Firefly-Inspired Sensor Network Synchronicity with Realistic Radio Effects

Geoffrey Werner-Allen, Geetika Tewari, Ankit Patel, Matt Welsh, Radhika Nagpal
Division of Engineering and Applied Sciences
Harvard University
{werner,gtewari,abpatel,mdw,rad}@eecs.harvard.edu

ABSTRACT

Synchronicity is a useful abstraction in many sensor network applications. Communication scheduling, coordinated duty cycling, and time synchronization can make use of a synchronicity primitive that achieves a tight alignment of individual nodes' firing phases. In this paper we present the *Reachback Firefly Algorithm (RFA)*, a decentralized synchronicity algorithm implemented on TinyOS-based motes. Our algorithm is based on a mathematical model that describes how fireflies and neurons spontaneously synchronize. Previous work has assumed idealized nodes and not considered realistic effects of sensor network communication, such as message delays and loss. Our algorithm accounts for these effects by allowing nodes to use delayed information from the past to adjust the future firing phase. We present an evaluation of RFA that proceeds on three fronts. First, we prove the convergence of our algorithm in simple cases and predict the effect of parameter choices. Second, we leverage the TinyOS simulator to investigate the effects of varying parameter choice and network topology. Finally, we present results obtained on an indoor sensor network testbed demonstrating that our algorithm can synchronize sensor network devices to within 100 μsec on a real multi-hop topology with links of varying quality.

Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]: Network Architecture and Design, Distributed Systems

General Terms

Algorithms, Design, Experimentation, Theory

Keywords

Synchronization, Wireless Sensor Networks, Biologically Inspired Algorithms, Pulse-Coupled Oscillators

SenSys'05, November 2–4, 2005, San Diego, California, USA.

1. INTRODUCTION

Computer scientists have often looked to nature for inspiration. Researchers studying distributed systems have long envied, and attempted to duplicate, the fault-tolerance and decentralized control achieved in natural systems. Those of us studying sensor networks also have every reason to be envious. Designing software coordinating the output of a collection of limited devices frequently feels as frustrating as orchestrating the activity of a colony of stubborn ants, or guiding a school of uncooperative fish. And yet ant colonies complete difficult tasks, schools of fish navigate the sea, and swarms of fireflies stretching for miles can pulse in perfect unison, all without centralized control or perfect individuals. The spontaneous emergence of synchronicity — for example, fireflies flashing in unison or cardiac cells firing in synchrony — has long attracted the attention of biologists, mathematicians and computer scientists.

Synchronicity is a powerful primitive for sensor networks. We define synchronicity as the ability to organize *simultaneous collective action* across a sensor network. Synchronicity is not the same as time synchronization: the latter implies that nodes share a common notion of time that can be mapped back onto a real-world clock, while the former only requires that nodes agree on a firing period and phase. The two primitives are complementary: nodes with access to a common time base can schedule collective action in the future, and conversely, nodes that can arrange collective action can establish a meaningful network-wide time base. However, the two primitives are also independently useful. For example, nodes within a sensor network may want to compare the times at which they detected some event. This task requires a notion of global time, however it does not require real-time coordination of actions.

Similarly, synchronicity by itself can be extremely useful as a sensor network coordination primitive. A commonly-used mechanism for limiting energy use is to carefully schedule node duty cycles so that all nodes in a network (or a portion of the network) will wake up at the same time, sample their sensors, and relay data along a routing path to the base station. Coordinated communication scheduling has been used both at the MAC level [18] and in multi-hop routing protocols [12] to save energy. Synchronicity can also be used to coordinate sampling across multiple nodes in a network, which is especially important in applications with high data rates. Previous work on seismic analysis of structures [1], shooter localization [13], and volcanic monitoring [15] could use such a primitive and avoid the overhead of maintaining consensus on global time until absolutely necessary.

In this paper, we present a biologically-inspired distributed synchronicity algorithm implemented on TinyOS motes. This algorithm is based on a mathematical model originally proposed by Mirollo and Strogatz to explain how neurons and fireflies spontaneously synchronize [10]. This seminal work proved that a very simple reactive node behavior would always converge to produce global synchronicity, irrespective of the number of nodes and starting times. Recently Lucarelli and Wang [8] demonstrated that this result also holds for multi-hop topologies, an important contribution towards making the model feasible for sensor networks.

The firefly-inspired synchronization described by Mirollo and Strogatz has several salient features that make it attractive for sensor networks. Nodes execute very simple computations and interactions, and maintain no internal state regarding neighbors or network topology. As a result, the algorithm robustly adapts to changes such as the loss and addition of nodes and links [8]. The synchronicity provably emerges in a completely decentralized manner, without any explicit leaders and irrespective of the starting state.

However, implementing this approach on wireless sensor networks still presents significant obstacles. In particular, the previous theoretical work assumes *instantaneous* communication between nodes. In real sensor networks, radio contention and processing latency lead to significant and unpredictable communication latencies. Earlier work also assumes non-lossy radio links, identical oscillator frequencies, and arbitrary-precision floating-point arithmetic which are unrealistic in current sensor networks.

We present the *reachback firefly algorithm* (RFA) that accounts for communication latencies, by modifying the original firefly model to allow nodes to use information from the past to adjust the future firing phase. We evaluate our algorithm in three ways: theory, simulation and implementation. We present theoretical results to prove the convergence of our algorithm in simple cases and predict the impact of parameter choice. Next we leverage TOSSIM, the TinyOS simulator, to explore the behavior of the algorithm over a range of parameter values, varying numbers of nodes, and different communication topologies. These simulation results validate the theoretical predictions. Finally, we present results from experiments on a real sensor network testbed. These results demonstrate that our algorithm is robust in the face of real radio effects and node limitations. Our results show that such a decentralized approach can provide synchronicity to within 100 μsec on a complex multiple-hop network with asymmetric and lossy links. To the best of our knowledge, this work represents the first implementation of firefly-inspired synchronicity on the MicaZ mote hardware, and demonstrates the ability of the model to achieve synchronicity given real radio and hardware limitations.

