

DuraNet: Energy-Efficient Durable Slot-Free Power Scheduling

Terence Tong, David Molnar, and Alec Woo

January 14, 2004

Abstract

Class project submission for CS262A. Please do not cite or distribute beyond NEST. Contact the authors for the latest version.

We present an effective distributed power scheduling algorithm for fixed, low bandwidth, many to one data collection sensor network applications. DuraNet reduces energy consumption by avoiding collision and overhearing while having nodes sleep most of the time. Because that it is hard to achieve global time synchronization while nodes are sleeping, DuraNet avoids the traditional approach where global hard bound time slots are assigned to nodes for communication. Instead, there is no notion of global time in DuraNet. Nodes allocate schedules for each link based on contention in a schedule formation phase, then re-use these schedules for a long period of time. By doing so, DuraNet is able to adapt to density, while providing significant energy savings. Given a route tree topology, DuraNet is also able to avoid congestion due to buffer queue overflow by careful scheduling.

We compare our algorithm to low power listening and classic CSMA through extensive simulation in TOSSIM[12]. We find that DuraNet achieves significant energy reduction over classic CSMA and low power listening, while providing similar end-to-end reliability.

1 Introduction

Sensor networks are an emerging technology for collecting physical information about our environment. A wireless sensor network consists of multiple nodes, each outfitted with a radio and sensors. Data gathered by these sensors can support many different applications, including earthquake monitoring, habitat monitoring, and hibernation studies. Nodes can send data to a central location for storage, allowing later processing and analysis.

One of the major challenges in sensor networks is the power limitations of sensor nodes. Current battery technology can support a node for only a short amount of time without power management. Because some applications place sensors in difficult-to-reach locations, changing batteries can be impractical or even impossible. Some mechanism for conserving power and extending battery life is necessary.

We focus on a *power scheduling* mechanism that creates a sleep/wake schedule for the radio of each node. The schedule maximizes each node's battery lifetime while meeting its network obligations. In particular, we perform scheduling above the MAC layer, an approach pioneered by Hohlt et al.[10] At the same time, during the execution of the schedule, we turn off the MAC layer and let the schedule itself manage media access. This approach allows us to leverage information available from the application and routing layer to form a better schedule.

When designing a practical power scheduling mechanism, we must consider the following requirements.

1. A scheduling algorithm should turn off radios whenever nodes are not involved with communication. Nodes should avoid idle listening and overhearing. It is a clear waste of energy to do sampling on a quiet medium or to handle a packet not addressed to the receiver.
2. The number of transmissions should be minimized, especially if transmission is comparatively expensive. For example, on the Berkeley mica2 mote platform, it costs roughly fifty percent more energy to

transmit a packet than to receive a packet. Therefore a good protocol will have a few control packets and will spend a minimum amount of energy to route packets to their destinations.

3. The protocol should be able to work in any ad-hoc environment, adapting gracefully regardless of whether there are thousands of neighbors or only a few.
4. The protocol must work gracefully in the presence of lossy links.
5. Clock skew must be handled carefully. Otherwise, it is possible for nodes to communicate at the same time, causing collisions. While global time synchronization has been done at microsecond resolution [7], it is not clear that these mechanisms work well in the power scheduling context. Without tight integration with the power scheduling layer, time synchronization will result in extra transmissions, incurring large energy costs.
6. The schedule must respect queue buffering. A receiver buffer queue can be overwhelmed by a stream of messages from the transmitter(s) resulting in packet loss. A protocol should be able to schedule the exact amount of packets to be sent to avoid overflow.
7. The protocol itself should have a low memory footprint. Limited memory may make it infeasible for each node to keep track of all neighbors in a dense network.
8. The protocol should adapt to changing network conditions. While for the applications that we consider for this work, nodes may be placed in a static network with stable connectivity, the protocol must detect and recover from catastrophic change.

We propose a protocol, DuraNet, that addresses these issues. We focus specifically on low data rate, many-to-one data collection applications with rarely changing topologies and stable traffic patterns. We leverage information available from these traffic patterns to derive schedules that are close to optimal and then re-use these schedules for as long as possible.

DuraNet aggressively minimizes energy waste due to communication. Because the traffic pattern is known, overhearing and idle listening can be almost eliminated by carefully turning on / off the radio at the right time. Collisions due to a one-hop hidden node problem are avoided by negotiating transmission times using exchanges of RTS and CTS.

Our schedule algorithm is *slot-free*: instead of dividing time periods into fixed slots, nodes are scheduled according to application traffic. Each link wakes up for however long is necessary to handle its packet load and then sleeps. Since there is no fixed number of global hard bound time schedules, DuraNet is able to dynamically allocate more schedules depending on the network density. To deal with clock drift, DuraNet incorporates a parent-child time synchronization algorithm into our schedule execution; however, this algorithm piggybacks on top of data ACK messages and so introduces minimal additional overhead for time synchronization. Our synchronization can efficiently prevent schedules from drifting into each other, thereby avoiding packet loss. In addition, we use data ACKs as a command channel.

Because DuraNet forms schedules based on the application traffic pattern, nodes are able to allocate schedules based on knowledge of future buffer queue length. In other words, a schedule will not be allocated unless there is room in the receiver.

In Section 2 we discuss other work in power scheduling and energy efficiency for sensor networks and our relation to this work. Then in Section 3 we describe the DuraNet protocol. In Section 4 we give the results of simulation of DuraNet in TOSSIM[12] and compare to CSMA and low-power listening. Finally in Section 5 we summarize lessons from DuraNet and point out future directions.

2 Related Work

Minimizing energy consumption has been a concern of sensor network research since its inception. One current approach is the *low-power listening*[16] implemented in the Mica2 ChipCon radio stack in TinyOS

1.1[9]. In low-power listening, motes sleep and periodically wake up to listen on a channel. When a mote wishes to send, it first detects that the channel is idle, then sends a *preamble* with length proportional to the sleep period. Low-power listening has been deployed to great effect in the Great Duck Island sensor application[13].

Despite its success, low-power listening has some drawbacks. First, low-power listening reduces the bandwidth available for data transmission; in a typical one percent duty cycle, this can be a factor of 30. Second, low-power listening is insensitive to the application data traffic; a node that may only “need” to receive messages once every 10 minutes may still wake up and listen much more often than its “real” period. Finally, the increased duration of a message raises the chance of collision due to a hidden node; low-power listening does not, by itself, address this problem. Low-power listening also allows nodes to overhear messages not intended for them, which is a waste of energy. In addition, low-power listening can not solve the problem of buffer queue overflow.

