DESYNC: Self-Organizing Desynchronization and TDMA on Wireless Sensor Networks

Julius Degesys, Ian Rose, Ankit Patel, Radhika Nagpal
Division of Engineering and Applied Sciences, Harvard University
ssr@eecs.harvard.edu

ABSTRACT

Desynchronization is a novel primitive for sensor networks: it implies that nodes perfectly interleave periodic events to occur in a round-robin schedule. This primitive can be used to evenly distribute sampling burden in a group of nodes, schedule sleep cycles, or organize a collision-free TDMA schedule for transmitting wireless messages. Here we present DESYNC, a biologically-inspired self-maintaining algorithm for desynchronization in a single-hop network. We present (1) theoretical results showing convergence, (2) experimental results on TinyOS-based Telos sensor motes, and (3) a DESYNC-based TDMA protocol. DESYNC-TDMA addresses two weaknesses of traditional TDMA: it does not require a global clock and it automatically adjusts to the number of participating nodes, so that bandwidth is always fully utilized. Experimental results show a reduction in message loss under high contention from approximately 58% to less than 1%, as well as a 25% increase in throughput over the default Telos MAC protocol.

Categories and Subject Descriptors: C.2.1 [Computer-Communication Networks]: Network Architecture and Design — Distributed Networks

General Terms: Algorithms, Design, Performance

Keywords: Desynchronization, self-organizing, sensor networks, pulse-coupled oscillators, medium access control, time division multiple access, resource scheduling

1. INTRODUCTION

The spontaneous emergence of synchronization from simple rules—cardiac cells beating together or fireflies flashing in unison—has long provided inspiration to biologists and mathematicians [12]. Lately, it has also become an important primitive in wireless sensor networks; several groups have shown how simple models of natural synchronization can be used to design decentralized algorithms for time and event synchronization [5, 14, 2]. A benefit of the biologically-inspired approach is that simple, local node behaviors result in the whole network robustly maintaining synchronization despite individual faults or changes in topology.

Here we introduce a related primitive: *desynchronization*. Desynchronization is the logical opposite of synchronization; instead of nodes attempting to perform periodic tasks at the same time, nodes perform their tasks as far away as possible from all other nodes. Imagine not fireflies flashing in unison, but in a uniformly distributed, round-robin fashion.

Desynchronization is a useful primitive for periodic resource sharing and applies to many sensor network applications. Consider a set of nodes that sample a common geographic region. By desynchronizing their sampling schedules, the requirements of the monitoring task can be equitably distributed. Similarly, one can use desynchronization to organize sleep cycles such that nodes take turns being awake, and therefore, consume less energy. Another application is to use desynchronization to implement time division multiple access (TDMA), a well-known medium access control (MAC) protocol in which nodes use a round-robin schedule for sending messages. In TDMA, scheduled nodes do not have to contend for the shared medium nor worry about message collisions. It is especially attractive in many settings where nodes are transmitting streams of data or there are real-time constraints on message latency, as is common in wireless sensor networks [13, 4, 11].

In this paper we present DESYNC, a biologically-inspired algorithm for achieving desynchronization in a single-hop network. Given a set of n nodes that generate events periodically with a common, fixed period T, the nodes adjust such that the events are evenly distributed throughout the time period (i.e. they are spaced at intervals of T/n). The algorithm is simple, decentralized, and requires constant memory per node regardless of network size. Furthermore, if nodes are added or removed, the system self-adjusts to re-equalize the event intervals. Thus, DESYNC implements a self-maintaining desynchronization primitive.

We evaluate DESYNC along three fronts: theory, implementation, and application. First, we show convergence and an upper bound on the time required. Second, we implement DESYNC on TinyOS-based Telos sensor nodes (a.k.a. "motes"). Our experimental results on a 20-mote single-hop network confirm that the system rapidly achieves desynchronization and seamlessly accommodates the addition and removal of motes. Finally, we present an implementation of DESYNC-TDMA, a self-organizing TDMA MAC protocol designed for single-hop wireless networks.

DESYNC-TDMA has two novel features compared to traditional TDMA implementations: (1) it does not require a

IPSN'07, April 25-27, 2007, Cambridge, Massachusetts, USA.

global clock or other infrastructure overhead and (2) the schedule automatically self-adjusts to the number of participating nodes so as to fully utilize the bandwidth. Our experimental results show that DESYNC-TDMA achieves over 90% bandwidth utilization (a 25% increase from the default Telos MAC implementation) and less than 1% message loss in high traffic (down from 58%). It also significantly outperforms Z-MAC, a representative hybrid TDMA protocol.

The rest of the paper is as follows: Section 2 presents related work. Sections 3 and 4 introduce the DESYNC and DESYNC-TDMA algorithms along with theoretical results. Sections 5 and 6 present the experimental results along with a comparison to existing MAC protocols. Section 7 presents directions for future work, and we conclude in Section 8.

2. BACKGROUND AND RELATED WORK

2.1 Models of Synchronization in Biology

Many natural synchronizing systems, such as networks of neurons or swarms of fireflies, are modeled as networks of *pulse-coupled oscillators*, where each node in the network represents an adjustable oscillator that pulses at a fixed frequency. Each oscillator observes other oscillators' pulses (e.g. a neuron firing or a neighboring firefly's flash) and uses this information to adjust its own oscillator. Ultimately, all oscillators pulse synchronously.

In a seminal paper, Mirollo and Strogatz proved that a complete network of n pulse-coupled oscillators, using a simple oscillator-adjustment function, would always converge to synchrony, irrespective of the initial state [7, 12]. Recently, this biological model has been extended and shown to be able to achieve decentralized time synchronization and coordinated sensor control in wireless sensor networks [5, 14, 2]. One of the key benefits of this model is its ability to adapt—the system adjusts automatically to nodes entering and leaving the system, even though the individual nodes are only using very simple, local rules. Thus, synchronization in this model is self-maintaining.

In some natural systems, the goal is not synchronization, but *patterned* synchronization. For example, in animal locomotion, limbs can be modeled by individual oscillators that are coupled so as to produce different gaits [12]. Similarly, in the intestines, a series of oscillators can be coupled to produce a systolic wave. In these cases, the oscillators do not first synchronize and then negotiate a schedule for the pulse pattern. Instead, they use different adjustment rules to directly generate the desired pattern, with the advantage being that these adjustment rules are also self-maintaining.

In our case of desynchronization, we are interested in the pattern in which all of the oscillators pulse at evenly spaced intervals (the oscillators are completely out of phase). We use the Mirollo and Strogatz framework to design a simple oscillator adjustment rule that causes the system as a whole to converge to desynchrony. As with the original model, the system self-adjusts to maintain desynchronization; if new nodes are introduced, or current nodes removed, the system automatically converges to a new state where the new set of nodes has evenly spaced pulses. Protocols built on top of this primitive inherit the same self-maintaining property.