Our paper is organized as follows. Section 2 presents related work. In Section 3 we present RFA in the context of the Mirollo and Strogatz model and describe current hardware and radio limitations. Sections 4-7 present our metrics and theoretical, simulation and experimental results. We conclude with future work.

2. BACKGROUND AND MOTIVATION

Time synchronization has received a great deal of attention in the sensor network community. The problem of establishing a consistent global timebase across a large network, despite message loss and delays, node failures, and local clock skew, has proven to be very difficult. As described in the introduction, our goal is not time synchronization, but rather *synchronicity:* the ability for all nodes in the network to agree on a common period and phase for firing pulses. Synchronicity can be used to implement time synchronization, although this requires mapping the local firing cycle to a global clock, which we leave for future work.

2.1 Time Synchronization

A number of protocols have been proposed that allow wireless sensor nodes to agree on a common global timebase. Here we briefly describe some of the protocols. In Receiver Based Synchronization (RBS) [2] a reference node broadcasts a message and multiple receivers within radio range can then agree on a common time base by exchanging the local clock times at which they received the message. This protocol avoids the uncertainty of transmission delays by using a single radio message to simultaneously synchronize multiple receiver nodes, however it does not apply in multi-hop networks. The TPSN [3] protocol works on multi-hop networks by constructing a spanning tree and then using hop-by-hop synchronization along the edges to synchronize all nodes to the root. They also introduce MAC level timestamping to estimate transmission delay. The FTSP protocol [9], simplifies the process of multi-hop synchronization by using periodic floods from an elected root, rather than maintaining a spanning tree. In the case of root failure, the system elects a new root node. FTSP also refines the timestamping process to within microsecond accuracy and provides a method for estimating clock drift which reduces the need to synchronize frequently.

Direct comparison of these protocols in terms of synchronization error is difficult, due to the differences in hardware and evaluation methodology. FTSP reports a per-hop synchronization error of about 1 μsec, although the maximum pairwise error is over 65 μsec in their testbed. The mean single-hop synchronization error reported for TPSN is 16.9 μsec, compared to 29.1 μsec for RBS [3]. The dynamics of these protocols in terms of robustness to topology changes and node population have not been widely studied.

2.2 Biologically-Inspired Synchronicity

Synchronicity has been observed in large biological swarms where individuals follow simple coordination strategies. The canonical example is the synchrony of fireflies observed in certain parts of southeast Asia [10]. The behavior of these systems can be modeled as a network of *pulse-coupled oscillators* where each node is an oscillator that periodically emits a self-generated pulse. Upon observing other oscillators' pulses, a node adjusts the phase of its own oscillator slightly. This simple feedback process results in the nodes tightly aligning their phases and achieving synchronicity.

Peskin first introduced this model in the context of cardiac pacemaker cells[11]. Mirollo and Strogatz [10] provide one of the earliest complete analytical studies of pulse-coupled oscillator systems. They proved that a fully-connected (all-to-all) network of N identical pulse-coupled oscillators would synchronize, for any N and any initial starting times. Recent work by Lucarelli and Wang [8] relaxes the all-to-all communication assumption. Drawing from recent results in multi-agent control, they derive a stability result based on nearest neighbor coupling and show convergence in simulation for static and time varying topologies. Their work

demonstrates that the same simple feedback process works, even when nodes only observe nearest neighbors and those neighbors may change over time.

Several groups have proposed using pulse-coupled synchronicity to solve various network problems. Hong and Scaglione [6, 5] introduce an adaptive distributed time synchronization method for fully-connected Ultra Wideband (UWB) networks. They use this as a basis for change detection consensus. Wakamiya and Murata [14] propose a scheme for data fusion in sensor networks where information collected by sensors is periodically propagated without any centralized control from the edge of a sensor network to a base station, using pulse-coupled synchronicity. Wokoma et al. [17] propose a weakly coupled adaptive gossip protocol for active networks. Each of these applications clearly demonstrates the utility of synchronicity as a primitive. However much of the prior work is evaluated only in simulation and does not consider real communication delay or loss.

Wireless radios exhibit non-negligible and unpredictable delays due to channel coding, bit serialization, and (most importantly) backoff at the MAC layer [3, 9]. In traditional CSMA MAC schemes, a transmitter will delay a random interval before initiating transmission once the channel is clear. Additional random (typically exponential) backoffs are incurred during channel contention. On the receiving end, jitter caused by interrupt overhead and packet deserialization leads to additional unpredictable delays. Radio contention deeply impacts the firefly model. Multiple nodes attempting to fire simultaneously will be unable to do so by the very nature of the CSMA algorithm. As nodes achieve tighter synchronicity, contention will become increasingly worse as many nodes attempt to transmit simultaneously. The goal of this paper is to address the limitations of current communication assumptions and realize a real implementation of firefly-inspired synchronicity in sensor networks.

3. FIREFLY-INSPIRED SYNCHRONICITY

In this section, we first describe the Mirollo and Strogatz model and discuss how the theoretical model differs from practice. Then we present our modified algorithm, which takes these differences into account.

3.1 Mirollo and Strogatz Model

In the Mirollo and Strogatz (M&S) model, a node acts as an oscillator with a fixed time period T. Each node has an internal time or phase t, which starts at zero and increments at a constant rate until $t = T$. At this point the node "fires" (in the case of firefly, flashes) and resets $t = 0$. Nodes may start at different times, therefore their internal time (phase) t is not synchronized.

In the absence of any input from neighbors, a node B simply fires whenever $t = T$. If B observes a neighbor firing, then B reacts by adjusting its phase forward, thus shortening its own time to fire (Figure 1(a,b)).

The amount of adjustment is determined by the function $f(t)$, which is called the *firing function*, and the parameter ϵ, which is a small constant < 1. Suppose node B observes a neighbor fire at $t = t'$. In response, node B *instantaneously jumps* to a new internal time $t = t''$, where

$$t'' = f^{-1}(f(t') + \epsilon) \quad (1)$$

However if $t'' > T$, then $t = T$ and the node immediately fires and resets $t = 0$. In a biological sense, $f(t)$ can be thought of as the charge of a capacitor within the neuron or firefly, which receives a boost of ϵ whenever a firing event is observed. Algorithmically, the effect is that a node instantaneously increments its phase by $\Delta(t') = (t'' - t')$, when it observes a firing event at $t = t'$.