We focus instead on *power scheduling*, in which we create a sleep/wake schedule for the each sensor that maximizes its lifetime while meeting its network obligations. In the case of power scheduling, most previous approaches can be divided into two categories: *centralized* and *ad hoc*. In a centralized system, network nodes report information on their local topology to a central base station. The base station then computes a power schedule from this information and pushes it back to the nodes. In an *ad hoc* system, computation of the schedule takes place entirely within the network. Our work has elements of both categories: most scheduling operations take place within the network, but DuraNet takes advantage of the global information known at the base station.

Centralized power scheduling has been implemented by Dust, Inc. as part of their sensor network architecture[6]. In addition, the masters thesis of Coleri gave basic definitions of power scheduling and algorithms based on graph coloring, as well as a concrete proposal called PEDAMACS[4]. Centralized scheduling has the advantage that all decisions are shifted to a base station. Upgrading the scheduling algorithm is easy and comparatively large amounts of computation can be applied to the task.

At the same time, centralized scheduling has some drawbacks. For an n -node network, the base station must know about $O(n^2)$ links. Each link must be probed, then send its information to the base, which costs time and effort. Failures in detecting network conditions may provide the base station with imperfect information, leading to poor schedules. The scheduling information itself must be passed out to nodes once computed. In addition, connectivity changes over time, so the base station needs to have some way to continually monitor the network and issue new schedules when current ones become “stale.”

In the area of ad-hoc power scheduling, pioneering work was performed by Hohlt et al.[10], who invented the term “power scheduling.” Their Flexible Power Scheduling work focuses primarily on adaptation in the face of changing network conditions, such as changing traffic patterns, interference topology, or joins and leaves of nodes. Their implementation divides time into explicit time slots, then has a distributed algorithm for computing power schedules based on supply and demand. Hohlt et. al. argue, as we do, that scheduling above the MAC layer allows for benefits beyond those possible by scheduling at the MAC layer. In the case of DuraNet, these benefits include awareness of topology, queue size, and workload.

Our work is similar to theirs in that both do not require global time sync, both schedule at the application layer (on top of MAC and routing), and both require only local time synchronization. We differ in that while Hohlt’s work provides constant adaptation to changing network conditions, DuraNet assumes that long-term network conditions will be similar to those present at schedule formation time. Hohlt et al. adapt to changing network conditions immediately, while DuraNet detects and recovers from catastrophic changes. In addition, DuraNet is designed for many-one applications which provide a base station, while Hohlt et. al. deal with arbitrary topologies. Finally, Hohlt et al. do not divide time into “phases” as we do, and the details of our schedule formation algorithms are different.

In addition, DuraNet is *slot-free*: it does not divide time into global hard bound time slots. Instead, schedules are defined by the behavior of nodes during schedule formation in the Sync Phase. Sohrabi and Pottie also use schedules within a schedule period that are variable size depending on application traffic, an approach they call “non-synchronous scheduled communication”[17]. We differ from their work in that they set the overall schedule period as a fixed parameter, while we infer ours in the Sync Phase from schedule

behavior and disseminate the period using the base station. In addition, they use frequency division to overlap several schedules in time, while we do not assume different frequencies and instead ensure non-overlap through the Sync Phase.

Shankar and Johnson take an alternative approach to ad-hoc scheduling based on channel division multiplexing [15]. In their scheme, nodes randomly choose a channel, which is a pair (s, t) of direct sequence spread spectrum spreading code and time slot. Nodes in channels that are too heavily loaded then switch to another channel with a certain probability p . When not in the time period corresponding to its channel, a node sleeps. They then argue that this process converges to a power-saving schedule. Nodes also listen on an “announcement channel” to overhear command messages.

This approach is elegant because no node need listen for messages which are not addressed to it. The use of different channels effectively segregates traffic. Their approach requires support for changing DSSS codes at the radio and assumes maximum use of aggregation. DuraNet differs from this work in that we do not have explicit slots, work without aggregation, and do not have an “announcement channel” in normal operation. Once DuraNet establishes and begins to execute a schedule, nodes do not require any nonessential listening.

TRAMA also assumes explicit slots, and adds a leader election protocol designed to achieve fairness in addition to low power consumption[14]. Nodes begin with random schedules and then adapt schedules based on the traffic patterns of their neighbors. Schedules can be exchanged and compared by nodes to ensure network schedule consistency. DuraNet, by contrast, does not have fairness as a goal, and so can achieve a much simpler protocol. In addition, DuraNet forms all schedules in advance during a separate schedule formation phase instead of trying to adapt schedules in real time.

The Sensor MAC (S-MAC) protocol takes a different approach entirely[19]. Unlike the previous approaches, S-MAC places itself at the MAC layer and takes a contention approach. Power schedules are formed by an RTS/CTS protocol for every packet sent and causing backoff, resulting in significant overhead. Our work, in contrast, only performs RTS/CTS during the creation of a schedule and avoids paying this penalty during schedule execution. Further, because S-MAC is unaware of routing restrictions, it can lead to buffering problems at higher layers. Because it takes a contention approach, S-MAC must also perform a large amount of idle listening.

Koushanafar et. al study the problem of power scheduling and prove a “Care-Free Sleep” theorem showing which nodes may safely sleep at any point in time of a network[11]. They then build a distributed power schedule inspired by the theorem. Unfortunately the protocol requires that nodes keep a large amount of state about their neighbors. By contrast, DuraNet requires a node keep only a small amount of state for each child.

Godfrey and Ratajczak propose a power-saving topology formation algorithm they call Naps[8]. Their work is similar to ours in that nodes pick a random time to wake up and begin interacting with neighbors. While in DuraNet, however, this interaction leads to a power schedule, in Naps the interaction leads to a routing graph. They show that the resulting graphs possess desirable routing properties with high probability. DuraNet sits above such protocols because it assumes a route tree has already been formed.

The idea of forming a power-sensitive topology is shared by the Intel Seattle ReOrg protocol[5]. ReOrg applies some simple heuristics to a routing topology to take advantage of cases in which some nodes have larger power budgets than others. The resulting routing topology guides messages through the nodes better able to handle high bandwidth.

ReOrg works in concert with a power scheduling protocol called ReSync[5]. In ReSync, time is divided into hard bound schedule slots. Nodes overhear and back off when their neighbors are sending in a particular slot. When a node wishes to send, it first sends an “intent” message to ask its parent to wake up at that slot. Once a slot has been reserved by a node, that node will send, even if the parent is asleep or sending at that slot would cause collisions.

