Distributed Scalable TDMA Scheduling Algorithm

Injong Rhee and Jangwon Lee
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-7534

ABSTRACT

Lack of scalability and channel efficiency is often touted as the main drawback of TDMA for wireless sensor networks. However, the results of this paper dictate otherwise. In the paper, a new distributed TDMA scheduling algorithm, called *DRAND*, is employed that gives a channel schedule as efficient as RAND - a commonly used centralized channel allocation scheme, but does so only in the expected running time and message complexity of $O(\delta)$ where δ is the maximum number of contending neighbors. This is the first distributed, scalable implementation of RAND. As the running time of DRAND does not depend on the total size of the input network, it is highly scalable, thus apt for large-scale sensor networks.

Keywords

Sensor networks, MAC, TDMA, Channel allocation

1. INTRODUCTION

Many contention-based MAC schemes [1, 2, 3, 4] are proposed for wireless ad hoc/sensor networks due to its easy deployment. In contention-based MAC schemes, there is no startup overhead incurred. That is, after deployment, nodes can instantly start communication among their neighborhoods while trying to avoid collision using RTS (Request-to-send) and CTS (Clear-to-send) signals. One more advantage of contention-based MAC schemes is flexibility in that it can accommodate with dynamic environments resulting from mobility of nodes in ad hoc networks, node failures, and energy depletion. However, broadcast in contention-based approaches is susceptible to packet loss. Furthermore, under a dense network or increased offered load, the performance of contention-based schemes can significantly degrade due to higher contention among neighborhoods. That is, there may cause reduction of battery life and huge control packet overhead for retransmission. This points to the need of collision-free MAC protocols based on scheduling.

TDMA is one of schedule-based MAC protocol which is a subject of an active and broad research (see [5]). TDMA provides collision-free transmission from nodes or links since a set of time slots are prearranged. Thus, TDMA can adapt well to various network densities and offered loads. An efficient TDMA schedule can save energy by allowing nodes to turn on the radio only during the scheduled transmission times of their neighbors, without wasting energy due to idle listening and overhearing unlike contention-based schemes. It is well-known that idle listening consumes as much as energy as receiving. Furthermore, as TDMA does not require any control message exchanges for communication (e.g., RTS/CTS), it limits overhead in communication. Finally note that since sensor networks are relatively stationary, the impact of dyanmic environments on TDMA MAC schemes can be lessened. Thus, the deployment cost can be amortized during the entire life time of sensor networks unless there is quite frequent channel assignment procedure. Despite these benefits, there are several challenges of a sensor MAC protocol that employs TDMA for sensor networks. Among these, the biggest challenge is lack of scalability and efficiency in existing solutions for producing an efficient time slot assignment.

- *Scalability*: Recent advances in processor and low power wireless communication technologies have enabled the deployment of large-scale sensor networks consisting of thousands, or even millions of nodes, of small and cheap sensor nodes. The performance of sensor MAC protocols must be scalable and independent of the total size of networks, as these networks are often deployed in large scale. Obtaining the optimal TDMA schedule is NP-hard [6]. Either existing heuristics require global topology information and centralized [7], or their performance still depends on the size of total networks [6, 8, 9].

- *Efficiency (Low latency)*: Recent distributed "more scalable" heuristic solutions [10, 11, 12] are originally designed for mobile ad hoc networks. These solutions typically run in a per-slot basis: for each slot, they determine a set of nodes to transmit safely without interference. Because of this per-slot property, it is difficult to find a schedule in which all nodes can transmit exactly once. Therefore, even if they are adapted to work in stationary sensor networks, the resulting schedules would be very inefficient. Furhtermore, schedule-based schemes are known for higher delays [13]. However, longer delays are an artifact of inefficient scheduling. The TDMA scheduling protocol being employed for sensor networks must efficiently take advantage of spatial resue of time slots.

To overcome these challenges with TDMA, we develop a new distributed TDMA scheduling scheme, called DRAND, that implements RAND [7], a commonly used centralized channel allocation scheme for mobile adhoc networks e.g.,

[14, 6]. This is the first scalable, distributed implementation of RAND to date, to the best of our knowledge. RAND sorts all the nodes in the graph in a random total order and assigns to each node, in that order, the minimum color (or channel) that has not been taken by its adjacent, but preceding nodes. Since RAND requires the knowledge of the global network topology, it is impractical for large-scale sensor networks. DRAND gives a channel schedule as efficient as RAND, but does so in the expected running time and message complexity of $O(\delta)$ where δ is the bound on the number of contending neighbors.