2.2 Channel Sharing in Wireless Networks

In wireless networks, nodes share the medium in which they transmit messages. It is the MAC protocol's responsibility to mediate their transmissions. Any of these protocols can usually be described as being either a contention-based protocol or a schedule-based protocol [3].

In contention-based, carrier sense multiple access (CSMA) protocols, nodes check the channel before transmitting, and if the channel is busy, they randomly back off for a short time and try again. This method is simple, adaptive, and frees nodes from having to maintain complex state about their environment. As a result, CSMA is often used when the expected contention is low (i.e. few nearby nodes transmitting) or when bursty traffic is expected.

In TDMA-based protocols, nodes use a round-robin schedule to transmit messages. Time is partitioned into fixed-size slots, and each node selects a time-slot during which it may regularly send messages collision-free. Since each node gets an equally sized slot, fairness is ensured. Message latency is bounded since nodes transmit at a fixed frequency.

TDMA is especially useful when nodes are transmitting streams of data, experience periods of high contention, have a high cost for message loss (e.g. energy cost of retransmissions), or require real-time constraints on message latency. These requirements are found in many sensor network applications due to their emphasis on periodic monitoring and local, event-triggered traffic [13, 4, 11]. As such, several TDMA protocols have been designed specifically for these settings [3]. However, almost all traditional TDMA implementations still encounter the following difficulties:

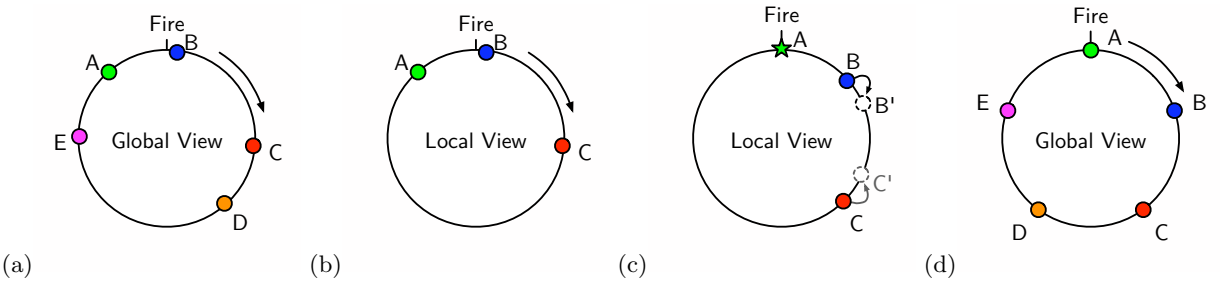
Overhead: Nodes must know when their slots begin and end, which usually requires accurate time synchronization among nodes and a negotiation of the slot schedule. The message overhead involved in maintaining these adds to the energy consumption and implementation complexity [6].

Wasted Slots: Nodes are assigned exclusive time slots. This means that slots go unused when nodes do not have data to send or have left the network. Thus, it is important for the network to be able to reclaim this lost bandwidth.

There is a large body of literature on modified TDMA protocols that attempts to address the second problem by either periodically renegotiating the schedule or by allowing nodes to contend for unused slots [3]. For example, TRAMA [9] periodically recomputes and reassigns slot schedules in order to utilize bandwidth and conserve energy. However, the protocol implementation is complex and assumes a global clock and application-level forecasting of traffic. Z-MAC, a hybrid protocol, focuses on recapturing wasted slots by allowing nodes to compete for all slots with a bias towards the owner of the slot. This method allows nodes to recapture unused bandwidth without having to renegotiate the slot schedule. However, this removes the collision-free guarantee on message transmission and often cannot fully recover the bandwidth. It also does not solve the problem of requiring time synchronization amongst communicating nodes.

In general, the complexity and cost of maintaining any TDMA schedule in the face of node and traffic changes can often outweigh the benefits of fairness, reliability, and high throughput. Hence, the default MAC protocols most used by sensor motes are CSMA protocols [8, 15].

Here we make a key observation regarding TDMA—there is no explicit need for nodes to agree upon a global time or to maintain information about each others' identities.



(a) (b) (c) (d)

Figure 1: Desync algorithm: (a) Global view of five nodes that are not yet desynchronized. (b) Node B's local neighborhood view. (c) When A fires, the node that fired immediately before it, node B, now knows both of its neighbors' positions—it heard C fire earlier and A just fired. Therefore, node B can now compute where it should have been if it were positioned ideally, B′, and jump towards it. However, C has since jumped to C′, unbeknownst to any other nodes. (d) The desynchronized state. All nodes are at the midpoints; thus, no node jumps and the system is stable.

Rather, TDMA only requires nodes to *desynchronize the timing of their transmissions*. If nodes could self-maintain desynchronization, then both weaknesses of TDMA would be addressed simultaneously. For example, if a node does not need to transmit, it can go to sleep and the remaining nodes will adjust to fully utilize the available bandwidth without message collisions.

3. ALGORITHMS

In this section, we first provide a description of the pulse-coupled oscillator framework, introduced by Mirollo and Strogatz [7]. We then use this framework to describe the DESYNC and DESYNC-TDMA algorithms.

3.1 Framework

Suppose there are n nodes that can communicate with each other (i.e. they are in a fully-connected network). Each node performs a task periodically with a period T. Thus, we can model each node as an oscillator with frequency $\omega = 1/T$. Let $\phi_i(t) \in [0,1]$ denote the phase of node i at time t where the phases 0 and 1 are identical and where $0 \leq i \leq n-1$. For example, if $\phi_i(t) = 0.75$, then node i is 75% of the way through its cycle. Upon reaching $\phi_i(t) = 1$, node i "fires" (or "pulses") indicating the termination of its cycle to the other nodes. Upon firing, the node resets its phase to $\phi_i(t^+) = 0$.

We can imagine the nodes as beads moving clockwise on a ring with period T (Figure 1). When a node reaches the top, it fires. All nodes observe this firing, and can use this information to jump forwards or backwards in phase. However, nodes are otherwise oblivious of the phases of other nodes; they can not observe the current state of the ring, only the firing events.

The goal is to have each node adjust the timing (phase) of its own firing such that eventually the network is *desynchronized*. For convenience, we can map phases into a set of delta-phase variable that represent the distance between oscillators on the ring. Let $\Delta_i(t) = \phi_i(t) - \phi_{i-1}(t)$. Thus, the desynchronized state can be defined as

$$\Delta_i^*(t) = \frac{1}{n} \tag{1}$$

for all $0 \leq i \leq n-1$. Visually, if the nodes are equally spaced around the ring, we have desynchronization.

3.2 DESYNC Algorithm

We now present a simple algorithm for achieving desynchronization among a set of n wireless sensor nodes in a single-hop network. We assume that a firing event corresponds to a node broadcasting a wireless *firing message* that all other nodes can hear.[1] Intuitively, the algorithm works as follows: each node adjusts its phase to be at the midpoint of the two nodes before and after it on the ring. In order to achieve this, a node must pay attention to the timing of the firings before and after its own. If each node can fire closer to the midpoint, then over successive periods this *jumping towards the average* will bring the system to a state in which all nodes are at the midpoints of their neighbors. This is exactly the desynchronized state.

