ABSTRACT

MIN, JEONG KI. Implementation of DRAND, the Distributed and Scalable TDMA Time Slot Scheduling Algorithm. (Under the direction of Professor Injong Rhee).

The problem of energy savings is the most important subject currently in the research area of wireless sensor networks. So, in order to present a better scheme for energy savings and system performance, the TDMA scheme is considered as a solution. Moreover, the TDMA time slot scheduling algorithm is an important issue in running the TDMA scheme. The distributed and scalable fashion is required in wireless sensor networks because it is very difficult and inefficient to manage many sensor nodes by the centralized method with small size of memory space and battery capacity on each sensor node deployed in the broad sensing field. So, we implemented DRAND, the TDMA time slot scheduling algorithm which supports the important requirements as we listed above. Even though a scheme shows good performance by the simulation result, the implementation as a real system is another problem to solve. This is because good simulation results could not guarantee that implementation of the algorithm would work properly in the real word due to various unexpected obstacles. Therefore, by implementing the DRAND scheme as a real system, we can confirm the analysis and simulation result with various real experiments. For the experiment, we use up to 42 MICA2 motes for one-hop and multi-hop test.

Implementation of DRAND, the Distributed and Scalable TDMA Time Slot Scheduling Algorithm

by

Jeong Ki Min

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial satisfaction of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh

2005

Approved By:

Dr. Rudra Dutta	Dr. Jaewoo Kang

Dr. Injong Rhee
Chair of Advisory Committee

To my parents,

Byung-Kook Min

and

Ran-Sook Kim.

Biography

Jeong Ki Min was born and grew up in Seoul, South Korea. He received his Bachelors degree in Computer Science from Hanyang University, Korea, in 2002. He has been a Masters student in the Department of Computer Science at North Carolina State University since August 2003. He will continue work on a Ph.D. in the same department starting August 2005.

Acknowledgements

A special thanks to Dr. Injong Rhee, my advisor, for his support and commitment. I would like to thank him for all his efforts guiding me throughout my research and sharing my personal difficulties. I am also grateful to him for his financial support and a great work environment he has provided me during my research. It has been my pleasure and honor working with him.

I would like to thank Dr. Rudra Dutta and Dr. Jaewoo Kang for being on my advisory committee and providing valuable comments on my research.

I would like to thank my colleagues in our Lab, Ajit Warrier, Sangjoon Park, Hyungsuk Won, Sangtae Ha and Gopala Krishnan for their advice and encouragement throughout the entire duration of my research and thesis preparation.

I would like to thank all my friends at North Carolina State University and in Korea for their concerns and precious time when I was in need. They have been of great help and pleasure to my life in Raleigh.

I would like to thank Gary Roy Weinberg and Robert Franklin Melvin. They have helped me a lot on my thesis.

I also would like to thank my parents and brother for their endless love, support, and everything they have provided me for the whole entirety of my life. I appreciate their patience and trust during my long pursuit of study. I would not have accomplished this without them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Wireless Sensor Network in Use

Small, but highly advanced technology such as low power wireless communication, microprocessor hardware, and microsensors mainly may lead to change in our life style. Using the sensor node which is very tiny and includes several sensing modules, we can run typical sensing tasks that measure temperature, light, vibration, sound, radiation, etc.

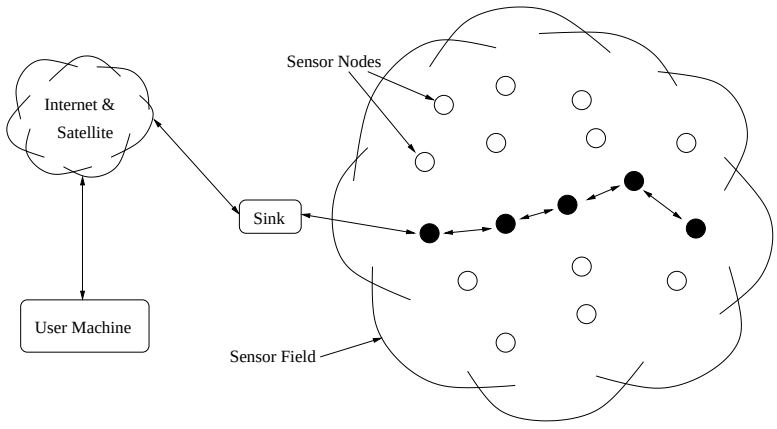

Figure 1.1: Sensor nodes scattered in a sensor field.

Currently, wireless sensor network is used in various fields for diverse purposes. For example, in the forest, sensors are monitoring and tracking birds, animals, and insects for habitat related research. And in the battlefield, they are used for detecting a nuclear, biological or chemical attack from the enemy and tracking the movement route of enemy weapons like tanks.

As Figure 1.1 shows, after sensing and translating the aimed phenomenon into a kind of data format from the field, these sensor nodes send their data toward the sink node using the wireless communication system, and the sink node forwards the gathered data to the user's computer by the traditional network systems, such as the Ethernet, which provides the specific desired information to the user.

While much research has been done on some important areas of the wireless sensor network such as architecture and protocol design, energy conservation, and localization, wireless sensor networks are still a largely unexplored research field. The reason is mainly because while sharing many commonalities with existing ad hoc network concepts, these networks of sensor nodes have a number of differences and specific challenges.

1.2 WSN Basics and Research Challenges

The wireless sensor network is an application specific and data centric system. The large number of combinations of sensing, process, and communication require many different and possible application scenarios. And most importantly, the low cost and low energy supply will require, in many application scenarios, redundant deployment of wireless sensor nodes. As a consequence, the most significant concern of any one particular node is considerably reduced as compared to traditional networks. A more important factor is the data that these nodes can observe.

Since these networks have to interact with the environment, their traffic characteristics can be expected to be very different from other, human-driven forms of networks. A typical consequence is that wireless sensor networks are likely to exhibit very low data rates over a large time scale, but can have very heavy traffic when something happens unexpectedly. Moreover, wireless sensor networks have to scale to much larger numbers of entities than current ad hoc networks, requiring different, more scalable

solutions.

Above all, energy consumption is considered the primary metric. Sensor nodes are small and energy is very scarce. Furthermore the battery of a sensor node is often not rechargeable. The need to prolong the lifetime of a sensor node has a deep impact on the system and networking architecture.

For saving energy, there are many approaches with various methodologies such as topology control, power aware routing, and sleep management. But most of these methods depend on the current BMAC [23] protocol which is based on the CSMA scheme. BMAC is the most popular now because of its simplicity, flexibility, and robustness on MICA2 [3] mote. This scheme doesn't require clock synchronization and global topology information, and it is easy to handle node adding or node deleting without any extra operation.

BMAC [23] is great for reducing the collision among the one-hop neighbors, but for the two-hop neighbors it causes the *hidden terminal* problem because the carrier sensing does not work beyond one-hop distance. Under a high data rate, the problem could cause heavy packet losses, and as a result, each node could waste too much energy. So, finally the entire sensor network could not work well because some nodes would be dead. Some solutions for the *hidden terminal* problem were suggested before. One example of a solution is RTS/CTS scheme [31, 23], but because this scheme also leads to a high overhead through the whole network, there is no conspicuous advantage for saving energy.

Then, how can we solve problems in the CSMA scheme and get a better data transmission rate without any packet loss using the smallest energy on the wireless sensor network? For a solution of this question, we thought that the TDMA(time-division multiple access) MAC protocol could be the answer. Because every node has its own time slot and sends data only during its own time slot by the time slot scheduling, we can solve the *hidden terminal* problem by the TDMA scheme.

But TDMA has many other disadvantages as well. First, it needs a time slot scheduling algorithm which must be energy efficient, distributed and scalable. The centralized method for a time slot scheduling algorithm induces high overhead and uses too much system memory to keep the whole network information. Moreover, because the deployed area is very broad and each node could be located under very different environments, it is very difficult to handle the whole topology altogether. Second,

usually TDMA requires the clock synchronization algorithm. The complicated clock synchronization also causes high overhead and memory waste through the whole network. Third, the sensor network might go through frequent topology changes because of node failing, link status variation, and physical environment change. In order to make a change in this case, each node could spend more energy. Fourth, under low contention, TDMA gives lower channel utilization and higher delay than CSMA, because each node in TDMA only uses its own scheduled time slot, and unused slots could be wasted.

The study for solving problems we mentioned before could be a useful research area, and if we could get good solutions from research, then we could get better energy saving results under high data rate compared with CSMA protocols including BMAC [23]. For making the whole TDMA protocol, we have to handle several required prerequisites as we mentioned earlier. These prerequisites are also important research area in wireless sensor networks. The algorithm related to the distributed TDMA time slot scheduling is provided by [27] without the real implementation with TinyOS [19]. This algorithm is named *DRAND*(Distributed RAND). And the NS2 [6] simulation of [27] shows good performance results.

On top of this basic protocol, for enhanced energy savings, we can utilize various improved schemes on the TDMA scheme. For example, the *sleep scheduling scheme* is a very famous and useful method to decrease the energy consumption of each sensor node in wireless sensor networks by replacing the idle time with sleeping time. When we combine the TDMA scheme and this *sleep scheduling scheme*, each node could sleep during some period of time which is not its own nor its neighbors' scheduled time slot. Furthermore, if the MAC protocol could use the routing information, some nodes which are not on the active routing path could sleep until those nodes are needed on the active routing path. We can call this cross-layer optimization technique the *route-awareness* approach. The idea of route-awareness scheme was designed by [25]. But, this route-awareness scheme was presented by only NS2 simulation result without any implementation and real experiment in the real life.

Most of the research in wireless sensor networks are supported by the implementation and experiment in the real life using a number of real sensor nodes such as MICA2 [3] mote. Even though the theoretical simulation is very important, because there are a lot of factors which could make the experiment result in the real life very different with the theoretical simulation result, the good result by the implementation

and experiment are seriously required currently for guarantee of the precise performance of the research work.

1.3 Thesis Contributions

As we said in the earlier section 1.2, because the TDMA scheme could be a good alternative to the CSMA scheme for energy savings, there has been several schemes of the TDMA time slot scheduling [10, 11, 28, 32]. But nothing of those schemes has implemented and run the experiment in the real life.

In this thesis, we implement the DRAND algorithm which was provided by [27] using TinyOS [19] and NesC [12]. And we run several different experiments under various environments of wireless sensor networks.

We describe how to implement the DRAND scheme specifically and how to solve some problems related with the implementation. And we show the experiment results using 20 MICA2 [3] motes for the single-hop test and 42 MICA2 motes on the testbed for the multi-hop test. We will see the running time, average round time duration, average number of message transmissions, average message transmission rate of each node, overhead cost, recovery time and energy under the single-hop case and multi-hop case.

So, we can see that these results follow the theoretical analysis in [27] very well. And we can get the practical data such as average running time and energy consumption to run the DRAND scheme in the real life.

1.4 Related Work

In this section, we describe the related work about MAC protocols including CSMA and TDMA schemes, and time slot scheduling algorithms.

The Low Power Listening and preamble are devised by [16] and [13]. And, WiseMAC [14] tried to reduce the long preambles of packets after an initial packet with a long preamble and showed the better energy efficiency over IEEE 802.11. But, the improvement of WiseMAC only works for special traffic patterns, and long preambles have to be used for all broadcast packets. Then, BMAC [23] was implemented using the

Low Power Listening, preamble scheme, and Clear Channel Sensing (CCA) based on the CSMA scheme. Currently, BMAC is a default MAC layer in the MICA2 [3] mote. Moreover, BMAC provides well-defined interfaces for the flexible control of various MAC functions depending on the user's different purposes.

The CSMA based protocols have very good performance with efficient energy consumption under low contention environments. But, under high contention and multi-hop case, CSMA can face heavy packet collision because of the hidden terminal problem over the two-hop neighborhood, and in order to overcome this problem it has to do extra operations which cause the useless energy consumption.

S-MAC [31] developed a distributed coordination scheme to synchronize node sleep schedules in a multi-hop network. By scheduling node wake-up times, S-MAC enables nodes to run at duty cycles of 1∼10% by the distributed and scalable method for the feature of wireless sensor networks. And T-MAC [29] is a modified scheme of S-MAC by reducing the wake-up duration controlled by an adaptive timer and introducing future-RTS (FRTS). In addition, T-MAC achieves power saving by trading throughput and latency. But it still suffers from the same complexity and scalability problems of S-MAC.