The seminal result by Mirollo and Strogatz is that if the function f is smooth, monotonically increasing, and concave down, then a set of n nodes will always converge to the same phase (i.e achieve synchronicity), for any n and any initial starting times [10]. The simple requirements on f ensure that a node reacts more strongly to events that occur later in its time period. One of the limitations of their proof was that it only held for the case where all n nodes could observe each others' firing (all-to-all topology). Recently Lucarelli and Wang [8] relaxed this condition and proved that this simple node behavior also results in synchrony in multi-hop topologies, a prerequisite for use in sensor networks.

3.2 From Theory to Practice

The M&S model has several salient features. The node algorithm and the communication are very simple. A node only needs to observe firing events from neighbors — there is no strength associated with the event or even a need to know which neighbor reported the event. Individual nodes have no state other than their internal time. Synchronicity provably emerges without any explicit leaders and irrespective of the starting state.

Because of these reasons, the model is particularly attractive as an algorithm for sensor networks. However, the theoretical results in [10, 8] make several assumptions which are problematic for wireless sensor networks. These include:

1. When a node fires, its neighbors instantaneously observe that event.
2. Nodes can instantaneously react by firing.
3. Nodes can compute f and f^{-1} perfectly using continuous mathematics and can compute instantaneously.
4. All nodes have the same time period T.
5. Nodes observe all events from their neighbors (no loss).

In a wireless setting, a *firing event* can be implemented as a node sending a broadcast message to its neighbors indicating that it fired. However, as mentioned before, nodes experience an unpredictable delay prior to transmission, based on channel contention. Thus, when a node A sends out a firing event message at time t, its neighbor B will not receive the message until time $t + \delta$ where the delay δ is not known in advance. This violates assumptions 1 and 2. Node B does not know when the actual firing event occurred and node B can not react instantaneously to node A's behavior. In addition, the best case for the theoretical model — i.e. all nodes fire simultaneously — constitutes a worst case scenario for channel contention because it creates the potential for many collisions, resulting in large message delays.

The other assumptions also pose potential problems, though not quite as problematic as message delays. Computation accuracy is limited due to the absence of efficient floating point arithmetic. Sensor nodes exhibit slightly different oscillator frequencies. Links between nodes exhibit varying quality and thus varying levels of message loss. At the same

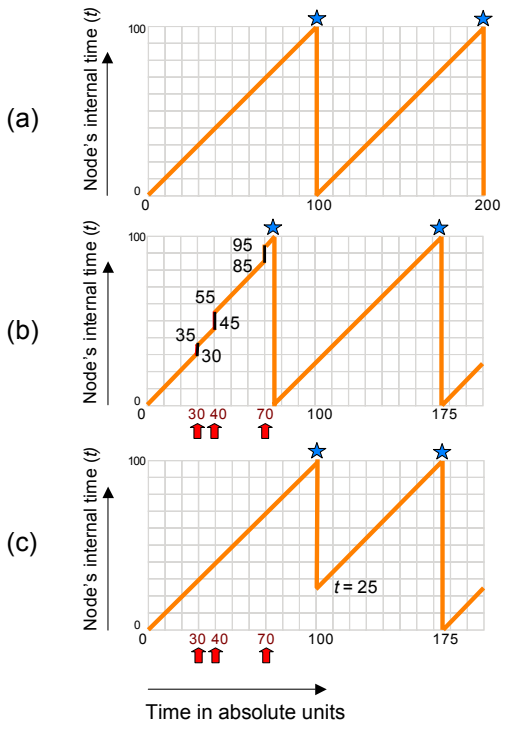


Figure 1: The firefly-inspired node algorithm. (a) A node fires whenever its internal time t is equal to the default time period T (=100). (b) In the M&S model, a node responds to neighbors firing (arrows) by instantaneously incrementing t. (c) In RFA, a node records the firing events and then responds all at once at the beginning of the next cycle.

time, real biological systems are known to have such variations. Therefore not all of the theoretical assumptions may be important in practice.

3.3 The Reachback Firefly Algorithm (RFA)

In this paper we focus mainly on the issues related to wireless communication. We tackle the three problems related to wireless communication in the following way: (1) We use low level timestamping to estimate the amount of time a message was delayed before being broadcast; (2) we modify the node algorithm and introduce the notion of "reachback" in which a node reacts to messages from the previous time period rather than the current time period, and (3) we preemptively stagger messages to avoid worst case wireless contention. Lastly, we use a simple, approximate firing function that can be computed quickly.

Timestamping Messages. In order to estimate the delay between when a node "fires" and when the actual message is transmitted, we use MAC-layer timestamping to record the MAC delay experienced by a message prior to transmission. The MAC delay can be measured by an event triggered by the TinyOS radio stack when the message is about to be transmitted, and is recorded in the header of the outgoing message. When a node receives a firing message, it uses this information to determine the correct firing time of the transmitting node by subtracting the MAC delay from the reception time of the message. This is very similar to the approach used in time synchronization protocols such as FTSP [9] to estimate message transmission delays.

The *Reachback Response.* The timestamping allows a node B to correctly identify when a neighbor A fired. However it only receives this information after some delay, and thus node B can not react instantaneously to node A's firing.

This causes two problems. First, node B may have already fired and thus no longer be able to react to node A. This is especially likely for neighbor firings that occur late in a node's time period. Furthermore, messages later in the cycle are important and have a larger adjustment effect (as a result of $f(t)$ being concave down). Secondly, as a result of the delays, a node may receive firing messages *out of order*. The effect of applying two firing events is not commutative. Suppose two firings occur at times t_1 and t_2 ($t_1 < t_2$). If a node learns of the events out of order, it will incorrectly advance its phase by $\Delta(t_2) + \Delta(t_1)$ instead of $\Delta(t_1) + \Delta(t_2 + \Delta(t_1))$. Therefore for the algorithm to be correct, a node would need to undo and redo the adjustments, quickly making the algorithm complicated and unmanageable.