ReSync tries to place signalling and schedule formation in the same channel as data sending. In contrast, PAMAS introduces a separate signalling channel and argues that a separate channel is necessary to achieve optimal power conservation[16]. In DuraNet, we provide a separate channel by doing all signalling during schedule creation and delaying schedule execution until a later phase.

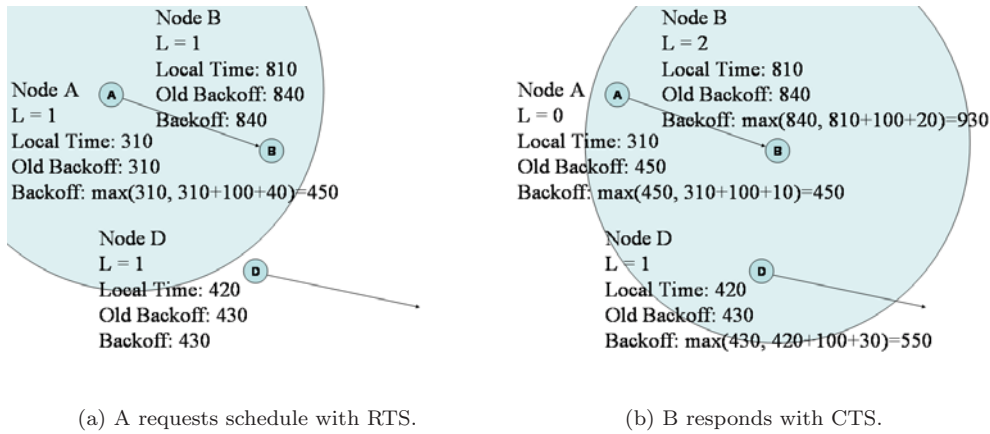


Figure 1: RTS/CTS Example

3 The DuraNet Protocol

3.1 A High Level Description

Our goal in DuraNet is to maximize the amount of time each node sleeps, while ensuring all messages reach their intended destination. The DuraNet protocol achieves this goal by forming a *power schedule* for each link in the network. A power schedule consists of a sequence of wake and sleep times for endpoint nodes.

At the beginning of network lifetime, DuraNet enters a Topology Formation Phase and obtains a network topology from a separate routing layer. Then the DuraNet protocol enters a *Sync Phase*. During the Sync Phase, parents and children negotiate transmission schedules that will be used repeatedly during the *Data Phase*. From the application's intended data generation period, the *application period*, and the time it takes to compute all power schedule, the *Settling Time*, the base station computes the *Schedule Period*, SP , and communicates this value to all nodes. During the Data Phase, nodes will reuse each schedule every SP seconds.

Our basic scheduling algorithm in the Sync Phase is a modification of an RTS/CTS protocol. During the Sync Phase, all nodes are on and listening. In addition, each node has a *Pending Packet Counter*, L , that keeps track of the number of packets it must schedule. It is an invariant that $L \leq Q$, where Q is the size of a node's buffer queue.

Every node picks a random time to send an RTS message asking its parent for a schedule for L packets. If the parent grants a schedule, it responds with a CTS. The child then decrements its value of L and the parent increments its L to reflect the committed schedule. Nodes that overhear the RTS and CTS note their local time as the *Latest Conflict Time*, LCT , and then back off.

After every link has formed a schedule, the base station notes the Settling Time, computes the Schedule Period, and then floods a phase change message to all nodes containing the Schedule Period. The network then transitions to the Data Phase, in which each link executes the formed schedule. In addition, DuraNet performs local parent-child time synchronization during the Data Phase.

During the Data Phase, the base station keeps track of the number of nodes that are still alive. This information is provided to the application, which may at any time decide to stop the Data Phase and switch back to the Topology Formation Phase. This closes the cycle and allows DuraNet to adapt to changing network conditions, such as changes in interference, joins and leaves of nodes, or changing application traffic.

3.2 An Example

We now begin an example that we will finish after discussing the details of the DuraNet phases. Consider a case with three nodes, A , B , and D , where B is the parent of A and D is neither parent nor child of A or B . Suppose the local times of these nodes are 310, 810, and 420. Then A sends an RTS requesting a schedule for sending L packets, which is one in this case, every SP seconds. The RTS is heard by B but not D . This is shown in Figure 1 above.

The parent B then responds with a CTS that is heard by all three nodes. On transmission of the CTS, the parent B has committed to wake up every SP seconds later and receive A 's packet. Therefore A decrements its L to 0 and B increments to 2, while D 's value of L remains 1. In addition, all three nodes compute a new backoff time in a way that will be explained in the following sections. This is shown in the second part of Figure 1.

We now describe each phase in more detail.

3.3 Topology Formation Phase

DuraNet treats topology formation as a service provided by a separate *routing layer*. This routing layer must provide the following to DuraNet:

1. **Termination-Notice:** the routing layer must signal DuraNet when the routing topology has completed construction and the Routing Phase has completed.
2. **Topology Information:** the routing layer must provide to DuraNet a node that acts as a parent. DuraNet does not require any other child information.
3. **Descendant Count:** the routing layer must provide to each node a count of its total number of descendants.
4. **Node Count:** the routing layer must provide to the base station a count of the total number of nodes in the network.

DuraNet acts as a queue interface for packets to be routed; packets are deposited into a queue maintained by DuraNet and flushed according to the transmission schedule.

3.4 Schedule Formation: The Sync Phase

We now give the details of the Sync Phase. Assume for simplicity that every node generates exactly one packet and aggregation is not used. Time is always quantized at the level of a hardware timer clock cycle.

3.4.1 Schedule Negotiation

To form a power schedule for a link, a node n_i sends an RTS message containing the name of its parent n_j , its node ID, and the value of its pending packet counter L_i . The RTS is a request to the parent to schedule L_i packets for transmission. The parent, on receiving the RTS, may ignore the RTS. We elected to have the parent drop the RTS instead of sending a negative acknowledgement to try again later because an RTS-NACK would require an extra message and the parent does not know when it will next be available. If the parent accepts the request, it responds with a CTS message containing the number of packets it will accept. Other nodes which overhear the RTS or CTS will set their LCT and increment their backoff counter in a way described in the next section.

We perform RTS/CTS to solve the one-hop hidden node problem. Because we plan to re-use this schedule many times, we want to avoid conflicts. Because that the interference range of a radio is larger than its communication range, we know that this cannot guarantee a conflict-free schedule; multi-hop hidden nodes can exist[1]. Nevertheless, RTS/CTS allows consistent schedules and avoids many common conflicts without requiring too much communication between all nodes in a local neighborhood.

