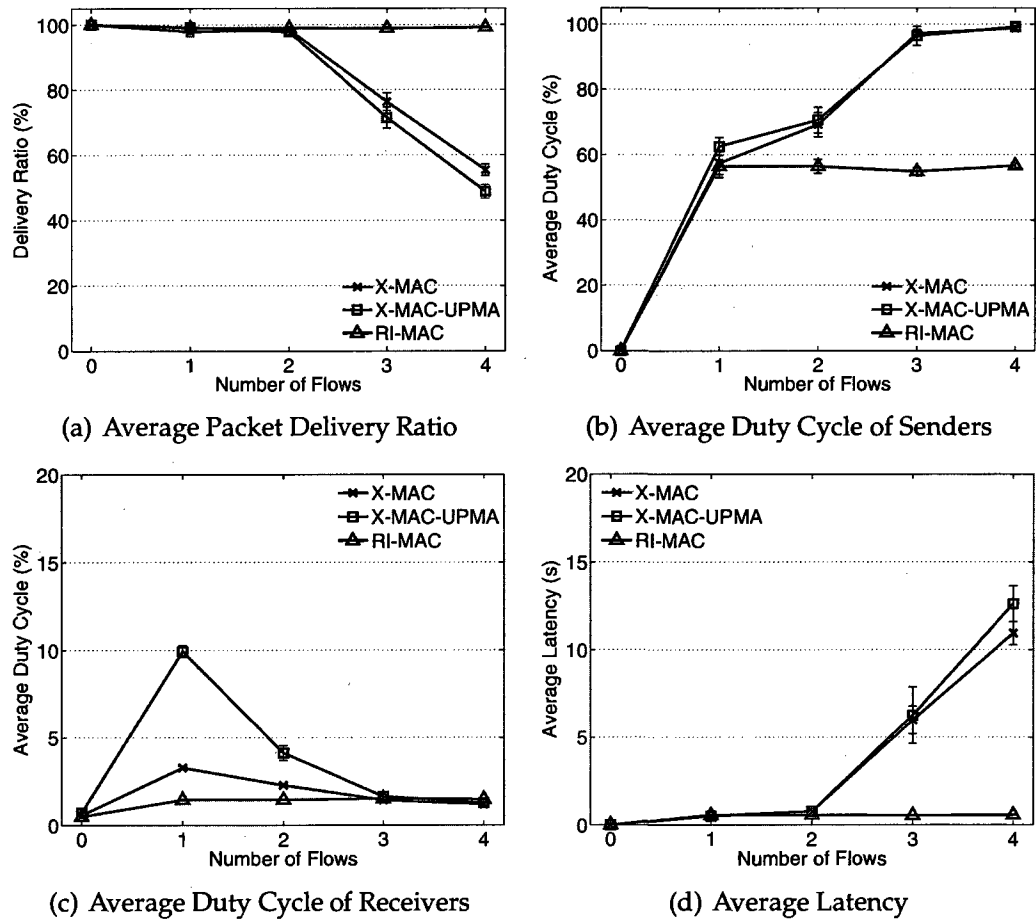
**Figure 6.3.** Performance for random correlated-event traffic in 50-node networks with sensing range of 250 m in simulation.

for X-MAC, X-MAC-UPMA, X-MAC with retransmission, X-MAC-UPMA with retransmission, and RI-MAC are 70.5%, 72.6%, 97.7%, 99.4%, and 100%, respectively. The addition of retransmissions in X-MAC and X-MAC-UPMA also come at the cost of increased energy consumption (Figure 6.3(c)). The average values for the duty cycles of all sensors for RI-MAC, X-MAC-UPMA, X-MAC, X-MAC-UPMA with retransmission, and X-MAC with retransmission are 0.37%, 0.89%, 0.95%, 1.21%, and 1.23%, respectively. The trends observed in these random networks for each of these three metrics are consistent with those observed in the 49-node ( $7 \times 7$ ) grid network, discussed above in Section 6.1.2.

## 6.2 Experimental TinyOS Evaluation

To validate the simulation-based evaluation reported above, and to explore hardware platform-dependent trends and problems, I also compared RI-MAC with X-MAC and X-MAC-UPMA in an implementation in TinyOS on MICAz motes. RI-MAC is implemented under the UPMA framework in TinyOS as described in Section 5.7.

Although both X-MAC and X-MAC-UPMA use short preambles to achieve LPL, I also implemented X-MAC under the UPMA framework, as X-MAC-UPMA differs from the original X-MAC design in several aspects, as discussed in Sections 2.2 and 6.1. The configuration of X-MAC is the same as that used in the simulations, except for the continuous CCA check interval, the duration to wait for an



**Figure 6.4.** Performance comparison in clique networks of MICAz motes with contending flows in TinyOS implementation.

ACK after each short preamble transmission, and the dwell time. As the duration of the continuous CCA check interval prior to preamble transmission should be longer than the gap between adjacent short preamble transmissions, the interval is set to the sum of the ACK transmission time, SIFS, and maximum propagation delay in the simulation. As discussed by Klues et al. [20], however, a longer interval is used in the TinyOS implementation in order to account for processing delays and to minimize false negatives. Therefore, the default value of 5.25 ms is used in the X-MAC-UPMA code for X-MAC. For the same reason, the duration to wait for an ACK after each short preamble transmission is set to 4 ms, which is also the default value in the X-MAC-UPMA code. Lastly, the dwell time should also be longer than the backoff window size in X-MAC, in order to account for possible processing delays such as post-processing of a just-transmitted packet, moving a queued packet to the MAC layer, and loading the packet to the hardware buffer. Therefore, the dwell time defined in X-MAC needs to be extended to account for these delays. For fair comparison, the extra dwell time for X-MAC is also 10 ms, the same with that in my implementation of RI-MAC. In order to minimize change to underlying radio core of TinyOS, a packet that contains only the CC2420 header is used as a short preamble. Both a short preamble of X-MAC and a beacon of RI-MAC are 12 bytes, although their minimum sizes could be 6 bytes, as discussed in Section 6.1. The default configuration of X-MAC-UPMA is used in the exper-

iments. Beacon-on-request is not included in the RI-MAC implementation, since nodes do not use multihop communication in these experiments.

### 6.2.1 Results in Clique Networks

In order to verify my simulation models, I present first experiments on MICAz nodes in clique networks; these TinyOS experiments are intended to replicate the simulation experiments performed for clique networks, discussed in Section 6.1.1. The network configurations and traffic model are the same as those used in simulations. The results are shown in Figure 6.4 and match closely the trends and results shown earlier in Figure 6.1 for the clique network simulations.

The duty cycles at sending nodes and at receiving nodes (Figures 6.4(b) and 6.4(c), respectively) are slightly higher than those in the simulations (Figure 6.1(b) and Figure 6.1(c)). This increase is mainly because the MICAz-specific processing delays in software are not simulated. For example, in my TinyOS implementation of RI-MAC, it takes around 3.75 ms for a DATA frame to arrive after a beacon transmission, mainly due to the processing delay of the beacon at a sender in software before it starts transmitting the DATA. In simulation, however, the beacon is assumed to be handled in hardware, so a DATA frame arrives just SIFS plus some propagation delay after the beacon transmission. In addition, in the TinyOS implementation of X-MAC and RI-MAC, I also add 10 ms to the dwell time from their original design to account for processing delays to handle the transmitted packet and to start transmitting new packets as discussed above. Although the simulation

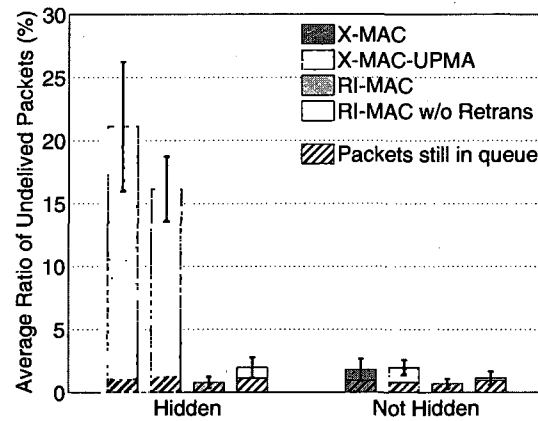
model does not account for these platform-specific delays, the simulation results still agree well with these TinyOS experimental results.

### 6.2.2 Results in a Network with Hidden Nodes

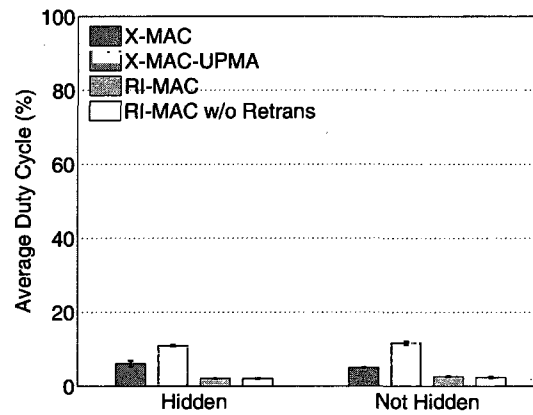
I evaluate here RI-MAC, X-MAC, and X-MAC-UPMA on networks of MICAz motes to determine how efficiently each of them detects collisions and performs retransmission to recover packets lost due to collisions; I chose to evaluate this on the TinyOS implementation rather than in simulation due to the simplified radio model used by *ns-2*.

In this set of experiments, each average value is calculated from the results of 10 experimental runs, in the same way as that for clique networks. Error bars show the 95% confidence intervals.

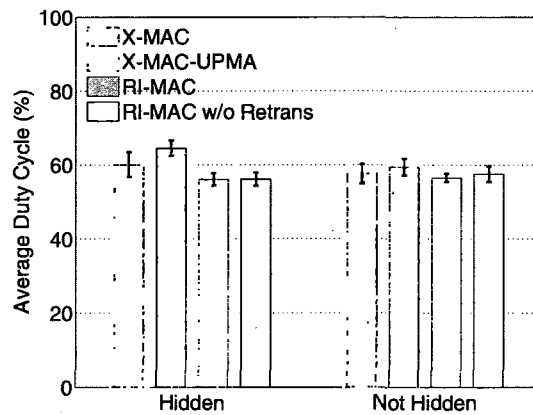
In this evaluation, I experimented with two separate network topologies: one in which hidden nodes were present, and one with no hidden nodes. Specifically, for each topology, I set up a network of 3 motes in which two senders transmit packets to a single receiver node. The distance from each sender to the receiver is the same and is within the transmission range of each sender. In the case with no hidden nodes, the two senders are also within range of each other, whereas in the case in which hidden nodes were present, the two senders are hidden to each other (i.e., the CCA check at each sender almost always indicates a clear channel, even while the other sender is transmitting packets). The two network topologies were otherwise identical. The same traffic model is used as that for clique networks in



(a) Average Ratio of Undelivered Packets



(b) Average Duty Cycle of The Receiver



(c) Average Duty Cycle of Senders

**Figure 6.5.** Performance comparison when two sender are hidden to each other and when they are not in a 3-node network in TinyOS implementation.



Section 6.2.1. To also evaluate how efficiently RI-MAC detects collisions, a variation of RI-MAC, in which a sender does no retransmissions or retries as defined in Section 5.5, is included in these experiments. This variation of RI-MAC is referred to as *RI-MAC w/o Retrans*.

Results for this set of experiments on MICAz motes are shown in Figure 6.5. I compare the ratio of undelivered packets for X-MAC, X-MAC-UPMA, and RI-MAC in Figure 6.5(a). A packet may be undelivered because of collisions; it is also possible that the packet is still in the transmission queue or is being transmitted at the end of experimental measurement period. Therefore, the ratio of undelivered packets for each protocol that are still in the queue (including those being transmitted) is indicated separately in Figure 6.5(a). In this way, it possible to evaluate separately how many packets are not delivered due to collisions. The labeling along the  $x$ -axis in Figure 6.5(a) indicates whether or not the two senders are hidden to each other.

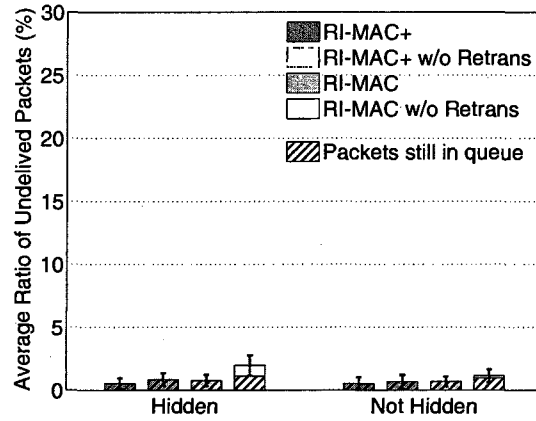
In both network topologies (with hidden nodes present and without), all protocols had a small fraction of undelivered packets still in the queue or still in transmission at the end of the experimental measurement period (Figure 6.5(a)). With hidden nodes present, X-MAC and X-MAC-UPMA both experienced a much larger number of additional undelivered packets due to other causes, though: about 20% of the generated packets are lost with X-MAC and 15% with X-MAC-UPMA. RI-MAC, on the other hand, experienced almost no such losses with hidden nodes.