Instead, in order to deal with delayed information, we introduce the notion of *reachback response*. In the reachback response, when a node hears a neighbor fire, it does not immediately react. Instead, it places the message in a queue, timestamped with the correct internal time t' at which the firing event occurred. When the node reaches time $t = T$, it fires. Then it "reaches back in time" by looking at the queue of messages received during the past period. Based on those messages, it computes the overall jump and increments t immediately (Figure 1(c)).

The computation is the same as in the M&S model described in Section 3.1; from the point of view of a node, it is as if it were receiving firing messages instantaneously. The only difference is that the messages it is receiving are actually from the *previous* time period. Thus a node is always reacting to information that is one time period old. In Section 4 we present theoretical results to support why the reachback response still converges.

Example: Here we illustrate how the algorithm works through an example, shown in Figure 1. We first show how the M&S model works, i.e. when messages are received instantaneously and the node reacts instantaneously. We then illustrate the reachback response using the same example.

Let the time period $T = 100$ time units. Let node B start at internal time $t = 0$ and increment t every unit time. Suppose firing events arrive at absolute times $30, 40$ and 70. Let $\Delta(t)$ be some jump function; here we simply pick jump values for illustration purposes.

In the M&S model, the node reacts as each event arrives, by causing an instantaneous jump in its internal time. $\Delta(t)$ represents the instantaneous jump at internal time t. When node B observes a firing at time $t = 30$, it computes an instantaneous jump of $\Delta(30) = 5$, and sets $t = 30 + \Delta(30) = 35$. Ten more time units from this point on it observes another event. While this event occurred 40 units of time since the beginning of the cycle, the node perceives it as having happened at internal time $t = 45$. The node again computes an instantaneous jump in internal time $t = 45 + \Delta(45) = 55$. After 30 more time units the node B observes another firing event. At this point $t = 85$ and the node computes an instantaneous jump to $t = 85 + \Delta(85) = 95$. After 5 more time units, $t = 100$ and node B fires.

It is also possible for the computed t to be larger than 100 (e.g. if $\Delta(85) = 20$ then $t = 85 + 20 = 105$), in which case the node sets $t = 100$, immediately fires, and resets $t = 0$.

The overall effect is that node B advances its phase (or shortens its time to fire) by 25 time units. It then continues to fire with the default time period of $T = 100$.

Now we use the same example to illustrate the reachback response. As before, let node B start with $t = 0$ and increment t every time unit. When node B receives a message, it uses the timestamping information to determine when that message would have been received had there been no delay. It then places this information in a queue and continues.

When $t = 100$, node B fires, resets $t = 0$, and then looks at the queue. In this example, the queue contains three events at times 30, 40 and 70. Using the same method described for M&S, the node computes how much it would have advanced its phase. Since all of the information already exists, it can compute the result in one shot. As in the previous case, the result is that the phase is advanced by 25 time units. Node B applies this effect by instantaneously jumping from $t = 0$ to $t = 25$. It then proceeds as before, firing by default at $t = 100$ if no events are received. The difference between the reachback scheme and the original M&S method is that the first firing event occurs at different absolute times (100 vs 75). This influences neighboring nodes' behavior and one must prove that the new scheme will still converge.

Pre-emptive Message Staggering. CSMA schemes attempt to avoid channel collisions by causing nodes to back-off for random intervals prior to message transmission. The range of this random interval is increased exponentially following each failed transmission attempt, up to a maximum range. If a small number of nodes are transmitting at any point in time, then this approach induces low message delays. However, if many nodes are transmitting simultaneously, delays may become very large. CSMA works very well with bursty traffic and non-uniform transmission times. However, for the M&S algorithm, the communication pattern is very predictable and represents the worst case for CSMA when many nodes are firing simultaneously.

In order to avoid repeated collisions and control the extent of message delay, we explicitly add a random transmission delay to node firing messages at the application level. We choose the delay uniformly random between 0 and a constant D. In addition, after a node fires, it waits for a grace period W (where $W > D$ and $W \ll T$) before processing the queue so that delayed messages from synchronized nodes are received. In Section 6, we discuss our choices for the parameter values and show that in practice this works well to control message delay.

Simplified Firing Function. In order to make the firing response fast to compute, we chose a simple firing function $f(t) = \ln(t)$. Using equation (1) along with $f^{-1}(x) = e^x$, we can compute the jump in response to a firing event, which is $\Delta(t') = f^{-1}(f(t') + \epsilon) - t' = (e^\epsilon - 1)t'$. To first order $e^\epsilon = 1 + \epsilon$ (Taylor expansion), leaving us with a simple way to calculate the jump.

$$\Delta(t') = \epsilon t' \tag{2}$$

3.4 Effect of Parameter Choices

The main parameter that affects the behavior of the system is ϵ, which determines the extent to which a node responds when it observes a neighbor firing. A node responds to a neighbor by incrementing its phase (shortening its time to fire) by ϵt, where t is the internal time at which the event was observed. Since $t < T$, the maximum increment a node could make is ϵT. Thus if $\epsilon = 1/100$, then a node can react to another node by at most $T/100$.

This gives us an intuitive feel for the effect of ϵ, which is made more concrete in the next section. Choosing a larger epsilon means that a node will take larger jumps in response to other nodes' firing, thus achieving synchrony faster. However if ϵ is too large, then nodes will "overshoot", preventing convergence. Making ϵ small avoids overshooting but only at the cost of nodes proceeding slowly towards convergence. In the next section we prove that the time to synchronize is proportional to $1/\epsilon$, for reasonable values of ϵ. Later in the paper we present simulation and testbed results that show that the system works well over a wide range of ϵ.

Other parameters such as the time period T and the message staggering delay D do not affect the ability to converge, nor the number of time periods to converge. The goal of D is to stagger messages within one broadcast neighborhood, therefore it should exceed network density. The choice of T affects overhead because it represents the frequency with which nodes communicate — one can choose that to be appropriate for the application. The main constraint is that $T \gg D$, so that there is enough time for all the messages from a previous time period to be collected. In the face of heavy congestion, this inequality may be violated in which case the delayed firing events can simply be discarded.

