A Middleware Framework for Robust Applications in Wireless Ad Hoc Networks

Gregory Chockler
grishac@csail.mit.edu

Murat Demirbas
demirbas@mit.edu

Seth Gilbert
sethg@mit.edu

Calvin Newport
cnewport@mit.edu

MIT Computer Science and Artificial Intelligence Laboratory
Cambridge, MA 02139, USA

Abstract

Wireless ad hoc networks are becoming an increasingly common platform for bringing computation to environments with minimal infrastructure. Increasingly, applications require robust fault-tolerance guarantees, despite a challenging network environment. In this paper, we introduce a new middleware framework for wireless ad hoc networks to aid the development of robust algorithms. Our framework is based on the following three components: (1) receiver-side collision detection, used for identifying inconsistencies caused by unreliable communication; (2) robust round synchronization, used for emulating a strictly synchronized multi-hop network using only basic timeliness assumptions about the environment; and (3) contention management, used for reducing message collision and supporting eventually reliable message delivery. We demonstrate the utility of our framework by showing how it can be used to implement a simple fault-tolerant broadcast protocol, and discuss algorithms to implement each of the components.

ng

1 Introduction

The uses for wireless ad hoc networks are expanding. Most current applications focus on data collection and aggregation. However, the ongoing proliferation of radio-equipped devices is likely to motivate interesting new directions as this technology evolves.

One such new direction is *reliable coordination*. For example, actuator-equipped devices coordinated by a wireless network could be used to control industrial regulator valves. Similarly, in a military scenario, wireless ad hoc networks could be used to guide precision weapon strikes. Or, unmanned missions to Mars could deploy a multitude of sensors to examine the local topology, and then safely navigate an expensive rover over dangerous terrain.

Reliability is crucial for each of these examples. If the devices coordinating regulator valves malfunction due to faulty communication, a pipe may burst and flood the factory. If two battlefield sensors provide inconsistent guidance information, the attack may fail or innocent civilians may be killed. For unmanned space exploration, reliability is similarly critical. An occasional missed warning about a dangerous topology feature could lead to a destroyed

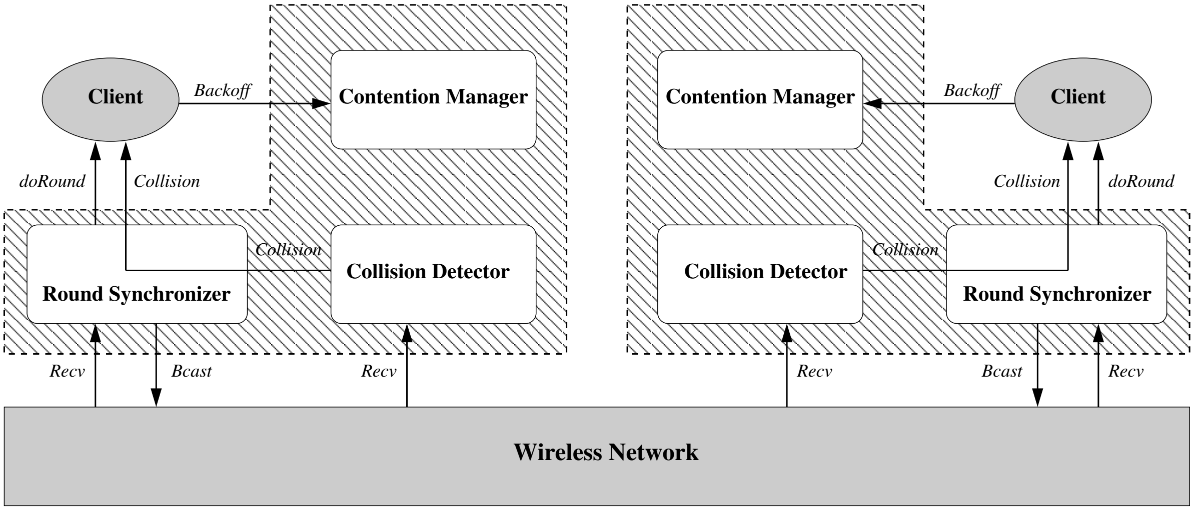

Figure 1: Middleware framework architecture diagram. The framework consists of three components, the round synchronizer, the collision detector, and the contention manager.

rover. In short, these uses require robust applications that always operate correctly, even under unpredictable conditions.