In the algorithm, node i keeps track of the times of two events: the firing that occurs just before it fires (from node $i+1 \pmod n$) and the firing that occurs just afterwards (from node $i-1 \pmod n$). We call the senders of those firing messages the *phase neighbors* of node i. The firing times of the previous and next neighbors are recorded relative to node i's firing as $\tilde{\Delta}_{i+1}$ and $\tilde{\Delta}_i$, respectively. In this way, node i can approximate the phases of its previous and next phase neighbors as $\tilde{\phi}_{i+1}(t) = \phi_i(t) + \tilde{\Delta}_{i+1} \pmod 1$ and $\tilde{\phi}_{i-1}(t) = \phi_i(t) - \tilde{\Delta}_i \pmod 1$. Using this information, node i can then calculate the midpoint of its neighbors:

$$\tilde{\phi}_{\text{mid}}(t) = \frac{1}{2}\left[\tilde{\phi}_{i+1}(t) + \tilde{\phi}_{i-1}(t)\right] \pmod 1 \tag{2}$$

$$= \phi_i(t) + \frac{1}{2}\left(\tilde{\Delta}_{i+1} - \tilde{\Delta}_i\right) \pmod 1 \tag{3}$$

Note that in calculating Equation 2, the modular arithmetic used to compute the approximations of the phase neighbors should be delayed to the end to yield the appropriate midpoint. Once the midpoint is known, node i jumps towards it:

$$\phi_i'(t) = (1-\alpha)\phi_i(t) + \alpha\tilde{\phi}_{\text{mid}}(t) \tag{4}$$

where $\alpha \in [0,1]$ is a parameter that scales how far node i moves from its current phase towards the desired midpoint. Thus, after hearing both neighbors fire, node i instantaneously jumps from $\phi_i(t)$ to $\phi_i'(t)$. Note that if node i jumps immediately when node $i-1$ fires, the estimate is exact, i.e. $\tilde{\Delta}_i = \Delta_i(t)$. This adjustment is not apparent to other nodes

[1] In Section 5 we describe how we deal with message delays.

until node i fires again. Furthermore, node i's neighbors will also make adjustments without node i's knowledge.

To further illustrate this point, consider Figure 1(b) where there are three nodes: A, B, and C. First, C fired followed by B, and now, A is about to fire. In Figure 1(c), A fires; thus, B has enough information to make a jump. However, at this point, C too has heard both of its neighbors, B and D, and has already jumped to C'. Thus, by the time B makes its adjustment, it no longer knows the true distance between C and B. It is in this way that nodes continually make adjustments based on stale information. However, as shown in Section 4, this system will still converge to a desynchronized state.

This algorithm has several key features:

- **Convergence to Desynchrony**: Regardless of the initial state and number of nodes, the system converges to a state in which all nodes are evenly spread out with a spacing of T/n.
- **Simple Implementation**: Nodes only record the timing of two firing events and are not concerned with the identity of the senders nor how many firings occur in a given period. Therefore, nodes use *constant memory*, regardless of network size and do not need to maintain any internal state on network composition.
- **Self-Adapting**: If the number of nodes changes (a node is added or removed) then the system is no longer desynchronized. This local imbalance causes nodes closest to the disturbance to adjust their phases, eventually leading the system back to a stable, desynchronized state. Nodes do not need to explicitly monitor the network membership. Furthermore, single-node failures are similarly accounted for in the normal operation of the algorithm. Thus, the system ensures a fair sharing of the time period, T, even when the network size changes or nodes experience faults.

3.3 DESYNC-TDMA Algorithm

In this section, we describe how one can implement TDMA using DESYNC. As discussed in Section 2, TDMA-based protocols suffer from overhead and wasted slots. DESYNC allows us to design a simple low-memory TDMA protocol that automatically regulates slot sizes, fully utilizing bandwidth without incurring any collision costs.

We define node i's TDMA slot as beginning at the previously computed midpoint between node i and its previous phase neighbor. Likewise, it ends at the previously computed midpoint between node i and its next phase neighbor. Intuitively, nodes use earlier firings to compute the TDMA slots near the time of their next firing. Figure 2 illustrates this slot definition.

Defining slots in this manner also guarantee that a node will never fire outside its own slot. Note that if this were not the case, node i would be unable to send its firing message as the channel would be occupied by the current slot owner's transmissions. To see that this is the case, we will consider the local behavior for a set of nodes: A, B, and C (see Figure 2). First, we note that if node B doesn't jump, its firing will always contained by its own time slot:

$$\begin{aligned} \phi_A < \quad \phi_B \quad &< \phi_C \\ \phi_A + \phi_B < \quad 2\phi_B \quad &< \phi_B + \phi_C \\ \text{mid}(AB) < \quad \phi_B \quad &< \text{mid}(BC) \end{aligned} \tag{5}$$

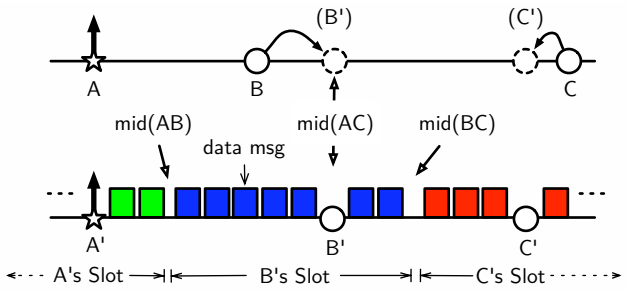


Figure 2: Desync-TDMA slots: Here, we have unravelled the ring into a line segment. The nodes in on the top line represent the current state in Figure 1(c) where node A is firing. The TDMA slots associated with the next set of firings are defined by the midpoints of the firings that occurred previously. Despite the information being old, if the nodes update according to Equation 4, firings will always occur during the nodes' TDMA slots.

Second, the target jump point of B' or $\text{mid}(AC) = (\phi_A + \phi_C)/2$, is also between $\text{mid}(AB)$ and $\text{mid}(BC)$:

$$\begin{aligned} \phi_B < \phi_C \qquad & \qquad \phi_A < \phi_B \\ \phi_A + \phi_B < \phi_A + \phi_C \qquad & \qquad \phi_A + \phi_C < \phi_B + \phi_C \\ \phi_A + \phi_B \quad < \quad \phi_A + \phi_C \quad & < \quad \phi_B + \phi_C \\ \text{mid}(AB) \quad < \quad \text{mid}(AC) \quad & < \quad \text{mid}(BC) \end{aligned} \tag{6}$$

This implies that if node B jumps towards the midpoint, it will always jump to a point that is within its time slot.

DESYNC-TDMA has the following characteristics:

The algorithm fully utilizes the channel regardless of the network's state of desynchronization. The algorithm always defines a set of non-overlapping slots that cover T, and nodes continually send collision-free data, even while they are desynchronizing. As the network approaches desynchronization, the slots converge to be of equal size. Thus, the system always provides *collision-free, fully-utilized bandwidth*, and constantly adjusts to increase fairness.

