

ADB: An Efficient Multihop Broadcast Protocol based on Asynchronous Duty-cycling in Wireless Sensor Networks

Yanjun Sun*

Omer Gurewitz[†]

Shu Du[‡]

Lei Tang*

David B. Johnson*

*Rice University
Houston, TX, USA

[†]Ben Gurion University
Israel

[‡]Texas Instruments
Dallas, TX, USA

yanjun@cs.rice.edu gurewitz@cse.bgu.ac.il s-du2@ti.com ltang@cs.rice.edu dbj@cs.rice.edu

Abstract

The use of *asynchronous* duty-cycling in wireless sensor network MAC protocols is common, since it can greatly reduce energy consumption and requires no clock synchronization. However, existing systems using asynchronous duty-cycling do not efficiently support broadcast-based communication that may be used, for example, in route discovery or in network-wide queries or information dissemination. In this paper, we 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 differs from traditional multihop broadcast protocols that operate above the MAC layer, in that it is integrated with the MAC layer 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. We evaluate ADB both through *ns-2* simulations and through measurements in a testbed of MICAz motes using TinyOS, and compare its performance to multihop broadcast based on X-MAC and on RI-MAC. In both evaluations, ADB substantially reduced energy consumption, network load, and delivery latency compared to other protocols, while achieving over 99% delivery ratio.

Categories and Subject Descriptors

C.2.2 [Computer-Communication Networks]: Network Protocols; C.2.5 [Computer-Communication Networks]: Local and Wide-Area Networks—*Access schemes*

General Terms

Algorithms, Design, Performance

Keywords

Sensor Networks, Medium Access Control, Duty-cycling, Broadcast, Multihop, Energy, *ns-2*, TinyOS

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SenSys'09, November 4–6, 2009, Berkeley, CA, USA.
Copyright 2009 ACM 978-1-60558-748-6 ...\$5.00

1 Introduction

Energy efficiency is important in wireless sensor networks, since the network nodes are often battery powered and must operate for long periods of time unattended. To reduce energy consumption, *duty-cycling* is widely used in MAC protocols for wireless sensor networks. With duty-cycling, each node alternates between active and sleeping states, leaving its radio powered off most of the time and turning it on only periodically for short periods of time.

Many duty-cycling MAC protocols, such as S-MAC [22], T-MAC [4], RMAC [5], and DW-MAC [16], assume synchronized duty-cycle schedules among the sensor nodes. This approach simplifies communication, as the transmitter and receiver of a packet are both easily awake at the same time, but it adds complexity and overhead for the necessary clock synchronization. With *asynchronous* duty-cycling protocols, such as B-MAC [14], WiseMAC [6], X-MAC [2], and RI-MAC [17], the nodes instead operate asynchronously, each on its own duty-cycle schedule. Such asynchronous protocols are particularly attractive, as they require no synchronization overhead and since this technique is used by the default MAC protocol in TinyOS [11].

However, existing systems using asynchronous duty-cycling do not efficiently support multihop broadcast-based communication. Multihop broadcast is an important network 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 neighbors of the node originating the broadcast, generally requiring multiple transmissions of the single packet from the originating node [8, 17]. The cost of such redundant transmissions is not well taken into account in existing broadcast protocols (e.g. [12, 13, 21]) designed for always-on networks such as ad hoc networks. With *multihop* broadcast, the problems of single-hop broadcast are amplified, as some neighbors attempt to forward the broadcast while the original transmitting node still attempts to transmit it to others of its neighbors, increasing wireless contention and the possibility of collisions.

In this paper, we present the design and evaluation of ADB (*Asynchronous Duty-cycle Broadcasting*), a new protocol for efficient multihop broadcast in wireless sen-

sensor networks using asynchronous duty-cycling. ADB is integrated with the MAC layer to exploit information only available at this layer. 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 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 and improving reliability through the use of Automatic Repeat Request (ARQ) as part of the unicast transmission. The sender also updates each receiver with information on the progress of the broadcast, helping a node avoid redundant transmissions and allowing delegating transmission for some neighbor to another neighbor with better link quality to it. This approach allows a node to sleep as early as possible and avoids transmissions over poor links, further reducing energy consumption and delivery latency.

The contributions of this paper are as follows:

- We 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. We prove that ADB achieves close-to-optimal delivery latency with error- and collision-free links.
- We evaluate ADB both through *ns-2* simulation and through implementation in a testbed of MICAz motes using TinyOS. Our evaluations show that ADB substantially outperform multihop broadcast based on X-MAC and on RI-MAC for power efficiency, packet delivery latency, and delivery ratio.

We discuss related work in Section 2. Section 3 presents the detailed design of ADB. Section 4 describes our implementation of ADB in TinyOS, and Section 5 presents an evaluation of ADB using both *ns-2* simulation and our TinyOS implementation in a testbed of MICAz motes. Finally, in Section 6, we conclude.

2 Related Work

The goal in *multihop* broadcast is for each node in a network to *receive* a broadcast packet. Multihop broadcast has been well studied in the context of mobile wireless ad hoc networks (e.g., [12, 13, 21]). For sensor networks, Trickle [9] and DIP [10] 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 [15], target multihop

broadcast for services such as routing and resource discovery, needing only propagation of small messages with high reliability 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 substantially 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 the 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. ADB is thus efficient in distributing small messages for services such as routing and resource discovery when asynchronous duty-cycling is used.