S-MAC and T-MAC don't need to have the precise global clock synchronization with small size time slot for communication among neighbors because of using the RTS/CTS scheme. Nodes maintain periodic duty cycles to listen to channel activities and transmit data. The protocols are mainly based on the RTS/CTS scheme and involve the SYNC period before each data transmission. S-MAC and T-MAC show the efficient energy consumption under the low data rate with low duty cycle, but under the high data rate, the network overhead is much larger over the multi-hop network.

Relatively, TDMA has not been used as a good MAC layer protocol in the wireless ad hoc network including wireless sensor networks because of major limitations of centralized control and strict time synchronization. Moreover, TDMA has worse adaptability to the frequently changed environment such as node joining and deleting. However, one of the most important characteristic of TDMA is collision-free. Because of this, we can get some advantages for the hidden terminal problem and energy saving comparing with the CSMA protocol. There has been several proposals [9, 18, 20, 30] for TDMA in wireless sensor networks. Some of these scheme use cluster head [9, 15, 20] with more capacity as centralized nodes for scheduling and coordination, or assume

regular grid topologies [18].

LMAC [30] use the distributed slot scheduling scheme. Each node picks time slot independently among the slots which are not assigned to its neighbor nodes. But there is a defect on this scheduling algorithm because two neighbors can choose the same time slot simultaneously. Then, some collision can happen because of same slot assignment. In other wise, there can be a lot of unused slots as well because the frame size must be big enough to reduce the chance of overlap. TRAMA [24] uses a distributed randomized TDMA scheduling heuristics called NAMA [10]. In TRAMA, all nodes decide on time slots dynamically depending on their bandwidth needs in the next time frame. But TRAMA requires high overhead in choosing slots at every frame and also requires each node to predict its future traffic. Furthermore, the protocol is too complicated for use in sensor networks. None of these TDMA schemes are implemented in real systems.

ZMAC [26] was proposed and implemented as a hybrid MAC scheme for wireless sensor networks that combines the strengths of TDMA and CSMA while offsetting their weaknesses. The main feature of ZMAC is its adaptability to the level of contention in the network so that under low contention, it behaves like CSMA, and under high contention, like TDMA. ZMAC uses the DRAND [27] which is a TDMA time slot scheduling algorithm right after deployment in the sensing field. So, after running this time slot assignment algorithm, ZMAC can have the conflict-free time slot schedule which ensures a broadcast schedule.

In order to use the TDMA scheme as a distributed and scalable MAC layer instead of the centralized method, the TDMA time slot scheduling has also been an related research issue. Most of early work is centralized and has performance dependency to O(n) where n is the total size of network. But several algorithms, [10], [11], [28], and [32], have improved the performance using the distributed method. These algorithms are developed for mobile environments where modes can frequently move and typically use many more time slots than $\delta + 1$ (for some protocols, e.g., [11],[32], these bounds are not given).

[11] tried to make the schedules independent of the detailed topology in order to improve the efficiency and robustness in a mobile environment. Instead, the schedule will depend only on global network parameters, namely, the number of nodes and the maximum degree (number of neighbors) a node can have.

NAMA [10] presented a contention resolution algorithm called neighborhood-aware contention resolution (NCR) by: each node maintaining the identifiers of its one-hop and two-hop neighbors, and making a new node or link activation decision during each contention context (e.g., each time slot).

FPRP (Five-Phase Reservation Protocol) [32] was developed as a distributed, parallel, and topology dependent scheduling protocol, which is arbitrary scalable, i.e.,its performance is not sensitive to the size of the network. By parallel, they mean that multiple reservations may be made simultaneously throughout the network - node do not need to wait for their turns to make reservations based on some ordering.

The idea of [28] is to employ a random schedule which is driven by a pseudo-random number generator. By exchanging the seeds of their pseudo-random number generators within a two-hop neighborhood, the nodes effectively publish their schedules to all hidden as well as exposed nodes.

FPS (Flexible Power Scheduling) [17] provided distributed power management protocol that exploited a tree based topology in combination with an adaptive slotted communication schedule to route packets, synchronize with neighbors, and schedule radio on-and-off times. FPS proposed coarse-grain scheduling at the routing layer to schedule all communication for the purpose of powering the radio on-and-off during idle times, and fine-grain medium access control at the MAC-layer to handle channel access.

1.5 Thesis Outline

Our thesis describes the implementation and performance result of a protocol, called DRAND (*Distributed RAND*), for scheduling the conflict-free TDMA time slot by the distributed and scalable fashion. The following describes each chapter of our thesis in detail.

Chapter 2 introduces DRAND (*distributed RAND*) algorithm for the efficient distributed time slot scheduling. Based on the theoretical algorithm, we describe how to implement the algorithm of DRAND using TinyOS and NesC following the sequence of each state step by step.

Chapter 3 describes the performance result of DRAND on the real testbed. We explain about the testbed which we use for evaluating the performance of DRAND.

In addition, we analyze the performance results through several kinds of graphs under various experiment environments.

Chapter 4 refers to an example to show how to apply the DRAND in the real TDMA system. So, as an example, we refer to the ZMAC protocol with its performance results.

Finally, chapter 5 concludes our thesis with future work and discussion.

Chapter 2

Distributed RAND

In order to get an efficient TDMA time slot schedule by the distributed and scalable method, we implement a distributed randomized channel-reuse scheduling algorithm, called *DRAND* [27]. In this chapter, we describe how the algorithm of DRAND works and how we implement the DRAND algorithm specifically for every state including the debugging method.

2.1 Introduction of DRAND

The DRAND algorithm is the modified *RAND* for the distributed and scalable network. RAND was the most popular channel scheduling algorithm because it produces time slot schedules with a high degree of channel reuse and concurrency. But RAND is not a scalable method. The main reason is that it is impossible to handle the information of the whole topology in the field because of the small size of memory in a sensor node (physical RAM memory : 4KB, EEPROM : 512KB for MICA2 [3]) and small battery capacity (2 AA size batteries for MICA2).

There exist some distributed heuristic algorithms [32, 10] to RAND. But because they do not implement channel reuse since they assign time slots dynamically for each transmission, overhead is constantly incurred. Moreover, because some slots may not even be assigned any nodes, its performance is still inferior to RAND, and it is hard

to convert it to a static version. DRAND can implement RAND in an exact sense – any schedule possible in RAND is also possible in DRAND [27].

For setting its time slot to use the channel, each node has to get the permission, GRANT message, from all one-hop neighbors and avoid using the duplicated time slot from two-hop neighbors. After setting the time slot, every two-hop neighbor shares that slot information. This method can ensure that only one node can use its time slot without any interference from other neighbors within the two-hop distance. We also keep "rounds" more synchronized by disallowing neighbor nodes to run a new round when a request is pending. This reduces the number of messages and failed rounds. In next Section, we describe the detail algorithm by the method of the implementation.

2.2 Design and Implementation of DRAND

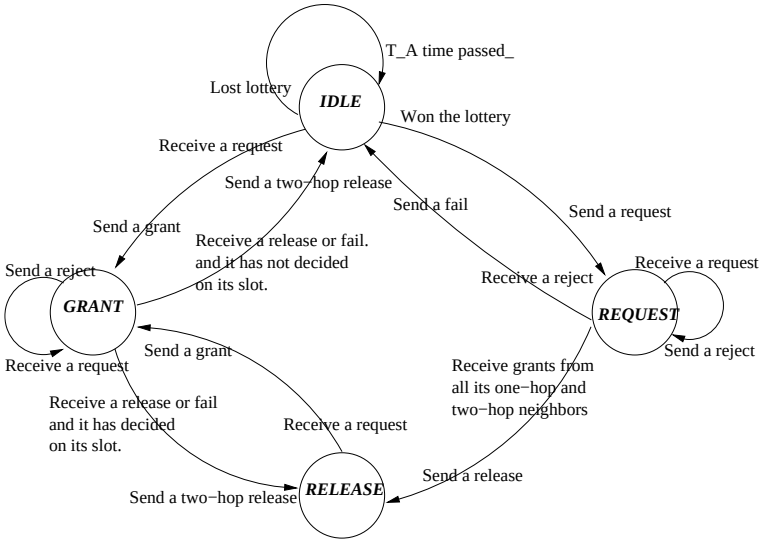

Figure 2.1: The state diagram of DRAND algorithm.

We describe how to implement DRAND specifically in this section. Because the DRAND algorithm is a small part of the TDMA scheme, in order to provide more flexible usage of small size memory to other protocol layers such as application, routing and MAC layers, the full implementation code size should be minimized. And importantly, DRAND shouldn't waste too much energy for itse1lf.

2.2.1 Basics for Implementation

DRAND has 7 states: HELLO, REQUEST, GRANT, IDLE, RELEASE, REPORT, and FRAME. Some states are changed by the timer after some amount of time. For example, after the HELLO state is starting, HELLO state goes on for 30 seconds and by the timer interruption the state goes to the next state, IDLE. But some states are changed by events which are made by one-hop neighborhood. For instance when a node is in IDLE state and receives a request message from its one of neighbors, the state is changed into GRANT state.

DRAND has 8 types of messages: hello, request, grant, release, reject, two-hop release, report, frame. Each message uses different send and receive interfaces to communicate among nodes. So, whenever a kind of message which was sent by a specific *send message function* from another node arrives, only one proper *receive message function* is called from a lower layer. We can define these specific send and receive functions for various messages using the different values of each message type. For the detail usage of each message, we will explain the detail usage of each message in the next subsections.

We use the *TIMER3* of ATmega128L processor on MICA2 motes for clock information. The TIMER3 is clocked at a frequency of $115.2KHz$. So, $115.2KHz$ means that 115200 numbers of clock interrupts happen within a second. The 16 bits timer register of TIMER3 is mapped to *sticks* variable of the *GTime* module in NesC [12]. And the TIMER3 fires and interrupts upon overflow of sticks. Then, the interrupt handler increases a 16 bit global variable which is mapped to *mticks* variable of the GTime module. So, one sticks is $(1/115200)sec$ and one mticks is $65535 * (1/115200) = 0.56888sec$. Therefore, we can use the real time value based on above calculation for the several kinds of timers through the whole DRAND implementation.

DRAND runs with the round unit. So, during a round, one time slot for a node is set with neighbors' agreements. But this algorithm doesn't require each node to synchronize on the round boundary. By the algorithm, most of the nodes converge into a similar round boundary. Figure 2.1 shows the state diagram for our DRAND implementation. This state diagram is the next stage of the HELLO state. So, first of all, we describe the HELLO state.

2.2.2 HELLO state

For the discovery of the neighbor node, we have the HELLO state in the beginning time of DRAND. During this state each node discovers its one-hop neighbors and two-hop neighbors. This small size of information is enough to run the DRAND scheme by a distributed and scalable fashion. Here, the most important element is how we can guarantee that one node, whom its one-hop neighbor already recognizes, recognizes that neighbor as well. For example, given node A and node B, it is possible that node A knows node B, but node B doesn't know about node A due to the asymmetric link. Then, after running the DRAND algorithm, we may not accomplish the conflict-free time slot scheduling. So, we try to use the three types of methods for the HELLO state as listed below. We assume that there are two nodes, A and B.

- Broadcast. A and B broadcast hello messages to each other periodically. And if A gets a hello message from B, it just records the B's address as a one-hop neighbor.
- Three-way-hand-shake. First, A broadcasts the hello message. Whenever B receives that message, B unicasts its hello message using the A's address. Then, if A gets B's hello message, A records B's address as a one-hop neighbor and sends a reply by unicasting to B. Finally, after B gets the message from A, B records A's address as its one-hop neighbor.
- Figure 2.2. Each node broadcasts hello messages periodically. And when each node gets a hello message from its neighbors, they use the algorithm in Figure 2.2. Here, *flag* means that the receiver should make the sender receive its hello message to let him know receiver.