RAND has been used as a benchmark for many distirubted dynamic channel access algorithms [12, 11]. The theoretical computer science community has also independently worked on distributed graph coloring extensively [ref]. The best distributed algorithms for graph coloring are randomized ones developed for a completely synchronous environment. In these algorithms, a randomly chosen color from a a palette of available colors is assigned to a node until that node has a different color from its adjacent (contending) nodes. However, the performance of both dynamic channel access schemes and palette-based algorithms are still inferior to that of RAND in terms of *worst chromatic number* which is defined by the maximum number of colors used by an algorithm for an input network. (See Section ? for details of related work.) Thus, to our best knowledge, DRAND is the most efficient distributed TDMA schemes working in asynchronous environments.

Clearly, large- scale sensor networks follow a pattern of unit-disk graphs [15], and the total network size is much larger than δ. DRAND is highly scalable and efficient, thus apt for large-scale sensor networks Our implementation of RAND is *exact* in the sense that for any input graph, DRAND can produce any channel assignments producible by RAND and vice versa. It is surprisingly simple and easy to implement. Finally, DRAND is inherently fair since every node in the network can transmit a packet within worst chromatic number of slots.

We formulate the channel assignment problem in Section 2, and present DRAND and its correctness, practical issues from Section 3 to Section 5. To study performance of DRAND in more realistic environments, we implemented DRAND in ns-2, and compared its performace with the other MAC protocol [12] in Section 6. Section 7 discusses related work, and Section 8 concludes the paper.

2. CHANNEL ASSIGNMENT

In this section, we formally define the channel assignment problem. The network is represented by a graph $G = (V, E)$ where V is the set of nodes, and E is the set of edges. An edge $e = (u, v)$ exists if and only if u and v are in V and u can hear from v and vice versa (i.e., all edges are bidirectional). A frame is divided into the *MaxSlot* number of non-overlapping equal time periods, called time slots. The slots are numbered from 1 to *MaxSlot*. We assume that *MaxSlot* is sufficiently large to handle all the assignment strategies. Once the channel assignment is finished, the frame size will be re-adjusted to fit to the maximum number of slots actually used (more on this in Section 5).

Informally, the objective of channel assignment is that each node picks a time slot during which it can transmit without "conflict". We say that two nodes u and v are in *conflict* if and only if the simultaneous transmission from u and v causes radio interference at some node. The definition of interference (or conflict) determines the channel assignment problems. For example, in the broadcast scheduling mode, conflict can happen among all the nodes within a two-hop distance. A good list of conflict relations in wireless networks is defined in [7].

We define the *channel assignment problem* as finding a time slot for each node, given an input graph G and conflict definition, such that if any two nodes are in conflict, they do not have the same time slot. The performance of an algorithm for the channel assignment problem can be determined by three quantities:

- *Worst-case chromatic number*: the maximum number of time slots required to solve the problem for all executions of the algorithm.
- *Running time*: the maximum time taken for all the nodes in V to decide on their time slots for all executions of the algorithm.
- *Message complexity*: the maximum number of messages transmitted for all the nodes in V to decide on the time slots for all executions of the algorithm.

3. DESCRIPTION OF DRAND

We first describe the algorithm in a distributed synchronous message-passing model [16] where every operation runs in a synchronous round and communication is reliable. We discuss in Section 5 how to implement the algorithm in a distributed asynchronous model where communication may not be reliable. We describe the algorithm only for broadcast scheduling and omit the generalization of the algorithm for other conflict relations. We assume that all nodes know their neighbors in a two-hop distance and the network is connected. These issues are discussed further in Section 5.

At each round i, the algorithm performs following steps:

> With a probability p_i where p_i is set to the inverse of the number of contending nodes (including itself) that have not decided their slots in earlier rounds, a node **A** broadcasts a *request* message to its one-hop neighbors if it has not decided on its time slot. When a neighbor **B** receives a *request* from **A**, if it is not aware of any of **B**'s one-hop neighbor (including **B**) that has sent a *request* message earlier than **A** *in the same round*, then it sends a *grant* message to **A**. If and when **A** receives *grant* messages from its entire one-hop neighbors, it decides on its time slot to be the minimum of the time slots that have not been taken by its two-hop neighbors before this round. Then **A** broadcasts a *release* message containing information about the selected time slot to its one-hop neighbors. When receiving a *release* message, a node re-broadcasts that message to its one-hop neighbors. Let us call this "forwarded" *release* message to be a *two-hop-release* message.