In order to confirm that these losses with X-MAC and X-MAC-UPMA are likely due to the collisions caused by the hidden node senders, these results are compared to those for the topology with no hidden nodes. In this case, almost all of these additional losses with X-MAC and X-MAC-UPMA were eliminated. With RI-MAC, however, with hidden nodes and without, no packets were lost other than those still in queue, indicating that *no packets are lost due to collisions with RI-MAC*.

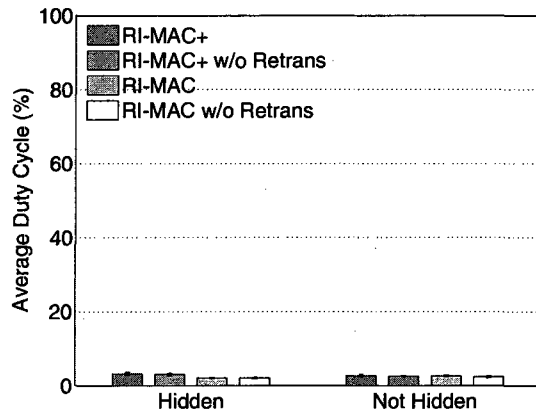
In addition to a much higher packet delivery ratio, RI-MAC achieves lower duty cycles both at the receiver and at the senders. The shorter dwell time in RI-MAC is the major reason for the lower duty cycles at the receiver with RI-MAC, as discussed above. Fast collision detection and retransmission with RI-MAC also helps to achieve lower duty cycles at the senders. With X-MAC, if short preambles from the two senders repeatedly collide with each other, each sender can do nothing but retransmit its short preamble. In RI-MAC, the receiver detects the collision quickly and uses a larger sender backoff window to avoid further collisions.

### 6.2.3 Extra Ending Beacons for MICAz

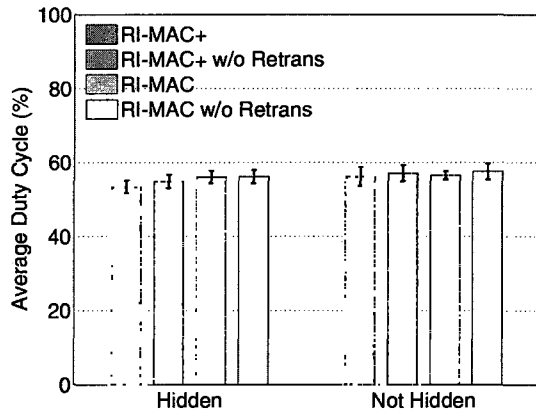
In Figure 6.5, the results for *RI-MAC w/o Retrans* are close to those for RI-MAC, except that around 2% of the packets are lost due to collisions. After extensive experimentation, this packet loss is discovered to be caused by a combination of the capture effect and the processing delays on the MICAz motes.



(a) Average Ratio of Undelivered Packets



(b) Average Duty Cycle of The Receiver



(c) Average Duty Cycle of Senders

**Figure 6.6.** Effectiveness of using an extra *ending beacon* in RI-MAC in TinyOS implementation.

For example, suppose two sender nodes, *A* and *B*, each have 2 packets in their queue to send to receiver *C* before they receive the first beacon without backoff window from *C*. Then *A* and *B* each start transmitting their DATA frames at the same time. Assume *C* receives the DATA from *A* but loses the DATA from *B* due to the capture effect in *C*'s radio. As *C* believes that no collision occurred since it received a DATA frame following its beacon, it sends an acknowledgment beacon without backoff window to *A*. Now *A* and *B* both are allowed to transmit DATA immediately. *B* has a DATA frame already loaded in its hardware buffer that is waiting for an acknowledgment beacon, but *A* has to get a DATA from its upper layer protocol or application and load it to its hardware buffer. Thus, *B* starts transmission immediately, but *A* can only start after this processing delay. If the later DATA transmission from *A* happens to overlap with the acknowledgment beacon transmission from *C* to *B*, *C* will not know that there is a sender with pending DATA for it and thus will not generate another beacon. As a result, *A* discards the DATA due to timeout.

This problem occurs on the MICAz motes because the CC2420 hardware transmission buffer can hold only one DATA frame. Thus, there is some delay before the queued DATA can be transmitted. If a radio could hold multiple queued packets in its hardware buffer and thus supported back-to-back DATA transmission, this problem would be much less likely to occur.

Although even on the MICAz hardware, this problem occurs only infrequently, to better handle this case, I experimented with adding an extra *ending beacon* to the original RI-MAC design. Suppose a node detects no incoming packet or collisions after the previous beacon transmission. In my original RI-MAC design, this node goes to sleep immediately. With this modification, instead, I let the node send another beacon without backoff window if the node has received at least one DATA frame after waking up in the current cycle. The node treats the beacon in the same way as the first beacon after waking up.

I compared this solution with the original RI-MAC design and show the results in Figure 6.6. RI-MAC with the extra *ending beacon* modification is referred to as *RI-MAC+*, and the modified RI-MAC without retransmissions is referred to as *RI-MAC+ w/o Retrans*. Figure 6.6(a) shows the average ratio of undelivered packets, Figure 6.6(b) shows the average duty cycle of the receiver, and Figure 6.6(c) shows the average duty cycle of senders. RI-MAC with this modification now does not lose any packets due to collisions, even with retransmission at the senders disabled (*RI-MAC+ w/o Retrans*). RI-MAC is thus very effective in detecting and recovering from collisions, even with the limitations of the real hardware in the MICAz nodes.

The receiver with *RI-MAC+* or *RI-MAC+ w/o Retrans* consumes more energy, as shown in Figure 6.6(b), due to the extra ending beacons, but these beacons help to reduce energy consumption at the senders, as shown in Figure 6.6(c), as some

DATA frames are delivered immediately following the ending beacons rather than waiting until the next cycle.

## **Chapter 7**

### **ADB Design**

This chapter presents the design of ADB. In addition to providing multihop broadcast support, ADB optionally maintains neighbor lists and estimates link quality in a power-efficient manner; these additional services may be used when they are not provided by other components in the system.

#### **7.1 Design Motivation**

Multihop broadcast over asynchronous duty cycling is challenging for many reasons. For example, the neighbors of a transmitter wake up asynchronously, requiring the transmitter to stay active long enough so that each neighbor has chance to receive the broadcast packet, resulting in increased energy consumption. In addition, transmission attempts over poor quality links can significantly decrease delivery ratio and increase delivery latency and energy consumption. When a transmission fails and the intended receiver goes to sleep, if the transmitter is to retransmit, it must wait until the receiver wakes up in next cycle. A transmitter may also substantially delay forwarding by other neighbors, if the transmitter occupies the medium while waiting to reach all of its neighbors. Finally, information about the progress a broadcast is often crucial for a node to avoid redundant

transmissions, but a node that has just waken up has no up-to-date progress information. A node cannot simply use overhearing to learn the information, as the progress may change when the node has its radio off.

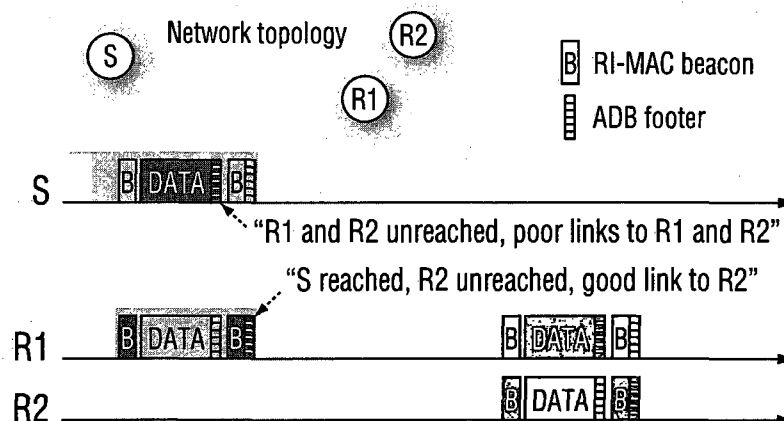
To address these challenges, I made a number of basic decisions in designing ADB. First, since with asynchronous duty cycling, the neighbors of a node wake up at different times, I chose to use unicast transmission of the DATA packet to each neighbor node as it wakes up. The acknowledgment in a unicast transmission also helps a transmitter to accurately learn whether a neighbor has been reached by the broadcast, and allows the transmitter to use retransmissions to increase the reliability of the broadcast to wireless transmission errors and collisions. Second, in order to avoid the transmitter occupying the wireless medium while waiting for each neighbor to wake up, I chose to integrate ADB with RI-MAC, in which each receiver announces its wakeup with a *beacon* packet, as described in Chapter 5. A transmitter starts DATA transmission upon receiving a beacon from its intended receiver, and then waits for an acknowledgment beacon (ACK) from the receiver. While waiting for the beacon before the DATA, the wireless channel is available for use by other nodes, such as neighbor nodes that have already received the DATA rebroadcasting it to their neighbors, helping to reduce delivery latency. By integrating with RI-MAC's unicast support, ADB can efficiently support multihop broadcast in the same system. Finally, I chose in ADB to passively measure link



quality based on the beacons, helping to avoid DATA transmission over a poor link when there is a better alternative.

## 7.2 Overview of ADB Operation

Figure 7.1 gives an overview of the operation of ADB. In this simple example, the network consists of three nodes, nodes *S*, *R1*, and *R2*, all within transmission ranges of each other. Node *S* wants to broadcast a DATA packet to all nodes. When *R1* wakes up, node *S* transmits the packet upon receiving *R1*'s beacon in the same way as for unicast in RI-MAC. However, ADB includes a new "footer" in DATA frames and acknowledgment beacons (ACKs), indicating the progress of the broadcast, including some transmissions that are about to happen. A receiving node uses this information to avoid unnecessary transmissions and to decide whether it should forward the packet to a neighbor that has not received it. In this example, the ADB footer in the DATA frame from *S* informs *R1* that *R2* has not been reached yet by the broadcast and that the quality of the link (*S*, *R2*) is poor. Suppose the quality of link (*R1*, *R2*) is good (e.g., because of the short distance). Node *R1* decides to deliver the packet to *R2* and indicates the good quality of (*R1*, *R2*) in the footer of the ACK to *R1*. Upon receiving this ACK, *S* learns that it is better for *R1* to transmit the packet to *R2*, so *S* "delegates" handling of *R2* to *R1*. As *S* has no other neighbor to be reached, *S* then goes to sleep immediately. When *R2* wakes up, *R1* unicasts the DATA frame to *R2* in the same way, except that the ADB



**Figure 7.1.** Overview of ADB. Node *S* broadcasts a DATA frame to node *R1* and *R2* via unicast transmission. The footer in DATA and ACK beacons helps *S* and *R1* to decide which node will deliver the DATA to *R2* and helps *R2* to learn that both *S* and *R1* have received the DATA.

footer in the DATA frame indicates that *S* has received the DATA frame, allowing *R2* to sleep immediately because all neighbors of *R2* have been reached.

The above example shows the following features of ADB:

- ADB allows a node to go to sleep once all its neighbors have been reached or have been delegated to other nodes;
- ADB attempts to avoid transmissions over poor links;
- ADB delivers a broadcast packet without occupying the medium while waiting for each receiver to wake up, to allow a neighbor to start rebroadcasting the packet immediately; and
- ADB informs a neighbor that has just woken up on the progress of a broadcast, to avoid unnecessary waiting and transmissions.

The coordination among direct neighbors is opportunistic, without relying on any network structure such as a tree or connected dominating set, allowing ADB to be efficient in handling broadcasts originated by any node in a network.

### 7.3 ADB Algorithm Details

ADB is composed of three basic procedures: (i) *Neighbor Detection and Link Quality Estimation*, which builds and distributes the neighbor list at each node and maintains the link quality to each neighbor on this list; (ii) *Coherent Encoding of ADB Control Information*, which helps to efficiently distribute information on the progress of a broadcast and information for delegation decisions; and (iii) *Delegation Procedure*, the basic procedure which runs whenever a broadcast packet or a beacon with an ADB footer is received or overheard; this procedure determines which nodes the packet should be forwarded to and which nodes should be delegated. Each procedure is described in turn below.

#### 7.3.1 Neighbor Detection and Link Quality Estimation

A node using ADB needs knowledge of its neighbors and the quality of the wireless link to each. Such information may be provided by existing mechanisms in the sensor node or by mechanisms provided by ADB. Packet delivery ratio (PDR) is used to estimate link quality in this thesis, as PDR can be measured in an energy efficient way and provides enough information for ADB to avoid poor links.

When a node  $v$  begins execution, it stays awake continuously for a short period of time, during which it counts the base beacons received from other nodes; each node transmits base beacons according to its normal schedule during this time, as shown in Figure 7.1. Let  $T$  denote the nominal duty cycle interval. Suppose the period of time for which node  $v$  counts base beacons in this way is 10 cycles ( $10 \times T$ ), and let  $n$  denote the number of base beacons received from some node  $w$  during this time. If  $n > 2$ , node  $v$  appends  $w$  to its neighbor list, denoted as  $N(v)$ . The estimated one-way link quality over link  $(w, v)$ , denoted as  $q(w, v)$ , is set to  $\min(1, n/10)$ , as on average 10 beacons should be expected. Initially, node  $v$  assumes that  $q(v, w) = q(w, v)$ ; its value will be updated passively later based on ongoing traffic. Subsequently,  $v$  begins normal operation. It waits for a random number of cycles before sending its neighbor list to its direct neighbors. When a node  $u$  receives the neighbor list of node  $v$ , node  $u$  will send its own neighbor list to  $v$  if it has not done so.