The TDMA schedule seamlessly adapts to nodes entering or leaving. When a node leaves, the neighboring nodes adjust their slot boundaries to fully utilize the bandwidth. The slot sizes equalize over time as the system approaches desynchronization, having the effect of leaving T fixed and increasing slot size. Thus, if a node does not need to transmit again for multiple periods, it can simply leave the protocol, sleep, and re-enter when it needs to send again. In the meantime, other nodes will have reaped the benefit of automatically acquiring the sleeping node's slot. We explore this experimentally in Section 6.

When a node enters the algorithm, it must first interrupt an existing node's data slot with its firing message (in Section 5, we show how this can be simply and reliably implemented). Here, the *costs of entering* for a node are the latency of one time period and the lost bandwidth that results from the one interrupted data slot. Other nodes remain oblivious to the entry and send data uninterrupted.

The algorithm is self-contained. Nodes do not need to know the network size or discover their neighbor IDs in order to create an initial schedule. The round-robin schedule order emerges as a result of the order in which nodes enter the process. Unlike other TDMA-style protocols, such as [10, 9], nodes do not need to agree on global time nor rely on a time synchronization protocol. While it is possible to write additional code to support each of these additional tasks (discovering neighbor IDs reliably, renegotiating schedule orders, electing leaders for global time consensus) this can add significant complexity to the implementation.

4. THEORETICAL RESULTS

THEOREM 1. *n nodes governed by* DESYNC *will always be driven to desynchrony (for $n < 500$).*

PROOF. In proving this, we will look solely at the delta-phase variables of the system and show that $\Delta_i = 1/n$ for all i. However, $\vec{\Delta}$ is insufficient to determine the next state of the system as the node performing the jump (node i) uses an approximation (or its memory) of Δ_{i+1}, namely $\tilde{\Delta}_{i+1}$. Thus, we add a dimension to the state of our system to store the memory, making the system state an $n+1$ dimensional vector, $\vec{x}$. We can then describe our system as a linear map, A, of dimension $n+1$ that is being repeatedly applied with each firing to $\vec{x}$. In total, our system is $A\vec{x} = \vec{x}'$, where

$$A = \begin{pmatrix} 0 & & & 1-\frac{\alpha}{2} & \frac{\alpha}{2} \\ 1 & 0 & & \frac{\alpha}{2} & -\frac{\alpha}{2} \\ & 1 & 0 & & \\ & & \ddots & \ddots & \\ & & & 1 & 0 \end{pmatrix}, \quad \vec{x} = \begin{pmatrix} \Delta_{i+1} \\ \Delta_{i+2} \\ \vdots \\ \Delta_{i-1} \\ \Delta_i \\ \tilde{\Delta}_{i+1} \end{pmatrix} \tag{7}$$

and $\alpha \in (0, 1)$ is the jump-size parameter, node $i-1$ is the node that is firing, and node i is jumping. Note that A not only performs the weighted average of Δ_i and $\tilde{\Delta}_{i+1}$, but it also stores Δ_i before the jump into the memory. Additionally, the first n elements are permuted in $\vec{x}$ to ready the next node to fire. Thus, A has zeros along the main diagonal, ones below it, and the weighted averaging portion in the top-right 2×2 sub-matrix. In terms of the state vector, $\vec{x}$, the next node to jump is the second-to-last element in $\vec{x}$.

It is easy to check that the desynchronized state $x_i^* = 1/n$ (for all $0 \le i \le n$) is a fixed point of the system (i.e. $\vec{x}^* = A\vec{x}^*$). Thus, $\vec{x}^*$ is an eigenvector of A with the eigenvalue $\lambda_0 = 1$. For matrices with the structure in Equation 7, the characteristic polynomial has a simple form:

$$\lambda^{n+1} - \frac{\alpha}{2}\lambda^2 - (1-\alpha)\lambda - \frac{\alpha}{2} = 0 \tag{8}$$

If we can show that all other eigenvalues of A lie strictly inside the unit circle in the complex plane (i.e. $|\lambda_l| < 1$ for $l > 0$), then we are guaranteed that $A^k\vec{x} \to \vec{x}^*$ as $k \to \infty$. Note that for $\alpha = 1$, $\lambda = -1$ is an eigenvalue. Thus, we must have $\alpha \in (0, 1)$.[2]

For purposes of a proof by contradiction, let us assume $\exists\lambda : |\lambda| > 1$. Bringing the negative terms across and taking the magnitude of both sides of Equation 8, we have

$$\left|\lambda^{n+1}\right| = \left|\frac{\alpha}{2}\lambda^2 + (1-\alpha)\lambda + \frac{\alpha}{2}\right| \le \left|\lambda^2\right|$$

contradicting our assumption. Thus, we have that $|\lambda| \le 1$. This implies that our system will either converge or settle to a limit cycle. For $\alpha \in (0, 1)$ and $n < 500$, direct computation of eigenvalues indicates absolute convergence. □

To conjecture a running time of $O(n^2)$, we observe that A^n mostly behaves like a random walk for large n. MATLAB simulation results confirm this (see Figure 3).

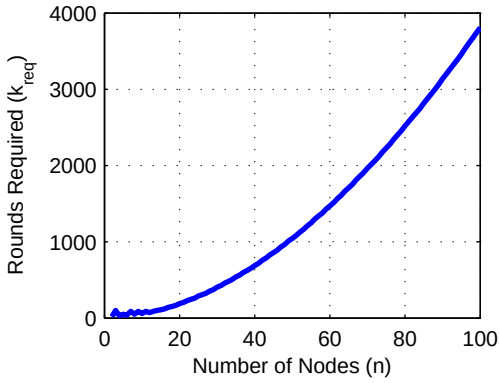


Figure 3: MATLAB simulation results on convergence rates showing an $O(n^2)$ running time: the number of rounds (n firings) required to reach an average desynchronization error of $\epsilon = 10\mu$sec for varying n with $\alpha = 0.95$. All simulations were started in a near-synchronized state and used A in Equation 7 for the updates.

5. IMPLEMENTATION

The DESYNC-TDMA algorithm was implemented on Telos wireless sensor motes running the TinyOS v1.1.14 operating system [1]. The 802.15.4-compliant motes use a 250 kbps, 2.4 GHz Chipcon CC2420 wireless transceiver. Radio messages all used TinyOS's standard, 35-byte active message format. Data packets used a 28-byte payload and included both sender IDs and message sequence numbers.

We used TinyOS's default CSMA radio interface to transmit messages, although the initial backoff when sending was reduced (to 1.2ms) since the DESYNC-TDMA algorithm should eliminate channel contention. We found in testing that sending streams of messages with 0 backoff between messages consistently led to the loss of every other message at the receiver. We concluded that under maximum send-speeds the receiver is unable to "keep up", although it is currently unclear exactly why this occurs. Experiments indicated that a delay of at least 1.2ms between messages was required to eliminate message loss due to this effect.