A node which sent a request, but could not get grant messages from all of its one-hop neighbors in the current round, simply gives up and repeat the above step in the next round. When receiving a *release* or a *two-hop-release* message, a node can update the number of contending neighbors in a two-hop distance that have decided. This information is used in setting the probability p_i for the next round. We note that the choice of probability p_i is critical in the performance of the algorithm which is discussed in the next section.

4. CORRECTNESS AND PERFORMANCE OF DRAND

The *safety* property of the algorithm (i.e., no two nodes within a two-hop distance can decide on a time slot at the same round) can be easily seen from the followings: (1) a node has to receive *grant* messages from all of its one-hop neighbors, (2) any nodes which are two-hop away from each other, share at least one common one-hop neighbor and (3) a node sends *at most* one *grant* message at each round. This property ensures that when a node decides, it always picks the minimum time slot that is not taken by two-hop neighbors and no other nodes within two- hop neighbors can pick the same time slot.

The *liveness* property of the algorithm is that every node in the network *eventually* decides its time slot in a finite time. To show this, it is sufficient to see that the probability that a node decides its slot is always larger than 0. Intuitively, it means that since at each round, some (fractional) node will decide and would not compete during ensuing rounds, the number of contenders (for grabbing a time slot) within a two-hop distance always decreases as rounds progress. Therefore, the convergence of the algorithm is guaranteed. Next, we show that the expected running time for DRAND algorithm is $O(\delta)$ where δ is the maximum number of contending neighbors.

Let C_i be the expected number of contending nodes including itself in a round i and we set p_i to be $\frac{1}{C_i}$ in a round i. Then, the probability that a node with C_i contending nodes (including itself) decides at a round i, $Pr_i(assigned)$, is greater than or equal to the probability that only one node among C_i sends a *request*:

$$Pr_i(assigned) \geq p_i(1-p_i)^{C_i-1} \tag{1}$$

Note that $C_i \leq \delta+1$ and if C_i becomes 1 (all of its contending neighbors decided slots), then $Pr_i(assigned) = p_i = 1/C_i = 1$. The expected number of nodes deciding their slots in a round i, D_i, is $C_iPr_i(assigned)$, i.e.,

$$D_i \geq \bar{D}_i = C_ip_i(1-p_i)^{C_i-1} \tag{2}$$

Note that when $p_i = 1/C_i$, $\bar{D}_i$ is maximized in the Eq. (2). Thus, in order to maximize the speed of convergence, at each round, we set probability p_i to the inverse of the number of contending nodes.

The next $i+1$ round has the expected number of contending nodes C_{i+1},

$$C_{i+1} = C_i - D_i = C_1 - \sum_{m=1}^{i} D_m \tag{3}$$

Putting $p_i = 1/C_i$ in Eq. (2), D_i has a lower bound e^{-1} as follows:

$$D_i \geq \bar{D}_i = (1-\frac{1}{C_i})^{(C_i-1)} \geq e^{-\frac{C_i-1}{C_i}} \geq e^{-1} \tag{4}$$

From Eq. (3) and (4), we have

$$C_k \geq C_1 - ke^{-1} \tag{5}$$

When C_k becomes less than 0, every node chooses its time slot. Thus, if the number of rounds is larger than eC_1, then DRAND algorithm completes channel assignments. Since C_1 is bounded by $\delta+1$, the expected running time of DRAND is $O(\delta)$ rounds.

Let us examine the message complexity of DRAND. According to the algorithm, each node sends $O(1)$ messages per round. In a round, if a node is requesting, it will send at most two messages, a *request* and a *release* - when it sends a *release* message, it does not have to send a *two-hop release* since, by the safety property, no two contending nodes decide at the same time. Also by the same token, a node does not receive more than one release message in a round. If a node is not requesting, then it will also send at most two messages, a *grant* and a *two-hop-release*. Since a node decides its slot in $O(\delta)$ rounds on average, a node sends at most $O(\delta)$ messages on average before it makes the decision.

We now prove that the algorithm implements RAND from the perspective of worst chromatic number. The intuition is clear, but its formal proof is a bit subtle.