To our best knowledge, the only prior work that optimizes multihop broadcast over asynchronous duty-cycling in wireless sensor networks is that of Wang et al. [19, 20]. 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. 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 are based on simulations, but they gave no information on details such as the mechanisms or overhead for learning the wakeup schedules of 2-hop neighbors or 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. We 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; this is then often used as a build-

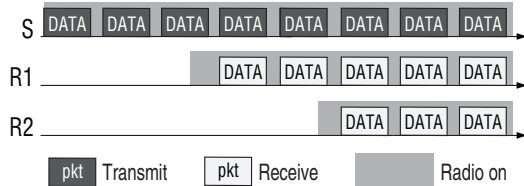


Figure 1. Broadcast support in X-MAC-UPMA [8, 18]. A transmitter S repeatedly transmits copies of a broadcast DATA packet over a duty-cycle interval, during which each neighbor (nodes $R1$ and $R2$) wakes up at least once and thus has an opportunity to receive the packet.

ing block for multihop broadcast when needed. B-MAC [14] can support single-hop broadcast in the same way as it handles 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 [2] 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 [8, 18] of TinyOS, in which a transmitter repeatedly transmits copies of a DATA packet over a duty-cycle interval, as illustrated in Figure 1. In the rest of this paper, we refer to this implementation of X-MAC as *X-MAC-UPMA*.

With X-MAC-UPMA [8, 18], 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.

RI-MAC is a receiver-initiated MAC protocol that works efficiently over a wide range of traffic loads [17]. Nonetheless, RI-MAC was mainly designed for unicast traffic. In order to support broadcast traffic, in our paper [17] we proposed that each node either can unicast a broadcast packet to each one of its neighbors, or can transmit the packet repeatedly back-to-back for a duty-cycle interval as is done in X-MAC-UPMA. However, neither scheme is efficient, as all neighbors of a node will try to transmit the same packet to the node. These redundant transmissions could cause multiple collisions if they happen simultaneously, which not only consumes energy but can result in undelivered packets to some nodes. In addition, a node has to stay awake and wait for a long time, at least a duty-cycle interval, in broadcasting a packet so that all its neighbors have a chance to receive the packet. In this paper, we compare and show that ADB significantly outperforms both previously proposed RI-MAC

broadcast mechanisms in terms of energy efficiency, packet delivery latency, and delivery ratio.

ADB avoids the problems faced by X-MAC-UPMA and RI-MAC by efficiently distributing information on the progress of each broadcast, allowing a node to go to sleep immediately if no more neighbors need to be reached. ADB also uses this progress information to coordinate the 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 transmitting throughout a duty cycle as in X-MAC-UPMA or waiting throughout a duty cycle for neighbors to wake up as in RI-MAC.

3 The ADB Protocol

This section presents the design of the *Asynchronous Duty-cycle Broadcasting (ADB)* protocol.

3.1 Design Motivation

It is challenging to achieve energy-efficient, low latency, and reliable multihop broadcast over asynchronous duty-cycling. For example, the neighbors of a transmitter wake up asynchronously, requiring the transmitter to stay active long enough so that each neighbor has a 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 of a broadcast is important for a node to avoid redundant transmissions, but a node that has just woken up has no up-to-date progress information; a node cannot simply use overhearing to learn this information, as the progress may change while the node has its radio off.

To address these challenges, we 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, we chose to use unicast transmission of the DATA packet to each neighbor 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 to use retransmissions to increase the reliability of the broadcast. Second, in order to avoid the transmitter occupying the wireless medium while waiting for each neighbor to wake up, we chose to integrate ADB with a receiver-initiated MAC protocol, RI-MAC [17], in which each receiver announces its wakeup with a *beacon* packet, as illustrated in Figure 2. 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,

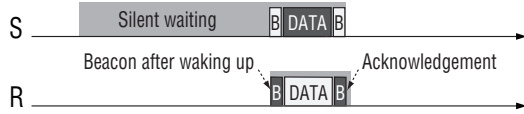


Figure 2. Unicast support in RI-MAC [17]. Each node wakes up on an independent schedule, with each wakeup interval varying randomly between $0.5 \times T$ and $1.5 \times T$, for a nominal duty-cycle period of T . A node transmits a *base beacon* when it wakes up, allowing a waiting sender to transmit a *DATA* packet to it. Upon receiving a *DATA* packet, the node transmits an *acknowledgment beacon* (ACK), confirming receipt and allowing other senders to also transmit a *DATA* packet to it, if needed. RI-MAC’s collision resolution mechanisms are not shown here.

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. Although in our presentation of ADB here, we have integrated ADB with RI-MAC, ADB could also be adapted for other efficient unicast MAC protocols such as WiseMAC [6]. In order to minimize the number of redundant transmissions, ADB coordinates transmissions to a node from its neighbors by efficiently distributing information on the progress of a broadcast together with *DATA* transmissions. Such information also indicates quality of the wireless links from the neighbors to the node, helping ADB avoid transmission attempts over poor links.

3.2 Overview of ADB Operation

Figure 3 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 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 example above shows a number of features of ADB:

- ADB allows a node to go to sleep once all its neighbors have been reached or been delegated to other nodes;

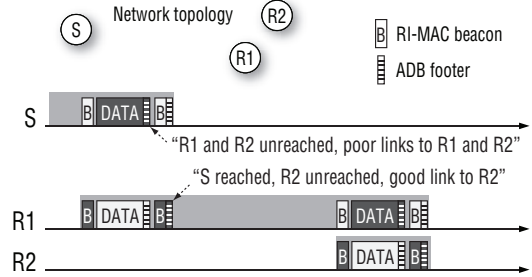


Figure 3. Overview of ADB. Node *S* broadcasts a *DATA* frame to node *R1* and *R2* via unicast transmission. The footer in *DATA* and ACK beacons helps *S* and *R1* to decide which node will deliver the *DATA* to *R2* and helps *R2* to learn that both *S* and *R1* have received the *DATA*.

- 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 the network.