For the first time, we tried to use a broadcast method which was very simple and very light to run. But it could not guarantee to get the symmetrical node information for every node under the asymmetric link environment. So, we used the three-hand-shake method to supplement the defects of the broadcast method. Then, we could get the symmetrical neighbor information even under the asymmetric link environment, but the message overhead was too large since each successful three-hand-shake operation required at least three messages. And additionally, the three-hand-shake took too long because if one of three messages failed, the whole three-hand-shake must start again. These weak points caused inefficiency of time and energy.

```
When A gets a hello message from B,
if A is in B's onewaylist
    if B is in A's onewaylist
        add B into A's twowaylist
        remove B from A's onewaylist
    else if B is in A's twowaylist
        flag
    else
        add B into A's twowaylist
else if A is in B's twowaylist
    if B is in A's twowaylist
        add B into A's twowaylist
        remove B from A's onewaylist
        flag
    else if B is in A's twowaylist
        nothing
    else if B is not in A's both lists
        nothing
else if A is not in B's both lists
    if B is A's onewaylist
        add B into A's onewaylist
    else if B is in onewaylist
        flag
    else if B is in A's twowaylist
        nothing
```

Figure 2.2: Pseudocode for the neighbor discovery

So, finally we devise the Figure 2.2 scheme. It works without any problem if there is enough time to exchange the message fully among one-hop neighbors. Each node sends a message periodically including two information arrays. One is *one-way* array which is for the asymmetric node information. For example, node A gets a message from node B for the first time, A records B's address in a one-way array. And the other is the *two-way* array which is for the symmetric node information. For instance, if node B receives a message from node A and its one-way array has node B's information, node B records node A's information in its two-way array. So, if node B already has node A's address in its one-way array, it removes that node A's address from that array. These

two arrays enable each node to have the symmetric one-hop neighbor information. The message overhead is the same as the overhead of the broadcast method, and Figure 2.2 can guarantee the same result as the three-hand-shake scheme.

In order to get the two-hop neighbors' information, once each node gets a hello message from its one-hop neighbors, it updates the two-hop neighbor array. For example, the one-hop neighbors of node B could be two-hop neighbors of node A except that one-hop neighbors of node B are node A's one-hop neighbors as well or node A itself. So, each node keeps updating its two-hop neighbor array until finishing the HELLO state whenever the arrived message has new node information in its two-way array.

We use 30 seconds for the HELLO state, and this time period is enough to get all one-hop and two-hop information in our experiment. And every node sends a hello message periodically with 500ms of time interval. By default, the data payload size of a message packet in TinyOS [19] is 29 bytes. With this size, we can handle only 27 neighbors (2 bytes are for sender's information). So, for the generous case, we increased the size of the data payload to 50 bytes. Even though transmission time increases because of larger packet size, there is no problem getting the proper result.

2.2.3 IDLE state

After the HELLO state finishes, the IDLE state starts directly. During this state, each node tries to run a lottery to be a winner who can send a request message to its one-hop neighbors for setting its own time slot. The winner is selected by the probability $P = 1/C$. C is the number of contenders who want to try to set their time slots during this round. The number of contenders is composed of one-hop neighbors and two-hop neighbors who have not yet decided on their time slot.

Given node A, after calculating the number of contenders, node A picks a random number and divides this random number by the number of contenders. Then if the remainder is zero, node A becomes a winner for this round, and its state is changed from the IDLE state into the REQUEST state (Section 2.2.4), or if not, node A waits for the next round and its state is still the IDLE state.

When node A is in the IDLE state, node A could get the request message from other one-hop neighbors. For example, node B becomes a winner, but node A fails to be

a winner. In this case, if node A gets a request message from node B, node A changes its state from the IDLE state into the GRANT state (Section 2.2.5). After receiving a release message from the neighbor who sent request message to node A, the GRANT state returns to the IDLE state.

2.2.4 REQUEST state

If node A is selected as a winner in a lottery, the state is changed from the IDLE state into the REQUEST state, and node A starts to broadcast request messages to its one-hop neighbors. By sending request messages, node A could get two types of messages, grant or reject messages. Whenever node A gets a grant message, it keeps the sender's information. If node A gets all grant messages from its all one-hop neighbors (This means that every neighbor allows node A to set its time slot for the current round), it can set its time slot with the smallest one among empty slots. And the state is changed from the REQUEST state into the RELEASE state. Otherwise, node A could get a reject message from node B which is already in the GRANT state (This means that node B already got a request message from its other one-hop neighbors before getting a request message from node A.). Once node A gets a reject message, it gives up the current round. Node A initializes senders' information and changes its state back to the IDLE state(Section 2.2.3).

Node A keeps broadcasting request messages until getting all grant messages or a reject message. At this point, if each one-hop neighbor responds to every request message by sending a grant message, during the REQUEST state there could be redundant grant messages because if node A already gets a grant message from node B, node B doesn't need to send grant messages anymore. So, in order to prevent useless grant messages flooding, whenever a node broadcasts a request message, that message includes senders' information about arrived grant messages. For instance, if node B gets a request message from node A, node B looks at the sender's information. If node B's information is in the request message, it doesn't send grant messages to node A. If not, it sends grant messages.

Using the first arrived grant message, node A calculates RTT (Round way Trip Time) and OTT (One way Trip Time). Using the measured value, node A can update its *ROUND_TIME* and *REQUEST_TIME*. ROUND_TIME is a estimated time dura-

tion value for one round considering all request, grant, and release messages. And REQUEST_TIME is same as the $OTT_{request}$ which is the time to transmit a request message from sender to receiver. We calculate ROUND_TIME with $OTT_{request}$ + (OTT_{grant} * *number of one-hop neighbors)* + $OTT_{release}$. So, the ROUND_TIME of DRAND can be adjusted depending on the current network transmission delay.

A grant message includes the time slot schedule information table which the sender has received from its one-hop neighbors before sending this grant message. In detail, when node A sends a grant message, it inserts the time slot schedule information table of its one-hop neighbors into the message. And, if node B gets this grant message, node B compares its time slot schedule information table and node A's. Then, if there is missing time slot information in node B's time slot schedule information table, it fills that empty slot using node A's information. As a result, before node B sets its time slot, as node B gets full time slot information about two-hop neighbors, it can set its time slot avoiding a duplicated one within the two-hop distance.

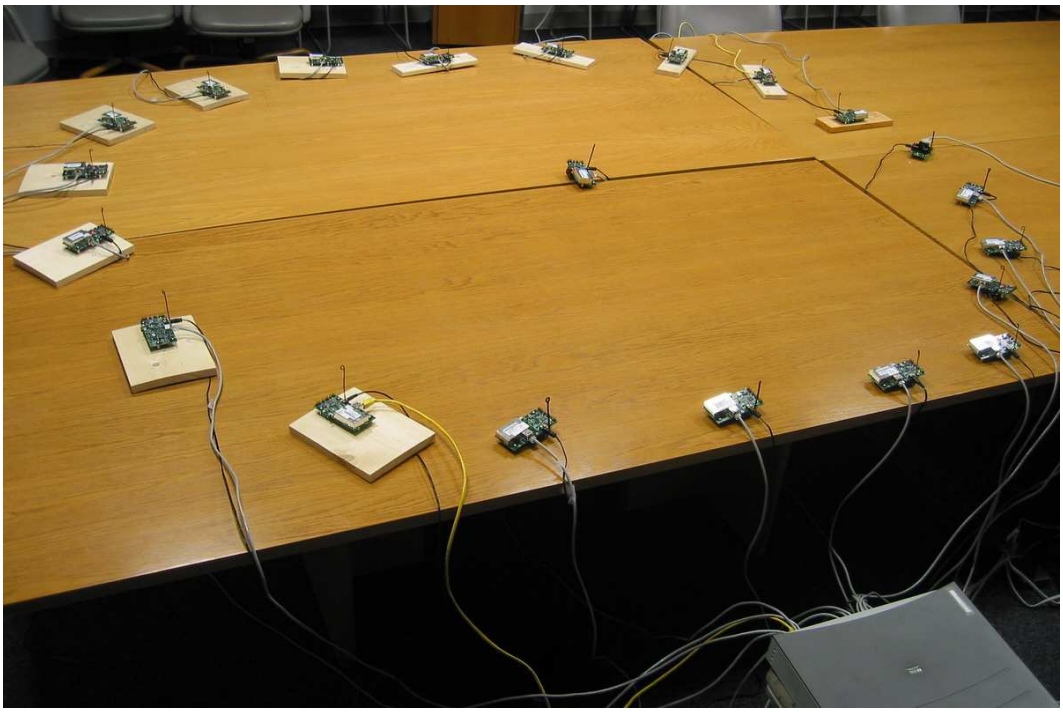

Figure 2.3: The photograph of the experiment using MICA2 sensor nodes and MIB600CA Ethernet programming boards

2.2.5 GRANT state

When node A gets a request message from node B, its one-hop neighbor, the state of node A goes to the GRANT state as long as it is not currently in the GRANT state or the REQUEST state. And node A unicasts a grant message to node B with small backoff time which is to prevent grant message flooding and collision from node A's one-hop neighbors. Moreover, because the meaning of the grant message is to allow the sender to set its time slot, node A cannot send any request message until getting the release message from node B. To do this, node A stops the round timer as well as the lottery for selecting a winner.

Sometimes the grant message from node A could be lost by relying on the network link status. Then, node B sends a request message again that includes the senders' information of arrived grant messages as we explained in Section 2.2.4. When node B gets a duplicated request message from node A, node B checks the information array in the request message to know whether node B's grant message was arrived at node A or not. If that grant message has arrived at node A, node A doesn't need to retransmit its grant message. If not, node A sends its grant message again. This method decreases the total number of grant messages which are created by the retransmission of request messages. Therefore, we can reduce the failure of the grant message retransmission.

During the GRANT state, node A can get a request message from a different node, node C who is located at minimum two-hop distance from node B. In this case, because node A already sent its grant message to node B, it cannot send a grant message to any other node again. But instead of a grant message, it sends a reject message to node C which means node A is already in the GRANT state, and it cannot allow node C to set its own time slot because only one time slot has to be set in a round. After sending a reject message, node A keep waiting for the release message from node B.

Once node B gets all grant messages from its one-hop neighbors, it broadcasts a release message to those neighbors. This message has node A's time slot information. So, when node B gets a release message from node A, it records the time slot information of node A. And node A changes its state into the IDLE state or the RELEASE state. If node A already set its time slot and if node A was in the RELEASE state when it got a request from node B, node A changes its state into the RELEASE state (Section 2.2.6). Otherwise, node B returns to the IDLE state (Section 2.2.3) to try to set its time slot.

Node A keeps retransmitting grant messages periodically to node B until node A gets a release message from node B. This is because the release message from node B could be lost. If node A cannot get a release message at all from node B, then, the DRAND algorithm fails due to a kind of dead-lock case. For instance, if node A could not get any release message from node B, node A keeps being in the GRANT state. And if other one-hop neighbors of node A send request messages, node A always replies with a reject message. Because of this reason, we have to make the release message more reliable against message loss.

2.2.6 RELEASE state

After setting own time slot during the GRANT state, a node goes to the RELEASE state. If node B gets all grant messages from its one-hop neighbors, it can set its time slot and change its state from the REQUEST state to the RELEASE state. Moreover, in order to let neighbors know its time slot information, node B sends a release message. Even after sending a release message, if node B gets a grant message from specific one-hop neighbors, node B retransmits its grant message again for the reliability of the release message as we described in Section 2.2.5.

If one-hop neighbors of node B including node A get a release message from node B, they send the two-hop release message to their one-hop neighbors in order to propagate node B's time slot information to node B's two-hop neighbors. By the two-hop release message, each node can set a non-duplicated time slot within the two-hop distance. But, we don't need to make this two-hop release message reliable because even though the two-hop release message is lost, the time slot schedule information in the grant message can take the place of the slot information in the two-hop release message as we discussed in Section 2.2.4.

During the RELEASE state, node B can get a request from its one-hop neighbors who have not set their time slots yet. Then, node B changes the state from the RELEASE state to the GRANT state. After that time, if node B gets a release message from that neighbor, it goes back to the RELEASE state.