Although these applications are exciting, their development is impeded by several significant challenges. First and foremost is the need to cope with communication that is inherently unreliable. Devices communicate using wireless radios, which use a shared spectrum that is subject to message collisions and other forms of electromagnetic interference.

Second, the devices are deployed in an ad hoc manner. There may be no convenient centralized servers (due to logistical or cost-related concerns), and devices have no *a priori* knowledge of which other devices may be nearby. Moreover, devices may be unreliable: the devices may be mobile and leave the computation; users may turn off devices or obstruct their antennas; devices may exhaust their batteries; and the hardware itself may fail.

Prior research has attempted to mitigate these difficulties by designing component services to tame the unpredictability of wireless networks. Examples of such services include clock synchronization, TDMA schedulers, and reliable MAC layers. Our goal is to identify a *coherent* set of such services that is sufficient to simplify the development of robust applications. Moreover, we aim to identify services that can themselves be built directly on an unpredictable, unreliable wireless network (rather than relying on more basic network services). Accordingly, in this paper, we describe a middleware framework consisting of three basic, modular services. Together, these services are powerful enough to simplify the development of a wide variety of protocols. At the same time, they are simple enough that they can be implemented in realistic—and unreliable—wireless networks.

Specifically, our framework offers the following three components: (1) a round synchronizer, used for emulating a strictly synchronized multi-hop network using only basic timeliness assumptions about the environment; (2) collision detectors, used for identifying inconsistencies caused by unreliable communication; and (3) contention managers, used for reducing message collisions and supporting eventual message delivery.

In the sections that follow, we describe these components in more detail. In Section 3, we present a simple broadcast protocol, using this framework, to demonstrate how these components are used in the context of building a robust application. Next, in Section 4, we present solutions for implementing each of the components.

2 Framework Overview

Our proposed framework consists of three components: the round synchronizer, the collision detector, and the contention manager. (See Figure 1.) The round synchronizer mediates the client's access to the wireless medium through the use of synchronized rounds; the collision detector offers indications of when messages may have been lost; the contention manager provides backoff suggestions that reduce contention on the wireless channel.

2.1 Round Synchronization

The goal of the round synchronizer is to coordinate nodes into a shared round structure. In each round, clients notify the synchronizer of messages to be broadcast in that round. The synchronizer then delivers to each node all the messages "successfully broadcast" by its neighbors in that round (with neighbors defined by geographic proximity). Notice, however, that some messages may not be successfully broadcast: the wireless medium in inherently unreliable, and some messages may be lost. The goal of the round synchronizer is not to provide reliable communication, but simply to provide a synchronous framework on which to build robust protocols.

2.2 Collision Detectors

In order to compensate for messages lost due to unreliable communication, we augment the framework with a collision detector. This component notifies a node when a collision seems to have occurred, indicating that message loss has occurred. In particular, if a neighbor of node p broadcast a message in round r, and yet node p did not receive this message, then in a perfect world, p would detect a collision. Notice that this is *receiver*-side collision detection; the transmitter learns nothing about collisions.

In an important break from previous work, we investigate collision detection that is less than perfect. As we demonstrate in a recent theoretical study [4], "weak" collision detectors are often sufficient for solving difficult problems. And as we discuss in Section 4, these weaker collision detectors are much simpler to implement in real world systems. For example, here are three of the collision detection classes we considered in [4]:

Complete (Perfect): *For any round r, if node p loses **one or more** messages broadcast by its neighbors during round r, then p receives a collision notification in its round r receive set.*

Majority Complete: *For any round r, if node p loses **a majority** of the messages broadcast by its neighbors during round r, then p receives a collision notification in its round r receive set.*

Zero Complete: *For any round r, if node p loses **all** of the messages broadcast by its neighbors during round r, then p receives a collision notification in its round r receive set.*

Notice that in all three cases, a collision notification is simply a single-bit indicator; it contains no information regarding the number of messages lost, their content, or their senders.

For many applications, even a zero-complete collision detector is sufficient to allow robust coordination. In particular, in a round where no messages are received, a node can be entirely certain that no message was broadcast; on the other hand, if a node performs a broadcast, it can be certain that every neighbor will receive that message—or a collision notification. These basic assurances allow enough information transmission to coordinate, even when communication reliability is minimal.