3.3 ADB Algorithm Details

ADB is composed of two basic components: (i) *efficient encoding of ADB control information*, which helps to distribute information on the progress of a broadcast and information for delegation decisions; and (ii) the *delegation procedure*, which runs whenever a broadcast *DATA* packet or a beacon with an ADB footer is received or overheard, determining which nodes the *DATA* packet should be forwarded to and which nodes should be delegated. We describe each in the sections below. A node using ADB needs knowledge of its neighbors and the quality of the wireless link to each, which can be provided by existing mechanisms such as the four-bit link estimation [7] and STLE [1]. In the rest of this paper, we denote the neighbor list of each node v , as $N(v)$, and we denote the quality of the wireless link from node v to each neighbor w as $Q(v, w)$; in particular, the value $Q(v, w)$ is the recent packet delivery ratio (PDR) over this wireless link.

3.3.1 Efficient 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, ADB propagates information on the progress of the broadcast and on link qualities by embedding this information into *DATA* packets and acknowledgment beacons.

In order to efficiently embed this information, each node v includes the *status* of each of its neighbors in the footer of *DATA* and ACK frames, as illustrated in Figure 3. 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 representing $Q(v, w)$, otherwise. We refer to $P(v, w)$ 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. We made this design choice 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 full progress of the corresponding broadcast. Second, having the transmitter instead include only the statuses 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 information in the footer for all neighbors.

ADB distributes the status of neighbors using a bitmap that is constructed based on an *append-only neighbor list*: once a node v detects a new neighbor, it appends the neighbor to the *end* of its neighbor list $N(v)$. A node v lists the status of neighbors using a bitmap with segments of equal length, with each segment corresponding 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, with the segments arranged in the same order as the corresponding node in $N(v)$. In order for a recipient node to be able 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 $N_w(v) \sqsubseteq N(v)$. With this property, even if a node w does not have a current 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 to avoid ambiguity, but we chose to use the append-only neighbor list to efficiently handle the common case where sensor nodes are essentially stationary. 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 node w .

In our implementation, each segment of a bitmap has 3 bits, which is able to represent the corresponding node status with a value from 0 to 7. We reserve the value 7 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. We use the total link

quality estimate $Q(v, w)$ to assign priority $P(v, w)$ values using the equation

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

In order to avoid using very poor links, when $Q(v, w)$ is less than some configurable threshold C , $P(v, w)$ 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 packet and the corresponding ACK have been successfully exchanged over link (v, w) , Equation 1 will be used to calculate status of w , and w is removed from $B(v)$.

3.3.2 ADB Delegation Procedure

In designing ADB, we strove to minimize redundant transmissions and to allow a node to sleep as early as possible. We also wanted to avoid transmissions over poor links.

ADB uses the following data structures to achieve these goals. For a broadcast DATA 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 packet to w on receiving a beacon from w .