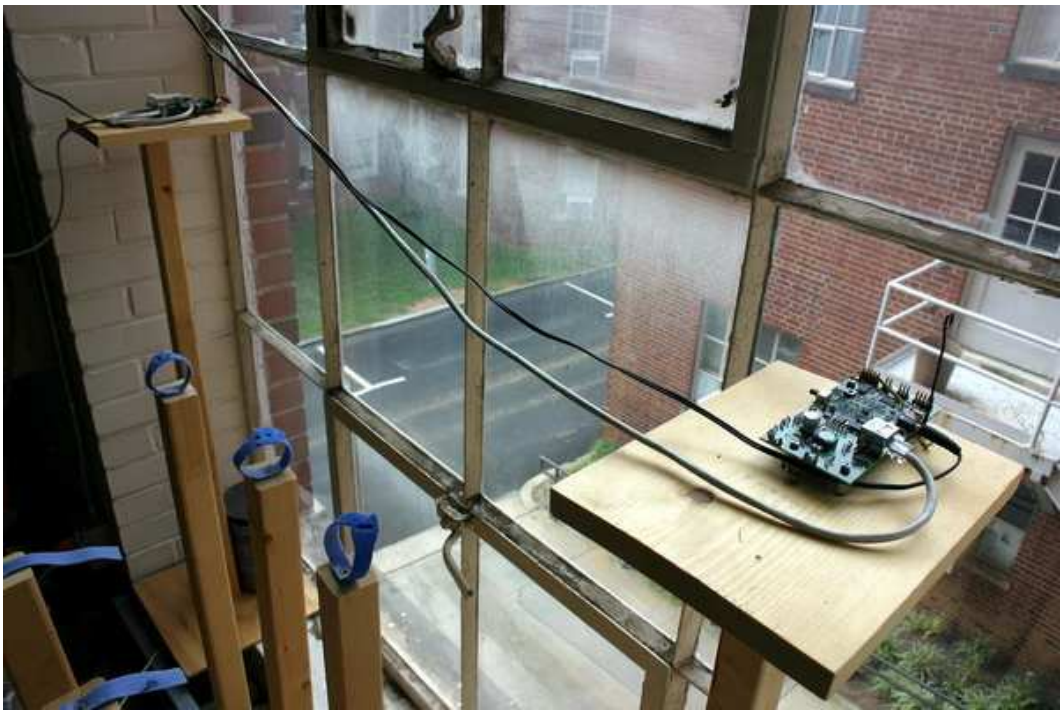

Figure 2.4: MICA2 sensor nodes and MIB600CA Ethernet programming boards which are deployed on the testbed

2.2.7 REPORT state

After receiving all grant messages from its one-hop neighbors, each node sets the round timer with periodical time duration (REPORT_PERIOD - We use 30 seconds.) considering the RELEASE state to send time slot information including its all one-hop neighbors. If each node gets a new request, the round timer would be rescheduled. Once this report timer is fired, the state is changed from the RELEASE state into the REPORT state. Or, if a node gets a report request message from its neighbors before firing its report timer, each node also changes its state in the same way and starts sending the report message. So, after timer interruption or receiving a report message, each node keeps sending report messages to its one-hop neighbors. And when each node changes its state from the RELEASE state into the REPORT state, it sets the round timer for the next FRAME state.

Whenever each node receives a report message from its one-hop neighbors, it updates the empty slot information for its one-hop or two-hop neighbors in the neighbor table. After 30 times of retransmission of the report message, if there are some neigh-

bors whose report messages have not been arrived yet, each node assumes that those neighbors are not available in this time because of the asymmetric link status, the dead node, and etc. So, each node drops those neighbors from its neighbor table.

2.2.8 FRAME state

After the round timer is fired, if the state is changed from the REPORT state into the FRAME state, each node calculates its *local frame size* and starts sending the frame message to its one-hop neighbors. All operations for the FRAME state are the same as that for the REPORT state.

The local frame size is very important for the efficient channel usage in the TDMA scheme. For instance, when a node A has one neighbor, its time frame size is 3 while node B in the network may have more neighbors and its time frame size is 10. If all nodes synchronize to the global time frame of 10, then node A has to wait for 9 slots before it can use its time slot again. As a result, 7 of those 9 slots are not used by any neighbors of node A. To avoid these problems, [26] presents a new scheme where each node maintains its own local time frame that fits its local neighborhood size, but avoids any conflict with its contending neighbors. We implement this scheme in the FRAME state of DRAND, and after DRAND finishes, the TDMA scheme can use the local frame information as well as the own time slot information.

So, after finishing REPORT state, each node calculates its local frame size. For example, node A sets its local time frame to be 2^a where a positive integer a is chosen to satisfy condition $2^{a-1} \leq$ *time frame of node* $A < 2^a - 1$. That is, node A uses its time slot in every 2^a time frame. Each node exchanges this local frame size to each other during the FRAME state. We can see the example of the local frame size in Figure 3.1. The largest number in the middle of each node is node ID. The small number on the bottom-left side is the assigned time slot, and another small number on the bottom-right side is the local frame size.

2.3 Debugging

During the entire implementation period, we faced some problems for debugging. Because we are running our code on the real sensor node, MICA2 [3] mote, the type of the debugging is very different with that of we have done mostly with the usual programming debugging method on personal computers. We cannot stop the application when errors happen, or because of the limitation of the memory size, we cannot make enough debug messages as we want. In addition, because debug messages make the mote system slower, the results could be different with that from the code without debug messages. We have tried to use several kinds of methods for debugging as listed below.

- The use of LED with three colors; green, red, and yellow.
- The use of JTAG [1] for the in-circuit debugging
- The use of TinyOS simulator, TOSSIM [22].
- The use of UART message
- The use of the SODbg [7] module for printing the message from the motes on the screen of the user's machine.

LED is too simple to know the current specific situation on the motes, but it is good for directly checking the present state. JTAG [1] is the one of the useful debugging devices which uses the GDB interface between user computer and MICA2 mote. Using this, we can get the exact information for the debugging, but when we have to use some number of motes together, it is very hard to control all JTAGs together. Besides, even though the TOSSIM [22] is good for the verification of whole algorithm, because it is impossible to reproduce the same situation as the real environment, we cannot encounter various error cases with this simulator. So, even though there is no problem on using TOSSIM, we can get more problems when we run the same code in MICA2 motes.

UART message uses the same module as message sending, but only the destination address is different. If we use TOS_UART_ADDRESS instead of any other mote's address, the message goes to the user computer which is connected by a serial

port. Then, we can see the raw data value of the message on the computer screen. UART message is better than previous methods when we want to know the specific situation inside each mote. But, because the UART message could fail to send by other message operations, sometimes it is hard to use this method to show the experiment results to the user.

We usually use the SODbg. The SODbg enables us to know what is happening inside motes by printing the debug message through the telnet connection to each mote. The process of SODbg is almost same as UART message. This also uses the serial port, but doesn't use the same module as the message sending. Besides, we usually use the SODbg for gathering the experimental data. However, this SODbg also has a disadvantage because the SODbg takes sometime to send the debug message and makes the whole system operation slower. So, we cannot use many messages for the proper experiment. For this reason, after debugging the code, we use only a few lines of SODbg messages for printing the experiment result.

2.4 Summary

In this chapter, we describe the implementation of DRAND. We implement this DRAND algorithm based on [27]. And in order to adjust this DRAND algorithm to the real environment, we build the DRAND code with small modification of some parts like the neighbor discovery.

In next chapter, we explain experiment environment and show the experiment result under various test cases.

Chapter 3

Performance Evaluation of DRAND

In this chapter, we explain the experiment environments including the testbed and what the performance result of DRAND is through the real various experiments using TinyOS [19] and NesC [12].

3.1 Experiment Environment

We use two kinds of experiment environments. One is for one-hop test and another one is for the multi-hop test. We can use the experiment environment of multi-hop test for the two-hop test as well. For each environment, we use MICA2 [3] motes and MIB600CA Ethernet programming boards [2]. MICA2 mote has an Atmel ATmega128L micro-controller with 4 KB of RAM, 128 KB of flash, and a CC1000 radio operating at 433 MHz. Raw throughput while using Manchester encoding is 38.4 Kbps; however, maximum single-hop data throughput achieved is about 10Kbps. All motes are configured to transmit at full power and with same radio frequency. By [8] the effective radio range with the full radio power and without any obstacle is about 50m. Moreover, we can assign the unique number to each node as its node ID and all nodes

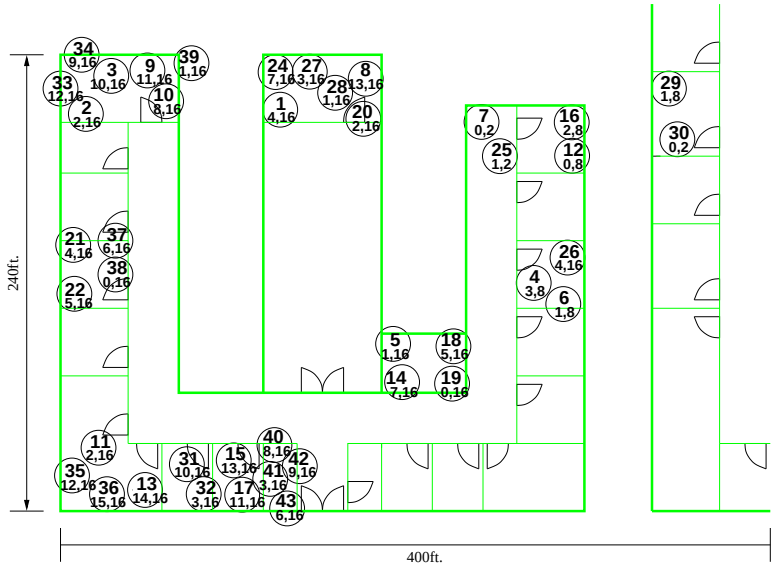

Figure 3.1: DRAND slot assignment

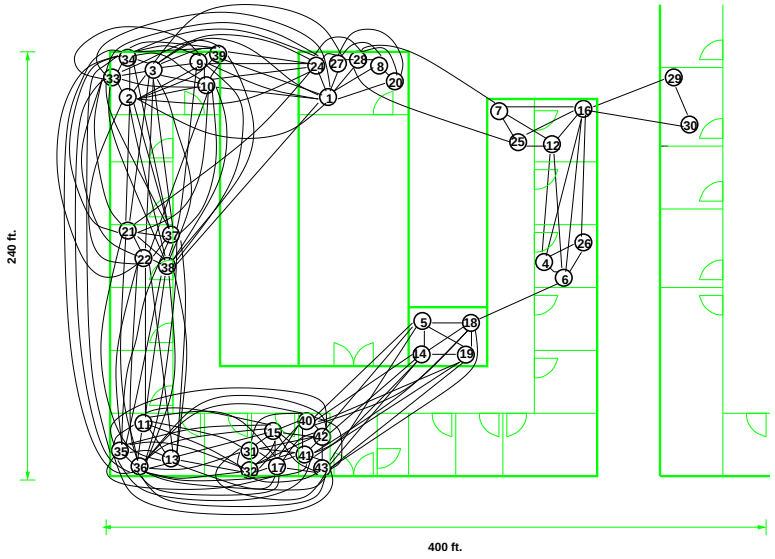

Figure 3.2: Network Testbed

on every experiment share same group ID.

Our one-hop test environment consists of a varying number of MICA2 wireless sensors placed within a one-hop neighborhood. We vary the number of nodes in the network from one node to twenty nodes one by one. Before running each experiment, we ensure that all nodes are within a one-hop transmission range, and they are also placed at least 2 feet above the ground in our experiments according to [8]. Every MICA2 mote is plugged into the MIB600CA Ethernet programming board. And all programming boards are connected to the Linux machine by Ethernet cables and hubs. Using the NAT (Network Address Translation) system, we can control all sensor nodes by the Linux machine. By uploading the program image concurrently from the Linux machine, each node starts at almost same time together. Figure 2.3 shows the one-hop experiment environment we used.

For a multi-hop test environment including two-hop test, we deploy 42 MICA2 motes in Withers Hall of North Carolina State University. Each mote is located in the several faculties and class rooms. Figure 3.1 shows the location of each MICA2 mote, and Figure 3.2 shows the connectivity between each node (represented by a line connecting them) after running a simple neighbor discovery algorithm for 30 seconds. The network is highly connected on the left side of the building with some nodes having as many as 13 one-hop neighbors. The radio connectivity between nodes varies in quality, with some links having loss rate as high as 30-40%. Figure 2.4 shows some of motes on our testbed in the building.

3.2 Performance of DRAND

In this section, we present the performance result from the real experiments using the DRAND implementation with TinyOS [19] and NesC [12]. We run the experiment with various situations under the single-hop and multi-hop case.

3.2.1 Single-Hop Experiment

For each experiment, we repeat the test ten times and report the average and its standard deviation errors. The error bars denote 95% confidence intervals of

our experiment results. We measure data related with running time and each message count taken for all nodes in a network to decide on their slots. According to the analysis in [27], we can see that they are linearly proportional to the number of neighbors.

Figure 3.3 shows the average of the maximum running time that a node has taken to decide on its slot and the average of the maximum number of rounds taken by a node in each setup. Our measurement shows that the running time is approximately a quadratic increase as the number of nodes increases. This is because when a node is added, each node needs to run one more round for the entire operation and get one more GRANT for each round to set the time slot.