After the above initialization, ADB maintains the neighbor lists and updates link qualities passively. Whenever a node  $v$  is awake, it passively monitors the quality of the link between itself and each neighbor node  $w$  by counting (i)  $n$ , the number of base beacons received from  $w$ ; (ii)  $A_e$ , the number of DATA packets it transmits to  $w$  and thus the number of ACKs expected from  $w$ ; and (iii)  $A_r$ , the number of ACKs received from  $w$ . Each of these counts are reset each time  $v$  wakes up. When  $v$  later goes to sleep, define  $t$  as the time for which  $v$  had been awake.

If node  $v$  has been awake because it was transmitting a broadcast DATA packet to its neighbors, these neighbors will have all woken up and had an opportunity to receive the packet in less than  $1.5 \times T$ , since each neighbor wakes up and transmits a base beacon at randomized intervals varying between  $0.5 \times T$  and  $1.5 \times T$ .

However, if due to transmission errors or collisions, node  $v$  is unable to deliver the DATA to one or more neighbors on the first attempt, node  $v$  may remain awake in order to complete deliver, thus remaining awake longer than  $1.5 \times T$ .

When  $v$  goes to sleep, if  $t \geq 1.5 \times T$ , then node  $v$  has had an opportunity to receive at least one base beacon from each of its neighbors  $w$  while being awake; in this case, node  $v$  has information it can use to update  $v$ 's estimate of  $q(w, v)$  for each of its neighbors  $w$ . If node  $v$  has been awake for less time ( $t < 1.5 \times T$ ) but, for some particular neighbor  $w$ ,  $A_e > 0$ , then  $v$  has transmitted at least one DATA packet to  $w$  while being awake, giving  $v$  an opportunity to receive the ACK following this DATA; in this case, node  $v$  has information it can use to update  $v$ 's estimate of  $q(w, v)$  for this particular neighbor  $w$ . To update the one-way link quality estimate  $q(w, v)$  for some neighbor  $w$ , node  $v$  uses the weighted moving average function

$$q(w, v) = \alpha(t) \times ((n + A_r)/(A_e + t/T)) + (1 - \alpha(t)) \times q(w, v) \quad , \quad (7.1)$$

where  $\alpha(t) = 1 - e^{-\frac{t}{10 \times T}}$ .

In order to ensure that each node  $v$  updates its link quality estimates for all of its neighbors from time to time when there is ongoing broadcast traffic, if  $v$  has

not recently been active for any period of at least  $1.5 \times T$ , then the next time node  $v$  transmits (originates or forwards) a broadcast DATA packet, node  $v$  is forced to remain active for at least  $1.5 \times T$  ( $v$  may be awake most of this time, or longer, simply transmitting this broadcast). When node  $v$  subsequently goes to sleep,  $q(w, v)$  is updated using Equation 7.1 for all neighbors  $w$ . The period of time considered “recent” above may be adaptively selected based on the rate of recent changes observed in the link quality estimates or may be a fixed interval. In my current design, an interval of 15 minutes is used, in order to reduce energy consumption from these measurements while still tracking long-term changes in link qualities.

The above procedure has measured the link quality estimate for the link  $(w, v)$  for each of  $v$ ’s neighbor nodes  $w$ . For the reverse link  $(v, w)$ , ACK is used to piggyback the information needed. When node  $w$  sends an ACK beacon for a broadcast DATA received from  $v$ , node  $w$  includes its local  $q(v, w)$  in the ACK. Node  $v$  then replaces its local  $q(v, w)$  with this new value.

With the estimated one-way link quality for both directions over a link between  $v$  and  $w$ , the total link quality estimate for a DATA frame transmission over this link, denoted as  $Q(v, w)$ , is defined as

$$Q(v, w) = q(w, v) \times q(v, w) \times q(w, v) \quad , \quad (7.2)$$

since a successful DATA transmission from  $v$  to  $w$  requires a 3-way handshake between these nodes:  $w$  first sends a wake-up beacon which is received by  $v$ ;  $v$

then sends the DATA which is received by  $w$ ; and finally,  $w$  sends the ACK beacon which is received by  $v$ .

When node  $v$  broadcasts a DATA, it defines a deadline time for its delivery effort. If  $v$  fails to reach one of its neighbor, say  $w$ , by this deadline,  $v$  reduces  $q(v, w)$  as described in the following section. The deadline time is calculated using the equation

$$T \times 1.5 \times 3 \times \frac{1}{(\min_{w \in N(v)}(Q(v, w)))} \quad (7.3)$$

The value  $1/(\min_{w \in N(v)}(Q(v, w)))$  is the expected number of duty cycles to successfully deliver a DATA over the poorest link. The maximum interval between two consecutive beacons is  $1.5 \times T$ , which are added to the equation to account for the worst case. The factor 3 is used to further increase reliability. Needed effort is estimated very conservatively in Equation 7.3. However, despite of the estimated long deadline time, a node rarely needs to wait this long, as it can go to sleep once all its neighbors are either reached or have been delegated to other nodes.

### 7.3.2 Coherent Encoding of ADB Control Information

When a node wakes up and receives a broadcast DATA packet, the node must decide whether or not to transmit it to each of its neighbors. To facilitate this decision, each node  $v$  includes the *status* of each of its neighbors in the footer of DATA and ACK frames, as illustrated in Figure 7.1. Node  $v$  assigns one of the following values as the status of each neighbor  $w$ : *REACHED*, if  $w$  has received the packet;

*DELEGATED*, if some other node is going to deliver the packet to  $w$ ; or  $P(v, w)$ , an integer representation of  $Q(v, w)$ , otherwise.  $P(v, w)$  is referred to as the *priority* of this link. If node  $w$ 's status is *REACHED* or *DELEGATED*,  $v$  does not attempt to transmit the packet to  $w$ .

Otherwise,  $v$  attempts to transmit the packet to  $w$ , and the quality of link  $(v, w)$  is indicated by priority  $P(v, w)$ . ADB includes the status of all direct neighbors in the footer of a frame to a node, rather than the status of a subset of neighbors that the receiver node might be interested in. This design choice is made for two practical reasons. First, in an environment with many packet losses due to link errors or collisions, overhearing any footer allows nodes to learn about the progress of the corresponding broadcast. Second, having the transmitter instead include only the status update that a receiver is interested in would add significant processing delays for the transmitter on sensor nodes with limited CPU resources. In particular, since a transmitter does not in general know which neighbor will wake up next, the transmitter could generate the appropriate footer only after receiving the beacon from a neighbor. In order to allow a node to go to sleep as soon as possible after transmitting a beacon (particularly in the common case in which no packet needs to be sent to this node after its beacon), we generate the footer in advance and include the same status update in the footer for any neighbor.

It is often impractical to put the node ID and status of each neighbor in a frame due to the transmission overhead and the limited frame size. For example, with the



CC2420 radio that is widely used by popular sensor nodes, the maximum frame size is 128 bytes.

To efficiently encode ADB footers, a node  $v$  lists the status of neighbors using a bitmap with segments of equal length, with each segment corresponds to a node in  $N(v)$ , the set of neighbors of node  $v$ . In order to refer to a node by its position in  $N(v)$ ,  $N(v)$  is organized as an array. The segments are arranged in the same order as the corresponding node in  $N(v)$ . In order for a recipient node to decode this bitmap, node  $v$  distributes the neighbor list to direct neighbors. Let  $N_w(v)$  denote  $w$ 's local view of  $v$ 's neighbor list. Due to packet losses caused by collisions or dynamics of wireless channels,  $N_w(v)$  could be stale and different from  $N(v)$ . ADB ensures that  $N_w(v)$  is a prefix of  $N(v)$ , denoted  $N_w(v) \subseteq N(v)$ , by employing an *incremental neighbor list*.

In ADB, once a node  $v$  detects a new neighbor, it appends the neighbor to the *end* of its neighbor list  $N(v)$ . Since sensor nodes are stationary in most WSNs,  $N(v)$  will converge quickly. Even if a node  $w$  does not have the up-to-date copy of node  $v$ 's neighbor list,  $w$  can still decode the beginning portion of a received bitmap without ambiguity. In a more dynamic network such as with mobility, we could assign a version number to each neighbor list and to avoid ambiguity, but I chose to use the incremental neighbor list to efficiently handle the common case. Also, a node  $v$  will not remove any existing neighbor, say  $w$ , from its neighbor list  $N(v)$  even if node  $w$  has moved away or has failed. Instead, a node  $v$  will use the value

zero for  $P(v, w)$  in its bitmap to tell its neighbors it does not currently have a valid link to the node  $w$ .

In my implementation, each segment of a bitmap has 3 bits, which is able to represent node status value from 0 to 7. The value 7 is reserved for *REACHED*, the value 6 for *DELEGATED*, and the value 0 to indicate an unreachable neighbor. The priority of  $w$  at  $v$ ,  $P(v, w)$ , is thus in the range of 1 to 5. The total link quality estimate  $Q(v, w)$  is used to assign priority  $P(v, w)$  values using the equation

$$P(v, w) = \begin{cases} 0 & \text{if } Q(v, w) < 2\% \\ \min(5, 1 + \lfloor Q(v, w) \times 5 \rfloor) & \text{if } Q(v, w) \geq 2\% \end{cases} \quad (7.4)$$

As a node updates some  $Q(v, w)$ , it also updates  $P(v, w)$ . Especially when a node fails to deliver a packet to a neighbor by the deadline calculated by Equation 7.3,  $q(v, w)$  is reduced so that the corresponding  $P(v, w)$  can be mapped to a lower priority, increasing the chance that the neighbor can be delegated to another node next time. In order to avoid using very poor links, when  $Q(v, w)$  is less than 2%,  $P(v, w)$  is also set to 0. Also, if  $v$  fails to deliver its neighbor list to  $w$  due to reasons such as asymmetric links,  $P(v, w)$  is set to 0 directly. When  $P(v, w)$  is 0, node  $w$  is added to set  $B(v)$  of “bad” neighbors. Node  $v$  still attempts to transmit to  $w$  but  $v$  will go to sleep if all neighbors whose priorities are greater than 0 have been reached or delegated, regardless of  $w$ ’s status. If a DATA and the corresponding

ACK have been successfully exchanged over link  $(v, w)$ , Equation 7.4 will be used to calculate status of  $w$ , and  $w$  is removed from  $B(v)$ .

### 7.3.3 Delegation Procedure

The goal of ADB is to minimize redundant transmissions and to allow a node to sleep as early as possible. ADB also attempts to avoid transmissions over poor links.

ADB uses the following data structures to achieve these goals. For a broadcast packet  $i$ , node  $v$  maintains  $Rd^i$  as the set of nodes whose status is *REACHED*, and  $Dl^i$  as the set of nodes whose current status is *DELEGATED*. Initially, both  $Rd^i$  and  $Dl^i$  are empty. A node updates these two sets when receiving or overhearing a frame with an ADB footer that contains information about the progress of the broadcast. If either set changes, ADB makes the following decisions:

- If  $Rd^i \cup Dl^i = N(v) - B(v)$ ,  $v$  can go to sleep immediately, as all neighbors are either *REACHED* or *DELEGATED*.
- Otherwise, if  $w \in N(v) - B(v)$  and  $w \notin Rd^i \cup Dl^i$ , node  $v$  transmits the DATA to  $w$  on receiving a beacon from  $w$ .

Figure 7.2 shows the way in which node  $v$  analyzes an ADB footer that contains information about the progress of packet  $i$ . This footer is *received* or *overheard* from  $w$  and contains an array  $S_w$  that lists the status of  $w$ 's neighbors. The separate  $S^i$  local array at  $v$  lists the priority of each of  $v$ 's neighbors with respect to packet  $i$ .

Each entry  $S^i(u)$  is initially set to  $P(v, u)$ . The variable  $N_v(w)$  denotes the most recent neighbor list  $v$  has received from  $w$ .