Figure 4 shows the algorithm *ANALYZE-FOOTER* used by node v to analyze 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 $N_w(w)$ 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 *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 a 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

```

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  $S_w[u] = \text{REACHED}$  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  $S_w[u] = \text{DELEGATED}$  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  $S_w[u] \neq \text{DELEGATED}$  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 4. Node v analyzes an ADB footer, received or overheard from w , containing information about the progress of packet i . The array S_w in the footer 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 .

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, node 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 packet i to u , u may still delegate the transmission to some node that has a better link quality to u . In order to minimize message overhead, we chose to set u 's status to *DELEGATED* at v .

Once a node starts the forwarding of packet i , the node does not reflect any change of link status into local array S^i until the node stops forwarding packet i , so that the node and its neighbors make consistent decisions on the delivery of packet i . An ACK beacon to a DATA frame 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 our implementation, this duration is set to 3 duty cycles. For each broadcast packet, there is also a predefined deadline beyond which a node gives up broadcasting the packet even if any of its neighbors has not been reached or delegated.

3.3.3 Memory Requirement of ADB

In ADB, each node is required to keep a list of its direct neighbors and their direct neighbors, hence at most a node

needs $O(D^2)$ memory space for such a list, where D is the maximum degree of a node in the network. In addition, since each segment in an ADB footer of a DATA or ACK frame is mapped from a direct neighbor of the sending node to indicate the status of the node, the maximum footer size will be $D \cdot \eta$, where η is the size of each segment. In this paper, we assume that a node has enough memory to hold the neighbor list and that an ADB footer is large enough to distribute the status of all direct neighbors. It is out of the scope of this paper to handle the case in which either memory or footers cannot hold all the information required by ADB. In our initial implementation, we set D to 16 and use 1 byte to hold a node ID. As a result, a node reserves 272 bytes in memory to hold the neighbor list and an ADB footer becomes 6 bytes as each segment has 3 bits.

In practice, however, as a node only needs the status of its direct neighbors in our delegation algorithm, less memory space is needed for non-direct neighbors. For example, when node v receives the neighbor list of w , $N_v(w)$, only for each common neighbor $x \in N(v) \cap N_v(w)$, node v remembers the offset of x in $N(v)$ and ignores the information on the other nodes in $N_v(w)$. Even if information of some neighbors cannot fit in memory or ADB footers, ADB may always attempt to deliver a broadcast packet to the neighbors once. This approach increases the collision probability at these neighbors, but collisions at other neighbors can still be reduced. Due to the great redundancy in a dense network, a single node does not have to deliver a packet to all neighbors. We leave to future work the possibility of taking advantage of such redundancy so that a node with ADB need only maintain information on a subset of neighbors in a very dense network.

3.4 Analysis of End-to-End Delivery Latency

To gain insight into the latency of ADB, we examine 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, we assume here that each node randomly picks a wakeup time that is independent of other nodes' wakeup times and of the traffic load. We show later in this section and in our simulation evaluation in Section 5 that our results based on this simplified model give good insights and are close to our simulation results based on a realistic simulation model.

Theorem 1 *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

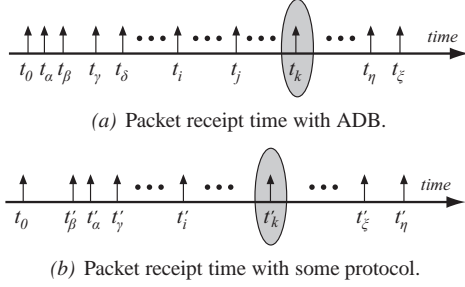


Figure 5. ADB achieves optimal latency under simplified assumptions.

protocol by t_j and t'_j , respectively, as shown in Figure 5. As t_0 is equal to t'_0 , the time the broadcast packet is originated by the source node 0 in both protocols, we replaced t'_0 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 5(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 ($t_\alpha \leq t'_\alpha, t_\beta \leq t'_\beta, \dots, t_j \leq t'_j$).

Assume that with the other protocol, node k received the broadcast from node v (possibly $v = 0$). 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 4. This means that with ADB, from time $t_v \leq t'_v$ until node k receives the packet, there is always 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). Since a node's wakeup time is determined independently, 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 to 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 0 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 of two neighboring nodes waking up within a packet transmission is low. Although the denser the network, the more likely that nodes will have to defer due to other transmissions, in the

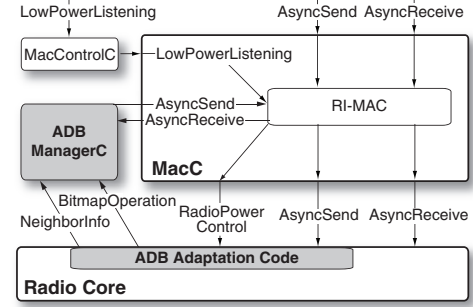


Figure 6. Interaction between ADB, RI-MAC, and the UPMA framework in our TinyOS implementation.

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 can take advantage of the collision resolution mechanism of the underlying RI-MAC protocol, resulting in small delivery latency.

4 ADB Implementation in TinyOS

We implemented ADB under the UPMA framework [8] of TinyOS. We tested our implementation on MICAz motes equipped with CC2420 radios [3]; these radios are also used in the popular TelosB motes. Figure 6 shows how the portions of our ADB implementation interact with RI-MAC and with the UPMA framework.