2.3 Contention Managers

The goal of the contention manager is to manage—and reduce—the contention on the wireless medium. When a node suspects that the contention is high (for example, due to a large number of collisions), it passes this information along to the contention manager. Similarly, if the channel seems available, the node notifies the contention manager. In return, the contention manager provides the node advice on whether or not it should attempt to gain access to the medium in the near future. The contention manager offers a well-defined, probabilistic guarantee about when the channel will become free. For example: *any node i that follows the advice of its contention manager will successfully gain solo access to the medium within* $\log n$ *rounds, with high probability.*

Reducing contention on a shared medium is, of course, a well-studied problem. Many solutions take the form of backoff protocols, where nodes that want access to the medium back away from trying for increasing periods of time until contention is low enough for some node to proceed by itself.

Much recent algorithmic work for wireless ad hoc networks has focused on this issue. This is especially true of work on the broadcast problem (e.g. [1–3, 8, 9]). In most of these papers, the authors start with a (well-behaved) collision model (see [2]), and propose algorithms that integrate complicated backoff protocols designed to reduce the broadcast contention. We expect that much of the complexity in these protocols—and their analysis—can be reduced by compartmentalizing the backoff protocols, separating the safety and liveness issues.

3 A Simple Example

To demonstrate the advantages of our framework, in terms of reduction to both code complexity and proof complexity, we present a simple solution to the well-known *reliable broadcast problem*. This problem has been well-studied, and there exist a plethora of solutions, both theoretical (e.g., [1–3]) and practical (e.g., [10, 11]). We begin with a description of the problem and the features of our solution. We then offer a closer examination of how the algorithm works in the context of the various components of our framework.

3.1 The Reliable Broadcast Problem

The reliable broadcast problem requires a single distinguished source to disseminate a message to all other processes. Any solution must guarantee that every node eventually receives the message, and each node must eventually halt. The performance of the algorithm is measured with respect to the last node to halt.

Most currently implemented reliable broadcast protocols fall into one of two categories: *flooding* or *tree-based*. The former approach has processes rebroadcast the message as soon as they receive it, causing a flood of the information throughout the network. A variety of schemes have been proposed to reduce the number of messages rebroadcast, while still maintaining a high probability of full delivery. For example, in [10], which is optimized for the delivery of large messages broken up into smaller pieces, processes peridocially gossip with their neighbors; more informed processes can then forward the missing information. See [13] for a more comprehensive survey of message flood techniques.

The tree-based approach has processes build spanning structures, such as a breadth-first search tree (as used in [11]), to expedite efficient forwarding of information throughout a network.

Unfortunately, both of these solutions can be difficult to implement in a collision-prone, failure-prone wireless network. Flooding, in an environment with collisions, is typically best-effort. Given enough rebroadcasts and gossiping, the message will *probably* be successfully disseminated. Tree-based approaches, on the other hand, are destabilized by both collisions and faults. The former might prevent a proper structure from being constructed, and the latter can disconnect (and disrupt) an already constructed structure.

3.2 A Simple Algorithm for Reliable Broadcast

We present a simple algorithm that solves the reliable broadcast problem using the framework described in Section 2. (See Figure 2.) The new algorithms is much simpler to implement, while performing competitively with prior theoretical solutions.

The algorithm consists of pairs of rounds. In *broadcast* rounds, each node that has already received the message rebroadcasts the message. As a result, some nodes may receive the message for the first time. Other nodes may receive collisions. In *veto* rounds, a node performs a broadcast only if it received a collision in the prior broadcast round.

If no veto messages are received after a veto round, a transmitter can be sure that its broadcast-round message was received. At this point, the transmitter can safely halt. Safety is guaranteed by the properties of the zero-complete collision detector: if node p receives no message or collision, then no neighbor of p broadcast.

The contention manager guarantees liveness. Before sending a message in the broadcast round, a node requests the advice of the contention manager. The contention manager guarantees that eventually each node will receive a chance to broadcast, and that eventually broadcast contention will be low, ensuring limited collisions. If all processes follow the advice of the component, eventually everyone will successfully deliver their message to their neighbors.

In particular, we analyze the performance under the following assumptions: (1) Each node has a bounded (constant) number of neighbors. (2) If only one neighbor of node p broadcasts a message in a round, then p receives that message. (3) D is the diameter of the network. The contention manager guarantees that if a constant number of nodes are competing to broadcast, they will all succeed within constant time. Since each node has at most a bounded number of numbers within two hops, the contention manager guarantees to resolve all collisions.

In this case, the algorithm will terminate in $O(D)$ rounds (with high probability). Notice how all the probabilistic analysis of the backoff protocols is relegated to the contention manager component.