THEOREM 1. *For any execution E_r of RAND, there exists a corresponding execution of DRAND that produces the same slot assignment as E_r, and conversely, for any execution of E_d of DRAND, there exists a corresponding execution of RAND that gives the same slot assignment as E_d.*

Suppose that S $(= v_1, v_2, v_3, ..., v_n)$ is the sequence in which the nodes are assigned the time slots in E_r, and $s(u)$ is the slot assigned to a node u in the execution of RAND (Here, $n = |V|$). We divide S into m non-overlapping partitions P_1, P_2, P_3, ..., P_i, P_{i+1}, ..., P_m for $m > 0$ where P_i is a subsequence of S, and the concatenation of P_1 through P_m yields S. No two nodes in each partition $P_i (1 \leq i \leq m)$, conflict with each other. That is, $S = P_1P_2P_3...P_m = v_1, v_2, v_3, ...v_n$. The following lemmas are sufficient to prove the first part of the Theorem 1.

LEMMA 1. *There exists an execution E_d of DRAND that all the nodes in P_i $(1 \leq i \leq m)$ decide their time slots in a round i.*

Proof: We will prove the lemma by induction. Consider $i = 1$. It is possible that each node in P_1 sends *requests* and

any contending nodes of nodes in P_1 do not send a *request* at the first round. This guarantees that nodes in P_1 decide their own slots in the first round because no two nodes in P_1 are in conflict by the definition of partition. By the induction hypothesis, suppose that there exists an execution of DRAND in which nodes in P_i decide their time slots in a round i ($1 \leq i \leq h-1$). That is, $P_1P_2...P_{h-1}$ is the node ordering up to round $h-1$ in some execution of DRAND. As the nodes in P_h have not decided in the earlier rounds than h, according to the algorithm, there exists a non-zero probability that only those nodes in P_h send a *request* in round h. Since by definition, those nodes are not in conflict, they decide on their slots in round h. ■

LEMMA 2. *In the execution E_d of DRAND, a node v_i ($\forall i, 1 \leq i \leq n$) chooses the same slot as in E_r (i.e., $s(v_i)$).*

Proof: By induction on i. Consider i =1. Since v_1 is the first node to decide in E_r and E_d, it will choose time slot 1 in both executions. Suppose that all the nodes before v_h, $(v_1, \ldots, v_{h-1})$ choose the same colors in both executions. By the algorithm of RAND, if v_h chooses $s(v_h)$, then $s(v_h)$ must be the minimum slot that has not taken by all of its conflicting nodes that have decided earlier than v_h. Since in both executions, v_h will have the same set of conflicting nodes that have decided before v_h by Lemma 1, and by the hypothesis, they have chosen the same colors in both executions. Therefore, the algorithm of DRAND will also dictate v_h to choose $s(v_h)$. ■

Lemmas 1 and 2 prove the first part of the Theorem 1. The second part of the Theorem 1 can be proved in the similar fashion and we omit its proof.

■

5. PRACTICAL ISSUES

5.1 Determining time slot size

TDMA requires clock synchronization. Typical clock synchronization algorithms [17] offer clock drifts in the order of microseconds or less. Also it is not uncommon to see sensors equipped with GPS [18] which can give much more accurate synchronization. To allow slack time for clock drift, the slot time must be several times larger than the drift. Typically, the slot size must be large enough to transmit at least one packet. The typical packet size in sensor networks are around 8-16 bytes [19, 17]. Given a 115 kbps network, assuming a packet size of 20 to 30 Byte packets, this runs around 1.4 ms to 2.1 ms. A slot size in order of 2 ms to 10 ms would be sufficient to handle the clock drift and one packet. The slot size must also be kept small because a large slot would result in larger delay.

Switching overhead is another factor in determining the slot size. When a sensor turns on the radio, it incurs non-negligible transition overhead. This overhead ranges from 20 μs [20] and a few hundred microseconds [21]. Thus, the slot size must also be long enough to compensate for the cost of the switching overhead. As this overhead time can overlap with the clock drift, we consider 2 to 10 ms a reasonable slot size.

5.2 Running under asynchronous environments

In an asynchronous environment, it may be difficult to run the algorithm in synchronous rounds as each node may start at a different time and communication may also be unreliable. But in TDMA environments where clocks are well synchronized within a bound, it is rather straightforward to implement DRAND.

To ensure the reliability of message communication and avoid deadlock due to message losses, the algorithm uses a three-way handshake with retransmission: a *request* message is always acknowledged by a *grant* message from a neighbor, and a *grant* message is always acknowledged by a *release* message, and *request* and *grant* messages are always retransmitted until their corresponding acknowledgments arrive.