The basic implementation for DESYNC-TDMA is summarized in Figure 4. The motes use their local clock to keep track of their own firing time as well as the firing times of their two phase neighbors. Once a mote has received these firing times, it performs a simple update via Equation 4.

As shown by Maróti et al. [6], sending radio messages in TinyOS can result in non-deterministic delays on the order of several milliseconds before they are actually transmitted.

[2]Ideally, α should be selected depending on n; however we found that since n dominated the running time, only negligible gains could be made by tweaking α from 0.95.

```
init() {
  T = 1 second
  alpha = 0.95
  just_fired = False
  [ next_fire, prev_fire, last_fire ] = 0
  call SetFiringTimer(T)
}
on_firing_timer_expire() {
  call SendFiringMessage()
  just_fired = True
  my_fire = Now
  prev_fire = last_fire
  call SetFiringTimer(T)
}
on_receive_firing_message(msg_time) {
  last_fire = msg_time
  if (just_fired == True) {
    just_fired = False
    next_fire = msg_time
    slot_start = T + (prev_fire + my_fire)/2
    slot_end = T + (next_fire + my_fire)/2
    goal_time = T + (1 - alpha) * my_fire
                + alpha * (prev_fire + next_fire)/2

    call SetFiringTimer(goal_time - Now)
    call SetSlotStartTimer(slot_start - Now)
    call SetSlotEndTimer(slot_end - Now)
}}
```

Figure 4: Pseudocode for Desync-TDMA

To compensate, we use MAC-level time stamping, which inserts a delay value into the message to indicate how long the message was delayed. The receiving mote subtracts this delay value from the received time in order to better estimate the intended send time. Thus, motes can use their local clocks to estimate when their neighbors intended to fire, even though the message containing that information may be delayed. We reused the time stamping code developed for the FTSP [6] project.

When a new mote enters an existing DESYNC-TDMA network, it first sends a series of short interrupt messages before sending its initial firing message. The interrupt messages are meant to notify the current slot-owner that a new node needs to send a firing message, and thus, the slot owner should temporarily pause data transmissions to avoid message collisions with the upcoming firing message.

6. EVALUATION

In this section, we investigate the performance of DESYNC-TDMA. Note that these results also apply to DESYNC alone, since the algorithm for sending data does not affect the desynchronization process.

6.1 Experimental Setup

We constructed a single-hop network by placing 20 Telos motes around a single Tmote Sky mote designated as a base station. The base logged all messages transmitted by the other motes over its USB port to a PC. As the base station did not send any messages once the experiment started, it was able to observe the algorithm without affecting its performance. For all experiments, we used the same fixed parameters: $T = 1$ sec and $\alpha = 0.95$. We performed two classes of experiments:

- **Fixed-size:** Here we chose a fraction of the total motes ($n = 4, 10, 20$) to transmit data. The base station triggered the motes, at which point each mote picked a random offset into the first round to begin its periodic events. Motes used their entire slots to transmit data, simulating the effect of heavy traffic load where there is enough data to fully saturate the channel. Five trials of 60 secs each were run for each n-value. This experiment tested the ability of the system to provide TDMA-like performance under peak load conditions.
- **Node Removal and Addition:** In order to evaluate the effect of motes entering and leaving the system, we conducted an experiment where $n = 8$ initially. At $t = 135$ one mote stopped transmitting. At $t = 180$ three motes woke up and entered the system. As in the previous experiment, motes simulated heavy-traffic load by continuously transmitting data during their slots.

6.2 Evaluation Metrics

Here we define the metrics that we use to measure the performance of the system. Let N be the total network size (20 in all cases) and n be the number of currently transmitting motes ($n \leq N$).

- **Average desync error:** For ease of comparison across varying n, we define error to be the average deviation from the desired slot size (T/n) for a given round. Using the notation from Section 4, this metric is expressed as $\frac{1}{n}||\vec{\Delta}^{(k)} - \vec{\Delta}^*||_1$.
- **Normalized throughput:** In order to estimate the best possible data throughput one could achieve, we allowed a single mote to transmit messages uninterrupted (with optimized back-off delays) and measured the throughput received by the base. The maximum measured throughput was found to be 62.8 Kbps. For each experiment, we measure the total number of data messages received by the base during each round. We do not include the DESYNC firing messages in this measurement. Here we define *normalized throughput* as the ratio between the measured data message throughput and the maximum measured throughput of 62.8 Kbps. Thus, a value of 1 implies a fully-utilized channel.
- **Fairness:** We computed the average throughput per node over the course of each experiment. We report the max and min of these average node throughputs.
- **Message Loss:** All messages sent by a mote include the sender's ID and a message sequence number, allowing the base to detect missed messages. We use the base station logs to compute the ratio between total number of missed messages and total number of expected messages.

6.3 Experimental Results

Figure 5 shows a single run of a fixed-size experiment for $n = 10$ motes. Figure 5(a) plots the times of each mote's firing events relative to those of a single mote. As can be seen, the motes quickly and smoothly achieved desynchronization. Figure 5(b) shows how the different performance

Nodes:	4	10	20
Total Throughput (Kbps):	**60.8**	**57.9**	**53.0**
(normalized, %)	(96.8)	(92.2)	(84.3)
Max Individual (Kbps):	15.2	5.8	2.8
Min Individual (Kbps):	15.2	5.6	2.4
Message Loss (%):	**0.2**	**0.3**	**0.5**

Table 1: Desync-TDMA's performance for different n over 60-second runs. In our experiments, the maximum rate at which a single node could transmit was 62.8 Kbps.

metrics changed over time. The average desync error decreased exponentially with time, reaching an error of less than 1 ms within 18 rounds (note that the desired slot size is $T/n = 100$ ms). However, the total normalized throughput was high ($\sim$ 92%, or 57.8 kbps) and roughly constant throughout the experiment, regardless of the desynchronization error. Message loss at the base station was $< 0.5\%$.

Figure 6 shows how the convergence rate scaled with n; the average desync error is plotted as an average over 5 runs. The systems reached convergence within 1 ms in 8, 20, and 48 rounds for 4, 10, and 20 nodes, respectively. As predicted, the average desync error decreased exponentially with a rate approximately proportional to n^2.

Table 1 shows the DESYNC-TDMA throughput and message loss. As n increased there were more firing messages sent per round, leading to a linear decay in the total throughput of approximately 0.8% (or 0.5 Kbps) per node. Extrapolating from these results suggests that for a fixed period of $T = 1$ sec, our system can support an approximate maximum of 125 motes before the bandwidth is completely consumed by firing messages. Although we have attempted to minimize the footprint of firing messages, a number of optimizations, such as piggy-backing firing messages on data messages, were not explored and could further increase system scalability. The table also shows that message loss for all numbers of motes was near zero. In addition to increasing throughput, this has the benefit of reducing latency and the number of re-trasmissions required. Additionally, the minimum and maximum per mote throughputs are very close, showing that the system is able to fairly distribute bandwidth. From these results, we conclude that DESYNC-TDMA is able to provide collision-free, high bandwidth utilization independent of the number of transmitting motes.