The *ADBManagerC* module provides most of the functionality of ADB, including maintaining the neighbor list and link qualities, 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, we added some 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.

We use a reserved bit in the IEEE 802.15.4 Frame Control Field (FCF) to indicate if an ADB footer is included in the frame, and we add to the frame's MAC header one byte, *bitmaplength*, giving the number of bytes in the ADB footer in the 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 $(\text{bitmaplength} + 1)$ if a footer is included. For a beacon with an ADB footer, we include the network layer source address and sequence number in order to identify the corresponding broadcast packet. In our experiments, we 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 we faced in our ADB implementation is *limited time in updating the destination address and footer of a pending DATA frame*. In order to update the destination or footer of a pending frame that is already in the CC2420 radio’s TX buffer, we discard the packet in the buffer and load an updated frame into the buffer; although we recently learned that the latency for this update can be reduced by using the CC2420 [3] RAM interface to only update the destination address and footer, ADB must still secure the SPI bus before the update. All of 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 does not completely solve this problem. This increased waiting time would allow for changing the destination to the intended receiver, at the cost of more energy consumption at all nodes when there is no traffic, and may still not give enough time to update the footer. 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 its information up-to-date. If a beacon is received from an intended receiver but the updates have not finished, we either have to wait until the receiver wakes up next time at the cost of much larger energy consumption and delivery latency, or transmit a frame with stale information in its footer.

Optimizing for the common case, we use a broadcast address for all DATA frames and do not delay DATA transmission due to pending updates. When a DATA frame is received, a node returns an ACK only if it has just transmitted a beacon and is waiting for an incoming packet. It is possible that two nodes send their beacons at almost the same time and both send ACKs upon receiving a DATA frame. When these ACKs collide at a DATA transmitter, the transmitter cannot learn of the successful delivery until it overhears future transmissions from those nodes. However, this is rare when using a low duty cycle. If a frame with a stale footer is sent, 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*. Node *B* will thus indicate the good link quality in the ACK to *A*. If *A* failed to receive this beacon, both *A* and *B* will transmit to *C*, leading to collisions. Now suppose *A* has successfully received the ACK from *B* and decides not to transmit to *C*, but, before *A* finishes updating the footer of the DATA, a beacon from node *D* is received. If we let *A* start transmission to *D* immediately, with a stale footer indicating that *A* will transmit to *C*, node *B* will give up its transmission to *C* to avoid collisions upon overhearing the transmission from *A*. As a result, neither *A* nor *B* will transmit to *C*. As such scenario was very rarely observed in our experiments, and in order to avoid the large energy consumption and delivery latency at node *A* when node *A* waits for the next beacon from *D*, we chose to allow DATA transmission with stale information in the footer. With future hardware support in efficiently updating the destination address and footers, the above limitation could be avoided.

5 Evaluation

We evaluated ADB both in detailed *ns-2* simulations and in a testbed running TinyOS on MICAz motes. We use simulation 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.

We compared ADB 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. We use the UPMA RESEND_WITHOUT_CCA option in X-MAC-UPMA, so that when a node repeatedly transmits a DATA frame to broadcast it, only the first transmission uses backoff; otherwise, each of these DATA transmissions would use backoff. We found that backoffs within this sequence could often lead to unreached nodes even in a simple chain topology, and we confirmed this problem with the author of the TinyOS CC2420 driver. 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, we use RESEND_WITHOUT_CCA to get the best performance for X-MAC-UPMA.

As in prior work [2, 8, 17], we use *effective duty cycle*, the percentage of time a node has its radio on, in evaluating power efficiency. When a broadcast packet has reached all nodes in a network, we use *end-to-end delay* to indicate the time between when the that packet was originated and 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 order to evaluate reliability, we measure the percentage of nodes that have been reached by each broadcast packet, indicated by *delivery ratio*.

In both the simulation and testbed evaluations, we use 1 second 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.

There are many schemes in the literature that estimate the quality of wireless links, such as the four-bit link estimation [7] and STLE [1]. We chose to implement a lightweight link quality estimation mechanism that takes advantage of the beacons used by RI-MAC. After a node is booted, it stays awake continuously for a short period of time (10 seconds in our evaluations), during which it counts the base beacons received from other nodes. These beacons are used to build neighbor lists and to estimate the quality of the link from a neighbor to this node. In addition, the quality of the link in the reverse direction is collected through acknowledgment beacons from each neighbor, which embed the link quality estimated at the neighbor based on beacons from this node.