3.3 The Framework Interface

The pseudocode in Figure 2 demonstrates how an algorithm can make use of the new framework. The structure of an algorithm in this framework consists of an implementation of the $\mathsf{doRound}()$ function. The round synchronizer calls doRound once for each round. The algorithm is pssed the current round number ($rnum$), the set of messages received during this round ($msgs[]$), and a boolean indication of whether or not a collision was detected during this round ($collision$). The return value of the procedure is the message to be sent during the next round (or $\emptyset$ to indicate that no message should be sent).

```
init(m)_i
   started ← false
   active ← false
   msg ← m

doRound(rnd, msgs[ ], collision)
   if rnd mod 2 = 0 then  // Receive a message in even rounds.
      if (not started) and (|msgs| > 0) then
         started ← true
         msg ← msgs[0 ]
      if collision then
         return veto  // Broadcast a veto if receive failed.
      else
         return ∅
   else if rnd mod 2 = 1 then  // Receive a veto in odd rounds.
      if started then
         if (not collision) and (|msgs| = 0) and active then
            halt()  // If no vetoes, then done.
         else if (collision) or (|msgs| > 0) then
            active ← Backoff(eTooMany)
         else if (not collision) and (|msgs| = 0) and (not active) then
            active ← Backoff(eTooFew)
         if active then
            return msg  // Broadcast message.
      else return ∅
```

Figure 2: Reliable broadcast protocol for node i. The protocol disseminates a single value to every (correct) node. The protocol works in phases: in even rounds, nodes receive messages and broadcast vetoes; in odd rounds, nodes receive vetoes and broadcast messages. The algorithm terminates when a node (a) receives the message and (b) broadcasts the message without receiving any vetoes.

The contention manager component is accessed through the $Backoff$ function. The function returns $true$ if the caller should attempt to broadcast during the next designated broadcast round, and it returns $false$ if the caller should remain silent. It accepts as an argument the state of the channel during the previous designated broadcast round–either $eTooMany$ to indicate there was contention, or $eTooFew$ to indicate that there was no contention. This information is enough to allow the component to run a local backoff protocol that eventually guarantees that all nodes will get a chance to successfully broadcast their message.

4 Implementation

In this section, we describe how to implement the framework presented in Section 2. There is a significant body of prior work related to each of the components. Our goal in this section is to provide one possible solution. In some cases, this represents new progress; in others (particularly, contention management), we draw heavily on existing efforts.

4.1 Collision Detection

We begin by describing a mechanism for implementing collision detection. Our collision detection (CD) implementation employs the following techniques, ordered according to their contribution towards the cumulative collision detection:

- Carrier sensing
- Cyclic redundancy check (CRC)
- Preamble detection

Receiver-side carrier sensing

Carrier sensing is widely employed in wireless networks with CSMA (Carrier Sense Multiple Access) MAC layers, which includes almost all of the modern wireless network MAC layers such as IEEE 802.11, IEEE 802.15.4, and all of the wireless sensor network MAC protocols [6, 14, 15, 18]. Before a transmitter starts it transmission, it senses the medium for any existing transmission, and only begins transmission if the medium is not already busy., it schedules its transmission to start. However, previously carrier sensing was used primarily by transmitters. We adopt this technique for use by the receiver to detect collisions.

The MAC layer is implemented as a state machine where the states are idle, synchronizing, receiving, prepare-to-transmit, and transmit. The node is in the idle state most of the time. In the idle state, when a node detects a preamble byte—a predefined byte signaling that a message is about to be transmitted—it switches to the synchronizing state, and after receiving the rest of the preamble bytes and synchronizing with the transmitter, it switches to the receive state.

To detect collisions at the receiver side, we employ carrier sensing in the idle state. The node detects a collision when its carrier sensing mechanism detects in the idle state that there is an intense activity on the medium. This is a good indication of a collision—if a message had been broadcast, a preamble would have been detected.

Due to noise, there is a lot of activity in the transceiver even in the idle state. However, it is easy to differentiate between noise and a genuine activity, such as a message or collision. The random noise has significant variance in channel energy (occasional pits below the noise floor) whereas a genuine activity has fairly constant channel energy (always stays above the noise floor). Our carrier sensing at the idle state searches for these pits: if for a long period no pit is found, this is a good indication of genuine activity in the radio.