Figure 7 shows how the average desync error and normalized throughput varied during the mote removal and addition experiment. At $t = 135$, when a mote stopped transmitting, the resulting imbalance in slot-sizes caused a jump in error of $\sim$ 25 ms, which decayed to less than 1 ms over the next 8 rounds. Likewise, an error of $\sim$ 40 ms was introduced when three new motes began transmitting at $t = 180$, requiring 19 rounds to reduce to 1 ms. The total throughput was slightly impacted at each event, suffering two-round throughput losses of 12.5% and 10.6%, respectively, during the removal and addition events. Within two rounds, the total throughput had returned to normal capacity.

Overall, this experiment shows that the cost of entry and exit for a mote is low and the system is able to adapt quickly to recover bandwidth and re-equalize slot sizes. The main costs are the single round latency that an entering mote must wait before transmitting data and a temporary drop in fairness as the slot sizes are re-equalized.

From these results we can conclude general trends: for DESYNC, the average desync error decreases exponentially with time and proportional to $O(n^2)$. This is in line with our theoretical results from Section 4. For DESYNC-TDMA, the bandwidth utilization is consistent, and message loss is near zero, regardless of the state of desynchronization and number of transmitting motes. Thus, nodes can easily enter and leave with a limited impact on total throughput.

6.4 Comparison to other MAC protocols

We next compare the performance of DESYNC-TDMA to other MAC protocols. As a reminder, we define N to be the network size and n be the number of participating motes.

- **Ideal TDMA:** Ideally, TDMA would utilize all bandwidth available and provide collision-free slots of exactly T/n size, without paying any price for renegotiating slot size or slot ownership or maintaining a common clock. This is not realizable but provides an upper bound on the attainable performance.[3]
- **Fixed TDMA:** In this variant of TDMA, the slot sizes are fixed at T/N. If a mote does not need to transmit during its assigned slot, then that bandwidth is wasted. This is simple to implement (given a global clock); however, it performs poorly since it only uses n/N of the available bandwidth. We do not implement this but it provides a lower bound on what TDMA should achieve.[3]
- **Hybrid TDMA**: Hybrid TDMA protocols modify the Fixed TDMA scheme so that unowned slots can be used by other motes. We use Z-MAC as a representative protocol of this group [10]. Here, motes use the platform's CSMA default protocol to contend for the unused slots. This is implemented by giving a mote a shorter back-off period during its own slot than during other slots.[4]
- **CSMA:** In CSMA protocols, a mote checks the channel before transmitting. If the channel is busy, then the mote backs off a random amount before checking the channel again. Message collision (and loss) occurs when motes check a free channel simultaneously and decide to transmit, or if the channel check is noisy. CSMA is a simple and adaptive protocol that works well for small numbers of motes and variable traffic, but experiences large backoffs and message loss under high load. For our experiments, we used the default CSMA (with initial and congestion back-offs selected randomly from [0.3–4.9] ms and [0.3–19.6] ms, respectively) as provided by TinyOS for the Telos platform.

[3]The Ideal and Fixed TDMA protocols assume a send backoff of 1.2ms to avoid the receiver problems mentioned in Section 5.

[4]At the time of writing, no Z-MAC implementations were known of for CC2420-based motes. In our implementation, we used a fixed slot size of 50 ms, a fixed frame size of 20, and set the backoff times to 1.3-3.2 ms and 3.2-12.8 ms for the owner and non-owners, respectively. For simplicity, and due to the experiments' short durations (60 secs), nodes were time-synchronized at the start of each experiment by a message from the base station but did not send any synchronization messages during the experiment.

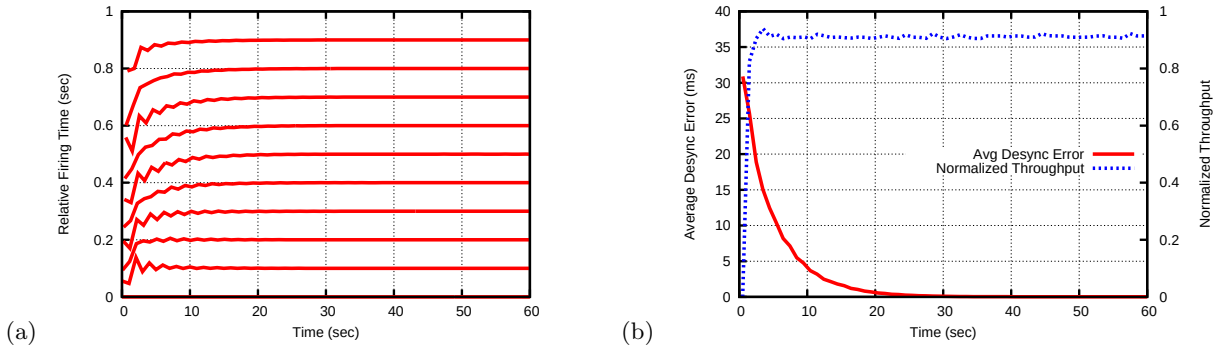


Figure 5: **Desynchronization on $n = 10$ sensor motes. (a) The firing times during each round are plotted relative to an arbitrarily chosen mote. The graphs show that the firing times stabilize to be evenly spaced and that the initial ordering of firing times is preserved throughout. (b) The average desync error and total throughput are plotted over time. Desync error decreases over time, but the total throughput remains high and roughly constant regardless.**

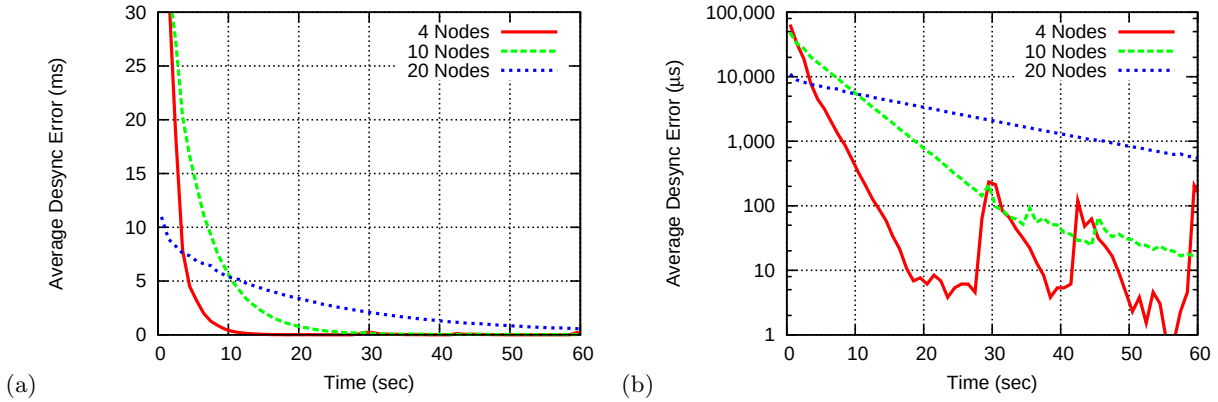


Figure 6: **Desynchronization error for different n, plotted using (a) normal and (b) log scales. For 4 motes, the system converges within 8 rounds to 1ms accuracy of an even spacing of 250 ms. The time taken to achieve similar accuracy increases with network size. The two "bumps" of ~ 100 μs error in the 4-node average in (b) were due to time-stamping failures which led to slightly inaccurate predictions of the send-times of two firing messages.**