For all protocols in our evaluations in this paper, we used a traffic model in which the sink node periodically originates a broadcast packet. When a broadcast packet is received by a node, the node forwards the packet to its neighbors if this is the first time it has received the packet.

5.1 Simulation Evaluation

We used the *ns-2* network simulator to evaluate ADB's performance in 100 random networks, each with 50 nodes randomly deployed in a 1000 m \times 1000 m area. We verified that each of these networks is connected. In each network, a random node is chosen as sink, which originates 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. We use the default *ns-2* combined free space and two-ray ground reflection radio channel model. We use the same radio parameters used in RI-MAC's evaluation [17], reflecting the CC2420 radio used in popular MICAz and TelosB motes.

We compared ADB with X-MAC-UPMA and with RI-MAC in supporting the 100 network-wide broadcasts in each random network. With X-MAC-UPMA, we considered two versions of the protocol. First, in the standard version of the protocol, which we refer to here as *X-MAC-UPMA-1*, a transmitter of a broadcast packet transmits the packet repeatedly over the duration of one duty cycle. Second, as discussed in Section 2, X-MAC-UPMA could experience collisions caused by transmissions from hidden nodes. In order to compensate for the packet loss caused by these collisions, we let each node transmit the packet over *two* 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.

For broadcast with RI-MAC, we chose to implement two variants of one of the two broadcast schemes suggested in our RI-MAC paper [17]; the other of our earlier suggested schemes is equivalent to broadcast with X-MAC-UPMA. In the first variant, 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 will generate 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. Also, if a node learns through receiving a transmission of a broadcast packet from some neighbor that the neighbor has received the packet, the node does to attempt to deliver the packet to that neighbor. 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*. The longer a node stays awake for broadcasting a packet, the higher the reliability, at a cost of lower energy efficiency. In order to explore such trade-offs, we increase the duration from $1.5 \times T$ to $4.5 \times T$ and refer to this variant as *RI-MAC-4.5*.

With the default radio 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 5.1.1. 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 5.1.2, we also study a modified channel model in which we introduce additional packet losses.

5.1.1 Results with Default Channel Model in *ns-2*

Figure 7 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 7(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 allowing 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; it does not know whether *all* neighbors have been reached due to the lack of a complete neighbor list. Moreover, the transmitter with RI-MAC attempts to send a broadcast packet to a neighbor regardless of 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 forwarding this packet. It is likely that the forwarding 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 possibly use 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 be delivered until this node wakes up again.

Figure 7(b) shows the packet delivery ratios (PDRs) 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% PDR, as ADB footers provide information on the progress of a multi-hop 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 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 PDR. However, this improvement comes at the cost of much increased energy consumption, as shown in Figure 7(a). X-MAC-UPMA shows the worst PDRs, as a node

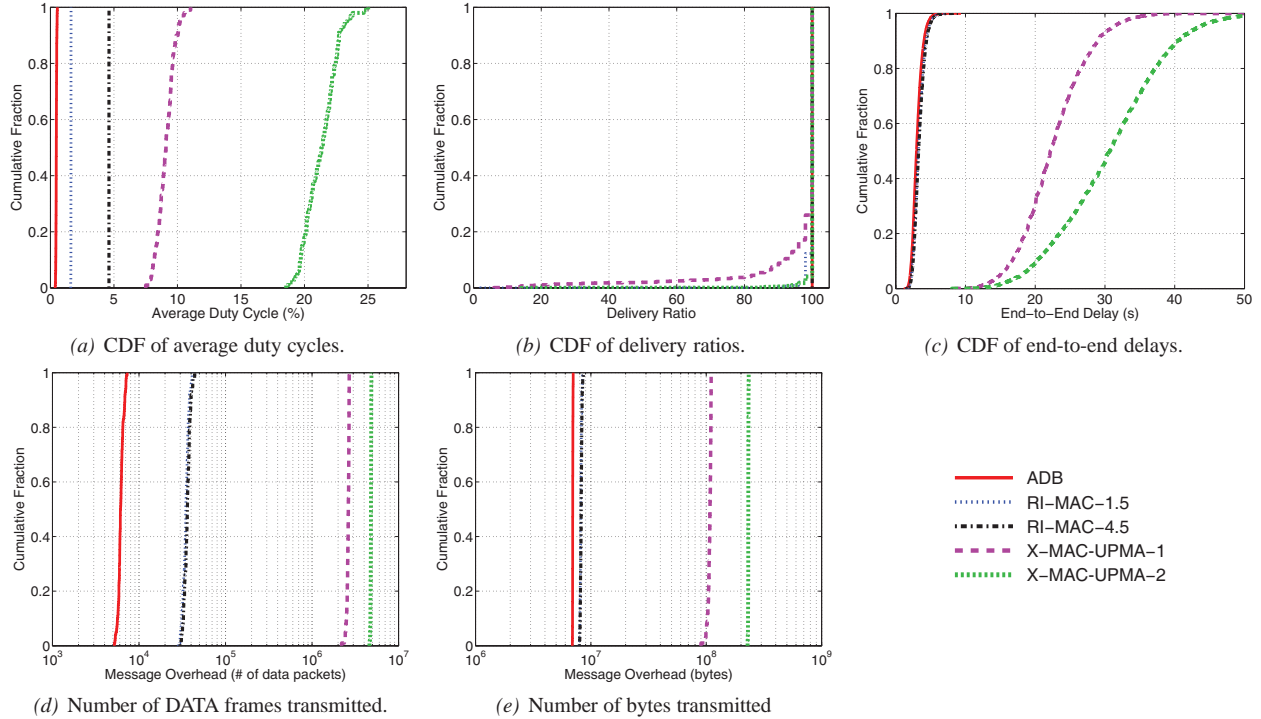


Figure 7. Performance comparison in 50-node networks with default channel model in *ns-2*.

may miss an incoming packet due to collisions caused by overlapping transmissions from hidden nodes, as discussed in Section 2. By rebroadcasting each newly received broadcast packet over two duty cycles with random backoffs, X-MAC-UPMA-2 improves PDRs, but also at the cost of much more energy consumption.

Figure 7(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 forwarding 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 with ADB, the extra delays are small.

Figure 7(d) shows the number of DATA frames transmitted 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 $6.19e+3$, $3.48e+4$, $3.62e+4$,

$2.61e+6$, and $4.79e+6$, respectively. ADB shows the lightest network load in this set of experiments. The number of DATA frames transmitted 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 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 cycles, respectively, from each node.

Besides DATA frames, beacons are also transmitted with ADB, RI-MAC-1.5, and RI-MAC-4.5. Figure 7(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 $6.96e+6$, $8.18e+6$, $8.24e+6$, $1.07e+8$, and $2.31e+8$ bytes. ADB still shows the lightest network load among these protocols.

5.1.2 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, we use a simple model 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). We then use linear interpolation 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.

Figure 8(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 7(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(b); thus, some nodes do not receive the packets and thus do not stay awake to forward 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 7(a). However, ADB still shows the lowest average duty cycle among these protocols.