Unlike a traditional RTS/CTS protocol, nodes back off if they overhear an RTS, even without overhearing a corresponding CTS. This gives a node two chances to back off in case an RTS or CTS is dropped. In addition, if we do not back off on the RTS, then the neighbors of the RTS sender need some mechanism to determine if the handshake occurred; one such mechanism is a CTS-ACK, which we elected not to include due to the extra cost and complication of the protocol.

In addition, our protocol allows parents to throttle child demand based on available buffer queue space. A parent may answer an RTS with a CTS allocating fewer packets than requested. In this case, the child backs off and tries again. Because we know that schedules will be executed every SP seconds, the value of L is the size of the future buffer queue. With this information, we can prevent buffer queue overflow in the data phase. Regardless of whether the child receives all of its desired allocation or not, after the CTS the parent increments its pending packet counter L_j and the child decrements its pending packet counter L_i by the number of packets in the *CTS*.

If aggregation is used, the child first determines how much aggregation it can perform, then asks for the number of aggregated packets from the parent. The parent responds as before based on the amount of space in its queue. The base station can still determine how many children have completed their schedules.

When a node hears concurrent *RTS* and *CTS* requests, we need to be careful to favor the reservation closer to the root of the routing tree. This means that a node services requests with the following priority order: 1) CTS receive 2) CTS send 3) RTS receive 4) RTS send. Messages with lower priority are interrupted by the higher priority messages.

As a consequence, we require the ability from MAC layer to perform *RTS/CTS abort*. An RTS/CTS abort occurs when the message has been queued waiting for a clear channel but not yet sent by the MAC layer and a higher priority message enters. In this case, the MAC layer should not send the pending message. In our implementation, we achieve this by making a small change to the MAC layer to ask the DuraNet layer if a message should be sent once channel idle is detected.

3.4.2 Backoff

When a node overhears an RTS or CTS, it sets its Latest Conflict Time LCT to be $\max(\text{old}LCT, \text{localTime} + L \cdot K)$, where L is specified in the incoming *RTS* or *CTS* and K is the packet time. We must back off by $L \cdot K$ because any smaller backoff would cause the LCT to be in the middle of the overheard reservation.

Every node keeps a counter *rtsFireTime* determining when it will next fire an RTS. Whenever the LCT of a node is updated, it adjusts its *rtsFireTime* to be $\max(\text{rtsFireTime}, \text{localtime}, LCT + H + \text{rand})$. Here H is a systemwide constant “handshake gap”, aimed at ensuring schedules have some minimum gap to account for the effect of clock skew. Here *rand* is an integer chosen uniformly from $(0, B]$, where we describe how B is set in the next section.

3.4.3 Decreasing the Settling Time

Settling time determines our maximum bandwidth, and therefore we would like it to be as low as possible. How we back off has a large effect on our settling time. Our backoff technique determines the value of B . For example, if we use an additive backoff technique, then we increment B by 1 every time we have an RTS preemption as mentioned above, while a multiplicative backoff would multiply B by some constant. We know that constant backoff is a bad idea due to channel capture. We also considered incorporating history into the backoff, but because there are fewer contenders over time in the Sync Phase, we believe that history will not predict the future very well.

In addition, we try to aggregate sending as much as possible. It is much better for our settling time if a node can send many packets at once instead of contending for each packet individually. Therefore, we introduce a “wait-queue” heuristic: a node delays scheduling with its parent until it receives packets from all its children.

This heuristic only helps if the queue size of the node is larger than the number of packets D it must handle from its unserved descendants. Therefore, there are three cases in which a node should contend for a schedule with its parent. Either $L = Q$, $D > Q$, or $D = 0$ will be the case. We have already covered $L = Q$.

For $D > Q$, we know that the node will have to contend at least twice, so it might as well go ahead. In the $D = 0$ case, the node is effectively a leaf and does not need to wait. We discuss the choices used by our implementation and show data on the effects of these heuristics on our settling time in Section 4.

3.4.4 RTS/CTS Sequence Numbers

We use sequence numbers to recover from RTS/CTS loss due to lossy links. The sequence number increments whenever a commit occurs. A commit is defined as a send of CTS for the parent, and the receive of the CTS for the child. If an RTS is dropped first, then the sender will simply retry. If a CTS is lost, the parent commits after the send of CTS and increments, but the child does not because the CTS is lost. The child will then retry with the old sequence number. At this point the parent detects the old sequence number and sends a recovery CTS message to the child with the parent's new sequence number and relative time of the old schedule. At this point the child commits and increments its sequence number. If the recovery CTS is dropped, the child simply retries sending RTS again, until it receives a recovery CTS. Because the recovery CTS does not correspond to a time when the channel will be busy during the Data Phase, other nodes do not back off on overhearing a recovery CTS.

We opted for sending a recovery CTS over CTS retransmission because the exact time of reception of CTS determines when the power schedule starts in the future. A retransmitted CTS would give an incorrect start time to the child and to all neighbors of the parent. To avoid this problem, we would need to put additional information into the retransmitted CTS and overhearing nodes would need to use this additional information to adjust their *LCT* values. These nodes would also need to account for hearing the same CTS twice. Therefore CTS retransmissions would complicate the protocol for negligible benefit over a recovery CTS.

3.5 Data Phase

In the Data Phase, nodes execute the power schedule formed during the Sync Phase. At the beginning of the Data Phase, the base station computes the Schedule Period SP as the maximum of the Settling Time and the application period and floods it to the network.

Then during the data phase, each pair of nodes runs its schedule every SP seconds. The child sends its scheduled packets to the parent, and the parent replies with an acknowledgement message. These packet will be routed up the tree and delivered to the base station in a number of seconds less than or equal to the schedule Settling Time.

3.5.1 Media Access

We chose not to enable CSMA during the execution of a power schedule. Because the schedule should already avoid collisions for all nodes in the network, retransmission should only be required due to link unreliability. As a node shuts off its radio at the end of its scheduled slot, enabling CSMA backoff means that less time is available for attempting retransmissions within the transmission schedule. Reduced retransmissions in this case translates directly to reduced reliability. In addition, CSMA may backoff due to environmental noise, which is useless. In fact, noise at the sender does not matter, because it is at the receiver where packets collide.

DuraNet does not require, but can take advantage of radios that support multiple frequencies. Nodes can pick a frequency at random during the Sync Phase for use during the Data Phase and then communicate this choice to their parent by including the frequency in the RTS/CTS during the Sync Phase. Parents then listen during the Data Phase on the child's frequency. This greatly reduces the chance of collisions.