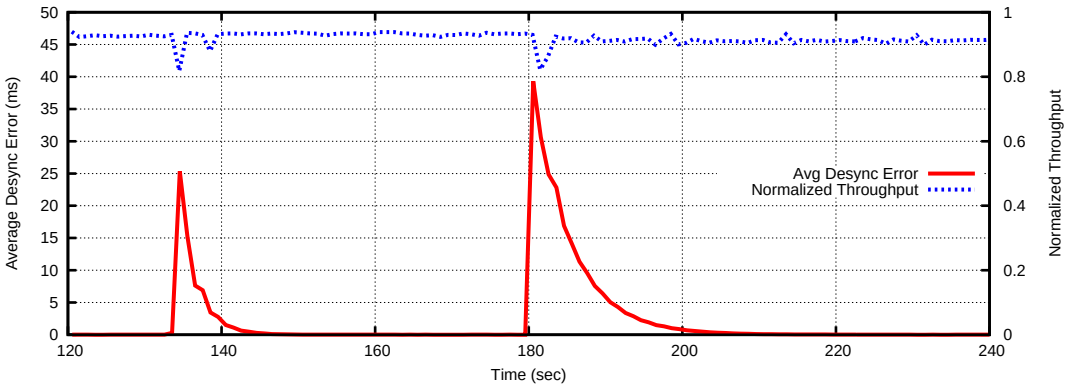


Figure 7: **Motes Arrival and Departure experiments. 8 motes were started at $t = 0$. At time $t = 135$ one mote left the system and stopped transmitting. At time $t = 180$ three motes woke up and entered the system. The system minimizes loss of throughput and rapidly re-equalizes slot sizes after addition or removal of motes.**

	Ideal TDMA			DESYNC-TDMA			Z-MAC			Telos-CSMA		
Nodes (n):	4	10	20	4	10	20	4	10	20	4	10	20
Throughput (Kbps):	**62.8**	**62.8**	**62.8**	**60.8**	**57.9**	**53.0**	**36.3**	**41.0**	**40.4**	**44.2**	**46.1**	**35.7**
Max Throughput/node:	15.7	6.3	3.1	15.2	5.8	2.8	10.0	5.1	2.8	11.6	4.4	5.1
Min Throughput/node:	15.7	6.3	3.1	15.2	5.6	2.4	8.0	2.9	1.4	9.7	3.5	0.4
Message Loss (%):	**0.0**	**0.0**	**0.0**	**0.3**	**0.2**	**0.2**	**15.4**	**32.4**	**50.7**	**35.9**	**57.1**	**87.1**

Table 2: Throughput and message loss characteristics across different protocols. In our experiments, the maximum rate at which a single node could send data was 62.8 Kbps.

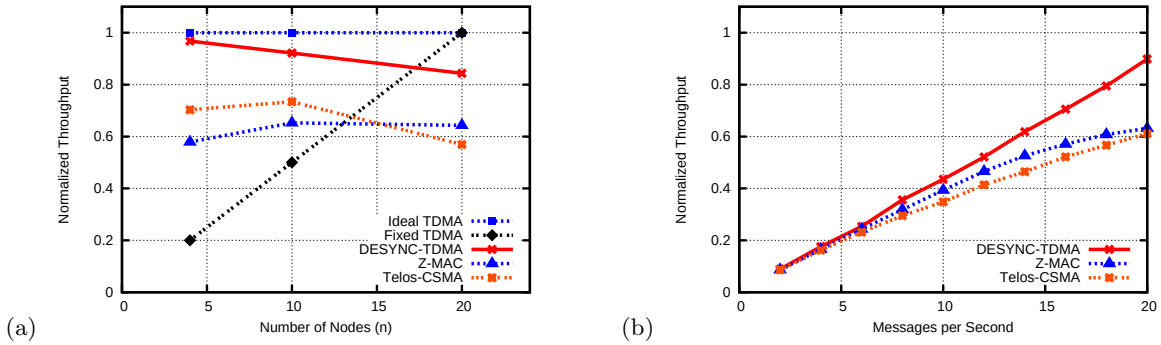


Figure 8: (a) High data rate: Normalized throughput vs n, for different protocols where each node attempted to send as much as possible. A normalized throughput of 1 represents a throughput of 62.8 Kbps. (b) Fixed data rate: the achieved total throughput on a network of 10 nodes across varying data rates. Note that even for low data rates of 1-2 messages per second, Desync-TDMA still matches or outperforms the other protocols.

6.4.1 *High Data-Rate*

We compared the performance and scalability of different protocols using the *fixed-size* scenario from Section 6.1. Table 2 shows our experimental results, and Figure 8(a) plots the normalized throughput relative to the number of transmitting motes. DESYNC-TDMA provided excellent bandwidth utilization, with a linear decay of approximately 0.8% per mote. In addition, fairness was quite high for all n, and message loss was near zero. In contrast, Telos-CSMA achieved a much lower bandwidth utilization. This was due to bandwidth wasted in backoffs as well as high message loss. We believe that the message loss was due in part to the listener saturation problem mentioned in Section 5.

For $n = 4$, our Z-MAC implementation achieved lower throughput than Telos-CSMA, as expected. In this case, most slots are not owned by any mote, so the behavior is analogous to CSMA, except that non-owner backoffs are larger than regular CSMA backoffs, thus reducing bandwidth. However, the remaining Z-MAC results were not as expected. As n increases the performance should approach that of Fixed-TDMA. Instead, at $n = 20$, we observed that throughput roughly remained constant and message loss increased substantially from $n = 10$. We suspect that Z-MAC is suffering from the same difficulties as Telos-CSMA.

6.4.2 *Low/Medium Periodic Data-Rates*

In many monitoring applications, nodes produce periodic sample data where the data rate is much lower than channel capacity. Given this, the question is whether there is still an advantage to using other protocols over CSMA. We conducted a second experiment with 10 transmitting motes, varying the periodic data rate from 4 message/sec to 20 messages/sec (which corresponds to the *fixed-size scenario*). Each experiment was run twice for 60 sec each with the averaged results plotted in Figure 8(b). DESYNC-TDMA, Z-MAC, and Telos-CSMA all performed similarly for data rates under 6 messages/sec. However, as the data rate increased, DESYNC-TDMA provided appropriate throughput, whereas the bandwidth utilization of the other protocols was suboptimal. This suggests that even for low data rates, DESYNC-TDMA can still be a useful protocol.