CRC-based collision detection

In wireless networks, CRC checks are widely adopted for filtering the messages that are received with errors. To this end, the receiver calculates a running CRC for the message it receives and compares this calculated CRC with the CRC appended to the transmitted message. The messages that fail the test are thrown away. Thus, a failed CRC is a good indication of collision, since it indicates that the receiver dropped a message. We raise a collision detection at the MAC layer whenever the CRC test fails for a message.

Preamble-based collision detection

While a node j is receiving a message, if the preambles of a stronger message arrives in the middle of the first message, the stronger message dominates the first message and renders it undeliverable. j synchronizes to this latter message and ignores the first message. CRC for the first message does not even get computed so a collision detection would not be triggered.

This effect is known as the shadowing or capture effect [17]. To be able to detect these types of collisions we use a preamble based collision detection technique first advocated by [17]. Note that, in the absence of any collision, the preamble bytes are only heard in the synchronizing state, and no preamble is heard in the receive state. So, when j receives a preamble byte in receive state, this is a good indication of a collision, and is reported as such.

4.2 Round Synchronization

The second component of our framework is a round synchronizer. In this section, we present a simple algorithm to implement a round synchronizer in a partially synchronous network. There is much prior work in clock synchronization, and if a synchronized clock is available, round synchronization is trivial. However, it seems unclear whether prior algorithms can work in a collision-prone, unpredictable environment. To that end, we develop a simple protocol that is quite robust. It remains an interesting open problem as to whether more powerful algorithms can be adapted to this environment.

The underlying partially synchronous network provides the following guarantees. First, we assume that messages are delivered in bounded time, d. A message may be lost due to a collision, but within some bounded period of time after a message is sent, either the receiver delivers a message, or a collision is delivered (as per the completeness guarantee of the collision detector). Second, we assume that the nodes have access to clocks with bounded drift. In other words, a duration x can take anywhere between $x(1-\epsilon)$ and $x(1+\epsilon)$ time to elapse on a local clock.

The round synchronizer measures out some constant number of constant-time rounds, followed by a linear-time resynchronization delay. The pseudocode for the round synchronization protocol is presented in Figure 3. When the node is initialized, it calculates certain constants, and then begins synchronization (line 8). Similarly, if a node every receives a message or a collision, it begins synchronization (lines 12 and 18). When synchronization begins, it propagates the start message, and begins simulating rounds. The node then relies solely on its timer to simulate a certain number of rounds, before resynchronization is necessary.

In particular, as time passes, the node's clock may drift continually further away from its neighbors, limiting the "safe" interval in which a message can be broadcast in a round. The delay function calculates the beginning of the safe broadcast window. Once the safe broadcast window has become too small, the nodes must resynchronize.

Notice that this protocol tolerates collisions and asynchronous initialization. If many nodes begin at the same moment, eventually their start messages propagate to each other, ensuring synchronization. If collisions obscure some message, again the resulting rounds are synchronized.

The key observation is that round synchronization requires only "gradient" synchronization, rather than global synchronization. As long as neighboring nodes are (nearly) synchronized, it is possible to emulate rounds.

4.3 Contention Manager

There are a variety of well-studied backoff protocols that can be used to implement a contention manager. The most common of these is exponential backoff, which guarantees that n competing nodes will achieve reduced contention in $O(\log n)$ rounds: that is, there will be only 1 active node.

```
init(m,r)_i
  rndLength ← desired round length
  delay(k) = (d+2ε(k−1)·rndLength) / (1−ε)
  round-offset ← r
  rcv-msgs ← ∅
  collisions ← false
  msg ← m
  start-synch()

recv(m)_i
  if (not started) then
    startSynch()
  else
    rcv-msgs ← rcv-msgs ∪ {m}

collision()_i
  if (not started) then
    startSynch()
  else
    collisions ← true

rndLimit(k)
  left ← rndLength(1-ε)
  right ← 2delay(k) + d
  return left > right

startSynch()_i
  cancelTimer(restart)
  started ← true
  rnd ← 1
  setTimer(round, rndLength)
  setTimer(bcast, delay(rnd))
  bcast(start)

timerFired(bcast)_i
  bcast(msg)

timerFired(round)_i
  r ← rnd + round-offset
  msg ← doRound(r, rcv-msgs, collisions)
  rcv-msgs ← ∅
  collisions ← false
  rnd ← rnd + 1
  if (not rndLimit(rnd)) then
    setTimer(round, rndLength)
    setTimer(bcast, delay(rnd))
  else
    init_i(msg,rnd)
    setTimer(restart,O(diameter))

timerFired(restart)_i
  startSynch()
```

Figure 3: Round synchronization algorithm for node i. The protocol receives inputs from the network (recv) and the collision detector (collision), and produces the doRound output. It uses a timer with drift bounded by ϵ.