A round is set to a time period large enough to finish the three-way handshake with some additional slack time for retransmission. If a requesting node does not receive a *grant* from the entire one-hop neighbors before one round expires, then it gives up on this round by sending a *release* message (with no slot assignment information), and moves on the next round where it restarts with probability p_i as described in Section 3. On the other hand, when a granting node does not receive a *release*, it has to retransmit the grant message until it receives a release message. This is because the requesting node may or may not decide on a slot in the current round, and the granting node must know its outcome to proceed. A granting node retransmits the *grant* message to its requesting node even after the current round until it receives a *release* message from the node. Therefore, when receiving a *grant* message, even if a node decided and sent a *release* message in the previous rounds or if it gave up requesting in the previous rounds and is not requesting any more in the current round, it must transmit a *release* message to "release" the granting node. If a requesting node receives a *grant* message that was triggered by a *request* sent in earlier rounds, it can still use this *grant* for the current round because the granting node must wait until it receives a *release* message from the requesting node.

One drawback of this round-based scheme is that if the period of one round is too small or too large, the assignment may take a longer time to finish. Another way to implement the algorithm in asynchronous environments is to have a new message called *reject*. If a node **B** receives a *request* from **A** when it has granted to another node **C** and is waiting for a *release*, then **B** sends a *reject* message to **A**. Once **A** receives a *reject* message, it gives up and sends a *release* (with no slot assignment information). In this scheme, the program does not run by a round and no requesting node gives up until it receives a *reject* or *grant*. The correctness of the algorithm does not change by the use of *reject* since the *reject* messages are used only to make requesting nodes give up earlier so that they can retry. It will, though, increase the message complexity by a factor of the number of one-hop neighbors. (We implement this algorithm in our simulation presented in Section **??**.)

Careful engineering on the message timing, however, avoids a substantial number of message exchanges. To avoid collision due to simultaneous transmission among neighbors, we force each node to wait a random amount of time before each transmission of a message. This random waiting helps reduce collision especially for *grant* or *reject* messages as they are triggered by a *request* broadcast. Further, by broadcasting and overhearing *grant* and *reject* messages as well as *request* and *release* messages, we can suppress some message transmissions: when a node **B** overhears a *reject* or *grant* message destined to another node from a node **A** while waiting to transmit a *request*, **B** can simply suppress the transmissions of the *request*, and wait for a longer period of time before it carries out the next *request* try. This is allowed because the *grant* or *reject* message indicates that its transmitter **A** has given a *grant* to some other node, so it's no use for **B** to send the *request*.

Our algorithm description has assumed that each node knows all of its two-hop neighbors. In fact, the safety of the algorithm still holds as long as one of any two one-hop neighbors is aware of the other. This is because a node needs to receive a *grant* message from the other before deciding. As the algorithm progresses, the two nodes become aware of each other, they will decide on their own slots. In an unlikely event that none of the two neighbors knows each other, the algorithm needs to be restarted. We are not aware of any algorithm that can work in such an event.

5.3 Reducing the frame size

We have assumed that the system fixes *MaxSlot* sufficiently large at the beginning of the execution of the algorithm. However, a large frame size leads to a long delay, as each node has to wait for its slot in a frame. In a greedy coloring-based technique such as RAND, it is sufficient to set *MaxSlot* to the maximum number of nodes in conflict, $\delta+1$ [22]. But typically in sensor networks where conflicts occur because of a radio range, the input network is very close to a unit-disk graph [15]. In such networks, the number of slots allocated by RAND can be far fewer than $\delta + 1$.

In order to reduce the frame size, we can start DRAND with a large enough *MaxSlot* and when we gather information about the maximum slot number selected by the network, we then set the frame size to that maximum slot number being assigned. In large-scale networks, we need a scalable way to accomplish this information gathering and propagation.

To gather and disseminate the frame size after deciding their slot numbers, all the nodes wait until their one-hop neighbors finish assignments. Then they synchronize to a predetermined periodic time (e.g., every minute by synchronized clocks). Using *MaxSlot* as the frame size, they then broadcast the maximum slot number taken by their neighbors, in their selected slot times. Transmitting during their own slot times reduces the possibility of collision in case when some nodes have not finished their assignments. If a node hears a slot number larger than the one it currently knows, it rebroadcasts the new number. Otherwise, it keeps quiet (i.e., suppresses it). After some predetermined time for the flooding to spread the entire network, all the nodes synchronizes to set their frame sizes to be the maximum slot number that they have heard. Again, the synchronization can occur at the predetermined periodic time.

As it is possible that some node may learn about a new maximum number even after it synchronizes to its previously known maximum slot number, the periodic synchronization of frames based on the global clock helps re-synchronize even after nodes use a wrong frame size. To enable this, we set the synchronization time to be always some multiple of a slot size. The slot size, synchronization time, and *MaxSlot* are among the global information that every node needs to know before deployment.