6.5 Summary Discussion

DESYNC-TDMA is a fundamentally new way of thinking about TDMA scheduling. Without explicit scheduling or time synchronization, DESYNC-TDMA is able to provide excellent total throughput and collision-free transmission under high loads, regardless of the state of desynchronization. Once desynchronized, it guarantees fairness and predictable (stream-like) message latencies. When nodes enter or leave, the system self-adjusts to accommodate the new nodes or to recapture the unused slots. Furthermore, unlike hybrid-TDMA methods, no contention is required for recapture.

However, DESYNC-TDMA also has some limitations and may not be appropriate for all types of traffic. One important limitation is that a node pays a "cost" when entering the system: (a) 1-round latency before being able to transmit data and (b) a smaller slot size for several rounds until the system re-converges to the desynchronized state. A second limitation is that DESYNC-TDMA provides nodes with equal slots which, although guaranteeing fairness, can also lead to inefficient bandwidth usage. If a node does not have enough data to fully utilize its slot, then the unused bandwidth is wasted. In hybrid protocols such as Z-MAC, nodes

can recover part of that bandwidth using CSMA contention, but DESYNC-TDMA does not allow this. In the future, we plan to extend the algorithm to provide variable slot sizes that can reflect each node's desired bandwidth.

Another potential issue is lossy radio links. Message loss can occur due to reasons other than collisions, and as a result, firing messages may occasionally be lost. Although DESYNC-TDMA is self-correcting (a missed message can be seen as analogous to a node leaving and then re-entering the system one round later), missed firing messages will still result in a temporary loss in desync accuracy. DESYNC-TDMA's α parameter affects how strongly nodes react to missed messages; larger values provide faster convergence at the cost of greater sensitivity to dropped messages. We have considered mechanisms by which nodes can dynamically vary α based on recent network conditions, but a complete evaluation is left to future work.

DESYNC-TDMA is not explicitly designed to achieve energy efficiency by scheduling time slots for receiving nodes to sleep, as is done in TRAMA. Instead, DESYNC-TDMA can achieve energy efficiency by utilizing asynchronous low power listening [8] as is done by Z-MAC. It has been shown that this can outperform duty cycling and does not require application-level forecasting of future senders and receivers.

7. FUTURE WORK

There are several avenues of future work, including coordinated sleep schedules and quality-of-service guarantees. However, a critical next step is extending DESYNC-TDMA to multi-hop networks.

Determining a slot schedule in multi-hop topologies is a much more complex problem for two reasons: nodes belong to intersecting and multiple-sized neighborhoods and overlapping broadcast regions create hidden terminals. A standard technique used in solving this problem is to color a constraint graph in which all two-hop neighborhoods in the communication graph are fully connected. Assigning each node in the graph its own color is equivalent to a global desynchronization, whereas minimal coloring constructs the fairest distribution of time amongst the nodes.

DESYNC-TDMA, however, allows for variable-sized slots. Thus, larger-sized slots can be given to nodes in less-dense areas of the graph. In this setting, DESYNC-TDMA could provide a simple algorithm for self-organizing multi-hop TDMA and automatically adjusting slot sizes as the traffic patterns change. It also provides a different way of looking at the multi-hop slot-assignment problem.

Our preliminary simulations suggest that DESYNC-TDMA converges on multi-hop topologies and produces a slot size comparable to T/c (where c is the chromatic number of the node's 2-hop neighborhood subgraph). However, proving that the algorithm converges on all multi-hop topologies and predicting the slot size and convergence times is currently an open question.

8. CONCLUSION

In this paper, we introduced a new primitive, *desynchronization*, and provided a self-organizing desynchronization algorithm for single-hop networks. Our theoretical results show that convergence to desynchronization is guaranteed for $n < 500$ in time $O(n^2)$. As an application of DESYNC, we presented DESYNC-TDMA, a self-organizing TDMA protocol. However, we note that each is a useful algorithm in its own right. DESYNC provides the ability to space out events in time, whereas DESYNC-TDMA constructs a method to share a medium fairly by simply having the events correspond to changes in ownership over the medium.

We implemented these algorithms on TinyOS motes, showing their simplicity and benefits. Experimental results on a network of 10 motes showed a reduction in message loss from 57.1% (Telos-CSMA) to less than 1% with a 25% increase in throughput. A unique feature of this approach to TDMA is that it does not rely on any external global infrastructure and self-adjusts to fully utilize the bandwidth.

Acknowledgements

The authors would like thank Matt Welsh for loaned equipment and advice. This work was supported by the NSF.

9. REFERENCES

[1] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System architecture directions for networked sensors. In *ASPLOS*, Nov 2000.

[2] Y. Hong and A. Scaglione. A scalable synchronization protocol for large scale sensor networks and its applications. In *IEEE Journal on Selected Areas in Communication*, Nov 2003.

[3] K. Langendoen and G. Halkes. *Energy-Efficient Medium Access Control (chapter in Embedded System Handbook).* CRC Press, Aug 2005.

[4] K. Lorincz, D. Malan, T. Fulford-Jones, A. Nawoj, A. Clavel, V. Shnayder, G. Mainland, S. Moulton, and M. Welsh. Sensor networks for emergency response: Challenges and opportunities. *IEEE Pervasive Computing*, Dec 2004.

[5] D. Lucarelli and I. Wang. Decentralized synchronization protocols with nearest neighbor communication. In *SenSys*, 2004.

[6] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi. The Flooding Time Synchronization Protocol. In *SenSys*, 2004.

[7] R. Mirollo and S. Strogatz. Synchronization of pulse-coupled biological oscillators. *SIAM Journal of Applied Math*, 50(6):1645–62, Dec 1990.

[8] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *SenSys*, 2004.

[9] V. Rajendran, K. Obraczka, and J. J. Garcia-Luna-Aceves. Energy-efficient collision-free medium access control for wireless sensor networks. In *SenSys*, 2003.

[10] I. Rhee, A. Warrier, M. Aia, and J. Min. Z-MAC: A Hybrid MAC for Wireless Sensor Networks. In *SenSys*, 2005.

[11] G. Simon, M. Maróti, A. Lédeczi, G. Balogh, V. Kusy, A. Nádas, G. Pap, J. Sallai, and K. Frampton. Sensor network-based countersniper system. In *SenSys*, 2004.

[12] S. Strogatz. *Sync: The Emerging Science of Spontaneous Order*. Hyperion, New York, NY, 2003.

[13] G. Werner-Allen, J. Johnson, M. Ruiz, J. Lees, and M. Welsh. Monitoring volcanic eruptions with a wireless sensor network. In *Proc. European Workshop on Wireless Sensor Networks (EWSN)*, Jan 2005.

[14] G. Werner-Allen, G. Tewari, A. Patel, R. Nagpal, and M. Welsh. Firefly-Inspired Sensor Network Synchronicity with Realistic Radio Effects. In *SenSys*, 2005.

[15] W. Ye, J. Heidemann, and D. Estrin. An Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *INFOCOM*, 2002.