The procedure *ANALYZE-FOOTER* is composed of three parts. First (lines 1–6), node  $v$  finds common neighbors with  $w$  that have been reached and adds each to  $Rd^i$ . Second (lines 7–14), if  $v$  has not received any ADB footer regarding packet  $i$ , then lines 9–13 are executed. If some common neighbor  $u$ 's status is equal to DELEGATED, some other node is about to transmit to  $u$ ; in order to avoid collisions, node  $v$  also sets  $u$ 's status to DELEGATED by adding  $u$  to  $Dl^i$ . Third (lines 15–28), node  $v$  and  $w$  negotiate which node is transmitting to a node that is neither REACHED nor DELEGATED. Line 19 is executed when  $v$  does not have better link quality to a common neighbor  $u$  compared with  $w$ . Thus,  $v$  gives up transmission to  $u$  and marks  $u$  as DELEGATED. Lines 21–25 are executed when  $v$  has better link quality to  $u$  compared with  $w$ . If this footer is from a DATA frame that is intended for  $v$ , node  $v$  removes  $u$  from  $Dl^i$  so that  $u$ 's status is set to  $P(v, u)$  in future outgoing frames (e.g., the ACK to this DATA). Once node  $w$  receives the ACK, node  $w$  will find that  $v$  has a better link quality to  $u$  and thus give up its own transmission to  $u$ . When the footer is from an *overheard* frame, node  $v$  sets  $u$ 's status to DELEGATED. There are two reasons for this design choice. First, if  $v$  wants to transmit the packet  $i$  to  $u$  itself,  $v$  would have to send a separate frame to notify  $w$  that  $v$  is in a better position to transmit  $i$  to  $u$ . Second, even if  $v$  decides not to transmit the packet  $i$  to  $u$ ,  $u$  may still delegate the transmission to some node that

---

```

procedure ANALYZE-FOOTER( $w, S_w, i, v$ ):
1: // find neighbors that are REACHED
2: for each vertex  $u \in N_v(w) \cap N(v)$  do
3:   if  $S_w[u] = REACHED$  then
4:      $Rd^i \leftarrow \{u\} \cup Rd^i$ 
5:   end if
6: end for

7: if  $v$  has never received any ADB footer regarding  $i$  before then
8:   // find neighbors that are DELEGATED
9:   for each vertex  $u \in N(v) \cap N_v(w)$  do
10:    if  $S_w[u] = DELEGATED$  then
11:       $Dl^i \leftarrow \{u\} \cup Dl^i$ 
12:    end if
13:  end for
14: end if

15: // delegation negotiation with  $w$  on unreached neighbors
16: for each vertex  $u \in (N(v) - Rd^i) \cap N_v(w)$  do
17:   if  $S_w[u] \neq DELEGATED$  then
18:     if  $S^i[u] \leq S_w[u]$  then
19:        $Dl^i \leftarrow \{u\} \cup Dl^i$ 
20:     else
21:       if  $S_w$  is from a DATA intended to  $v$  then
22:          $Dl^i \leftarrow Dl^i - \{u\}$ 
23:       else
24:          $Dl^i \leftarrow Dl^i \cup \{u\}$ 
25:       end if
26:     end if
27:   end if
28: end for

```

---

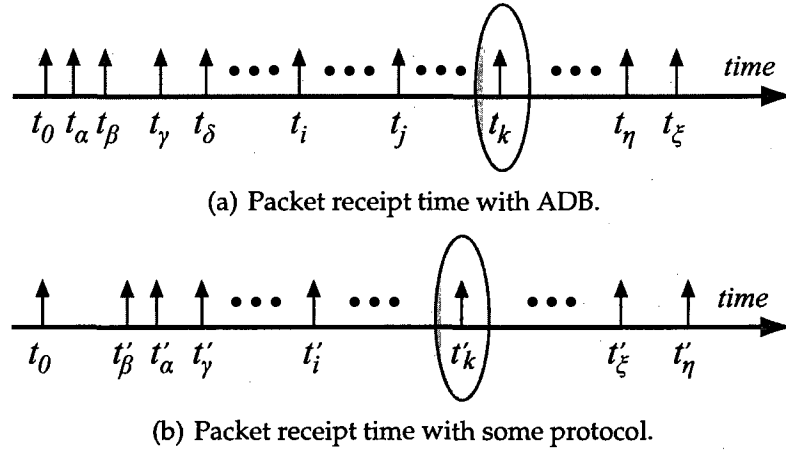
**Figure 7.2.** Node  $v$  analyzes an ADB footer that contains information about the progress of packet  $i$ . This footer is *received* or *overheard* from  $w$ , containing an array  $S_w$  that lists the status of  $w$ 's neighbors. The separate  $S^i$  local array lists the status of  $v$ 's neighbors with respect to packet  $i$ .

has a better link quality to  $u$ . In order to minimize message overhead,  $u$ 's status is set to DELEGATED at  $v$ .

Once a node starts forwarding of packet  $i$ , the node does not reflect any change of link status into local array  $S^i$  until the node stops the forwarding, so that the node and its neighbors make consistent decisions on the delivery of packet  $i$ . An ACK beacon to a DATA might get lost due to collisions or link errors. To make ADB robust to such packet losses, a node continues to include in each beacon the source address and sequence number of the most recently received broadcast packet for some period of time. In my implementation, this duration is set to 3 duty cycles.

## 7.4 Analysis of End-to-End Delivery Latency

To gain insight into the latency of ADB, I considered a simplified model in which the actual transmission time of a broadcast packet is negligible (0 time), and in which no collisions or link errors occur; that is, if two nodes transmit to the same node at the same time, the receiver will receive both packets successfully. Furthermore, each node is assumed to randomly pick a wakeup time that is independent of other nodes' wakeup times and of the traffic load. As shown later in this section and in the simulation evaluation in Section 8, the results based on this simplified model give good insights and are close to my simulation results based on a realistic simulation model.



**Figure 7.3.** ADB achieves optimal latency under simplified assumptions.

**Theorem 2** *Under the assumptions of 0 packet-transmission duration and error- and collision-free channels, each node receives, for the first time, each broadcast packet in minimum possible time using ADB.*

*Proof:* By contradiction. Assume without loss of generality that a node 0 in a network  $G = (V, E)$  originates a broadcast packet in the network; call this node the source node. Also assume that there exists at least one node that receives its broadcast packet with ADB later than it would with some other protocol. We run an instance of both protocols, ADB and the other protocol, and compare the time at which each node receives the packet for the first time.

Denote the times at which node  $j \in V$  received the broadcast packet for the first time based on ADB and the other protocol by  $t_j$  and  $t'_j$ , respectively, as shown in Figure 7.3. As  $t_0$  is equal to  $t'_0$ , the time the broadcast packet is originated by the source node 0 in both protocols,  $t'_0$  is replaced with  $t_0$  in the figure. Assume,

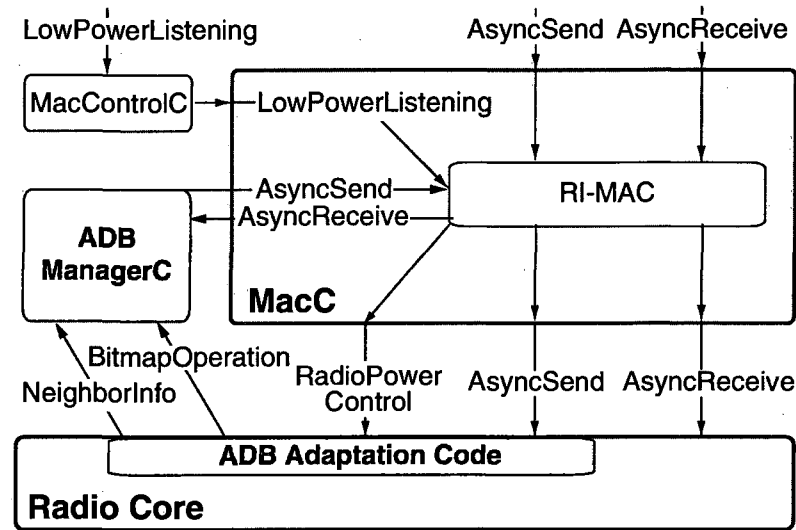
without loss of generality, that the first node that received the packet with the other algorithm before the time it would have received it with ADB, is node  $k$ , i.e.,  $t_k > t'_k$  and  $t_i \leq t'_i \ \{\forall i | t'_i \leq t'_k\}$ . For example, in Figure 7.3(a), all nodes  $\alpha, \beta, \gamma, \delta, \dots, i, j$  received the packet with ADB no later than the time they received the packet with the other protocol ( $t_\alpha \leq t'_\alpha, t_\beta \leq t'_\beta, \dots, t_j \leq t'_j$ ); node  $k$  is the first node that received the packet with the other protocol earlier than with ADB ( $t_k > t'_k$ ). Although there might be multiple nodes that receive the packet with the other protocol earlier than with ADB (e.g., node  $\xi$  in Figure 7.3(b)), we concentrate here on only the first of such node, that is  $k$ .

Assume that with the other protocol, node  $k$  received the broadcast from node  $v$  (node  $v$  could potentially be node 0). Since node  $k$  received the packet from  $v$ , then  $t'_v < t'_k$ . Hence, based on our assumption that  $k$  was the first node that received the packet with the other protocol earlier than with ADB, node  $v$  received the packet with ADB not later than when  $v$  got the packet with the other protocol, i.e.,  $t_v \leq t'_v$ . Furthermore, with ADB, a transmitter delegates a neighbor to some other node if and only if this other node wakes up and receives the broadcast packet prior to the wakeup time of the delegated neighbor as shown in Figure 7.2. This means that with ADB, from time  $t_v \leq t'_v$  until node  $k$  receives the packet, there is at all times at least one of  $k$ 's neighbors that is awake waiting for  $k$  to wake up in order to deliver the packet (it can be either  $v$  itself or some other node that has already received the packet such as node  $j$  in Figure 7.3(a)). Since a node's wakeup time



is determined independently from broadcast protocols, and since there is at least one node waiting for  $k$  to wake up in order to deliver the packet, at time  $t'_k$  when  $k$  is awake, it must either have already received the packet with ADB or it must be receiving the packet. Since there are no collisions, the packet will be delivered to node  $k$  successfully, which means that  $t_k \leq t'_k$ , contradicting the assumption that node  $k$  received the broadcast packet with the other protocol earlier than the node does with ADB. ■

Remarks: (i) The delivery times with ADB as shown in the theorem are the optimal delivery times, as a node receives the packet as soon as it wakes up, and one of its neighbors has already received the broadcast packet. (ii) The assumption of zero packet transmission duration implies that nodes do not defer due to other transmissions; in a real system, the duration to transmit a packet is relatively small compared to a duty cycle interval, so the probability that two neighboring nodes wakes up within a packet transmission is low. It is true that the denser the network, the more likely that nodes will have to defer due to other transmissions. In the worst case, a transmitter cannot start transmission even if the intended receiver wakes up, as the transmitter is deferring due to some other transmissions. In this case, the receiver goes back to sleep immediately, unaware of the waiting transmitter, resulting in a penalty in latency of one duty cycle. (iii) ADB is not immune to collisions, but ADB can take advantage of the collision resolution mechanism of the underlying RI-MAC, resulting in small delivery latency.



**Figure 7.4.** Interaction between ADB, RI-MAC, and the UPMA framework in TinyOS.

## 7.5 ADB Implementation in TinyOS

I implemented ADB under the UPMA framework [20] of TinyOS, integrated with the RI-MAC implementation described in Section 5.7. I tested this implementation on MICAz motes equipped with CC2420 radios [5]; these radios are also used in the popular TelosB motes. Figure 7.4 shows how the portions of the ADB implementation interact with RI-MAC and the UPMA framework.

The *ADBManagerC* module in Figure 7.4 provides most of the functionality of ADB, maintaining the neighbor list and status, encoding and decoding ADB footers, checking whether neighbors have been reached or delegated, and deciding whether to transmit to a neighbor that has just woken up. This module uses the *AsyncSend* and *AsyncReceive* interfaces to distribute and receive neighbor lists through RI-MAC. In order to minimize processing delays of ADB operations, I

added additional code to the radio core of TinyOS, indicated by *ADB Adaptation Code* in the figure. This adaptation code reports bitmaps in received frames to *ADBManagerC* directly and retrieves the most up-to-date bitmaps for packets to be transmitted. The adaptation code also notifies *ADBManagerC* of incoming beacons and ACKs for neighbor detection and link quality estimation.

I use a reserved bit in the Frame Control Field (FCF) of an IEEE 802.15.4 frame to indicate whether an ADB footer is included in the frame. I also add one byte named *bitmapslength* to the MAC header of the frame. The *bitmapslength* field gives the number of bytes used by the ADB footer in the corresponding frame. The footer is placed after the original data payload of the frame and before the Frame Check Sequence (FCS) field. Therefore, the additional number of bytes introduced by ADB is  $(bitmapslength + 1)$  if a footer is included. For a beacon with an ADB footer, the network layer source address and sequence number must be included, in order to identify the corresponding broadcast packet. In my experiments, I use 1 byte for the source address and 2 bytes for the sequence number.

Since ADB operations are handled by software and since a node goes to sleep soon after its beacon transmission, one challenge I faced in this ADB implementation is *limited time in updating destination address and footer of a pending DATA frame*. In order to update the destination or footer of the pending frame that is already in the CC2420 radio's TX buffer, ADB has to discard the packet in the buffer and load an updated frame into the buffer. Before loading, ADB also needs to secure

the SPI bus. All these operations take time, and thus ADB may not be able to start transmission before the intended receiver has gone back to sleep.

A slight increase in the waiting time after beacon transmission at each node does not completely solve this problem. This will make room for change the destination to the intended receiver, at the cost of more energy consumption at all nodes when there is not traffic. However, we may still not have enough time for footer update. As a node may update the footer of a pending frame upon receiving or overhearing frames. The footer may need multiple updates during a very short period of time in order to keep up-to-date information in the footer. If a beacon is received from an intended receiver but the updates haven't been finished, we either has to wait until the receiver wakes up next time at the cost of much larger energy consumption and delivery latency, or transmits a frame with stale information in its footer.