For our implementation, we choose T = 1sec and D = 25ms. These choices are somewhat arbitrary; our experimental results suggest that the application layer delay of 25ms works well to eliminate packet loss during synchronized firings for neighborhoods of upto 20 nodes.

4. THEORETICAL ANALYSIS

In this section, we present an analysis of the reachback scheme for two oscillators. Our analysis follows that of Mirollo and Strogatz. The ideas and mathematical constructs used are similar, though they differ in important ways that require slightly more complex analysis.

Each oscillator is characterized by a state variable s that evolves according to $s = f(\phi)$ where f is the firing function and ϕ is a phase variable representing where the oscillator is in its cycle. For example, if an oscillator has finished 3/4 of its cycle, then $\phi = 3/4$. Thus $\phi \in [0, 1]$ and $d\phi/dt = 1/T$ where T is the period of the oscillator's cycle. We assume that the function f is monotonic, increasing, and concave down. For our purposes, we choose $f(\phi) = \ln(\phi)$.

Now consider two oscillators A and B governed by f. They can be visualized as two points moving along the fixed curve $s = f(\phi)$ at a constant horizontal velocity $1/T$, as shown in Figure 2. When A reaches $\phi = 1$ it will fire, and B will record the phase at which it hears A's firing. In the RFA scheme, unlike the M&S model, B will *not* jump immediately upon hearing A's fire; instead, B will record the time and then execute the appropriate jump after its next firing. The jump is defined as $\Delta(\phi) = g(f(\phi) + \epsilon) - \phi$, where $g = f^{-1}$ and $\epsilon \ll 1$. For example, if B records ϕ_1 as the time A fired, when B reaches $\phi = 1$ it will fire and then jump forward to $\phi = \Delta(\phi_1)$. In the the RFA scheme, $f(\phi) = \ln(\phi)$ (and thus $g(\phi) = e^\phi$), therefore $\Delta(\phi) = (e^\epsilon - 1)\phi$. The question is whether the RFA scheme leads to synchrony.

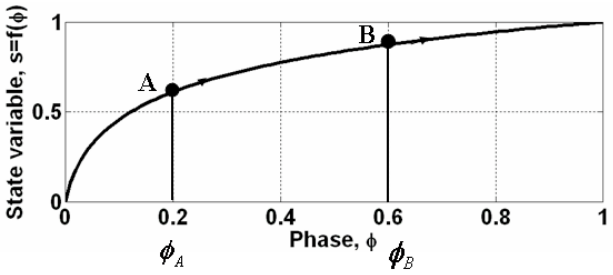


Figure 2: Two nodes A and B moving along $s = f(\phi)$

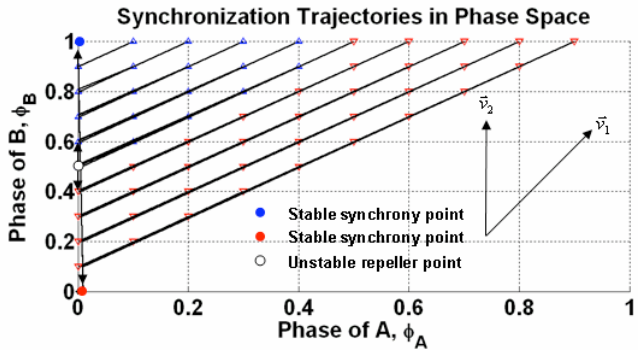


Figure 3: Trajectory of the oscillator phases

THEOREM 1. *Two oscillators A and B, governed by RFA dynamics, will be driven to synchrony irrespective of their initial phases.*

Proof. Consider two oscillators A and B. Consider the moment after A has fired and jumped. In the instant after the jump, let $\vec{\phi} = (\phi_A, \phi_B)$ denote the phases of oscillators A and B, respectively. The *return map* $R(\vec{\phi})$ is defined to be the phases of A and B immediately after the *next* firing of A (which is necessarily *after* the next firing of B since A cannot jump past B[1])

We now calculate the return map $R(\vec{\phi})$. Without loss of generality assume $\phi_A < \phi_B$. Since A has just fired, B records A's firing time as ϕ_B. Both oscillators move forward in their cycles until B fires. After B fires, according to our algorithm, B jumps to phase $\Delta(\phi_B)$. In the meanwhile, A has moved forward a distance $1 - \phi_B$, reaching phase $\phi_A + 1 - \phi_B$, and recording this as B's last firing time. After A's next firing, it jumps to $\Delta(\phi_A + 1 - \phi_B)$, and B is at $\Delta(\phi_B)+1-(\phi_A+1-\phi_B) = \Delta(\phi_B)+\phi_B-\phi_A$. Substitution of the expression for $\Delta(\phi)$ and algebraic simplification yields:

$$\vec{\phi}_{n+1} = R(\vec{\phi}_n) = M\vec{\phi}_n + \vec{b} \tag{3}$$

Here n denotes the cycle number. The vector $\vec{b}$ is defined as $(e^\epsilon - 1, 0)$, and the matrix M is defined as

$$M = \begin{bmatrix} e^\epsilon - 1 & -(e^\epsilon - 1) \\ -1 & e^\epsilon \end{bmatrix} \tag{4}$$

Hence the algorithm can be described as a linear dynamical system in $\vec{\phi}$, where $\vec{\phi} \in [0,1] \times [0,1]$. The unique fixed point of this dynamical system is easily shown to be:

$$\vec{\phi}^* = \begin{bmatrix} 0 \\ \frac{1}{2} \end{bmatrix} \tag{5}$$

At $\vec{\phi}^*$ both A and B would be exactly half a cycle apart. We now show that $\vec{\phi}^*$ is unstable (i.e.. a "repeller") such that the phases gets pushed to either (0,0) or (1,1) where the dynamics no longer change. Introducing the change of variables $\vec{\varphi}_n = \vec{\phi}_n - \vec{\phi}^*$ we can rewrite (3) as

[1]If A fires and then jumps to ϕ_A, then $\phi_A \leq \phi_B$ for the following reason: If the phase of B is ϕ_B when A reaches $\phi = 1$, then A must have observed B fire at $\phi_x \geq 1 - \phi_B$ (since B would have fired and then taken a positive jump). After firing A takes a jump of $\phi_A = \Delta(\phi_x)$. $\Delta(x)$ is always $\leq 1 - x$ because it is truncated to never cause a jump past the end of the cycle. Therefore $\phi_A \leq 1 - \phi_x \leq \phi_B$.