# of nodes	Max. # of slots	Run time(sec)	Avg. # message
100	8	6.04	5.19
200	9	7.56	5.91
300	10	9.25	5.74
400	11	8.42	6.24
500	9	8.31	5.93

Table 1: DRAND performance results

6. DRAND PERFORMANCE EVALUATION

The main goals of the performance evaluation of DRAND is to show that (1) the performance of DRAND is scalable and independent of the total size of the network, and (2) DRAND produces an efficient channel assignment (i.e., a small worst chromatic number) with low overhead in a distributed environment.

First to show that the performance of DRAND is independent of the network size, we test DRAND in various network configurations with differing numbers of sensor nodes. We generate five sensor networks of different sizes, ranging from 100 to 500 nodes with an increment of 100 nodes. The 100 node sensor network configuration is generated by randomly placing the nodes in a 1000 x $1000m^2$ square. We use 200 Kbps radio bandwidth and $100m$ radio transmission range. The other configurations are generated by increasing the number of nodes as well as the area of the sensor field by the same factor as the increase in the number of nodes. This scaling of the sensor field while keeping the radio range constant keeps the average density of nodes approximately constant.

We measure (1) the number of slots assigned (worst chromatic number), (2) running time, and (3) message complexity, i.e., the average number of messages transmitted per node during the run time of the protocol.

Table 1 shows the performance of DRAND for the five different configurations. Since we approximately make density of nodes constant, the maximum numbers of slots assigned are approximately the same for all the tests. We note that the running time and message complexity are unaffected as we increase the number of nodes. This indicates that DRAND is scalable to the total size of network.

To compare the performance of DRAND to that of the existing TDMA scheduling schemes, we implemented a modified version of a well-known distributed TDMA scheduling algorithm called *Five-Phase Reservation Protocol* (FPRP) [12] in *ns-2*. FPRP is a very efficient TDMA protocol developed

Transmission ranges	100m	150m	200m	250m
Avg. # of one hop neighbors	2.7	5.82	9.84	14.54
Avg. # of two hop neighbors	4.84	12.9	24.8	36.82
Max # of two hop neighbors	13	30	42	61

Table 2: Characteristics of sensor fields (100 nodes in 1000 x 1000 m^2)

for mobile ad-hoc networks but can be adapted to provide a static slot assignment. FPRP alternates between a reservation frame and a data transmission frame. In a reservation frame, a segment of time slots is assigned to a set of nodes for data transmission, and in the following data transmission frame, the nodes that are assigned a slot in the previous reservation frame can transmit during their assigned slots. We can adapt the reservation frame for our purpose as follows. During a reservation frame of FPRP, for a given slot, each node that has not assigned a slot repeats a number of five-phase reservation cycles to compete for that slot. After a "sufficient" number of trial, FPRP assumes that a "sufficient" number of nodes have been selected for transmission during that slot. Clearly as more cycles are employed, it returns a better channel assignment, but with an increased running time. Unfortunately, in a distributed environment, nodes do not know in advance how many reservation cycles are required. Although this scheme may not be practical for a static assignment required for our purpose, for a comparison purpose, we can test the algorithm with various numbers of the reservation cycles. We denote FPRP-x to be the run with x number of reservation cycles for each slot.

We generate the input topology to the algorithms by randomly placing 100 nodes in a 1000 x 1000 m^2 network. At this time, we vary transmission ranges of each node from $100m$ to $250m$ to control the number of neighbors (i.e., density). We set $2.5ms$ to be the duration of each reservation cycle. Table 2 shows the characteristics of the various network configurations being tested. Table 3 shows the performance results of DRAND and FPRP. As the density increases, we observe that the number of slots, the message complexity, and the running time increase. DRAND outperforms all of (modified) FPRPs on the number of slots and message complexity. For the most dense case (a $250m$ transmission range), the number of slots by DRAND is far less than that of FPRP (up to 34%). Note that any percentage reduction in the number of slots can be translated into the same percentage improvement of the overall performance in TDMA since it represents the size of frame in RASMAC. The running time of DRAND is comparable to that of FPRP-30, and is less than FPRP-50. Considering that DRAND produces much more efficient channel assignments than FPRP-50, this result is encouraging. The number of transmitted messages in DRAND is far less than that in FPRP, which implies much less energy consumption for DRAND.