However, the number of rounds required for each run grows linearly with the number of neighbors. Figure 3.4 shows the average per-round time which, indeed, grows linearly as the number of neighbors increases. This increase is almost entirely attributable to that in the grant message delays, shown in a dotted line.

The discrepancy between the asymptotic analysis in [27] and our measured data comes from the different ways in accounting for a message delay. The asymptotic analysis does not account for any message delay increase due to the increased number of senders. But in practice, wireless communication delays increase proportionally as the number of transmitting neighbors increases because all share the same channel. If the additional message delays incurred by contention is negligible, our result follows the analysis.

The growth in the number of per-node message transmissions shown in Figure 3.5 is clearly linear. The number of messages transmitted by each node per second, shown in Figure 3.6, also decreases as the neighbor size increases. This indicates that although the running time grows quadratically, the amount of energy spent by each node to run DRAND grows linearly because radio communication is the dominant source of power consumption in sensor nodes.

3.2.2 Multi-Hop Testbed Experiment

On this topology, each node first runs a neighbor discovery protocol to get its neighborhood information. Then the DRAND algorithm is executed and a TDMA time slot is assigned to each node. MAC-layer protocols like Z-MAC [26] and RA-SMAC [25] may require additional information about their two-hop neighborhood. In Z-MAC, for

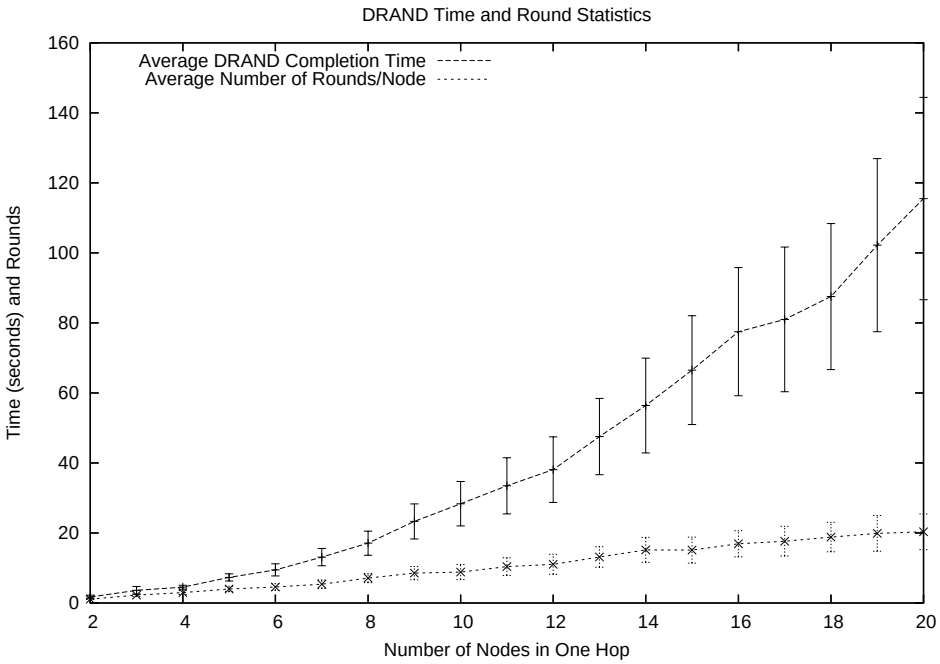

Figure 3.3: The running time of DRAND and the number of rounds as the size of neighbors increases.

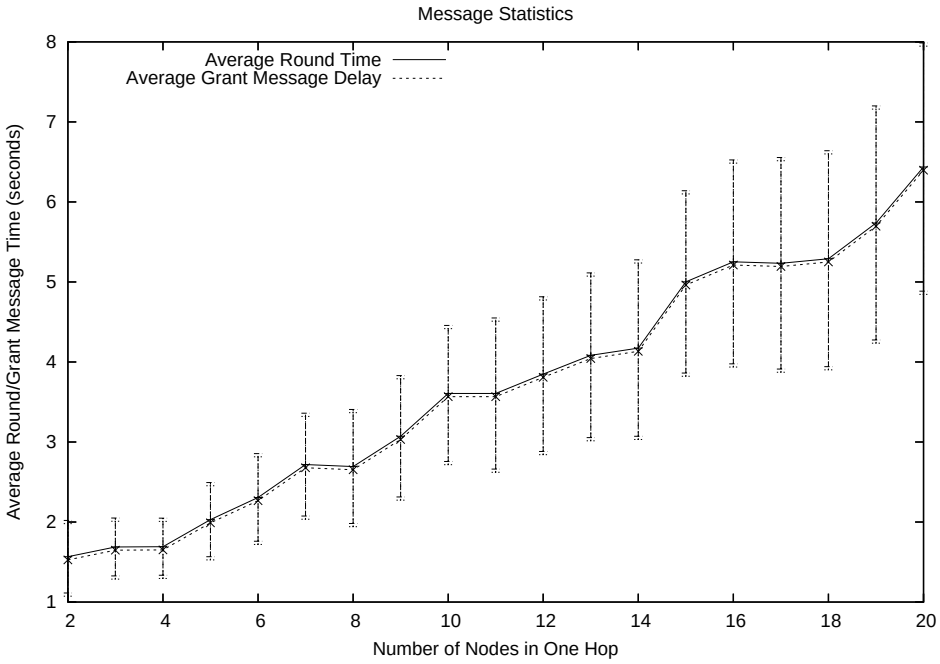

Figure 3.4: The average round time duration and the grant message delay.

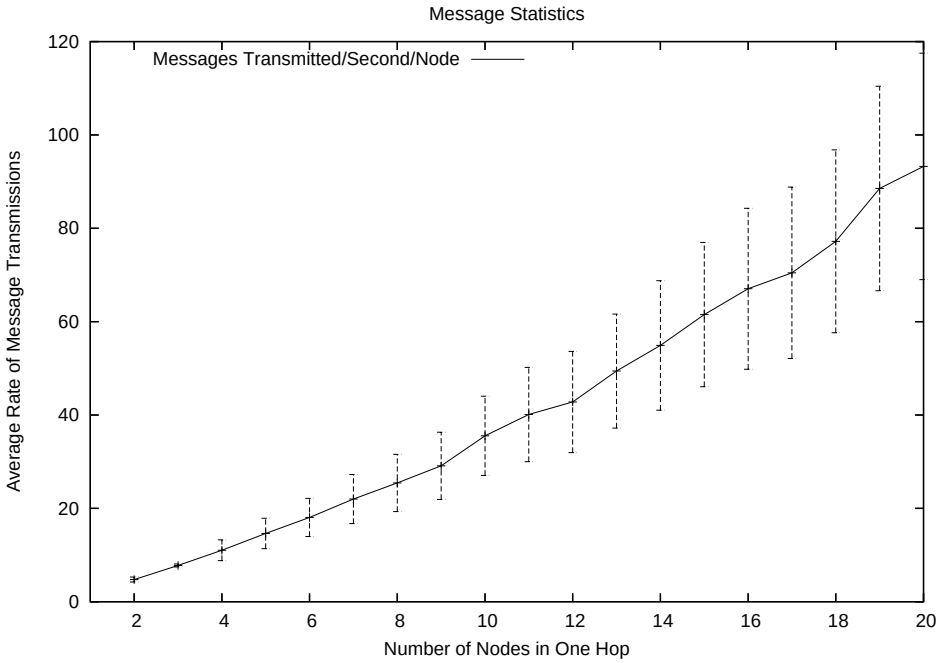

Figure 3.5: The average number of message transmissions per node during the execution of DRAND.

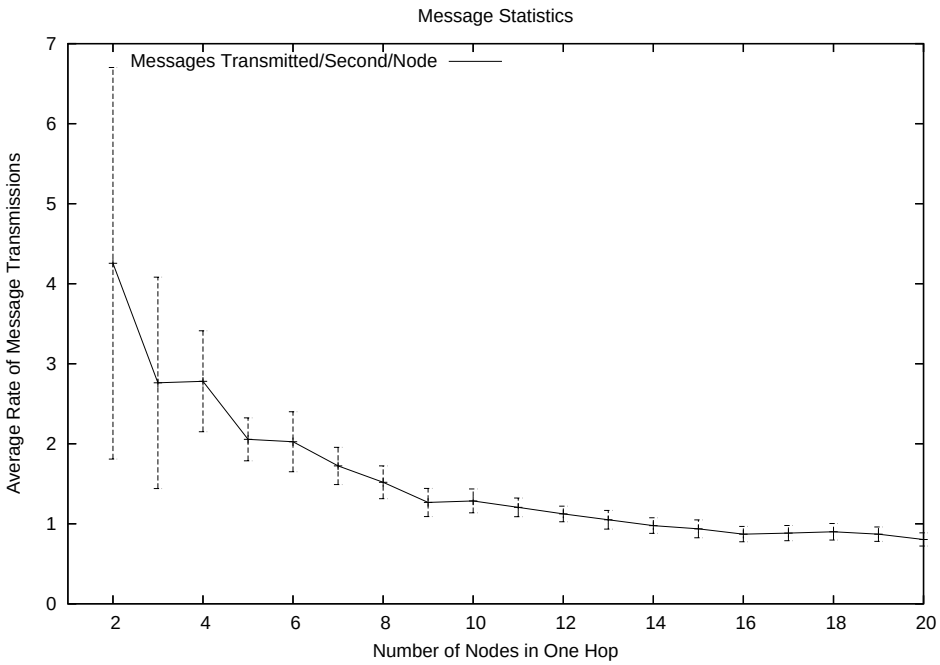

Figure 3.6: The message transmission rate by each node (per second) during the execution of DRAND.

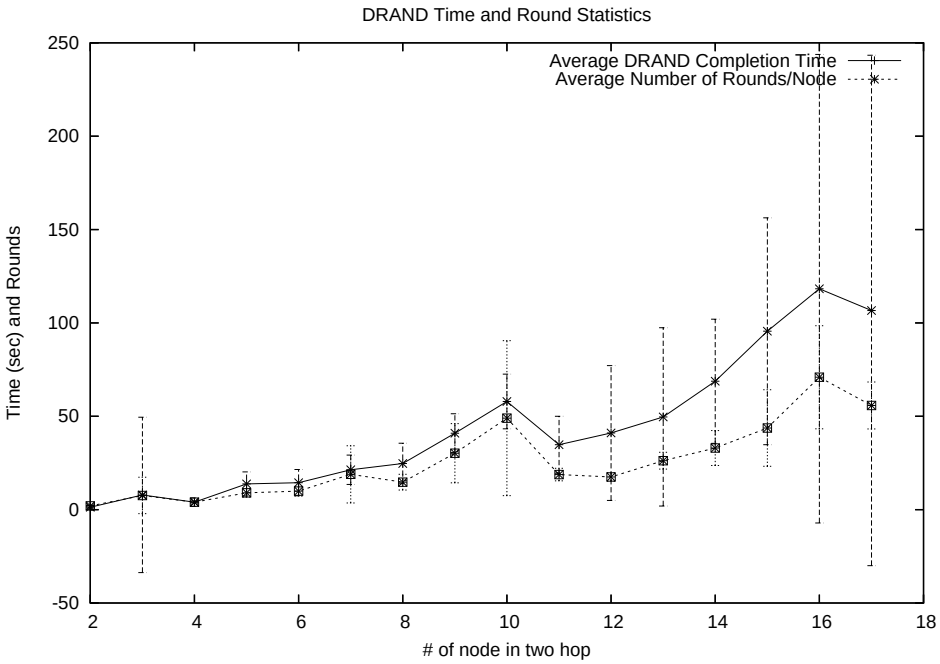

Figure 3.7: The running time of DRAND and the number of rounds for the two-hop tests.