$$\vec{\varphi}_{n+1} = M\vec{\varphi}_n \tag{6}$$

By the eigendecomposition theorem, we can decompose M as $M = V\Lambda V^{-1}$, where V is the matrix of composed eigenvectors and Λ is a diagonal matrix containing the eigenvalues of M. To leading non-vanishing order in ϵ, the eigenvalues (in Λ) are $\lambda_1 = \epsilon^2$ and $\lambda_2 = (1+\epsilon)$, and the principal eigendirections (the rows of V^{-1}) are $\bar{v}_1 = (1,1)$ and $\bar{v}_2 = (0,\epsilon)$. The decomposition allows us to rewrite (6) as

$$\vec{\theta}_{n+1} = \Lambda\vec{\theta}_n \tag{7}$$

where $\vec{\theta}_n = V^{-1}\vec{\varphi}_n$ is a change-of-basis transformation that maps all vectors into a new coordinate system spanned by the basis $\mathcal{B} = \{\bar{v}_1, \bar{v}_2\}$. Equation (7) shows us that the evolution of the system is most simply described in terms of $\mathcal{B}$. The system's evolution along the directions $\bar{v}_1$ and $\bar{v}_2$ is illustrated in Figure 3. First, all trajectories rapidly approach the $\phi_A = 0$ axis along the vector $\bar{v}_1 = (1,1)$ since $\lambda_1 = \epsilon^2 \ll 1$. Upon reaching the axis, trajectories are repelled away from $\vec{\phi}^* = (0, 1/2)$, along the direction $\bar{v}_2 = (0,\epsilon)$, since $\lambda_2 > 1$. If a trajectory approaches the axis from below $\vec{\phi}^*$, it will slide down the axis to the state of synchrony $\vec{\phi} = (0,0)$. Otherwise, it will climb up to $\vec{\phi} = (0,1)$, another state of synchrony. Thus ϕ^* is a repeller and the oscillators are driven to synchrony, irrespective of initial phases. QED

Rate of synchronization. How quickly the system synchronizes depends on how fast it moves in the $\bar{v}_2$ direction away from $\vec{\phi}^*$ before it reaches a state of synchrony. We can estimate the time to synchronization, starting from $\vec{\phi}^{(0)} = (\phi_A^{(0)}, \phi_B^{(0)})$. Given such an initial state, the system's trajectory will intersect the $\phi_A = 0$ axis at approximately $\delta = \phi_B^{(0)} - \phi_A^{(0)}$. The distance from the fixed point grows exponentially fast with eigenvalue $\lambda_2 = (1+\epsilon)$ in the $\bar{v}_2$ direction. Let k denote the number of iterations required. Solving $\lambda_2^k(\delta - \frac{1}{2}) = \frac{1}{2}$ yields

$$k = \frac{1}{\ln(1+\epsilon)} \ln(\frac{1}{2\delta - 1}) \approx \frac{1}{\epsilon} \ln(\frac{1}{2\delta - 1}) \tag{8}$$

Thus, *the time to synchrony is inversely proportional to* ϵ.

Note that these proofs are very similar to the two oscillator case for Mirollo and Strogatz, and most likely can be extended to n nodes. However extending these results to multi-hop topologies requires considerably more sophisticated analysis [8]. Instead we evaluate the algorithm in simulation for different n and network topologies.

5. EVALUATION TOOLS AND METRICS

Both our simulation and testbed experiments output a series of node IDs and firing times. In order to discuss the accuracy of the achieved synchronicity, it is necessary to identify groups of nodes firing together.

For this purpose, we identify sets of node firings that fall within a prespecified time window. We call each cluster of node firings a *group*. Given a time window size w, the clustering algorithm outputs a series of firing groups that meet two constraints. First, every node firing event must fall within exactly one group. Second, groups are chosen to contain as many firing events as possible.

We define the *group spread* as the maximum time difference between any two firings in the group. The time window size w represents the upper bound on the group spread.

5.1 Evaluation Metrics

The two evaluation metrics that we are concerned with involve the amount of time until the system achieves synchronicity (if at all), and the accuracy of the achieved synchronicity.

Time To Sync: This is defined as the time that it takes all nodes to enter into a single group and stay within that group for 9 out of the last 10 firing iterations. The value chosen for the time window w does impact the measured time to sync; a very small w will result in a time to sync that is longer than with a larger w, because it takes longer for all nodes to join a firing group within a smaller time window. Also, as will be discussed in the next sections, the simulator has lower time resolution than the testbed hardware which means there is a limit on the accuracy it can achieve. Therefore, for the simulator we set $w = 0.1$sec and in the real testbed we set $w = 0.01$sec.

50th and 90th Percentile Group Spread: Recall that the group spread measures the maximum time difference between any two events in a firing group. We wish to characterize the *distribution* of group spread for all groups after the system has achieved synchronicity. Although synchronicity may be achieved according to the time to sync metric above, we wish to avoid measuring group spread while the system is still settling. Given the first sync time t_s and the time the experiment ends t_e, we calculate the group spread distribution across all groups in the interval $[t_s + \frac{(t_e - t_s)}{2}, t_e]$. In this way we are measuring the distribution across all "tight" groups rather than settling effects. We plot the 50th and 90th percentile of the distribution.

Lastly, we define the *Firing Function Constant* (FFC) to be the value $1/\epsilon$, which is the main parameter in the RFA algorithm. As discussed in Section 3.4, this parameter limits the response of a node to be at most T/FFC and thus the time to synchronize is directly proportional to FFC.