In previous work [4], we have used an even simpler variant, approximating a well-known back-off strategy [5, 7, 12, 16]: (1) If a node indicates that too many devices are active, then with probability $1/2$, it recommends becoming passive; (2) If a node indicates that too few devices are active, it recommends becoming active with probability $1/2$. This simple strategy guarantees that within $O(\log^2 n)$ rounds (with high probability) there is only one active node.

5 Conclusion

In this paper, we have presented a basic new middleware framework for wireless ad hoc networks. By providing a small number of fundamental services, we believe that the development of algorithms can be significantly simplified. Round synchronization helps algorithms to overcome hardware failures and unknown deployment topologies; collision detection helps algorithms to overcome lost messages and unreliable communication; and a contention manager compartmentalizes the backoff protocols, necessary to reduce contention on the wireless medium. Together, these three components allow the simple development of powerful protocols.

References

[1] R. Bar-Yehuda, O. Goldreich, and A. Itai. Efficient emulation of single-hop radio network with collision detection on multi-hop radio network with no collision detection. *Distributed Computing*, 5:67–71, 1991.

[2] R. Bar-Yehuda, O. Goldreich, and A. Itai. On the time-complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization. *Journal of Computer and System Sciences*, 45(1):104–126, 1992.

[3] I. Chlamtac and S. Kutten. On broadcasting in radio networks - problem analysis and protocol design. *IEEE Transactions on Communications*, 33(12):1240–1246, 1985.

[4] Gregory Chockler, Murat Demirbas, Seth Gilbert, Calvin Newport, and Tina Nolte. Consensus and collision detectors in wireless ad hoc networks. In *PODC '05: Proceedings of the twenty-fourth annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 197–206, New York, NY, USA, 2005. ACM Press.

[5] R. Gallager. A perspective on multiaccess channels. *IEEE Trans. Information Theory*, IT-31:124–142, 1985.

[6] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for network sensors. *ASPLOS*, pages 93–104, 2000.

[7] S. Olariu K. Nakano. A survey on leader election protocols for radio networks. *Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN)*, pages 71–79, 2002.

[8] E. Kranakis, D. Krizanc, and A. Pelc. Fault-tolerant broadcasting in radio networks. In *Proceedings of the 6th Annual European Symposium on Algorithms*, pages 283–294, 1998.

[9] Eyal Kushilevitz and Yishay Mansour. An
u03a9(d log(n/d)) lower bound for broadcast in radio networks. In *PODC '93: Proceedings of the twelfth annual ACM symposium on Principles of distributed computing*, pages 65–74, New York, NY, USA, 1993. ACM Press.

[10] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. *First USENIX/ACM Symposium on Networked Systems Design and Implementation*, 2004.

[11] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. Tinydb: An acqusitional query processing system for sensor networks. *ACM TODS*, 2005.

[12] K. Nakano and S. Olariu. Uniform leader election protocols in radio networks. In *ICPP '02: Proceedings of the 2001 International Conference on Parallel Processing*, pages 240–250. IEEE Computer Society, 2001.

[13] S. Ni, Y. Tseng, Y. Chen, and J. Sheu. The Broadcast Storm Problem in a Mobile Ad Hoc Network. In *MOBICOM*, 1999.

[14] J. Polastre and D. Culler. Versatile low power media access for wireless sensor networks. *The Second ACM Conference on Embedded Networked Sensor Systems (SENSYS)*, pages 95–107, 2004.

[15] T. van Dam and K. Langendoen. An adaptive energy-efficient MAC protocol for wireless sensor networks. *The First ACM Conference on Embedded Networked Sensor Systems (SENSYS)*, pages 171–180, 2003.

[16] D. E. Willard. Log-logarithmic selection resolution protocols in a multiple access channel. *SIAM Journal of Computing*, 15(2):468–477, 1986.

[17] A. Woo, K. Whitehouse, F. Jiang, J. Polastre, and D. Culler. Exploiting the capture effect for collision detection and recovery. *The Second IEEE Workshop on Embedded Networked Sensors (EmNetS-II)*, May 2005.

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