3.5.2 Dealing With Clock Skew

Because clocks drift over time, we pad schedules with a *left-guard time*. Each parent wakes up ahead of schedule by the left-guard time, which is a systemwide constant. In addition, a child corrects its timer based

on when it receives the parent acknowledgement message. The parent, in turn, can calculate how much a child skews by comparing the child's scheduled data arrival time with its actual arrival time. By relying on the timing of existing messages, we achieve parent-child time synchronization without extra overhead. Parent always adjust the child's local time to match the parent's local time.

3.5.3 Recovery From Network Change

Instead of spending energy to change schedules as network conditions change, DuraNet tries to come up with a schedule is robust enough to survive small changes. The parent-child time synchronization in DuraNet means schedule collision due to clock drift is less likely, especially in small networks. If link reliability changes, this just means more or fewer retransmissions within the schedule.

In addition, DuraNet tries to detect and recover from catastrophic changes, such as permanent change in interference topology. We take advantage of the parent acknowledgement to provide a command channel from the base station and perform link quality estimation. Because the base station has a global view of the network, it can easily tell if something goes wrong based on end-to-end reliability, then send a command to switch phase through the acknowledgement message.

In addition, the base station places a sequence number of its own in acknowledgements. This special sequence number is not incremented by other nodes in the network and is propagated to all nodes. Nodes keep track of the time between increments in this sequence number. If this time becomes too large, the node assumes it has been partitioned. In this case, the node ceases to follow its power schedule and switches to low-power listening mode. At phase change time, other nodes that have not been partitioned can then beacon a phase change message with low-power listening preamble in an attempt to reacquire these partitioned nodes.

In effect, we use low-power listening to create a special announcement channel in case of emergency. During normal Data Phase operation, no extra listening for announcements is needed because all commands can be carried on parent acknowledgements. If a node times out on hearing a base station increment from its current parent, however, then that parent is most likely permanently disabled. Low-power listening allows the child to listen for phase change messages from any node, not just its parent.

3.5.4 Schedule Durability

The base station is the final arbiter of schedule lifetime, but the durability of the schedule may be impacted by several factors. First, nodes may drop out of the network due to hardware failure or battery exhaustion. We increase battery lifetime by following the DuraNet schedule, but if a node does fail, the base station will detect the failure by end-to-end reliability and signal the application. The application may then decide to switch to the Topology Formation Phase, which will form a new topology with the surviving nodes. Partitioned nodes will go to low-power listening as mentioned above and can be reacquired at phase change time. In the case where aggregation is used, the base station can no longer reliably determine the status of single nodes, but it can still detect degradation in the network.

Second, given a static network with a schedule, link reliability should not change very much because noise from collisions is greatly reduced. At the same time, links may decrease in reliability to the point of becoming unusable. Again, the base station will notice this link due to end-to-end reliability and will signal the application. The application may then decide whether to wait out the decrease in reliability or to switch to Topology Formation Phase.

Third, interference neighborhoods may change over time. Two nodes not previously neighbors may begin to interfere with each other, thereby compromising the DuraNet schedule. While the base station cannot directly detect changes in interference, it can notice which nodes have become unreliable in the current schedule. At this point, DuraNet can signal the application, which then decides whether to wait out the change or switch to Topology Formation Phase.

Finally, clocks may drift between nodes. Our parent-child time synchronization ensures that schedule drift between neighbors is bounded by a function of the number of hops to their common ancestor. This follows because errors accumulate over each hop. In small networks, we expect the number of hops to be small, and therefore the amount of schedule drift to be small. In larger networks, the number of hops to

common ancestor may be large, and therefore the errors may cause schedule conflicts from drift. We believe that such long-hop networks are best broken into smaller ones and run DuraNet on each network with a different frequency.

3.6 Towards Building Schedules for Fix-Up

In addition to performing maintenance during the Data Phase, we might build schedules in the Sync Phase that allow for “fix-up” at some small cost. We now consider a variant of schedule formation and a companion schedule maintenance protocol that allows a schedule to perform local fix-up. We have not implemented or measured this variant; we include it to illustrate how DuraNet might be extended in future work. These ideas touch on the area of “adaptivity” considered in full by Hohlt et. al., but are not nearly as comprehensive as that work[10].

In the Sync Phase, we change the semantics of an RTS. As before, every node sends an RTS to its parent in the route tree. All nodes pick a constant number of potential parents to overhear during the Data Phase; the schedules of these potential parents are noted by overhearing their RTS and CTS times. In addition, we require that each node queue all packets from its descendants and transmit only once during the Data Phase.

Then during schedule execution in the Data Phase, we perform maintenance as follows. Each node during each period randomly picks a time for a receive slot to listen for new children. During a node’s transmission slot, it sends the name of its intended parent, the time of its next transmit slot, the time of a short receive, then a route msg, its own data message, and finally all messages from its descendants. We refer to parent name, time of next transmit slot, and time of receive slot as the Schedule Beacon.

Every node wakes up to overhear the Schedule Beacon of all its potential parents, but does not need to hear the rest of the transmission. Overhearing the Schedule Beacon allows the node to perform time synchronization with the potential parent and learn the time of the receive slot. If the node’s original parent dies, the node can then tell the potential parent during the potential parent’s receive slot when to listen so as to replace the original parent. In effect, the child tells the potential parent to pick up the original parent’s schedule. The child uses CSMA and gives up if it detects a collision.

Notice that because we simply switch out one parent for another in the same schedule, the RTS/CTS protocol in the Sync Phase should prevent collisions from other nodes. This depends on the schedule being large enough to accomodate the traffic of the new parent and child; we can enforce this by careful potential parent selection. Notice also that the new parent does not need to know anything about the child before picking up the child’s schedule. Each node keeps track only of potential parents.

This maintenance protocol allows for children to switch parents in the case of untimely death at the cost of waking up to overhear schedule beacons and communication through the receive slot. Further, because each node keeps track of a constant number of potential parents, the protocol requires only a constant amount of space.

The number of potential parents and method of parent selection requires investigation. We would like to minimize the number of potential parents kept by each node, while maximizing the chance that the node can reach at least one parent during network failures. The work of Godfrey and Ratajczak suggests that tools from percolation theory may be useful here [8]. Another approach might take node neighborhoods into account and attempt to build graphs with high reliability but respecting locality; here we might look at ideas from work such as Brewer, Chong, and Leighton for building fault tolerant graphs that consider “metanodes” (sensor neighborhoods), within each of which are many “channels” (sensors) [2]. Part of the challenge will be finding an algorithm with good properties that requires a minimum amount of communication between nodes and minimizes the number of restrictions on the routing layer.