To optimize the implementation for most common cases, I always use broadcast address as destination in each DATA frame and do not delay DATA transmission due to pending updates. When a DATA frame is received, a node sends back an acknowledgement beacon only if the node has just transmitted a beacon and waiting for incoming packets. It is possible that two nodes send their beacons almost at the same time and both send acknowledgement beacons upon receiving a DATA frame. When these beacons collide at the transmitter of the DATA, the transmitter cannot learn the successfully delivery until it overhears transmissions

from those nodes. However, this is a rare event when a network operates with a low duty cycle configuration. If a frame with stale information in the footer is sent out, ADB could have more redundant messages or even unreached nodes. For example, node *A* transmits a DATA to node *B* and *B* has better link quality to their common neighbor *C*. In this case, node *B* will indicate the good link quality in the acknowledgement beacon to *A*. If *A* failed to receive this beacon due to link errors, both *A* and *B* will transmit to *C*, leading to collisions. Now suppose *A* has successfully received the beacon from *B* and decides not to transmit to *C*. However, before *A* finishes updating the footer of the DATA, a beacon from node *D* is received. If we let *A* start transmission immediately, with a stale footer indicating that *A* will transmit to *C*, node *B* will suppress its transmission to *C* to avoid collisions. As a result, neither *A* nor *B* will transmit to *C* any more. As such scenario was rarely observed in my experiments, and in order to avoid the large energy consumption and delivery latency at *A* when *A* waits for next beacon from *D*, I chose to allow DATA transmission with stale information in the footer. With future hardware support in efficiently updating destination and footers, all the above problem would be avoided.

## Chapter 8

### Evaluation of ADB

In this chapter, I evaluate ADB both in detailed *ns-2* simulations and in a testbed running TinyOS on MICAz motes. Simulation is used to evaluate networks that are hard to deploy and experiment with, and use the testbed in order to explore the details not completely captured by simulation.

ADB is compared with X-MAC-UPMA rather than the original X-MAC, since the X-MAC paper did not explicitly explain how broadcast is supported and its code is not available in TinyOS. The `RESEND_WITHOUT_CCA` option in UPMA is used so that when a node repeatedly transmits a DATA frame to broadcast it in X-MAC-UPMA, only the first of the sequence uses backoff before transmission. Without this option used, each DATA transmission in the sequence is subject to backoff, as is the default in the TinyOS code. However, I found that the backoffs within the sequence could often lead to unreached nodes even in a simple chain topology, and I confirmed this problem with the author of the TinyOS code [30]. The problem occurs because the sequence from a transmitter could be interrupted by a neighbor's transmissions when the transmitter is doing backoff; if an intended receiver wakes up during the transmitter's backoff, the receiver cannot detect incoming packets and thus goes to sleep immediately. Since an improved TinyOS

code to solve this problem is under construction, RESEND\_WITHOUT\_CCA is used to get the best performance for X-MAC-UPMA. Two possible schemes to support broadcast with RI-MAC are also simulated. ADB specific features are not used in these schemes, so that we can tell how much the new features of ADB contribute to performance gains. Details of these two schemes are discussed in Section 8.1.

As in prior work [3, 20], *Effective duty cycle*, the percentage of time a node has its radio on, is used in evaluating power efficiency. When a broadcast packet has reached all nodes in a network, *End-to-end delay* used to indicate the time between when the that packet was first generated and the time when the packet reaches the last node. If the packet fails to reach all nodes in a network, the end-to-end delay value is infinity and is *not* included in the following figures. In order to evaluate reliability, the percentage of nodes that have been reached by each broadcast packet is reported as *delivery ratio*.

In both the simulation and testbed evaluations, 1 second is used as the duty cycle interval for all MAC protocols, and randomize the initial wakeup time of each node. Data payload size is always 28 bytes, the default value in the UPMA package.

## 8.1 Simulation Evaluation

I use the *ns-2* network simulator to evaluate ADB's performance in 100 random networks. As with the other simulation evaluations presented in this thesis, I used version 2.29 of the *ns-2* network simulator, using the standard combined free space and two-ray ground reflection radio propagation model commonly used with *ns-2*. Each sensor node is simulated with a single omni-directional antenna. In each simulated network, 50 nodes randomly deployed in a 1000 m  $\times$  1000 m area. Each of these networks is connected. In each network, a random node is chosen as sink, which initiates 100 broadcast packets during each run. The interval between two consecutive broadcast originations is 100 seconds so that all forwarding for one packet completes before the next packet is originated. The simulation uses the default *ns-2* combined free space and two-ray ground reflection radio model and the same radio parameters used in RI-MAC's evaluation shown in Section 6, in order to simulation the CC2420 radio used in popular MICAz and TelosB motes.

ADB is compared with X-MAC-UPMA in each random network in supporting the 100 network-wide broadcasts. With X-MAC-UPMA, Two versions of X-MAC-UPMA are considered. First, in the standard version of the protocol, which is referred to as *X-MAC-UPMA-1* in the rest of this work, a transmitter of a broadcast packet transmits the packet repeatedly over the duration of one duty cycle. Second, as discussed in Section 2.3, X-MAC-UPMA could experience collisions caused by transmissions from hidden nodes. In order to compensate for the



packet losses caused by these collisions, each node transmits the packet over two different duty cycles, indicated by *X-MAC-UPMA-2*: the first transmission cycle takes place in the same way as in *X-MAC-UPMA-1*, and the second takes place after a randomly chosen delay, up to 5 duty cycle intervals, following the first one.

As ADB is integrated into RI-MAC, in order to show the advantage brought by ADB-specific features, such as delegation and the ability to adapt to link qualities, I simulated two schemes for broadcasting for RI-MAC, by varying the amount of effort each node spends in attempting to reach its neighbors. In the first scheme, when a node receives a new broadcast packet, the node stays awake for  $1.5 \times T$  (RI-MAC varies duty cycle interval between  $0.5 \times T$  and  $1.5 \times T$ ), during which each neighbor generates at least one beacon. When a beacon is received from a neighbor, the node unicasts the packet to the neighbor and waits for an ACK corresponding to this packet. If an ACK is received, the node does not attempt to transmit the packet to the same neighbor again during this time. After staying awake for  $1.5 \times T$ , the node discards the packet and goes to sleep if the medium is idle. This scheme is referred to as *RI-MAC-1.5*. A duration of 1.5 duty cycles ( $1.5 \times T$ ) was chosen as that is the minimum duration the node has to stay awake in order to be able to receive a beacon from all neighbors. This duration may be too short for reliable packet deliveries in case some beacons or DATA frames are lost, so in the second scheme, this duration is increased to  $4.5 \times T$ ; in the rest of this thesis, this scheme is referred to as *RI-MAC-4.5*.

With the default channel model of *ns-2*, if a receiver is within 250-meters of a sender, packets from the sender will be successfully received by the receiver unless there is a collision. Results in this channel model are shown in section 8.1.1. In a real network, in addition to collisions, errors caused by factors such as wireless fading and interference could also cause packet losses. In order to evaluate ADB's robustness and efficiency with such packet losses, in Section 8.1.2, the simulation uses a modified channel model in which additional packet losses are introduced.

#### 8.1.1 Results with Default Channel Model in *ns-2*

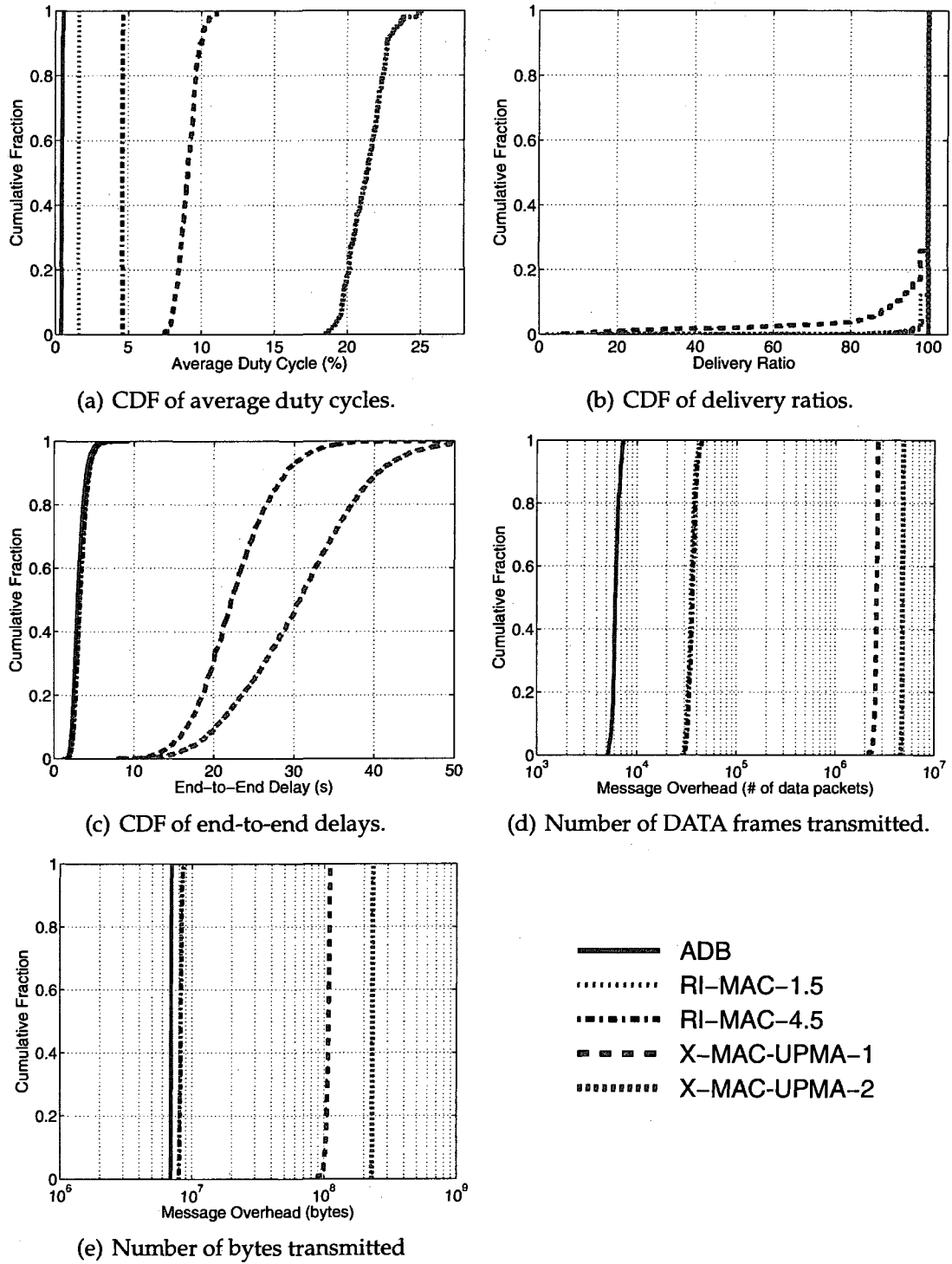
Figure 8.1 shows our simulation results with the default channel model in *ns-2*. The results are shown as cumulative distribution functions calculated based on the results from the 100 random runs.

Energy efficiency is shown in Figure 8.1(a) as average duty cycles. The average values with ADB, RI-MAC-1.5, RI-MAC-4.5, X-MAC-UPMA-1 and X-MAC-UPMA-2 are 0.46%, 1.62%, 4.61%, 9.08% and 21.30%, respectively. The energy consumption of ADB is only around 28% of that of RI-MAC-1.5, 10% of that of RI-MAC-4.5, 5% of that of X-MAC-UPMA-1, and 2% of that of X-MAC-UPMA-2.

ADB achieves such substantial savings by putting a node to sleep immediately when all of its neighbors are reached or delegated. Such optimization is not possible with RI-MAC, as a transmitter only knows which neighbors have been reached; the transmitter does not know whether *all* neighbors have been reached due to the lack of a complete neighbor list. Moreover, the transmitter with RI-MAC at-

tempts to send a broadcast packet to a neighbor, regardless whether the neighbor has received the packet or will receive a copy from some other node. With X-MAC-UPMA, a transmitter must continue sending a DATA packet for a whole duty cycle interval, as feedback from neighbors is unavailable. Moreover, overhearing consumes significant energy. Suppose a node has finished broadcasting a DATA packet, and then one neighbor starts rebroadcasting this packet. It is likely that the rebroadcast is still ongoing when this node wakes up again for its next cycle, and thus the node will receive duplicate copies of the DATA from the neighbor. This node could have used some bookkeeping to avoid receiving such duplicated broadcast packets and go to sleep immediately, but this would require careful consideration as to when to turn the node on again later. If the transmitting neighbor has some queued packets to this node, they cannot get delivered until this node wakes up again.