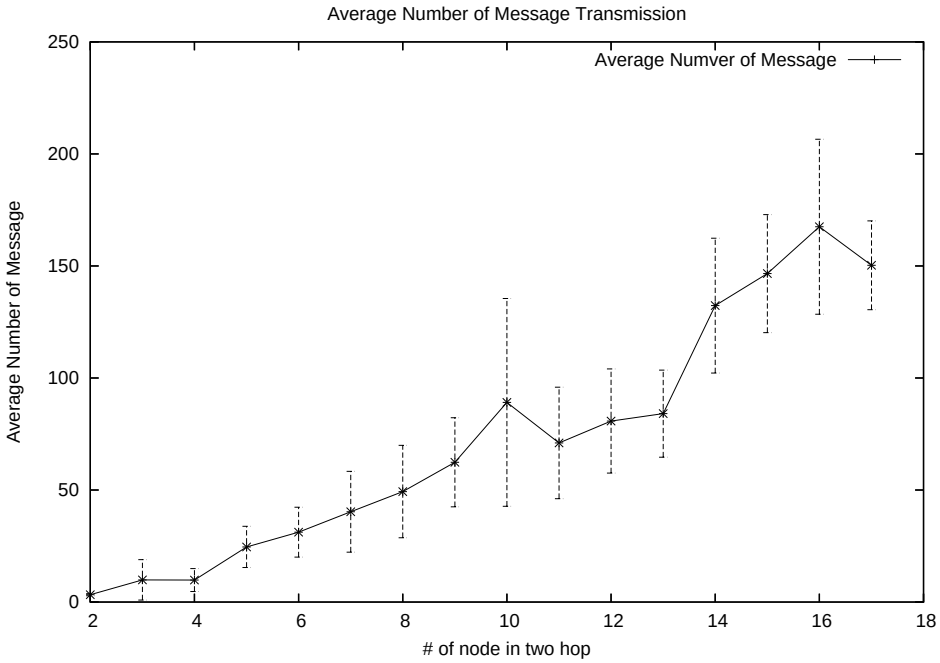

Figure 3.8: The average number of message transmissions in the two-hop tests.

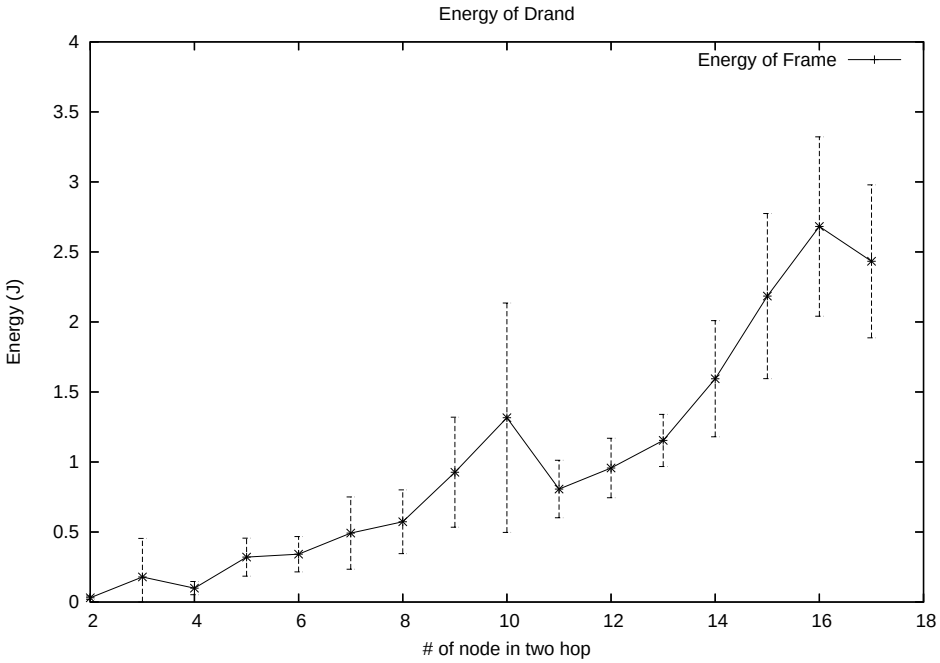

Figure 3.9: The maximum energy experiment of DRAND for any node in the two-hop test.

instance, a node requires the local frame size of all nodes in its two-hop neighborhood, which it obtains right after the DRAND phase is over as we explained in Section 2.2.8. In the remainder of this section we run several tests on subsets of this topology to analyze the performance of DRAND in a real world scenario.

We have an experiment for the impact of the duty cycling on the performance of DRAND. We execute DRAND on top of BMAC which supports several duty cycle modes. We can see the specific information of each duty cycle of BMAC in Table 3.1 from [23]. Figure 3.10 shows the running time and energy cost of the DRAND algorithm when it runs on the full testbed with different duty cycle mode, 0 to 4. The running time reported is the maximum value for any node in the testbed. And the energy is the average of whole nodes on the testbed. Each node records the total number of primitive radio operations (receiving a byte including the garbage data, transmitting a byte, and taking a CCA sample including the whole sequence of operations: initializing radio, turning on radio, switching to RX/TX, sampling radio, and evaluating the radio sample). The total energy is the sum of these operations weighted by the cost of each operation as reported in [23] and [5]. The experiment is reported 5 times for each duty

Table 3.1: Duty Cycle Modes of BMAC

Mode	**Duty Cycle (%)**	**Effective Data Rate (kbps)**
0	100	12.364
1	35.5	5.671
2	11.5	2.488
3	7.35	1.737
4	5.61	1.336
5	2.22	0.559
6	1.00	0.258

Table 3.2: DRAND Overhead Cost

Operation	**Average Time**	**Average Energy**
Neighbor Discovery	30s	0.732J
DRAND Slot Assignment	194.38s	4.88J
Slot/Local Frame Dissemination	60s	1.33J

cycle.

For calculation of energy, we use the Joule unit. So, Joule = Watt × Time (second) and Watt = Volt × Ampere. So, Joule = Volt × Ampere × Time (second). As we mentioned above, for the receiving and transmitting operations, we count the number of bytes inside the MAC layer. For channel sampling, we just count the number of sampling operations. The time period for receiving and transmitting a byte is 400μs and we sum all time periods of every operation for one channel sampling. According to [23] and [5], we can get the respective Ampere values.

Using a lower duty cycle allows each node executing DRAND to sleep during the idle periods of its round, hence saving energy. If we lower the duty cycle too much, the packet lengths grow larger (due to increased preambles), consequently increasing the round time, running time, and energy consumption. So, when we use low duty cycles, because the running time is too long we don't consider 5, 6 modes for the experiment.

For all consequent experiments, we maintain a duty cycle of 80%. The average cost of 30 runs of the full testbed with 80% duty cycle, for each of the three phases - neighbor discovery, DRAND and the frame size dissemination is in Table 3.2. The slots assigned to each node (as well as the local frame sizes) for one particular run are shown

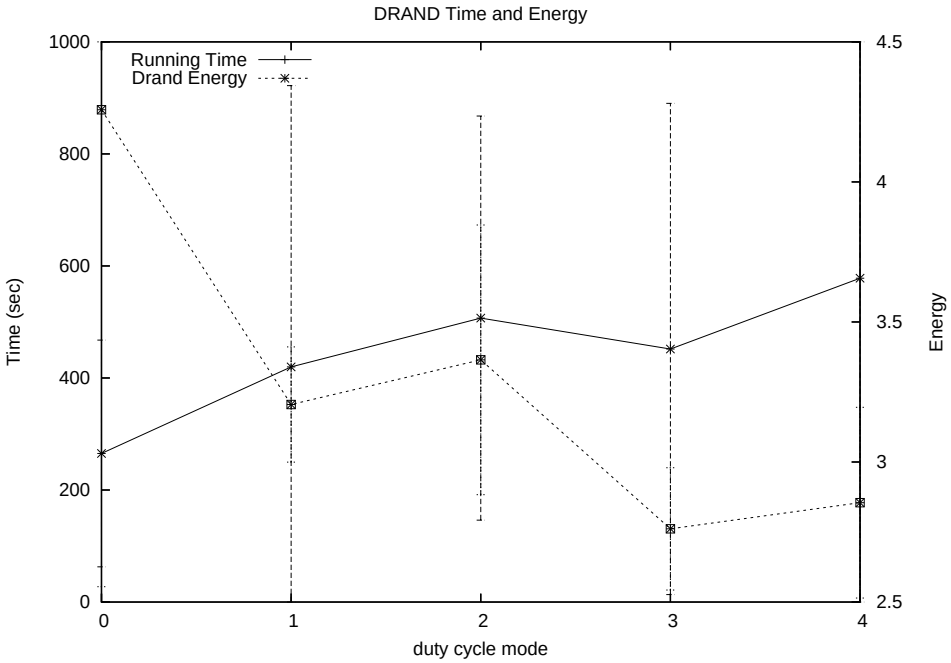

Figure 3.10: Energy and time of DRAND by different duty cycle modes

in Figure 3.1. Note that all nodes within two-hop distance of each other are assigned unique time slots. Most of the nodes on the testbed have 16 local frame size.

We have experiments for the DRAND under the two-hop case on the testbed. We run DRAND on a subset of the full testbed, consisting of 12 nodes on the left side of the building. All nodes are within two-hop distance of each other. We report the DRAND maximum energy expenditure for any node in this topology as the number of nodes is increased. Each data point in the following figures shows the 90% confidence intervals of the average of 30 runs. Figure 3.7 shows the DRAND runtime and the number of rounds taken to acquire a slot. Note that the general trend is similar to the one hop tests in Figure 3.4, since both parameters are a function of the maximum degree (number of neighbors) which is the same for both tests, regardless of one-hop or two-hop. Figure 3.8 shows the growth in the maximum message transmissions per node which follows a linear curve.

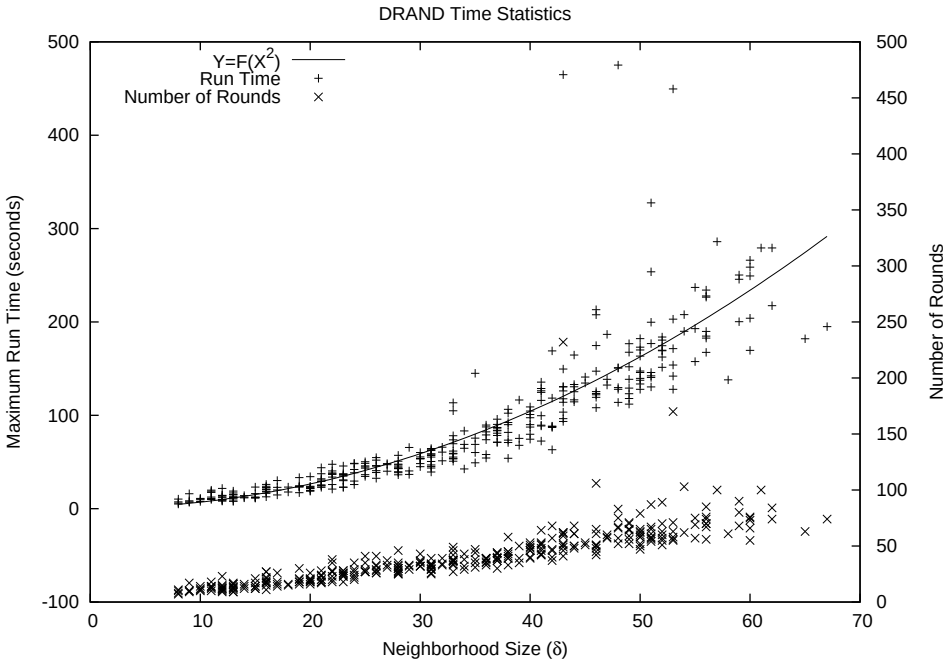

Figure 3.11: The running time of DRAND as the neighborhood size increases.

3.2.3 Multi-Hop Simulation Experiment

For the large scale test of DRAND, we refer to the NS2 [6] simulation result in [27]. For the simulation, the network topology consists of nodes placed randomly on a 300m $\times$ 300m surface. The radio range of each node is 40m and a link capacity is 2Mbps. The density of the network of the network is changed by varying the number of nodes from 50 to 250. As before, nodes run a neighbor discovery protocol on startup to get their neighborhood information. And the experiment is repeated 15 times and the maximum of the values for all nodes are reported.