On the negative side, the protocol now requires that each node be able to route through two potential parents, which may constrain the routing layer. In particular, this fix-up protocol does not allow for re-scheduling of buffer queue space without performing a full Sync Phase. Therefore, to avoid a Sync Phase, potential parents must have enough queue space to schedule all potential children and all their descendants. In addition, massive localized failure might disable all of a node’s potential parents. In such a case, the base

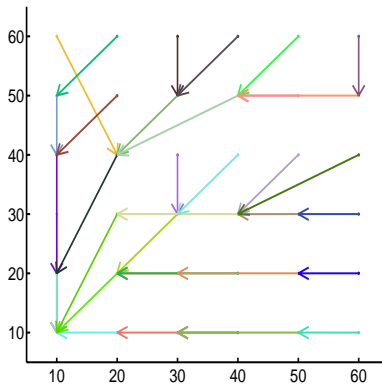


Figure 2: Sample MT Topology on 6x6 Grid.

station signals a phase change, the network is re-formed, and new schedules created.

4 Protocol Simulation

We simulated DuraNet using TOSSIM[12] and a custom version of TinyOS 1.1 [9]. For DuraNet timers, we used TOSSIM wall clock time instead of the TinyOS 1.1 Timer component. For random number generation, each mote used the LFSR component from TinyOS 1.1 seeded using a constant unique to each test run added to the node’s NodeID. For a routing layer, we used Blast version 0.2, which is described by Woo, Tong, and Culler [18]. We implemented DuraNet as a separate module for Blast that controls the sending queue.

We compare against low-power listening and CSMA. For a fair comparison, we considered low-power listening with a 1 percent duty cycle. This is the lowest duty cycle currently supported by the Mica2 platform with ChipCon 1000 radio. For CSMA, we used the method implemented in the TinyOS 1.1 radio stack.

Then we ran DuraNet on simulated network topologies and interference graphs. For our simulation test runs, we obtained topologies by first generating grids with 10 foot spacing. These grids were then provided with interference patterns produced by the TOSSIM tools. These tools use an empirical trace to create realistic interference patterns. We then used Matlab to run the Minimum Transmission path metric on the interference graph; the output was hard-wired as a routing tree into our simulation code for all runs of the same size, including CSMA and low-power listening. The routing tree and interference graph was fixed for all runs on the same grid size. A sample MT topology for a 6x6 grid is shown in Figure 2. In this topology, the base station is located at coordinate (10,10); arrows represent parent-child relations. The maximum distance from the base station in this topology, therefore, is five hops. The expected number of transmissions and retransmissions from the farthest node is about six.

As part of our implementation, we made choices for DuraNet specific parameters. Unless otherwise specified, our implementation of DuraNet uses an additive backoff regime. In addition, we set the Handshake Gap to be 5 ms, and our right-guard time to be 50ms. Our packet time is 30 ms, and the default queue size was 20. As a default, we ran all simulations for 10,000 seconds unless otherwise specified. We also implemented clock skew, at a rate of 1ms per 50,000ms, which is realistic for the Mica2 platform.

4.1 Simulation Experiments and Results

We now present the results of our simulations of DuraNet. First we examine how the Sync Phase settling time depends on network size, backoff, and queue behavior. Then we compare DuraNet’s reliability and power consumption to CSMA and Low-Power Listening.

4.1.1 Settling Time

Because the Sync Period settling time determines the maximum bandwidth of DuraNet, we need to know how it scales with the size of the network. We would also like to know how the usage of the “wait-queue” heuristic detailed in Section 3.4.3 affects our final settling time. We investigated how settling time increases with the size of a grid network with MT routing topology as defined above. In particular, we considered DuraNet with additive backoff both with and without the wait-queue heuristic. In addition, we simulated an ideal version of DuraNet using additive backoff with wait-queue heuristic and an unlimited queue. Each parameter setting was run four times on grid sizes ranging from 3x3 to 13x13. All nodes began with $L = 1$. A scatterplot of our data points is in Figure 3

The settling time for DuraNet with infinite queue and wait-queue heuristic seems to increase linearly with the size of the grid. This result makes sense, because with an infinite queue size the wait-queue heuristic ensures maximum aggregation. On the other hand, with a limited size queue, the wait-queue heuristic can only help if most nodes have a number of unserved descendant packets D that is smaller than Q . In a grid which is very large compared to Q , many nodes have $D > Q$. As a result, these nodes will go ahead and schedule with their parent as described in 3.4.3, causing contention; in effect it is as if the heuristic were not in use.

Without the wait-queue heuristic, the settling time increases exponentially as the size of the grid increases. With the wait-queue heuristic and the default queue size of 20, the settling time is close to that of the infinite queue case until the 11x11 grid. At this point it begins to increase rapidly.

Our data shows that queue size in conjunction with the wait-queue heuristic is an important parameter for reducing the settling time. In addition, for the workload and topology given, we see that the settling time is on the order of two minutes for grid sizes up to 11x11.

The data also shows that the DuraNet Sync Phase schedules packets at a high rate. In our 11x11 MT topology, we need 619 transmissions in order to forward all packets from their sources to the base station. DuraNet is able to fit $\frac{619}{120}$ packets and their retransmissions into a second. Because the channel bandwidth is roughly 20 packets per second, DuraNet utilizes a significant fraction of the channel.

4.1.2 End-to-End Reliability

Our next goal is to understand the end-to-end reliability of DuraNet and how it compares to CSMA and Low-Power Listening. Saving power is of little use if it comes at the expense of reliability. For our experiment, we fixed the grid size at 10x10 and again used an MT topology over TOSSIM-generated interference graph. We then used a synthetic, periodic workload and varied the data period.

We used DuraNet with additive backoff and wait-queue heuristic. Each setting was run for four times at each data rate. The results are shown as a scatter plot in Figure 3 .

At high data rates, we see that low-power listening suffers due to its long preamble, leading to network congestion. At lower data rates, however, low-power listening matches the reliability of CSMA.

DuraNet, in contrast, has lower reliability than CSMA, although it still correctly delivers at least 75 percent of packets in all cases tested, with most runs around 85 percent or more. We believe this decreased reliability is due in part to the fact that a node in the Data Phase will not retransmit a packet if the retransmission would cause the node to be awake past the end of its schedule.