Max # of slots					
	FPRP-10	FPRP-20	FPRP-30	FPRP-50	DRAND
100m	12	10	11	9	8
150m	18	19	19	17	14
200m	31	31	30	34	24
250m	59	40	45	41	34
Average # of message transmitted					
	FPRP-10	FPRP-20	FPRP-30	FPRP-50	DRAND
100m	13.41	18.32	22.29	29.56	6.88
150m	35.01	41.15	48.7	52.69	18.37
200m	80.36	81.36	92.13	109.03	30.78
250m	185.73	132.81	163.4	160.32	52.06
Run time (sec)					
	FPRP-10	FPRP-20	FPRP-30	FPRP-50	DRAND
100m	1.91	4.63	11.03	17.03	7.32
150m	5.43	11.87	20.01	39.03	14.86
200m	11.47	26.23	33.87	61.03	32.53
250m	22.63	34.27	50.63	81.15	64.87

Table 3: The performance results of DRAND and (static) FPRP

7. RELATED WORK

Wireless MAC has been a subject of an active and broad research [2], [1], [3]. In this section, we relate our work only to MAC schemes for sensor networks. Stankovic et al. [5] gives a good survey of them.

Sohrabi et al. [23] propose a distributed MAC scheme that combines both TDMA and FDMA. Use of two different mediums (time and frequency) reduces the chance of collision, but it incurs high cost, as it requires essentially two radio systems in each sensor. As SMAC [24] is extensively discussed in the earlier sections, we skip the discussion here. Guo et al. [25] gives a new CDMA scheme that adopts a graph coloring similar to $\delta + 1$ coloring algorithms by Luby [26] and Johansson [27], which color any graph with $\delta+1$ colors where δ is the maximum number of two- hop neighbors. TMAC [28] enhances SMAC by sending data in burst in a shorter active period and allowing nodes to sleep if no signals are detected for some period of time even during the scheduled active period. DMAC [29] also improves on the delay problem of SMAC especially for data gathering applications where routes follow a tree-like structure. By staggering active periods between the parent and children, it achieves low delays for some specialized networks (where routes are predetermined). Although DMAC uses route information to reduce delay, it is not clear how this information can be obtained and how route adaptation can be performed. Both TMAC and DMAC still carry the same drawback as SMAC where they are route-oblivious and node outside route paths waste energy via unnecessary idle listening.

Rajendran et al. [13] proposes a schedule-based TDMA scheme called TRAMA that bears some similarity to our RA-TDMA. Like RA-TDMA, TRAMA allows nodes that are not transmitting and receiving to sleep opportunistically. It requires each node to periodically transmit packet information such as sources, destinations, and the size of packets to transmit. Based on this information, a TDMA scheduling scheme, called NAMA [11], is used to produce transmission schedules for the next period. Due to its route awareness, TRAMA saves a lot of energy, because each node knows exactly how and when packets are transmitted. However, this incurs a lot of delay because of scheduling overhead. Their experiments indicate that its delay characteristics are several orders of magnitude worse than SMAC.

TDMA scheduling is an extensively studied subject (see [5]). Most of early work is centralized and has performance dependency to $O(n)$ where n is the total size of the network. Recent distributed solutions [11], [12], [30], [10] improve

the performance by removing global topology dependency. NAMA [11] and FPRP [12] obtain dynamic channel assignments where without a notion of frames, every time slot is contended by some of the neighboring nodes. Dynamic and topology independent assignments are inapplicable for route-awareness since channels being used for transmission by a node is not known a priori. NAMA uses a hash function to determine priority among contending neighbors. One main drawback of this hashing based technique is priority chaining; even though a node gets a higher priority in one neighborhood, it may still have a lower priority in other neighborhood. This chaining can build up to $O(n)$, yielding a very inefficient schedule. Thus the worst chromatic number of NAMA is $O(n)$. SEEDEX [30] uses a similar hashing scheme based on random seed exchanged in a two-hop neighborhood. However, its worst case chromatic number is $\delta + 1$ as each node can pick randomly (instead of the minimum) a channel among those not taken by the others. FPRP [12] is discussed extensively in Section 6.

8. CONCLUSION

Existing TDMA schemes are not scalable. We claim that this is not an inherent feature of TDMA, but rather an artifact of inefficient schemes. In this paper, we develop DRAND that gives a very efficient channel assignment with only $O(\delta)$ time and message complexity on average.

9. REFERENCES

[1] A. Woo and D. Culler, "A transmission control scheme for media access in sensor networks," in *ACM MobiCom*, 2001, pp. 221–235.

[2] LAN MAN Standards Committee of the IEEE Computer society, "Wireless lan medium access control (mac) and physical layer (phy) specification," in *IEEE Std 802.11*, 1997.