Figure 8.1(b) shows the packet delivery ratios achieved by these protocols. The average delivery ratios with ADB, RI-MAC-1.5, RI-MAC-4.5, X-MAC-UPMA-1 and X-MAC-UPMA-2 are 100%, 99.64%, 100%, 96.55% and 99.78%, respectively. ADB always achieved 100% delivery ratios, as ADB footers provide information on the progress of a multihop broadcast, which reduces many redundant transmissions that could cause collisions. These collisions are not avoided with RI-MAC. Sometimes collisions caused by transmissions from multiple neighbors to a node cannot be resolved in time and the node goes to sleep after that. When the node



**Figure 8.1.** Performance comparison in 50-node networks with default channel model in *ns-2*.

wakes up again, all of its neighbors have finished their broadcasts and gone to sleep already. This is why RI-MAC-1.5 experienced some undelivered packets. With longer waiting time and thus more effort spent in delivering a broadcast packet, RI-MAC-4.5 allows a node to receive the broadcast packet when it wakes up again, which helped to improve the delivery ratio. However, this improvement comes at the cost of much increased energy consumption, as shown in Figure 8.1(a). X-MAC-UPMA shows the worst delivery ratios, as a node may miss an incoming packet due to collisions caused by overlapping transmissions from hidden nodes, as discussed in Section 2.3. By rebroadcasting each newly received broadcast packet over two duty cycles with random backoffs, X-MAC-UPMA-2 improves delivery ratios, but the improvement also comes at the cost of much more energy consumption.

Figure 8.1(c) shows the CDF of end-to-end delays for all packets in the 100 runs. The average values with ADB, RI-MAC-1.5, RI-MAC-4.5, X-MAC-UPMA-1 and X-MAC-UPMA-2 are 3.08, 3.34, 3.39, 22.61 and 30.61 seconds, respectively. ADB shows an average end-to-end latency that is around 14% of that of X-MAC-UPMA-1, and 10% of that of X-MAC-UPMA-2. With ADB, because a transmitter occupies the wireless medium only for a small amount of time, neighbors of this transmitter can start rebroadcasting the received packet immediately, whereas with X-MAC-UPMA, the neighbors have to wait until the end of the long transmitting sequence from the current transmitter. The long repeated transmission sequences

may even block transmission of nodes that are not direct neighbors of the current transmitter when they can sense the busy medium caused by the transmitting sequence. RI-MAC-1.5 and RI-MAC-4.5 show only slightly larger end-to-end delays than does ADB. The extra delays are mainly caused by collisions. As collisions can be quickly solved by RI-MAC, the extra delays are small.

Figure 8.1(d) shows the number of DATA frames transmitted over the air with these protocols. Due to the wide range among the results, a log scale is used for the  $x$ -axis. The average numbers with ADB, RI-MAC-1.5, RI-MAC-4.5, X-MAC-UPMA-1 and X-MAC-UPMA-2 are  $6.19\text{e}+3$ ,  $3.48\text{e}+4$ ,  $3.62\text{e}+4$ ,  $2.61\text{e}+6$  and  $4.79\text{e}+6$ , respectively. ADB shows the lightest network load in this set of experiments. The number with ADB is only 18% of that with RI-MAC-1.5 and 17% of that with RI-MAC-4.5, for two reasons. First, delegation in ADB greatly helps in reducing redundant transmissions. Second, the reduced redundancy also helps to reduce collisions and thus the number of retransmissions. There are significantly more DATA frames transmitted over the air with X-MAC-UPMA-1 and X-MAC-UPMA-2, as copies of a broadcast packet must be repeatedly transmitted for 1 or 2 duty cycle intervals, respectively, from each node.

Besides DATA frames, beacons are also transmitted over the air with ADB, RI-MAC-1.5, and RI-MAC-4.5. For a fair comparison, Figure 8.1(e) shows the total number of bytes transmitted by each protocol, which include all DATA and control frames. The average numbers with ADB, RI-MAC-1.5, RI-MAC-4.5,

X-MAC-UPMA-1 and X-MAC-UPMA-2 are  $6.96\text{e}+6$ ,  $8.18\text{e}+6$ ,  $8.24\text{e}+6$ ,  $1.07\text{e}+8$  and  $2.31\text{e}+8$  bytes. ADB still shows the lightest network load among these protocols.

### 8.1.2 Results with Increased Packet Losses

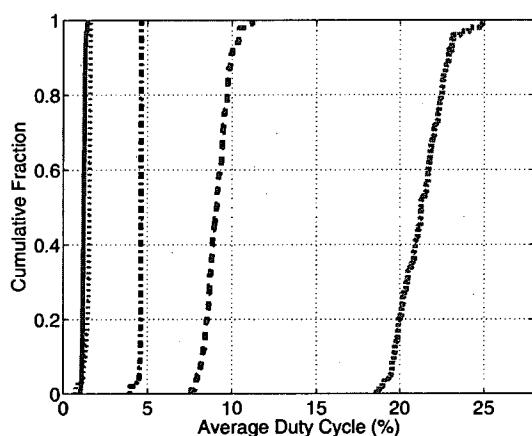
In a real wireless sensor network, a packet may be lost due to errors caused by factors such as wireless fading and interference. To evaluate the effect of such increased packet losses, a simple model is used to introduce random losses based on the distance between transmitter and receiver, as longer distances will generally result in lower received signal strength and thus increased probability of loss. In these simulations, a link with a span of 0 meters has 0% probability of additional packet loss, and a link with a span of 250 meters has 50% probability of additional loss; these probabilities refer to the random losses introduced by this modified channel model, beyond those caused by any collisions in the default *ns-2* channel model, for each individual transmission (e.g., of a DATA frame or an ACK). Then linear interpolation is used to calculate the probability of loss based on a link's span. The maximum communication range possible is still 250 meters, the default value in *ns-2*. The results with this modified channel model are shown as CDFs in Figure 8.2.

Figure 8.2(a) shows the average duty cycles. The average values with ADB, RI-MAC-1.5, RI-MAC-4.5, X-MAC-UPMA-1 and X-MAC-UPMA-2 are 1.22%, 1.50%, 4.60%, 9.11% and 21.29%, respectively. Compared with the results using the default *ns-2* channel model shown in Figure 8.1(a), all protocols except for

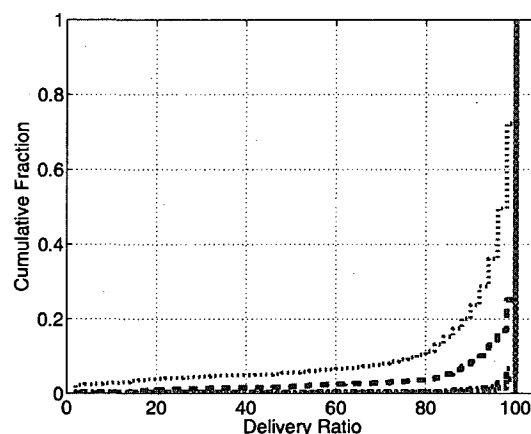
ADB show similar energy efficiency because each node stays awake for essentially a fixed amount of time regardless of channel condition. Some runs even show smaller energy consumption between RI-MAC-1.5 and RI-MAC-4.5, since significantly more packets fail to reach the whole network, as shown in Figure 8.2(b); thus, some nodes do not receive the packets and thus do not stay awake to rebroadcast them. As ADB adapts to link qualities, ADB attempts more retransmissions, as needed, in order to compensate for increased packet losses. ADB thus consumes more energy with this channel model than it does with the default model as shown in Figure 8.1(a). Even though, ADB still shows the lowest average duty cycle among these protocols.

With increased packet losses over the wireless channel, ADB still maintains 100% delivery ratios, as shown in Figure 8.2(b). Average delivery ratios achieved by the other protocols, RI-MAC-1.5, RI-MAC-4.5, X-MAC-UPMA-1 and X-MAC-UPMA-2 are 90.09%, 99.33%, 96.70% and 99.78%, respectively. Delivery ratios with RI-MAC-1.5 and RI-MAC-4.5 decrease as these protocols do not adapt to channel conditions. With more redundancy, RI-MAC-4.5 shows much higher delivery ratios than RI-MAC-1.5, with the trade-off that RI-MAC-4.5 consumes more energy. Both X-MAC-UPMA-1 and X-MAC-UPMA-2 show almost the same performance compared to the results in Figure 8.1(b), because of extraordinary redundancy in their DATA transmissions. Even with increased packet losses, when a DATA is retransmitted repeatedly for a whole duty cycle in X-MAC-UPMA-1,

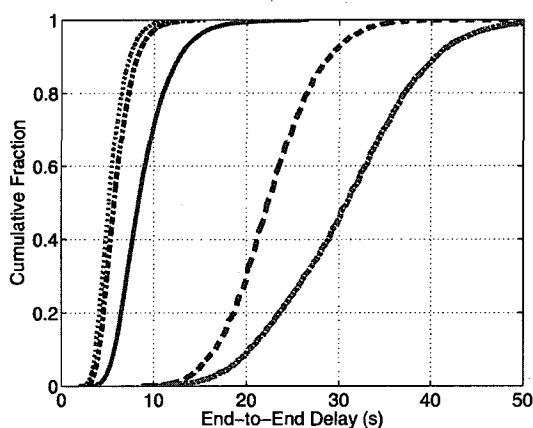




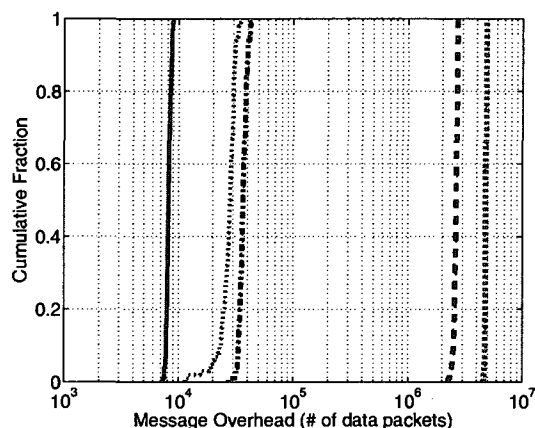
(a) CDF of average duty cycles.



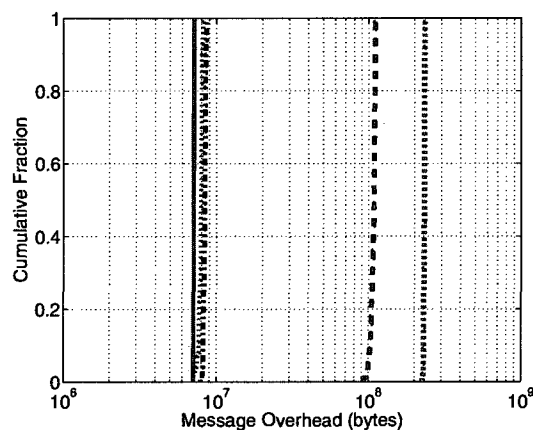
(b) CDF of delivery ratios.



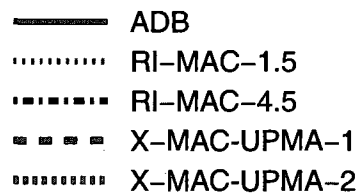
(c) CDF of end-to-end delays.



(d) Number of DATA frames transmitted.



(e) Number of bytes transmitted



**Figure 8.2.** Performance comparison in 50-node networks with increased packet losses.

a receiver is very likely to get at least one copy of the DATA; X-MAC-UPMA-2 increases that likelihood. Therefore, broadcast using X-MAC-UPMA is more robust to packet losses, but this redundancy still causes many collisions, resulting in lower delivery ratios than with ADB.

Figure 8.2(c) shows the CDF of end-to-end delays for all packets in the 100 runs. The average end-to-end delays with ADB, RI-MAC-1.5, RI-MAC-4.5, X-MAC-UPMA-1 and X-MAC-UPMA-2 are 8.89, 5.38, 5.99, 22.63 and 30.69 seconds, respectively. ADB, RI-MAC-1.5 and RI-MAC-4.5 show much longer end-to-end latency than they do with the default channel model in *ns-2*, since when a beacon from the intended receiver is lost, a transmitter has to wait until another beacon arrives in the next cycle. With extraordinary redundancy, X-MAC-UPMA-1 and X-MAC-UPMA-2 show similar results to those in Figure 8.1(c), for the same reason I've discussed above. RI-MAC-1.5 and RI-MAC-4.5 show lower end-to-end latency than does ADB for this channel model since the latency is not included for any broadcast packets that have not reached all nodes; For those packets, which occur with RI-MAC but not with ADB, the end-to-end delay is *infinity*.