4.1.3 Energy Consumption

For our investigation of the energy consumption of DuraNet, we fixed the grid size at 6x6 and used an MT topology over TOSSIM-generated interference graph. We then used a synthetic, periodic workload and varied the data period as before. For our power numbers, we wrote a shim over GenericComm to count transmit, receive, idle listening, and radio sleep time. We then used the current draws reported in the CC1000 data sheet to calculate energy consumption[3].

We again compare to CSMA and low-power listening with 1 percent duty cycle. In addition, we introduce a “no-listen CSMA.” This is an ideal variant of CSMA in which idle listening is free; only transmissions are

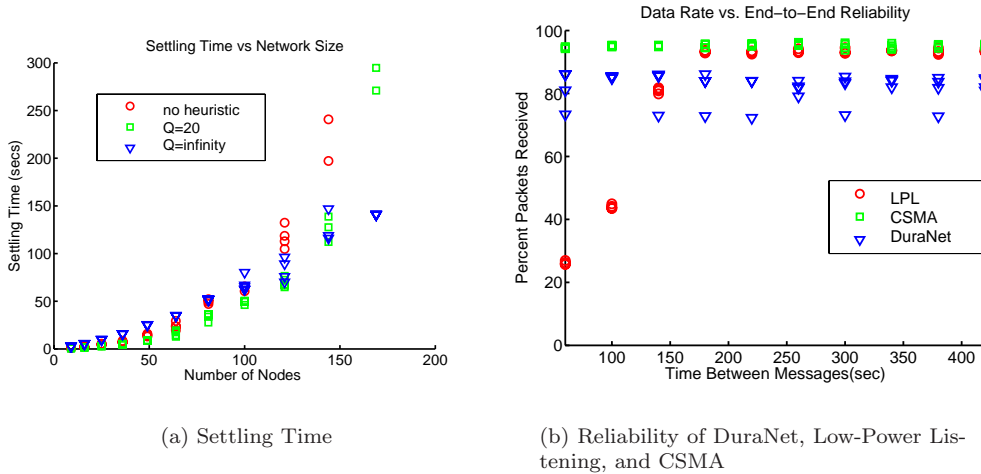


Figure 3: Settling Time and Reliability

charged. We include no-listen CSMA as an estimate of the minimum energy required to reliably deliver the given workload to the base station. Each was run for 4 trials at each data rate. As we can see in Figure 4 (a) CSMA uses *vastly* more power than no-listen CSMA, DuraNet, or low-power listening, which makes sense as it was not designed to save power. A more reasonable comparison is between DuraNet, no-listen CSMA, and low-power listening with 1 percent duty cycle. The results are in Figure 4 (b).

All three methods use less power as the data rate decreases, because they send less. Low-power listening, however, uses more power than either DuraNet or no-listen CSMA, and decreases its power consumption more slowly than DuraNet. In contrast, DuraNet can achieve a power schedule that matches or beats no-listen CSMA. In part, DuraNet can beat no-listen CSMA because the RTS/CTS avoids one-hop hidden nodes, while no-listen CSMA does not. Hidden nodes may lead to conflicts, which in turn require energy-consuming retransmissions. Another part of DuraNet’s low power result, however, may be due to its lower reliability, leading to fewer packets sent. We also see that while DuraNet can achieve a schedule with energy consumption close to no-listen CSMA, some runs of DuraNet obtain a schedule that consumes significantly more energy, although still less than low-power listening; we do not currently have a good explanation for these runs.

The graph is misleading in the sense that it suggests that the gap between DuraNet and low-power listening to close as data rate drops. In fact, low-power listening curves downward on our graph simply because it sends fewer packets; as the simulation time is fixed at 10,000 seconds, a lower data rate implies fewer packets. We expect the gap between low-power listening and DuraNet will widen with increased application duration, as low-power listening incurs overhead on every packet.

4.2 Energy Distribution

Comparing the amount of time spent by DuraNet in transmission, receiving, and idle listening with that spent by CSMA and low-power listening is instructive. We measured the time devoted to each operation on a 6x6 grid with synthetic periodic workloads of three different data rates, with four runs each. The data rates and the average energy consumption for each operation are shown in Figure 5. In this graph, T denotes energy cost due to transmission of messages, R cost from receiving, and I cost from idle listening. The 100s, 260s, 380s bars show results from a workloads with periods of 100 seconds, 260 seconds, and 380 seconds, respectively.

The graph shows us that while low-power listening has very low amounts of idle listening, it pays by

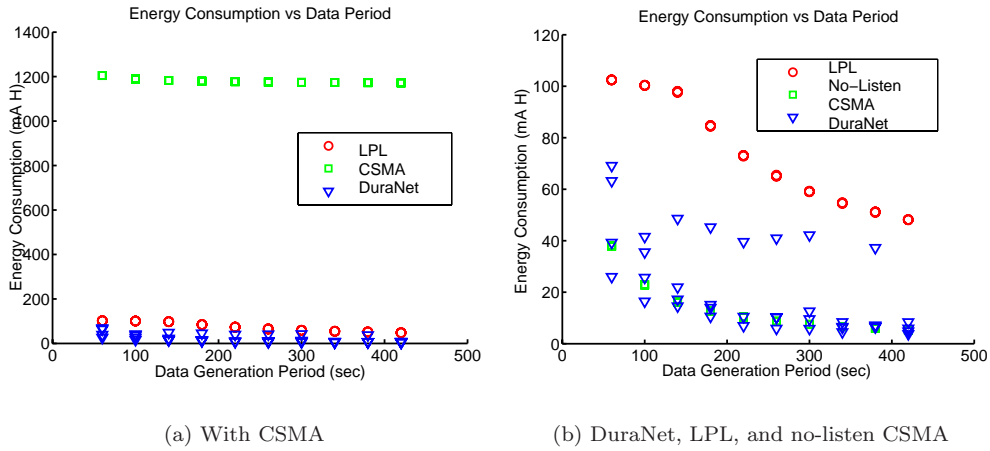


Figure 4: Energy Consumption

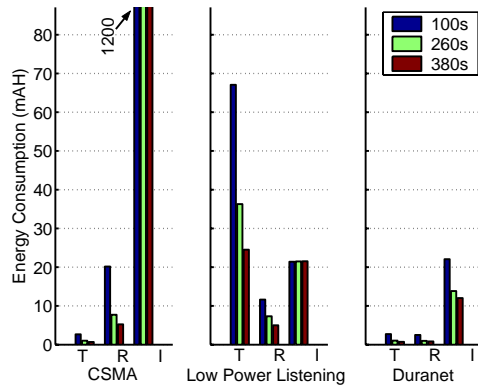


Figure 5: Transmission, Receiving, and Idle Listening Time

spending energy in transmission of the preamble. CSMA, by contrast, spends very little energy in transmission, but has an unacceptable idle listening cost as it never turns off the radio. While low-power listening still pays some idle listening cost to spike and probe for a preamble, it achieves a cost of 20 while CSMA has a cost of 1200, a remarkable improvement. DuraNet’s transmission cost, on the other hand, is close to that of CSMA while its idle listening cost is close to that of low-power listening.