With increased channel packet losses, ADB still maintains 100% delivery ratios, as shown in Figure 8(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 does 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 7(b), because of substantial redundancy in their DATA transmissions. Even with increased packet losses, when a DATA frame is retransmitted repeatedly for a whole duty cycle in X-MAC-UPMA-1, 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(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 with the default channel model in *ns-2*, since if a beacon from the intended receiver is lost, a transmitter must wait until another beacon arrives in the next cycle. With substantial redundancy, X-MAC-UPMA-1 and X-MAC-UPMA-2 show similar results to those in Figure 7(c), for the same reason 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 we do not include the latency for any broadcast packets that have

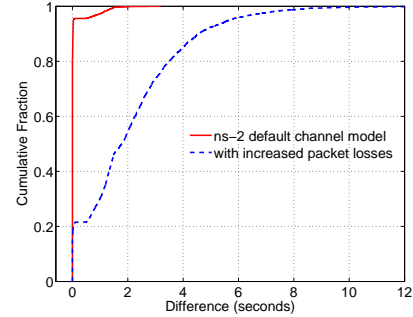


Figure 9. Difference between optimal delivery latency to each node and that with ADB.

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(d) shows the overhead in terms of number of DATA frames transmitted from all nodes, and Figure 8(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 $8.20e+3$, $2.73e+4$, $3.65e+4$, $2.61e+6$, and $4.79e+6$, respectively. Compared with the results in the default *ns-2* channel model, more DATA frames are 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 for forwarding. The similar trend is visible for the number of bytes transmitted (Figure 8(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 $7.12e+6$, $7.90e+6$, $8.38e+6$, $1.07e+8$, and $2.31e+8$, respectively.

5.1.3 Comparison to Optimal Latency

In Figure 9, we show 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 latency. The latencies achieved by ADB used 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 5.1.1) and the model with increased packet losses (Section 5.1.2).

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 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

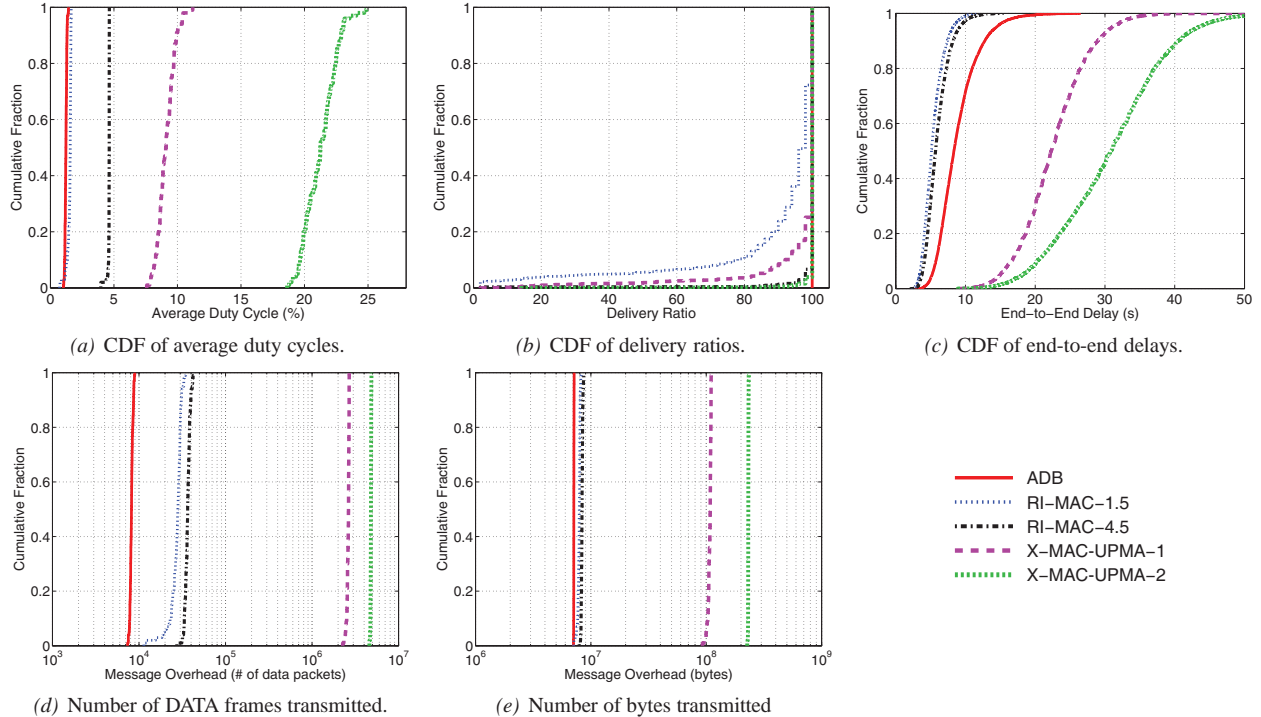


Figure 8. Performance comparison in 50-node networks with increased packet losses in ns-2.

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 channel model with increased packet losses, delivery latencies are larger due to the greater number of lost beacons and DATA frame transmissions.

5.2 Experimental TinyOS Evaluation on MICAz Motes

To explore hardware-dependent issues and details not completely captured in simulation, we implemented ADB in TinyOS, as described in Section 4, and evaluated it on MICAz motes in a clique network and in a random network. Our ADB implementation uses the UPMA framework [8, 18] in TinyOS, integrated with the RI-MAC unicast module [17]. We compared ADB against X-MAC-UPMA, for which the code is available in the UPMA package in TinyOS. The configuration of payload size and duty-cycle interval are the same as those in our simulations presented in Section 5.1.

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

Table 1: Performance comparison in a 5-node TinyOS MICAz 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

5.2.1 Results in a Clique Network

We first present the experimental results for a clique network of 5 TinyOS MICAz nodes, in which all nodes are placed close to each other. We randomly chose one node 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 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% PDR, 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 wake up asynchronously. ADB shows slightly larger latency, mainly due to the difference in generated random numbers that determine the wakeup schedules of each node.

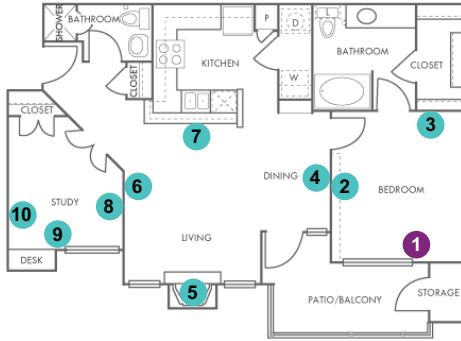


Figure 10. Topology of a 10-node random TinyOS MICAz network deployed in an apartment.

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 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 4, we use 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.

5.2.2 Results in a Random Network

Finally, we also evaluated ADB in a multihop random TinyOS MICAz network of 10 nodes deployed in an apartment, as show in Figure 10. Each node is placed below a wall power outlet, in order imitate (for example) a simple 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 help to ensure that the transmissions for one broadcast packet have finished before another broadcast packet is originated.

The measured performance of X-MAC-UPMA and ADB in this network is shown in Table 2 and Table 3. The average duty cycle with ADB is about 10% of that with X-MAC-UPMA. The large energy consumption with X-MAC-UPMA is mainly due to overhearing. The average duty cycles at all nodes with X-MAC-UPMA are above 22% (Table 3). 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.

Table 2: Performance comparison in the 10-node TinyOS MICAz 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

Table 3: Average duty-cycle percentage of each node in the 10-node TinyOS MICAz 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

Unlike the results in the clique network in Section 5.2.1, ADB here achieves a lower average delivery latency than with X-MAC-UPMA; each node with X-MAC-UPMA occupies the medium for an entire duty cycle, introducing a large delay before any neighbor can begin forwarding the packet; in particular, transmitting a packet over h hops thus requires *at least* h duty cycles. The improvement in latency with ADB over X-MAC-UPMA, however, is not as significant as in our simulation evaluation in Section 5.1, mainly due to the different network size. With X-MAC-UPMA, the sink node in the TinyOS network directly delivered 19 out of 75 broadcasts to all nodes in the network, and 47 of the rest broadcasts reached all nodes in 2 hops; thus, the delay in forwarding at intermediate nodes did not add significantly to the delivery latency. With the larger network used in our simulations, more forwarding hops, and thus more delay, were required, increasing the difference between the ADB and X-MAC-UPMA results in our simulations. In addition, in the default radio channel model in *ns-2*, if a node is beyond a predefined transmission range of a sender, the node can never receive a packet transmitted by that sender. In the real network in our TinyOS experiments, however, no strict cut-off like this exists; with the repeated transmissions over a duty cycle made by X-MAC-UPMA, it possible to deliver some packets even over very long wireless links, resulting in a network with smaller effective diameter than a simulated network with the same topology.