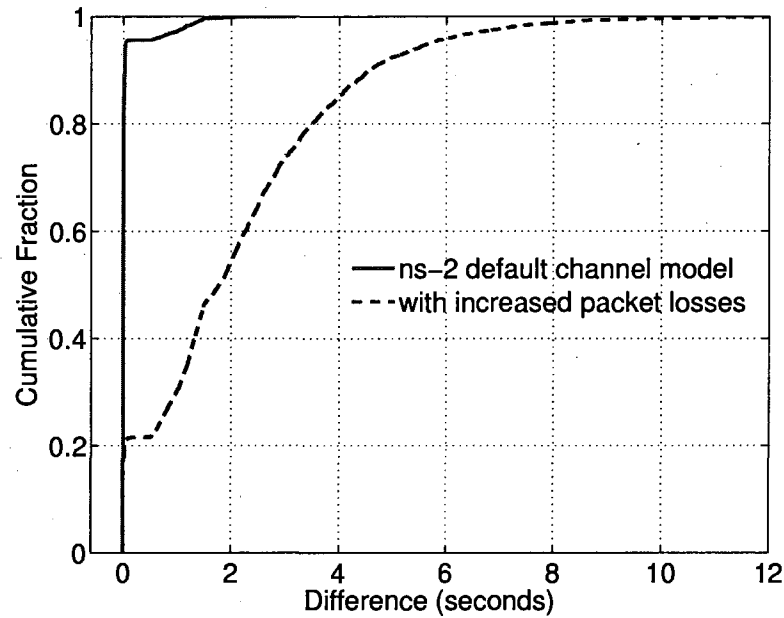
Finally, Figure 8.2(d) shows the overhead in terms of number of DATA frames transmitted from all nodes, and Figure 8.2(e) shows the total number of bytes transmitted with each protocol. The average number of DATA transmissions with ADB, RI-MAC-1.5, RI-MAC-4.5, X-MAC-UPMA-1 and X-MAC-UPMA-2 are  $8.20\text{e}+3$ ,  $2.73\text{e}+4$ ,  $3.65\text{e}+4$ ,  $2.61\text{e}+6$  and  $4.79\text{e}+6$ , respectively. Compared with the

results in the default channel model, more DATA frames have been transmitted with ADB, RI-MAC-1.5 and RI-MAC-4.5 due to more retransmissions. In some random runs RI-MAC-1.5 shows a much smaller value than its average value, as many packets fail to reach all nodes in the network, and thus there are fewer rebroadcasts. The similar trend can be observed for the number of bytes transmitted (Figure 8.2(e)). The average numbers of bytes transmitted with ADB, RI-MAC-1.5, RI-MAC-4.5, X-MAC-UPMA-1 and X-MAC-UPMA-2 are  $7.12\text{e}+6$ ,  $7.90\text{e}+6$ ,  $8.38\text{e}+6$ ,  $1.07\text{e}+8$  and  $2.31\text{e}+8$ , respectively.

### 8.1.3 Comparison to Optimal Latency

Figure 8.3 shows the difference between the delivery latency achieved by ADB and the optimal delivery latency to each node. The optimal delivery latencies are calculated based on the topology of a network, wakeup schedules of each node, and origination time of each broadcast; transmission delay, link errors and collisions are ignored in calculating the optimal delivery latency. The latencies achieved by ADB used in this figure are those for the 100 broadcast packets originated during one randomly selected run for each wireless channel model described above; the default *ns-2* model (Section 8.1.1) and the model with increased packet losses (Section 8.1.2).

With the default *ns-2* channel model, the differences from optimal are essentially 0 for more than 80% of the packet deliveries. For around 15% of the packet deliveries, the latencies with ADB are slightly larger due to the delays for ADB to



**Figure 8.3.** Difference between optimal delivery latency to each node and that with ADB.

resolve collisions. For the remaining 5%, the differences increase almost uniformly between 0.5 and 1.5 seconds. These increases occur when a transmitter missed the opportunity to deliver a packet immediately when an intended receiver wakes up, either due to failure to receive the beacon from the receiver or due to busy medium around the transmitter which stops the transmitter from transmitting the packet in time. Once the transmitter is unable to deliver the packet while the receiver is still awake, the transmitter must wait until the receiver wakes up for the next cycle. The next wakeup time is a uniform distribution between 0.5 and 1.5 seconds, which matches well with the distribution of the large differences at the top of the figure. When using the modified channel model with increased packet

losses, delivery latencies get larger due to the greater number of lost beacons and DATA frame transmissions.

## 8.2 Experimental Evaluation on MICAz Motes

In order to explore platform-dependent issues and details not completely captured in simulation, I implemented ADB in TinyOS and evaluated it on MICAz motes in a clique network and in a random network. As described in Section 7.5, this ADB implementation uses the UPMA framework [20, 43] in TinyOS, integrated with the RI-MAC unicast module [41]. The configuration of payload size and duty cycle interval are the same as those in the simulations described in Section 8.1.

In each experiment with the testbed, no DATA packets are generated for the first 2 minutes, during which time ADB collects and distributes information for the neighbor lists. Clock synchronization among nodes is also done during this time for later trace analysis; this clock synchronization is not used by the protocols. As all the operations during the first 2 minutes happen only once during the lifetime of a network and are protocol dependent, count energy consumption and message overhead are not counted during this time. After this initialization, the sink node periodically originates a broadcast DATA packet, for a total of 75 originated broadcast packets.

### 8.2.1 Results in a Clique Network

I first present the experimental results for a clique network of 5 nodes, where all nodes are placed close to each other. one random node is chosen as sink, which originates a broadcast packet every 10 seconds. This interval is large enough to ensure that a new broadcast packet is originated only after all transmissions of the previous broadcast packet have finished.

The measured performance for X-MAC-UPMA and ADB in this clique network is shown in Table 8.1. The average duty cycle with ADB is only 6.2% of that with X-MAC-UPMA, since a node with ADB goes to sleep immediately once all its neighbors are either reached or delegated for a given broadcast, but X-MAC-UPMA must repeatedly transmit the DATA over an entire duty cycle. Both ADB and X-MAC-UPMA achieve 100% delivery ratio, but ADB uses much less energy. The average delivery latencies with both X-MAC-UPMA and ADB are about half a duty cycle interval, as nodes wakes up asynchronously. ADB shows slightly larger latency, which is mainly due to the difference in generated random numbers that determine the wakeup schedules of each node.

As a node with X-MAC-UPMA repeatedly transmits copies of a DATA packet for an entire duty cycle interval, many more frames are transmitted over the air compared with ADB. The number of bytes transmitted with ADB is only 2.3% of that with X-MAC-UPMA, substantially reducing channel contention and leaving additional capacity for traffic from other nodes, if needed. There are 300 total

**Table 8.1.** Performance comparison in a 5-node TinyOS clique network

	X-MAC-UPMA	ADB
Average duty cycle (%)	53.47	3.36
Delivery ratio	100	100
Average latency (s)	0.53	0.60
Message overhead (bytes)	2,875,125	65,586
DATAs transmitted	70,125	300
ACK beacons transmitted	–	302
Other beacons transmitted	–	3,332

DATA frame transmissions with ADB, translating to exactly 4 DATA frame transmissions (one to each non-sink node) per originated broadcast packet. This result shows that ADB efficiently avoids redundant transmissions.

The total number of ACKs is 302 rather than 300 because, following 2 different DATA transmissions, two nodes returned an ACK rather than just one node. As discussed in Section 7.5, The implementation uses a broadcast destination address for all DATA transmissions. Infrequently, a node may mistakenly believe the DATA was intended for it and return an ACK in addition to the ACK from the intended receiver, causing a collision. Such a collision is not very harmful, as the receiver has received the DATA and the following beacon or a frame with an ADB footer from that receiver will notify the transmitter of this.

### 8.2.2 Results in a Random Network

Finally, I also evaluated ADB in a multihop random network deployed in an apartment, as show in Figure 8.4. Each node is placed below a wall power outlet,

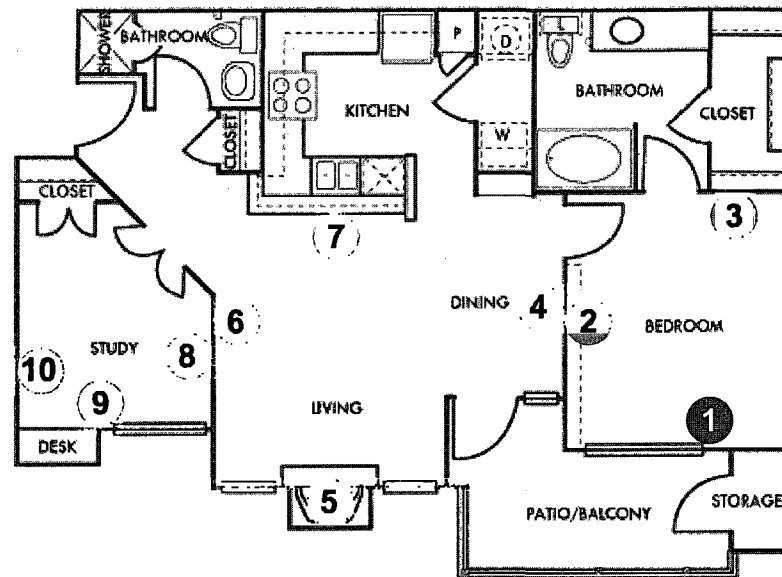


Figure 8.4. Topology of a 10-node random network deployed in an apartment.

in order imitate (for example) a sensor network deployment for monitoring energy consumption of household appliances. Node 1 is the sink and generates one broadcast packet every 20 seconds. This interval is twice the interval used for the clique network above, as the number of nodes is doubled. In this way, we ensure the transmissions for one broadcast packet have finished before another broadcast packet is originated.

The performance of X-MAC-UPMA and ADB in this 10-node network is shown in Table 8.2. As it is difficult to sniff all packets in a multihop network, the message overhead is not compared here. The average duty cycle with ADB is about 10% of that with X-MAC-UPMA. Unlike the results in the clique network, ADB shows a smaller average delivery latency than with X-MAC-UPMA, as each node occu-



**Table 8.2.** Performance comparison in the 10-node TinyOS network

	X-MAC-UPMA	ADB
Average duty cycle (%)	27.00	2.77
Delivery ratio	99.47	99.47
Average latency (s)	0.71	0.64

pies the medium for a duty cycle interval with X-MAC-UPMA, which introduces longer delay before next node can begin forwarding the packet.

Both X-MAC-UPMA and ADB achieved high packet delivery ratio of 99.47%. X-MAC-UPMA was able to maintain a high delivery ratio since in this network, it is unlikely to have transmissions from hidden nodes, avoiding the problems from collisions observed in the larger networks in the simulations. Unlike in the simulations where ADB always achieved 100% delivery ratio, a few packets failed to reach all nodes in the experiments. This is due to the design choice made in my TinyOS implementation that allows a footer with stale information to be transmitted. This design choice is made to reduce delivery latency and energy consumption as discussed in Section 7.5. In simulations, no delay is assumed in updating ADB footers; with hardware support for quickly updating destination and ADB footers, the above problems should be eliminated in the implementation. Even with platform-specific limitations, ADB still achieves the same delivery ratio with X-MAC-UPMA with much less energy consumption.

**Table 8.3.** Average duty cycle % of each node in the 10-node TinyOS network

Node ID	1	2	3	4	5	6	7	8	9	10
X-MAC-UPMA	22.8	25.8	25.5	28.2	24.9	26.9	28.2	29.3	29.6	28.8
ADB	7.2	4.8	2.5	1.9	3.6	1.5	1.7	1.1	2.0	1.4

Table 8.3 shows average duty cycle at each node. With X-MAC-UPMA, the average duty cycles are all above 22%, mainly due to unnecessary overhearing. Nodes 1, 2, and 3 show slightly lower energy consumption, as they are at one side of the network that has lower density, and thus they overhear fewer redundant transmissions. Nodes at the other side of the network (nodes 6, 7, 8, 9 and 10), however, show higher energy consumption due to increased overhearing.

## Chapter 9

### Conclusions

In this thesis, I have presented both a synchronous duty cycle MAC protocol, DW-MAC, and an asynchronous duty cycle MAC protocol, RI-MAC, which are designed to efficiently operate under a wide range of traffic loads. In addition, I have presented a multihop broadcast protocol, ADB, to efficiently distribute small messages in a wireless sensor network using asynchronous duty cycling.

DW-MAC is an energy efficient protocol designed to reduce packet delivery latency for a wide range of traffic loads, including both unicast and broadcast traffic. Compared to prior protocols, DW-MAC adaptively increases effective channel capacity during an operational cycle as traffic load increases, allowing DW-MAC to achieve low delivery latency under dynamic traffic loads. The scheduling algorithm in DW-MAC integrates scheduling and access control to maintain a proportional one-to-one mapping function between a Data period and the subsequent Sleep period, which minimizes scheduling overhead while ensuring that data transmissions do not collide at their intended receivers. I compared DW-MAC with S-MAC (with and without adaptive listening) and with RMAC through extensive simulations. I found that DW-MAC outperforms these protocols, with lower latency, higher power efficiency, and higher packet delivery ratios, and with increas-

ing benefits as traffic load increases. For example, under high unicast traffic load, DW-MAC reduces delivery latency by 70% compared to S-MAC and RMAC, and uses only 50% of the energy consumed with S-MAC with adaptive listening. Under broadcast traffic, DW-MAC reduces latency by more than 50% on average, always reducing energy consumption by more than 15%. In addition, DW-MAC improves packet delivery ratios under all scenarios in my simulations.

RI-MAC uses receiver initiated data transmission in order to efficiently and effectively operates over a wide range of traffic loads. To achieve this, RI-MAC attempts to minimize the time a sender and its intended receiver occupy the wireless medium to find a rendezvous time for exchanging data, while still decoupling the sender and receiver's duty cycle schedules. I evaluated RI-MAC through detailed *ns-2* simulation and through measurements of an implementation in TinyOS in a testbed of MICAz motes. Compared to X-MAC, RI-MAC achieves higher throughput, packet delivery ratio, and power efficiency under a wide range of traffic loads. Especially when there are contending flows, such as bursty traffic or transmissions from hidden nodes, RI-MAC significantly improves throughput and packet delivery ratio. In my experimental evaluation in my TinyOS testbed, when there are 4 contending flows in clique networks, RI-MAC improves throughput by 100%, reduces delivery latency by 90%, and reduces duty cycle by 50% at sending nodes compared to X-MAC. In the 3-node network with hidden senders, RI-MAC achieves 0 packet loss compared to the more than 15% packet loss in X-MAC. Sim-

ilar trends were also observed in my simulations for large networks. Even under light traffic load for which X-MAC is optimized, RI-MAC achieves the same high performance.