Both CSMA and low-power listening pay an overhearing penalty as well. DuraNet uses less power for receiving messages than either because it avoids overhearing packets aimed at other nodes.

In addition, we see that while the transmission cost of low-power listening decreases at the lower data rate, the listening cost remains similar. In contrast, DuraNet can take advantage of the lower data rate to decrease its listening cost; the only idle listening in DuraNet occurs due to guard times and sync overhead, which can be decreased at lower data rates. In fact, for the periods shown, DuraNet uses less than 2 mAH of power for protocol overhead. This gives us insight into why DuraNet showed less energy consumption than low-power listening in our simulations.

5 Conclusions and Future Work

We learned several lessons from DuraNet. First, radio power schedules are extremely effective at saving energy, especially if they can be reused many times. In our simulations, the schedule built by DuraNet consistently required less energy than low-power listening with 1 percent duty cycle and came close to no-listen CSMA on synthetic, periodic workloads.

Second, the cost of building a schedule that gives us desirable properties is low. DuraNet's Sync Phase uses RTS/CTS to avoid the one-hop hidden node problem and prevent routing buffer queue overflow. In our simulations, the settling time of DuraNet's Sync Phase was low, especially in the case where nodes have a queue size that is large compared to the size of the network.

Third, we can make a schedule more durable by leveraging messages exchanged during schedule execution. DuraNet piggybacks time synchronization on parent acknowledgements, allowing for easy parent-child synchronization. Parent acknowledgements also give DuraNet an effective command channel that reduces the need for global announcements during the Data Phase. Our proposed schedule maintenance protocol allows for children to switch parents if necessary by overhearing a special beacon message with a parent's receive slot.

Finally, by managing network state at the base station, we can simplify the creation and monitoring of schedules. In the DuraNet Sync Phase, the base station determines the Schedule Period and when phase change occurs. During the Data Phase, the base station can signal the application when schedule performance degrades. In addition, the base station sequence number in the parent acknowledgement allows nodes to detect when they have been partitioned and stop following a schedule.

A major question for future work will be how well DuraNet schedules endure in the real world. A related question is how to build local schedule maintenance protocols to perform fix-up as schedules fail. To investigate these questions, we can implement DuraNet on the Mica2 mote platform and take many measurements. As an intermediate step, we could simulate DuraNet on a wider variety of workloads and changing topologies. Also interesting would be theoretical models and approaches to increasing durability of schedules, route trees, or other objects on top of changing interference graphs.

6 Acknowledgements

We thank Eric Brewer, David Culler, and David Wagner for feedback and support at various points of writing this paper. We thank Barbara Hohlt for helpful discussions about power scheduling in general and her work in particular. We also thank Umesh Shankar, Rob Johnson, Brighten Godfrey, and David Ratajczak for pointing us to their work. Phil Levis suggested the term "slot-free" scheduling. Rob Szewczyk showed us the data repository from Great Duck Island. Finally, we thank the NEST group as a whole for providing motes and TinyOS as a platform for this research.

References

- [1] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang. MACAW: Media access protocol for wireless lans. In *Proceedings of the ACM SIGCOMM Conference*, 1994. <http://citeseer.nj.nec.com/bharghavan94macaw.html>.
- [2] Eric A. Brewer, Frederic T. Chong, and F. Thomson Leighton. Scalable expanders: Exploiting hierarchical random wiring. In *STOC '94*, 1994. <http://tinyurl.com/zk0x>.
- [3] ChipCon. Cc1000 data sheet, 2002. <http://tinyurl.com/z1r9>.
- [4] Sinon Coleri. PEDAMACS: Power efficient and delay aware medium access protocol for sensor networks, 2002. Master Thesis, University of California Berkeley, Dec 2002 supervised by Prof. Pravin Varaiya.

- [5] W. Steven Conner, Jasmeet Chhabra, Mark Yarvis, and Lakshman Krishnamurthy. Experimental evaluation of topology control and synchronization for in-building sensor network applications. Technical report, Intel Corporation, AUGUST 2003.
- [6] Inc Dust. Dust, inc. web page. <http://www.dust-inc.com>.
- [7] Saurabh Ganeriwal, Ram Kumar, , and Mani B. Srivastava. Timing-sync protocol for sensor networks, 2003. UCLA.
- [8] Brighten Godfrey and David Ratajczak. Naps: Scalable, robust topology management in wireless ad hoc networks, 2003. Personal Communication.
- [9] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for network sensors, November 2000.
- [10] Barbara A. Hohlt, Lance R. Doherty, and Eric A. Brewer. Flexible power scheduling for sensor networks, 2003. UCB CSD-03-1293, under submission.
- [11] F. Koushanfar, A. Davare, and DT Nguyen. Low power coordination in wireless ad-hoc networks, 2002. EECS Dept. UC Berkeley.
- [12] Phil Levis, Nelson Lee, Matt Welsh, and David Culler. Tossim: Accurate and scalable of entire tinyos applications. In *First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, 2003.
- [13] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler, and John Anderson. Wireless sensor networks for habitat monitoring. In *2002 ACM International Workshop on Wireless Sensor Networks and Applications. WSNA '02, Atlanta GA, September 28, 2002.*, 2002. (also Intel Research, IRB-TR-02-006).
- [14] V. Rajendran, K. Obraczka, and J.J. Garcia-Luna-Aceves. Energy-efficient, collision-free medium access control for wireless sensor networks. In *ACM SenSys 2003*, November 2003.
- [15] Umesh Shankar and Rob Johnson. A multichannel communication system for sensor networks, 2003. Personal Communication.
- [16] S. Singh and C. Raghavendra. Pamas: Power aware multi-access protocol with signalling for ad hoc networks, 1999.
- [17] K. Sohrabi, J. Gao, V. Ailawadhi, and G. Pottie. Protocols for self-organization of wireless sensor network. In *IEEE Personal Communications*, 2000.
- [18] Alec Woo, Terence Tong, and David Culler. Taming the underlying challenges of multihop routing in sensor networks. In *First ACM Conference on Embedded Networked Sensor Systems*, November 2003.
- [19] Wei Ye, John Heidemann, and Deborah 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)*, June 2002.