6. SIMULATION RESULTS

We have implemented the Firefly algorithm in TinyOS [4] using the TOSSIM [7] simulator environment. This simulator has several limitations. It does not model radio delay correctly, and nor does it take into account clock skew that occurs from variations in clock crystals in individual wireless sensors. Despite these limitations, the simulator is useful for exploring the parameter space of our algorithm. This can help us determine optimal parameter settings for the algorithm on a real testbed as well as better understand the impact of the parameter values on the level of synchronicity achieved. In our simulator experiments, we explore the impact of varying:

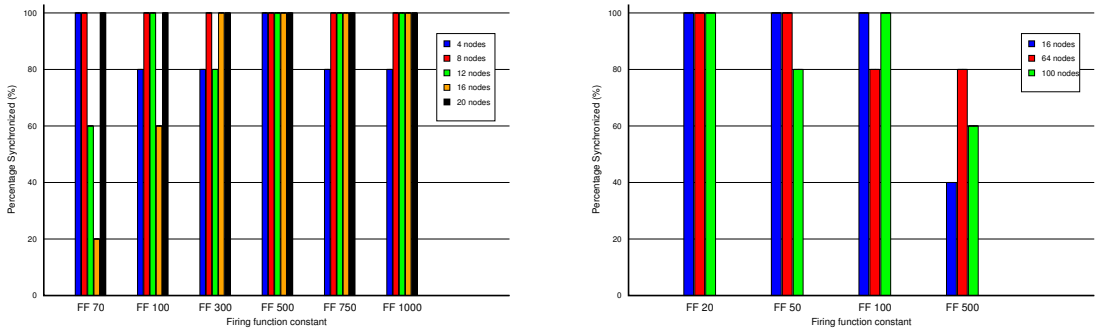
1. **Node topology**: *all-to-all* where each node can communicate with every other node, and a *regular grid* topology where a node can directly exchange messages with at most four other nodes.
2. **Firing function constant value**: ranging from 10-1000. Theoretically, the time to synchronize is proportional to the firing function constant value.
3. **Number of nodes *(n)***: We examine whether the impact of the firing function constant and node topology varies with the number of nodes. The size of the all-to-all topologies is varied between 2-20 nodes with 2 node increments, and grid topologies are varied from 16, 64, to 100 nodes.

All-to-all Topology Results. Figures 4(a) and 5 show the results of simulations on the all-to-all topology. We ran simulations for firing function constant values (FFC) 10,20,50,70,100,150,300,500,750 and 1000, repeating this combination for number of nodes ranging from 2-20 in 2 node increments. For each parameter choice, we ran 10 simulations using different random seeds to start the nodes at different times. Each experiment was run for 3600 seconds of simulation time. Also the time period $T = 1$sec for all experiments.

Fig. 4(a) shows the percentage of simulations that synchronized for a selection of parameter values. This percentage represents the fraction of runs that achieved synchronicity out of the 10 total runs, for a given parameter choice of n and FFC. We can see that the FFC values displayed in the figure (70,100,300,500 and 750) are fairly reliable since these cases achieved synchronicity in a majority of the simulation runs. Most experiments with small firing function constants (10,20,50) did not achieve synchronicity. One likely reason for this behavior is that small FFC values lead nodes to make extremely large jumps, causing them to overshoot (see Section 3.4).

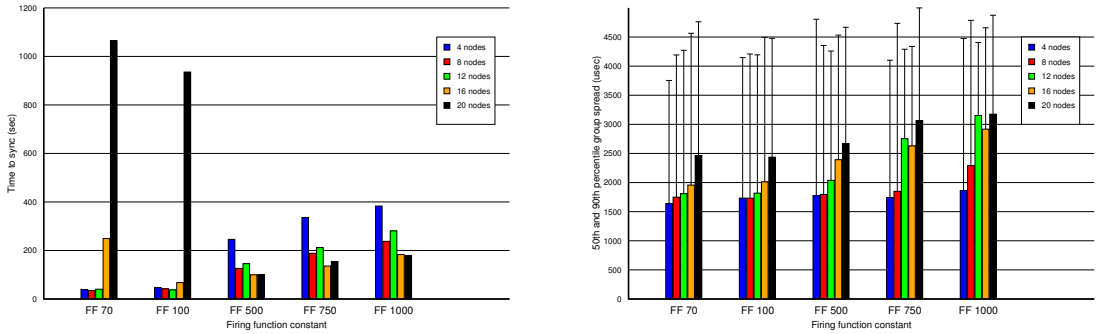
Fig. 5(a) shows the time to synchronize as a function of FFC value and the number of nodes. The graph shows that most FFC constants work well. The time to sync increases with increasing FFC value but not beyond 400 time periods. There is no clear trend with increasing numbers of nodes, although small FFCs do not work as well with large numbers of nodes. This is possibly because the effect of overshoot is worsened when there are more neighbors (and thus more total firing events per cycle to react to).

Fig. 5(b) shows the corresponding group spreads. For most FFC values and most n, the 90th percentile group spread remains the same. The 50th percentile shows a slight increase with increasing FFCs and a slight increase with



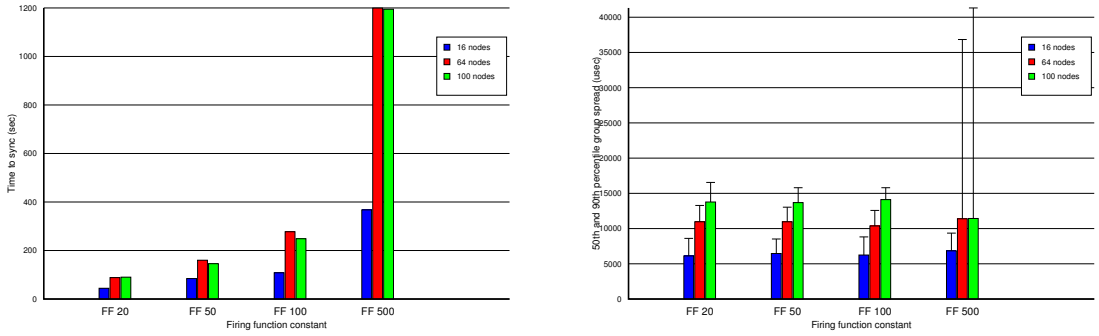
(a) (b)

Figure 4: Percentage of simulations that achieved synchronicity for different firing function constants and numbers of nodes. (a) All-to-all topology. Small firing function constants (E.g. 10,20,50,150) did not achieve synchronicity most of the time. (b) Grid Topology. Experiments with very small firing function constants (E.g. FFC=10) or very large firing function constants (E.g. > 500) did not achieve synchronicity.