[3] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang, "MACAW: A media access protocol for wireless LAN's," in *ACM SIGCOMM*, 1994, pp. 212–225.

[4] S. Singh and C. S. Raghavendra, "PAMAS- power aware multi-access protocol with signalling for ad hoc networks," *ACM SIGCOMM Computer Communication Review*, vol. 28, no. 3, pp. 5–26, July 1998.

[5] J. A. Stankovic, T. E. Abdelzaher, C. Lu, L. Sha, and J.C. Hou, "Real-time communication and coordination in embedded sensor networks," *Proceedings of IEEE*, vol. 91, no. 7, pp. 1002–1022, Jul. 2003.

[6] A. Ephremides and T. truong, "Scheduling broadcasts in multihop radio networks," *IEEE Trans. Communications*, vol. 38, no. 4, pp. 456–460, Apr. 1990.

[7] S. Ramanathan, "A unified framework and algorithms for (T/F/C)DMA channel assignment in wireless networks," in *IEEE INFOCOM*, 1997, pp. 900–907.

[8] I. Cidon and M. Sidi, "Distributed assignment algorithms for multihop packet radio networks," in *IEEE Trans. Computers*, 1987, pp. 739–746.

[9] L. C. Pond and V.O.K. Li, "A distributed time-slot assignment protocol for mobile multi-hop broadcast packet radio networks," in *IEEE MILCOM*, 1989, pp. 70–74.

[10] I. Chlamtac and A. Farago, "Making transmission schedules immune to topology changes in multi-hop packet radio networks," *IEEE/ACM Trans. Networking*, vol. 2, no. 1, pp. 23–29, Feb. 1994.

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

[12] C. Zhu and M. S. Corson, "A five phase reservation protocol (FPRP) for mobile ad hoc networks," *Wireless Networks*, vol. 7, no. 4, pp. 371–384, Sep. 2001.

[13] 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, pp. 181–192.

[14] R. Ramaswami and K. K. Parhi, "Distributed scheduling of broadcasts in a radio network," in *IEEE INFOCOM*, 1989, pp. 497–504.

[15] W. K. Hale, "Frequency assignment: Theory and applications," *Proceedings of IEEE*, vol. 68, no. 12, pp. 1487–1514, Dec. 1980.

[16] H. Attiya and J. Welch, *Distributed Computing*, McGraw-Hill, UK, 1998.

[17] J. Elson and D. Estrin, "Time synchronization for wireless sensor networks," in *International Parallel and Distributed Processing Symposium*, 2001, pp. 1965–1970.

[18] "Crossbow technology inc.," http://www.xbow.com.

[19] K. Sohrabi and G. J. Pottie, "Performance of a novel self-organization protocol for wireless ad hoc sensor networks," in *IEEE 50th Vehicular Technology Conference*, 1999, pp. 1222–1226.

[20] "Mote," http://www.cs.berkeley.edu/~awoo/smartdust.

[21] E. Shih, S. Cho, N. Ickes, R. Min, A. Sinha, A. Wang, and A. Chandrakasan, "Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks," in *ACM MobiCom*, 2001, pp. 272–287.

[22] D. A. Grable and A. Panconesi, "Fast distributed algorithms for brooks-vizing colourings," in *ACM-SIAM Symposium on Discrete Algorithms*, 1998, pp. 473–480.

[23] K. Sohrabi, J. Gao, V. Ailawadhi, and G. Pottie, "Protocols for self-organization of a wireless sensor network," *IEEE Personal Comm. Mag.*, vol. 7, no. 5, pp. 16–27, Oct. 2000.

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

[25] C. Guo, L. C. Zhong, and J. M. Rabaey, "Low power distributed MAC for ad hoc sensor radio networks," in *GLOBECOM*, 2001, pp. 2944–2948.

[26] M. Luby, "Removing randomness in parallel computation without processor penality," *Journal of Computer and System Sciences*, vol. 47, no. 2, pp. 250–286, Oct. 1993.

[27] O. Johansson, "Simple distributed δ +1 - coloring of graphs," *Information Processing Letters*, vol. 70, no. 5, pp. 229–232, 1999.

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

[29] G. Lu, B. Krishnamachari, and C. S. Raghavendra, "An adaptive energy-efficient and low-latency MAC for data gathering in sensor networks," submitted for publication.

[30] R. Rozovsky and P. R. Kumar, "SEEDEX: a MAC protocol for ad hoc neworks," in *International Symposium on Mobile ad hoc networking & computing*, 2001, pp. 67–75.