Finally, ADB provides efficient multihop broadcast support, despite the challenges of broadcast over asynchronous duty cycling. ADB optimizes the progress of a broadcast at the level of transmission from a node to each of its neighbors individually. Information about the progress is efficiently distributed, based on which ADB uses delegation to avoid redundant transmissions and transmissions over poor links. In my evaluation of ADB using *ns-2* simulation in 100 random networks, compared to network-wide broadcast with X-MAC-UPMA and RI-MAC, ADB shows much higher energy efficiency, 100% delivery ratio, and lowest network load. I also implemented ADB in TinyOS on a testbed of MICAz motes and evaluated it in a clique network and a multihop random network. Compared to an implementation of multihop broadcast X-MAC-UPMA, ADB shows much higher energy efficiency and significantly reduces network load, while maintaining low delivery latency and high packet delivery ratio.

## Bibliography

- [1] Muneeb Ali, Umar Saif, Adam Dunkels, Thiemo Voigt, Kay Römer, Koen Langendoen, Joseph Polastre, and Zartash Afzal Uzmi. Medium Access Control Issues in Sensor Networks. *SIGCOMM Computer Communications Review*, 36(2):33–36, 2006.
- [2] G. Anastasi, A. Falchi, A. Passarella, M. Conti, and E. Gregori. Performance Measurements of Motes Sensor Networks. In *Proceedings of the 7th ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM 2004)*, pages 174–181, October 2004.
- [3] Michael Buettner, Gary V. Yee, Eric Anderson, and Richard Han. X-MAC: A Short Preamble MAC Protocol for Duty-Cycled Wireless Sensor Networks. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, pages 307–320, 2006.
- [4] Qing Cao, Tarek Abdelzaher, Tian He, and John Stankovic. Towards Optimal Sleep Scheduling in Sensor Networks for Rare-Event Detection. In *Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks (IPSN 2005)*, pages 20–27, April 2005.

- [5] CC2420 Datasheet. <http://www.ti.com>.
- [6] Chipcon. Single Chip Very Low Power RF Transceiver (CC1000 Datasheet), April 2002.
- [7] Crossbow MICAz motes. <http://www.xbow.com>.
- [8] Tijs van Dam and Koen Langendoen. An Adaptive Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *Proceedings of the First International Conference On Embedded Networked Sensor Systems (SenSys 2003)*, pages 171–180, November 2003.
- [9] Shu Du, Amit Kumar Saha, and David B. Johnson. RMAC: A Routing-Enhanced Duty-Cycle MAC Protocol for Wireless Sensor Networks. In *Proceedings of the 26th Annual IEEE Conference on Computer Communications (INFOCOM 2007)*, pages 1478–1486, May 2007.
- [10] Amre El-Hoiydi and Jean-Dominique Decotignie. WiseMAC: An Ultra Low Power MAC Protocol for Multi-hop Wireless Sensor Networks. In *Proceedings of the First International Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS 2004)*, *Lecture Notes in Computer Science, LNCS 3121*, pages 18–31, July 2004.

- [11] Amre El-Hoiydi and Jean-Dominique Decotignie. Low Power Downlink MAC Protocols for Infrastructure Wireless Sensor Networks. *Mobile Networks and Applications*, 10(5):675–690, 2005.
- [12] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-Grained Network Time Synchronization using Reference Broadcasts. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, pages 147–163, December 2002.
- [13] Deborah Estrin, Ramesh Govindan, John Heidemann, and Satish Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks. In *Proceedings of the Fifth Annual International Conference on Mobile Computing and Networking (MobiCom 1999)*, pages 263–270, August 1999.
- [14] Rajiv Gandhi, Srinivasan Parthasarathy, and Arunesh Mishra. Minimizing Broadcast Latency and Redundancy in Ad Hoc Networks. In *Proceedings of the Fourth ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2003)*, pages 222–232, June 2003.
- [15] Saurabh Ganeriwal, Ram Kumar, and Mani B. Srivastava. Timing-Sync Protocol for Sensor Networks. In *Proceedings of the First International Conference On Embedded Networked Sensor Systems (SenSys 2003)*, pages 138–149, November 2003.



- [16] J. J. Garcia-Luna-Aceves and Asimakis Tzamaloukas. Reversing the Collision-Avoidance Handshake in Wireless Networks. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 120–131, 1999.
- [17] Bret Hull, Kyle Jamieson, and Hari Balakrishnan. Mitigating Congestion in Wireless Sensor Networks. In *Proceedings of the Second International Conference On Embedded Networked Sensor Systems (SenSys 2004)*, pages 134–147, November 2004.
- [18] Kyle Jamieson, Hari Balakrishnan, and Y.C. Tay. Sift: a MAC Protocol for Event-Driven Wireless Sensor Networks. In *Third European Workshop on Wireless Sensor Networks (EWSN)*, 2006.
- [19] Abtin Keshavarzian, Huang Lee, and Lakshmi Venkatraman. Wakeup Scheduling in Wireless Sensor Networks. In *Proceedings of the Seventh ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2006)*, pages 322–333, May 2006.
- [20] Kevin Klues, Gregory Hackmann, Octav Chipara, and Chenyang Lu. A Component-Based Architecture for Power-Efficient Media Access Control in Wireless Sensor Networks. In *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems*, pages 59–72, 2007.

- [21] Andrzej Kochut, Arunchandar Vasan, A. Udaya Shankar, and Ashok Agrawala. Sniffing Out the Correct Physical Layer Capture Model in 802.11b. In *Proceedings of the Network Protocols, 12th IEEE International Conference (ICNP 2004)*, pages 252–261, 2004.
- [22] Sandeep S. Kulkarni and Mahesh Arumugam. TDMA Service for Sensor Networks. In *Proceedings of the 24th International Conference on Distributed Computing Systems Workshops (ICDCSW 2004)*, pages 604–609, March 2004.
- [23] Jeongkeun Lee, Wonho Kim, Sung-Ju Lee, Daehyung Jo, Jiho Ryu, Taekyoung Kwon, and Yanghee Choi. An Experimental Study on the Capture Effect in 802.11a Networks. In *Proceedings of the the Second ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WinTECH 2007)*, pages 19–26, 2007.
- [24] Philip Levis. TEP 111: message.t. TinyOS 2.0 Documentation, <http://www.tinyos.net/tinyos-2.x/doc/>.
- [25] Philip Levis, Neil Patel, David Culler, and Scott Shenker. Trickle: a self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation (NSDI 2004)*, 2004.

- [26] Yuan Li, Wei Ye, and John Heidemann. Energy and Latency Control in Low Duty Cycle MAC Protocols. In *Proceedings of the 2005 IEEE Wireless Communications and Networking Conference (WCNC 2005)*, March 2005.
- [27] Kaisen Lin and Philip Levis. Data Discovery and Dissemination with DIP. In *Proceedings of the 7th international conference on Information processing in sensor networks (IPSN 2008)*, pages 433–444, 2008.
- [28] Gang Lu, Bhaskar Krishnamachari, and Cauligi S. Raghavendra. An Adaptive Energy-Efficient and Low-Latency MAC for Data Gathering in Wireless Sensor Networks. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*, April 2004.
- [29] Gang Lu, Narayanan Sadagopan, Bhaskar Krishnamachari, and Ashish Goel. Delay Efficient Sleep Scheduling in Wireless Sensor Networks. In *Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2005)*, pages 2470–2481, March 2005.
- [30] David Moss. personal communication, 2008.
- [31] David Moss, Jonathan Hui, Philip Levis, and Jung Il Choi. TEP 126: CC2420 Radio Stack. <http://www.tinyos.net/tinyos-2.x/doc/>, 2007.
- [32] Razvan Musaloiu-E., Chieh-Jan Mike Liang, and Andreas Terzis. Koala: Ultra-Low Power Data Retrieval in Wireless Sensor Networks. In *Proceedings*

*of the 2008 International Conference on Information Processing in Sensor Networks (IPSN 2008)*, pages 421–432, April 2008.

- [33] Sze-Yao Ni, Yu-Chee Tseng, Yuh-Shyan Chen, and Jang-Ping Sheu. The Broadcast Storm Problem in a Mobile Ad Hoc Network. In *Proceedings of the Fifth Annual International Conference on Mobile Computing and Networking (MobiCom 1999)*, pages 151–162, August 1999.
- [34] Stefan Pleisch, Mahesh Balakrishnan, Ken Birman, and Robbert van Renesse. MISTRAL: Efficient Flooding in Mobile Ad-Hoc Networks. In *Proceedings of the Seventh ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2006)*, pages 1–12, May 2006.
- [35] Joseph Polastre, Jason Hill, and David Culler. Versatile Low Power Media Access for Wireless Sensor Networks. In *Proceedings of the Second International Conference On Embedded Networked Sensor Systems (SenSys 2004)*, pages 95–107, November 2004.
- [36] Nazanin Rahnavard and Faramarz Fekri. CRBcast: A Collaborative Rateless Scheme for Reliable and Energy-Efficient Broadcasting in Wireless Sensor Networks. In *Proceedings of the Fifth International Conference on Information Processing in Sensor Networks (IPSN 2006)*, pages 276–283, April 2006.

- [37] Venkatesh Rajendran, Katia Obraczka, and J. J. Garcia-Luna-Aceves. Energy-Efficient Collision-Free Medium Access Control for Wireless Sensor Networks. In *Proceedings of the First International Conference On Embedded Networked Sensor Systems (SenSys 2003)*, pages 181–192, November 2003.
- [38] Victor Shnayder, Mark Hempstead, Bor-rong Chen, Geoff Werner Allen, and Matt Welsh. Simulating the Power Consumption of Large-Scale Sensor Network Applications. In *Proceedings of the Second International Conference On Embedded Networked Sensor Systems (SenSys 2004)*, pages 188–200, November 2003.
- [39] Fred Stann, John Heidemann, Rajesh Shroff, and Muhammad Zaki Murtaza. RBP: Robust Broadcast Propagation in Wireless Networks. In *Proceedings of the Fourth International Conference On Embedded Networked Sensor Systems (SenSys 2006)*, pages 85–98, October 2006.
- [40] Fred Stann, John Heidemann, Rajesh Shroff, and Muhammad Zaki Murtaza. RBP: Robust Broadcast Propagation in Wireless Networks. In *Proceedings of the Fourth International Conference On Embedded Networked Sensor Systems (SenSys 2006)*, pages 85–98, October 2006.
- [41] Yanjun Sun, Omer Gurewitz, and David B. Johnson. RI-MAC: A Receiver Initiated Asynchronous Duty Cycle MAC Protocol for Dynamic Traffic Loads in

- Wireless Sensor Networks. In *SenSys '08: Proceedings of the 6th ACM Conference on Embedded Networked Sensor Systems*, 2008.
- [42] Y.C. Tay, Kyle Jamieson, and Hari Balakrishnan. Collision-Minimizing CSMA and its Applications to Wireless Sensor Networks. *IEEE Journal on Selected Areas in Communications*, 22(6), 2004.
- [43] UPMA Package. <http://tinysos.cvs.sourceforge.net/tinysos/tinysos-2.x-contrib/wustl/upma/>.
- [44] Feng Wang and Jiangchuan Liu. RBS: A Reliable Broadcast Service for Large-Scale Low Duty-Cycled Wireless Sensor Networks. In *Proceedings of the 2008 IEEE International Conference on Communications (ICC 2008)*, May 2008.
- [45] Feng Wang and Jiangchuan Liu. Duty-Cycle-Aware Broadcast in Wireless Sensor Networks. In *Proceedings of the 28th Annual IEEE Conference on Computer Communications (INFOCOM 2009)*, April 2009. To appear.
- [46] Brad Williams and Tracy Camp. Comparison of Broadcasting Techniques for Mobile Ad Hoc Networks. In *Proceedings of the Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2002)*, pages 194–205, June 2002.

- [47] Wei Ye, John Heidemann, and Deborah Estrin. Medium Access Control with Coordinated Adaptive Sleeping for Wireless Sensor Networks. *IEEE/ACM Transactions on Networking*, 12(3):493–506, 2004.
- [48] Wei Ye, John S. Heidemann, and Deborah Estrin. An Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002)*, pages 1567–1576, June 2002.
- [49] Wei Ye, Fabio Silva, and John Heidemann. Ultra-Low Duty Cycle MAC with Scheduled Channel Polling. In *Proceedings of the Fourth International Conference On Embedded Networked Sensor Systems (SenSys 2006)*, pages 321–334, October 2006.
- [50] Hongwei Zhang, Anish Arora, Young-ri Choi, and Mohamed G. Gouda. Reliable Bursty Convergecast in Wireless Sensor Networks. In *Proceedings of the Sixth ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2005)*, pages 266–276, May 2005.