As same as the single-hop experiment in Section 3.2.1, we can see the maximum DRAND running time with with different number of nodes in Figure 3.11. This result shows the quadratic curve which we already saw in Figure 3.3. As explained before in Section 3.2.1, this is because of the linear increase in the DRAND round time and the number of GRANT messages as the neighborhood size increases as is seen in the Figure 3.11. And as we can see in the Figure 3.12, the average number of message transmissions per node shows the linear increase same as the Figure 3.5 in Section 3.2.1

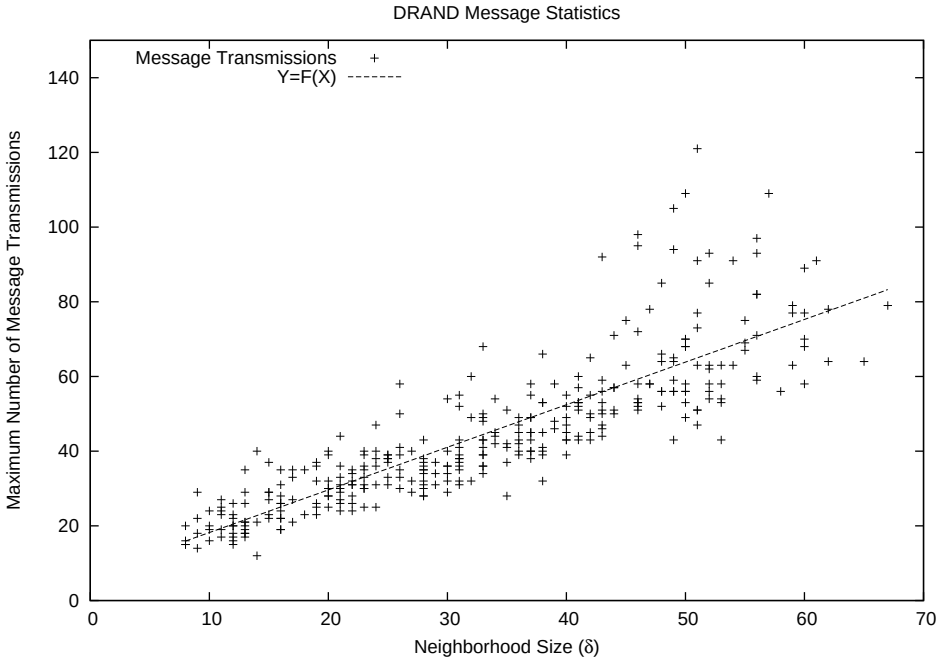

Figure 3.12: The average number of message transmissions per node during the execution of DRAND.

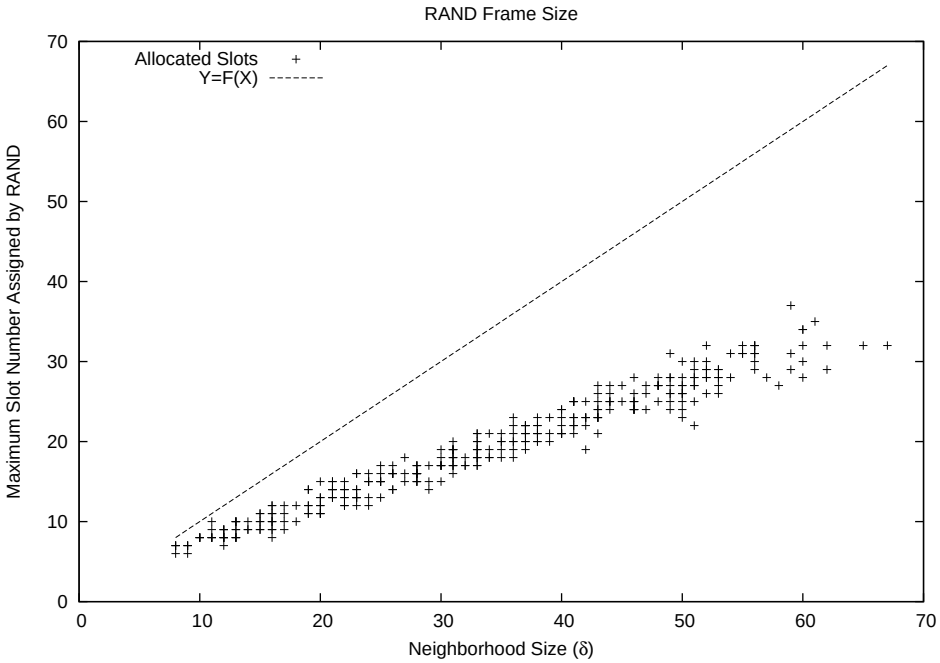

Figure 3.13: The maximum number of time slots being assigned by DRAND for input graphs with various densities.

and the Figure 3.8 in Section 3.2.2.

In DRAND, the number of time slots being assigned is bounded by the number of nodes + 1 by the analysis in [27]. But when we see the simulation result, the number of time slots that DRAND assigns can be far less than that. Figure 3.13 shows the number of time slots used for input graphs with various densities. The dotted line indicates number of nodes. Each data point represents the maximum number of time slots being assigned by DRAND for a different network. In all runs except one, the maximum slot number used by DRAND is far less than number of node + 1. This is very different result with other algorithms such as [10], [11], [21], [28], and [32] which require number of nodes + 1 or more.

3.3 DRAND Overhead and Local Recovery

We can see the life time of MICA2 mote with 2 AA type batteries in [4]. They continuously run an application in TinyOS with some operations: blinking three LEDs and transmitting a TOS packet once every four seconds with 433.02 MHz and maximum radio power. Power supply load was 10 to 15 mA in this configuration. [4] determines that end of service life is when TOS packets could not be received by a reference MICA2 mote that was 30 meters away. Finally, they measured 172 hours as the observed battery service life time. By Table 3.2 shows that DRAND algorithm takes 30 seconds on average and 4.88 J. By the energy calculation, DRAND algorithm needs 8.3 mA during 30 seconds of the operation period. In this case, DRAND uses only 0.03% of the full battery capacity with 2500mAh and 3V. Then, comparing the result of [4], we can check that the energy consumption of DRAND is not too big.

In DRAND, when a new node A joins the network (or a node, already assigned a slot, fails for some reason, and restarts), it can secure a time slot by (re-)running DRAND within its one-hop neighborhood - it is not necessary for nodes more than one hop away from node A to re-run DRAND. This is because the node which is two-hop distance away from node A doesn't need to have the information of node A. We simulate node joining by restarting specific nodes on our testbed, allowing all nodes within one-hop of the restarted node to run DRAND and then measure the average time and energy for all such nodes to secure new conflict-free time slots. We choose 5 such nodes - 2,

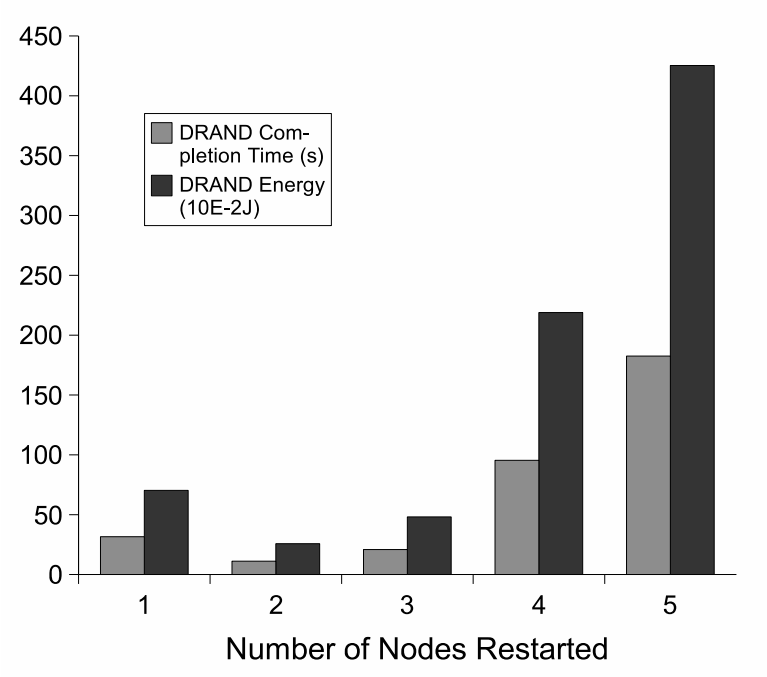

Figure 3.14: DRAND Recovery Time and Energy

14, 17, 26, and 29. Figure 3.14 shows the average time and energy expended for the recovery of 1, 2, 3, 4, and 5 node failures corresponding to simultaneous restart of nodes (2), (2, 14), (2, 14, 17), (2, 14, 17, 26), and (2, 14, 17, 26, 29) respectively. When all 5 nodes fails together, their one hop neighborhood could be all nodes on the testbed, and hence the cost is approximately the same as re-running DRAND again for all nodes.

3.4 Summary

In this chapter, we describe the performance result of DRAND from the various experiments under single-hop and multi-hop case. All results follow the theoretical analysis and simulation results in [27]. Moreover, we can see each running time, number of message, and energy consumption of the DRAND implementation in the real life unit.

Chapter 4

Practical Use of DRAND

In the chapter, we show how DRAND works for the TDMA scheme. So, we take the ZMAC [26] as an example for the purpose. ZMAC could be a partial example, but we can confirm that DRAND is working properly.

4.1 ZMAC

ZMAC is a hybrid MAC schemes for wireless sensor networks that combines the strengths of TDMA and CSMA while offsetting their weaknesses. The main feature of ZMAC is its adaptability to the level of contention in the network so that under low contention, it behaves like CSMA, and under high contention, like TDMA.

ZMAC uses the DRAND algorithm right after deployment in the sensing field. So, after running this time slot assignment algorithm, ZMAC can have the conflict-free time slot schedule which ensures a broadcast schedule where no two nodes within a two-hop communication neighborhood are assigned to the same slot.

Figure 4.1 shows the throughput result under one-hop case. All senders are transmitting at their full transmission power and the receiver has its radio on always. The frame size is 20 which means that the size of all time slot schedule from DRAND is less than or equal to 20.