Both X-MAC-UPMA and ADB achieved a high packet delivery ratio of 99.47%. X-MAC-UPMA was able to maintain a high delivery ratio since in this small network, it is unlikely to have transmissions from hidden nodes, avoiding the problems from collisions we observed in the larger networks in our simulations. Unlike in our simulations where ADB always achieved 100% delivery radio, a few packets failed to reach all nodes in our experiments. This is due to the design choice we made in our TinyOS implementation that allows a footer with stale information to be transmitted. We made this choice to reduce delivery latency and energy consumption as discussed in Section 4. In simulations, we assume ADB footers are updated with no delay; with hardware support for quickly updating the destination address and ADB footers, the above problems should be eliminated in an implementation. Even with these limitations, ADB still achieves the same delivery radio as with X-MAC-UPMA with much less energy consumption.

6 Conclusion

In this paper, we have presented the design and evaluation of the *Asynchronous Duty-cycle Broadcasting (ADB)* protocol, which efficiently supports multihop broadcast in wireless sensor networks using 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. Compared to RI-MAC and X-MAC-UPMA, in our evaluation of ADB in *ns-2* simulations in 100 random networks, ADB shows much higher energy efficiency, 100% delivery ratio, and the lowest network load. We 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 a TinyOS implementation of X-MAC-UPMA, ADB shows much higher energy efficiency and significantly reduces network load, while maintaining low delivery latency and over 99% packet delivery ratio.

Acknowledgments

We thank the anonymous reviewers and Philip Levis, our shepherd, for their valuable feedback that helped to improve this paper. This work was supported in part by the U.S. National Science Foundation under grants CNS-0520280, CNS-0435425, CNS-0338856, and CNS-0325971; and by a gift from Schlumberger. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of NSF, Schlumberger, Rice University, Ben Gurion University, or the U.S. Government or any of its agencies.

References

- [1] Alexander Becher, Olaf Landsiedel, and Klaus Wehrle. Towards Short-Term Wireless Link Quality Estimation. In *Proceedings of Em-Nets 2008*, 2008.
- [2] 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 SenSys 2006*, pages 307–320, 2006.
- [3] CC2420 Datasheet. <http://www.ti.com>.
- [4] Tijds van Dam and Koen Langendoen. An Adaptive Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *Proceedings of SenSys 2003*, pages 171–180, November 2003.
- [5] Shu Du, Amit Kumar Saha, and David B. Johnson. RMAC: A Routing-Enhanced Duty-Cycle MAC Protocol for Wireless Sensor Networks. In *Proceedings of INFOCOM 2007*, pages 1478–1486, May 2007.
- [6] Amre El-Hoiydi and Jean-Dominique Decotignie. WiseMAC: An Ultra Low Power MAC Protocol for Multi-hop Wireless Sensor Networks. In *Proceedings of ALGOSENSORS 2004*, pages 18–31, July 2004.
- [7] Rodrigo Fonseca, Omprakash Gnawali, Kyle Jamieson, and Philip Levis. Four Bit Wireless Link Estimation. In *Proceedings of HotNets VI*, 2007.
- [8] 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 SenSys 2007*, pages 59–72, 2007.
- [9] 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 NSDI 2004*, 2004.
- [10] Kaisen Lin and Philip Levis. Data Discovery and Dissemination with DIP. In *Proceedings of IPSN 2008*, pages 433–444, 2008.
- [11] David Moss, Jonathan Hui, Philip Levis, and Jung Il Choi. TEP 126: CC2420 Radio Stack. TinyOS 2.0 Documentation, <http://www.tinyos.net/tinyos-2.x/doc/>.
- [12] 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 MobiCom 1999*, pages 151–162, August 1999.
- [13] Stefan Pleisch, Mahesh Balakrishnan, Ken Birman, and Robbert van Renesse. MISTRAL: Efficient Flooding in Mobile Ad-Hoc Networks. In *Proceedings of MobiHoc 2006*, pages 1–12, May 2006.
- [14] Joseph Polastre, Jason Hill, and David Culler. Versatile Low Power Media Access for Wireless Sensor Networks. In *Proceedings of SenSys 2004*, pages 95–107, November 2004.
- [15] Fred Stann, John Heidemann, Rajesh Shroff, and Muhammad Zaki Murtaza. RBP: Robust Broadcast Propagation in Wireless Networks. In *Proceedings of SenSys 2006*, pages 85–98, October 2006.
- [16] Yanjun Sun, Shu Du, Omer Gurewitz, and David B. Johnson. DW-MAC: A Low Latency, Energy Efficient Demand-Wakeup MAC Protocol for Wireless Sensor Networks. In *Proceedings of MobiHoc 2008*, pages 53–62, May 2008.
- [17] 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 *Proceedings of SenSys 2008*, 2008.
- [18] UPMA Package: Unified Power Management Architecture for Wireless Sensor Networks. <http://tinyos.cvs.sourceforge.net/tinyos/tinyos-2.x-contrib/wustl/upma/>.
- [19] Feng Wang and Jiangchuan Liu. RBS: A Reliable Broadcast Service for Large-Scale Low Duty-Cycled Wireless Sensor Networks. In *Proceedings of ICC 2008*, May 2008.
- [20] Feng Wang and Jiangchuan Liu. Duty-Cycle-Aware Broadcast in Wireless Sensor Networks. In *Proceedings of INFOCOM 2009*, April 2009. To appear.
- [21] Brad Williams and Tracy Camp. Comparison of Broadcasting Techniques for Mobile Ad Hoc Networks. In *Proceedings of MobiHoc 2002*, pages 194–205, June 2002.
- [22] Wei Ye, John S. Heidemann, and Deborah Estrin. An Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *Proceedings of INFOCOM 2002*, pages 1567–1576, June 2002.