Figure 4.2 shows the throughput result under two-hop case. We measure the

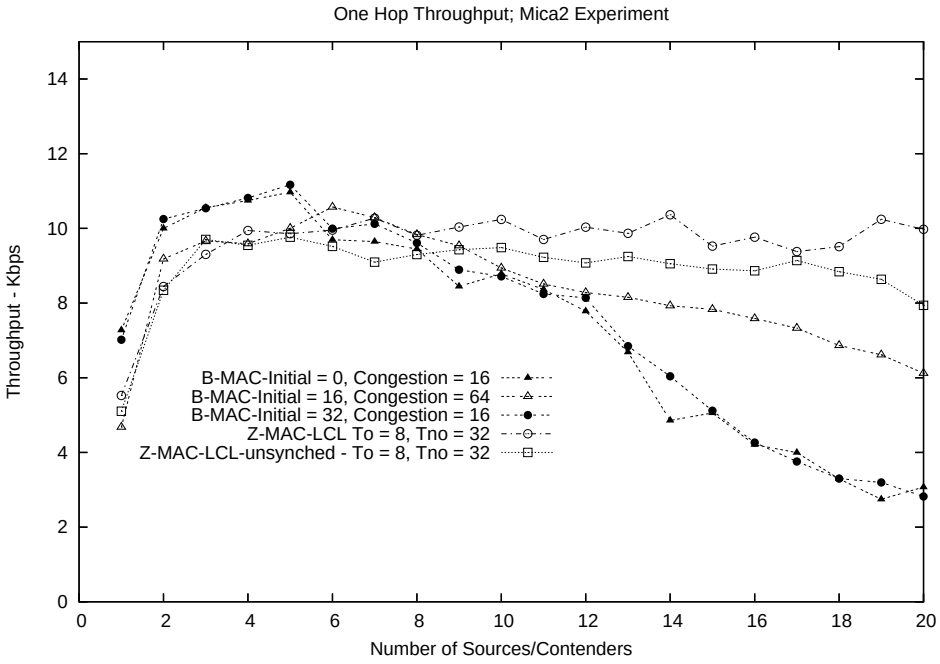

Figure 4.1: The data throughput comparison in the one-hop MICA2 benchmark

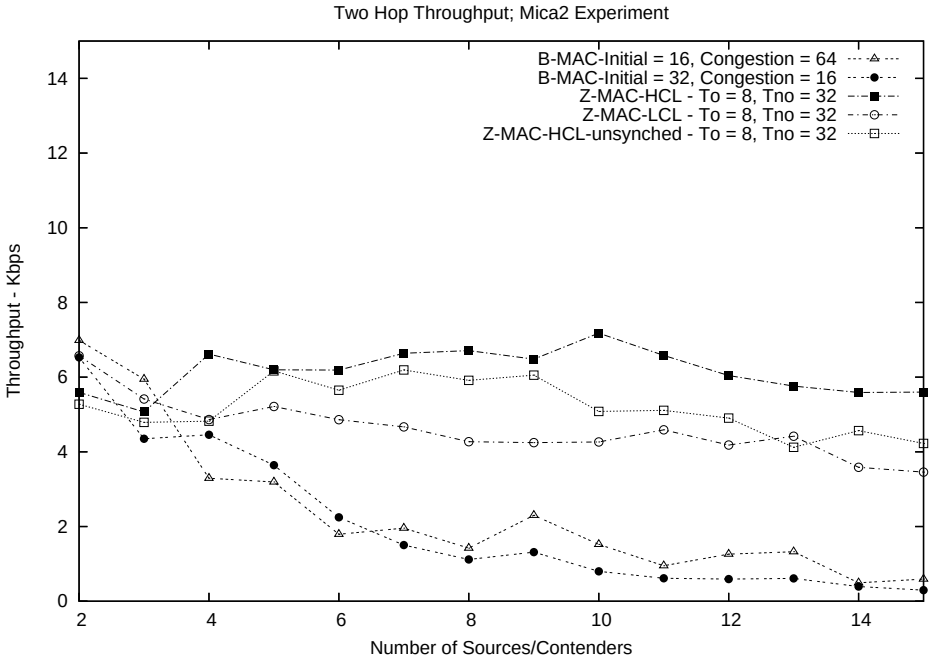

Figure 4.2: The data throughput comparison in the two-hop MICA2 benchmark

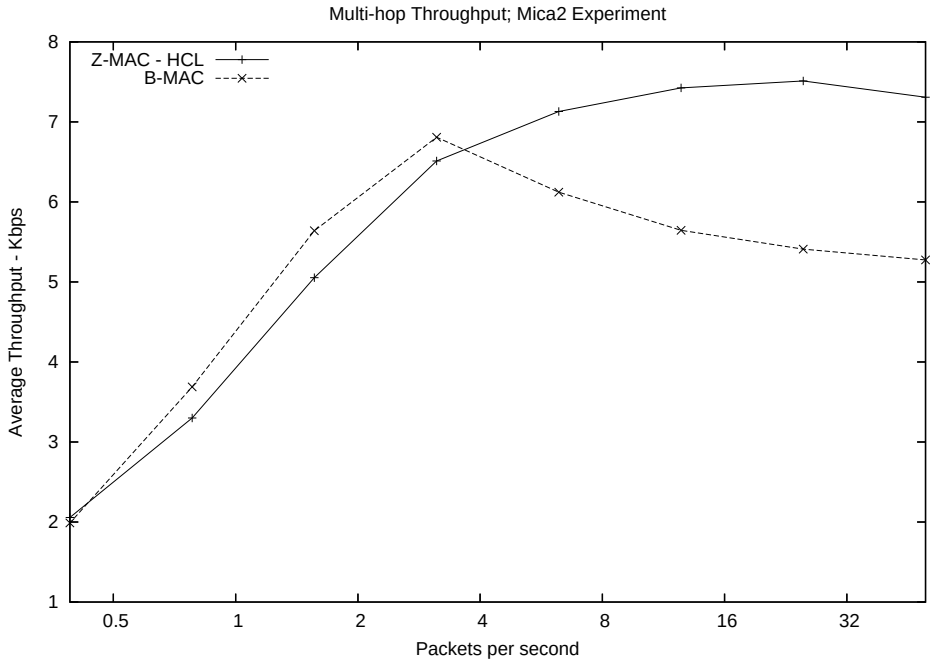

Figure 4.3: The data throughput comparison in the multi-hop MICA2 benchmark

data throughput when hidden terminals are present. We vary the number of senders while fixing the number of neighbors. For this experiment, the frame size is 16.

For the multi-hop test result, Figure 4.3 shows the data throughput result on our testbed with 42 MICA2 motes. Each node sends data with low data rate and delivers the data to the sink node.

4.2 Summary

In this chapter, we show the result of ZMAC experiment as an example of the application of the DRAND implementation. ZMAC works very well based on the TDMA time slot schedule from DRAND. This means DRAND can be applicable very well on not only ZMAC but also any other TDMA scheme.

Chapter 5

Conclusion and future Work

5.1 Conclusion

Nowadays, a lot of new MAC scheme are coming out with different algorithms for making better performance than others. But we can see that there are a few implementations among the published algorithms. Nobody can know whether some of the algorithms which are not implemented are working properly in the real life. The reason is because even though the algorithm has no defects theoretically on itself, if it is too complicated to implement or it causes other unexpected problems, consequently it would be a useless algorithm. So, the implementation and the proper result from the real experiment is very important to support the theoretical algorithms well.

We implemented the DRAND scheme by the algorithm of [27] with TinyOS and NesC. And we run various experiments with the DRAND implementation on the MICA2 motes and MIB600CA programming boards. The reason we implement DRAND is to build the full TDMA scheme finally. The TDMA scheme is considered as a kind of solution for preventing the defects of the CSMA scheme. For running the TDMA scheme on wireless sensor networks, the TDMA time slot scheduling algorithm with distributed and scalable method is very essential. Moreover, the scheduling algorithm should guarantee a small amount of energy consumption and a constant finishing time.

By the requirement of the TDMA time slot scheduling algorithm as we said

above, using the result of the various experiments under the single-hop and multi-hop case, we can confirm that the DRAND is working well as we expected by the analysis. The number of rounds of DRAND is increased linearly when we vary the number of nodes in the network. And we can see the average running time of DRAND is quadratically increasing based on the number of rounds and GRANT messages' frequency. The results are also similar when we run the experiment under the two-hop case and the simulation by NS2. In addition, the energy consumption of the DRAND algorithm is very reasonable value.

5.2 Future Work

We design and implement the DRAND scheme to run the TDMA scheme. One of the TDMA weak points is that it takes more overhead to react the topology change in the network. For example, when the link status is very lossy and the link connectivity is changed frequently, the one-hop and two-hop neighborhood among each node is also changed frequently. So, new nodes joining and existing nodes dropping happen a lot in this unstable network. Then because this TDMA scheme using the original neighbor table which was made by DRAND in the beginning time of the entire protocol cannot know the whole topology change immediately, there could be unexpected heavy packet loss during the routing time. In this case, DRAND should deal with this problem. But, at the present time, the best solution of this problem is to run the DRAND algorithm periodically. Or, if the packet loss is more than some threshold, then we can run the DRAND scheme again.

If we run the DRAND again, it is almost same as the entire protocol starts again. So, after running the DRAND again, sender nodes try to set up the routing path to the destination node again. The process for running DRAND again takes a while, so the latency from the source node to destination node could get longer and the amount of the energy consumption also could be more. This problem is not only the one of our DRAND scheme, but also the main problem of the traditional TDMA MAC protocol in wireless sensor networks. So, the final goal for working on the TDMA time slot scheduling algorithm is to solve all problem as listed above.

Moreover, the main reason we choose the TDMA MAC protocol is to enhance

the energy efficiency in wireless sensor networks against the defects of the CSMA scheme like BMAC. So, based on the DRAND scheme, using the TDMA scheme, we will implement an entire TDMA MAC protocol which would be a modified and improved scheme for the efficient energy consumption.

Bibliography

[1] Atmel Co. JTAG ICE, a Real Time In-Circuit Emulator for megaAVR devices with JTAG interface.
http://www.atmel.com/dyn/resources/prod_documents/DOC2475.PDF.

[2] Crossbow Technology Inc. Base Station/Ethernet Gateway for MICA2 and MICAz Motes.
http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MIB600CA_Datasheet.pdf.

[3] Crossbow Technology Inc. Mica2 wireless measurement system datasheet.
http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf.

[4] MICA2 AA Battery Pack Service Life Test.
http://www.xbow.com/Support/Support_pdf_files/MICA2_BatteryLifeTest.pdf.

[5] PowerTOSSIM: Efficient Power Simulation for TinyOS Applications.
http://www.eecs.harvard.edu/ shnayder/ptossim/.

[6] The Network Simulator - NS2.
http://www.isi.edu/nsnam/ns/index.html.

[7] TOS Debuggin and Development Techniques.
http://xbow.com/Support/Support_pdf_files/Motetraining/DebuggingEtc.pdf.

[8] G. Anastasi, A. Falchi, A. Passarella, M. Conti, and E. Gregori. Performance measurements of motes sensor networks. In *MSWiM04*. ACM, October 2004.

[9] K. Arisha, M. Youssef, and M. Younis. Energy-aware tdma-based mac for sensor networks. In *IEEE Workshop on Integrated Management of Power Aware Com-*

munications, Computing and NeTworking (IMPACCT 2002), New York City,NY, May 2002.

[10] L. Bao and J. J. Garcia-Luna-Aceves. A new approach to channel access scheduling for ad hoc networks. In *ACM MobiCom*, pages 210–221, 2001.

[11] I. Chlamtac and A. Farago. Making transmission schedules immune to topology changes in multi-hop packet radio networks. *IEEE/ACM Trans. Networking*, 2(1):23–29, Feb. 1994.

[12] R. von Behren M. Welsh E. Brewer D. Gay, P. Levis and D. Culler. The nesc language: A holistic approach to networked embedded systems. SIGPLAN, June 2003.

[13] A. El-Hoiydi. Aloha with Preamble Sampling for Sporadic Traffic in Ad Hoc Wireless Sensor Networks. In *Proc. IEEE Int. Conf. on Communications*, New York, USA, April 2002.

[14] A. El-Hoiydi, J. D. Decotignie, C. Enz, and E. Le Roux. Wisemac: an ultra low power mac protocol for the wisenet wireless sensor network. In *Poster, ACM Sensys 2003*, Los Angeles, CA, November 2003.

[15] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *HICSS '00: Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 8*, page 8020, Washington, DC, USA, 2000. IEEE Computer Society.

[16] J. Hill and D. Culler. A wireless embedded sensor architecture for system-level optimization, 2001.

[17] Barbara Hohlt, Lance Doherty, and Eric Brewer. Flexible power scheduling for sensor networks. In *IPSN'04: Proceedings of the third international symposium on Information processing in sensor networks*, pages 205–214, New York, NY, USA, 2004. ACM Press.

[18] T. Inukai. An efficient ss/tdma time slot assignment algorithm. *IEEE Trans. Communications*, 27:1449–1455, 1979.

[19] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. Tinyos: An operating system for wireless sensor networks, 2004.

[20] J. Li and G. Lazarou. A bit-map-assisted energy-efficient mac scheme for wireless sensor networks. In *3rd Int. Symp. on Information Processing in Sensor Networks (IPSN04)*, pages 55–60, Berkeley,CA, April 2004.

[21] M. Luby. Removing randomness in parallel computation without processor penalty. *Journal of Computer and System Sciences*, 47(2):250–286, Oct. 1993.

[22] Matt Welsh Philip Levis, Nelson Lee and David Culler. Tossim: Accurate and scalable simulation of entire tinyos applications. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, New York, NY, USA, 2003. ACM Press.

[23] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 95–107, New York, NY, USA, 2004. ACM Press.

[24] V. Rajendran, K. Obraczka, and J. J.Garcia-Luna-Aceves. Energy-efficient, collision-free medium access control for wireless sensor networks. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Los Angeles, CA, November 2003.

[25] Injong Rhee. Route-aware mac protocols for sensor networks. Technical report, Computer Science Department, North Carolina State University, Raleigh, NC, 2004.

[26] Injong Rhee, Ajit Warrier, Mahesh Aia, and Jeongki Min. Z-mac: a hybrid mac for wireless sensor networks. In *SenSys '05: Proceedings of the 3nd international conference on Embedded networked sensor systems*, pages 90–101, New York, NY, USA, 2005. ACM Press.

[27] Injong Rhee, Ajit Warrier, and Lisong Xu. Randomized dining philosophers to

tdma scheduling in wireless sensor networks. Technical report, Computer Science Department, North Carolina State University, Raleigh, NC, 2004.

[28] R. Rozovsky and P. R. Kumar. SEEDEX: a MAC protocol for ad hoc networks. In *International Symposium on Mobile ad hoc networking & computing*, pages 67–75, 2001.

[29] T. van Dam and K. Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. In *ACM SenSys*, pages 171–180, 2003.

[30] L. van Hoesel and P. Havinga. A lightweight medium access protocol (lmac) for wireless sensor networks: Reducing preamble transmissions and transceiver state switches. In *INSS 2004*, Tokyo, Japan, June 2004.

[31] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *IEEE INFOCOM*, pages 1567–1576, 2002.

[32] C. Zhu and M.S Corson. A five-phase reservation protocol (fprp) for mobile ad hoc networks. In *IEEE INFOCOM '98*, volume 1, pages 322–331, San Francisco, CA, March-April 1998.