(a) (b)

Figure 5: All-to-all topology. (a) Time to Sync, which for most cases increases with increasing FFC values. (b) Group Spread. The solid bars represent the 50th percentile spread, while the error bar indicates the 90th percentile. For most FFC values, the group spreads remain similar, with a slight increase as the number of nodes increases



(a) (b)

Figure 6: Grid Topology. (a) Time to Sync, which increases with increasing FFC values and network diameter. (b) Group Spread. The solid bars represent the 50th percentile spread, while the error bar indicates the 90th percentile. Group spread remains similar over varying FFC values, but increases with network diameter. For large grids, FFC=500 does synchronize as well, as also shown in Figure 4.

increasing numbers of nodes. However, the difference in the spreads is not large and thus group spreads remain fairly similar over all parameter values. The error bars for this data (not shown here) show that there is not much variation across different experimental runs.

Grid Topology Results. Fig. 4(b) and 6 show the simulation results for regular grid topologies of 4x4, 8x8, and 10x10 nodes. Fig. 4(a) shows the percentage of cases that synchronized for a selection of parameter values. The results show that FFC values in the range of 20-500 almost always achieve synchronicity in a grid topology.

The behavior of nodes in a grid topology reflects the impact of network diameter on performance. Fig. 6 (a) shows that large values of the firing function constant increase the time taken to achieve synchronicity, and that this effect is more pronounced for larger grids. Larger FFC values imply that nodes make smaller jumps and thus converge to synchrony more slowly. For a given FFC, the time to sync also increases slightly with network diameter, but not by much for the smaller FFC constants. The error bars for the FFC=500 simulations (not shown here) show that there is a large variation in time to sync and group spread across runs, most likely caused by the initial phase distribution of nodes in the grid which can only be corrected slowly. Barring that case, Fig. 6 (b) shows that the group spread does not vary significantly with FFC value. However there does seem to be an increase in spread (i.e. decrease in accuracy) for larger grids, indicating that the network diameter may have an impact on how well the system can synchronize.

7. WIRELESS SENSOR NETWORK TESTBED EVALUATION

Having proven our algorithm correct in simple cases and explored the parameter space using TOSSIM, we then tested RFA on a real sensor network testbed. The experiments carried out on the 24-node indoor wireless sensor network testbed, show the performance of our algorithm running on real hardware, with a complex topology, and experiencing communication latencies over lossy asymmetric links. The table in Figure 10 summarizes our results, showing that RFA can rapidly synchronize all the nodes to approx. 100 μsec.

This section is structured as follows. First, we describe our testbed environment, focusing on the significant ways in which the testbed differs from the TinyOS simulator. We then describe our use of FTSP to provide a common global time base for nodes participating in our experiments. We then discuss our experiments and results, and compare them to our expectation from theory and simulation.

7.1 Testbed Environment

Our experiments ran on MoteLab [16], a wireless sensor network testbed consisting of 24 MicaZ motes distributed over one floor of our Computer Science and Electrical Engineering building. The MicaZ motes have a 7.3MHz clock. Each device is attached to a Crossbow MIB600 interface backchannel board allowing remote reprogramming and data logging. Messages sent to the nodes' serial ports are logged by a central server. Using this data-logging capability, nodes report their firing times as well as information about firing messages they observe. This information is then used to evaluate the performance of our algorithm, as well as better understand its behavior.

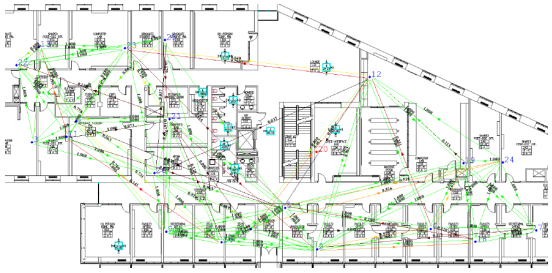


Figure 7: Connectivity Map: The distribution and connectivity of sensor nodes (detailed image at `http://motelab.eecs.harvard.edu/`).

7.1.1 Network Topology

We conducted our experiments on the most densely populated floor, with 24 nodes. Statistics on message loss rates are calculated periodically and tend to vary over time. Examining the connectivity map and graph shown in Figures 7 and 8, one can see that this is a complex multi-hop topology. The layout of the building produces two cliques of nodes that are connected by high quality links (less than 20% message loss) and these cliques are connected by only a few bridge nodes. Further examination of the cliques shows that a large fraction of the links are asymmetric in quality and some nodes (e.g. 2, 26) may have no incoming links of good quality. This type of complex topology is representative of sensor networks distributed in a complex environment.

7.1.2 Time Stamping with FTSP

In order to evaluate the performance of RFA, we need to time stamp the firing messages so that we can determine the accuracy with which the firing phases align. However, this proved to be difficult in our testbed environment. Unlike TOSSIM, our sensor nodes do not have access to a global clock. Furthermore, since they are distributed throughout the building, there is no single base station that can act as a global observer and provide a common time base [9]. In an ironic way, evaluating RFA requires an independent and accurate implementation of time stamping.

To address this difficulty we deployed the Flooding Time Synchronization Protocol (FTSP) [9] described in Section 2. FTSP provides nodes access to a stable global clock and allows them to time stamp events with precisions reported in the tens of microseconds. We characterized the errors in FTSP on our MoteLab topology in the following way. Nodes log all firing messages they hear from other nodes and the time stamp that FTSP assigned to them. For every firing message heard by more than two nodes, we compute the differences between the times that FTSP reported on each node, taking the maximum difference between any two stamps. This is only done for messages where FTSP on both the sender and receiver reported that the nodes were well-synchronized to the global clock. The cumulative distribution frequency (CDF) of these errors for all of our testbed experiments is shown in Figure 9. Note however that the FTSP errors are only calculated for nodes that are within one hop of each other, therefore we do not know the error in FTSP between two arbitrary nodes. Given this caveat, our results show that FTSP can quickly synchronize nodes to within one hop errors of tens of microseconds.

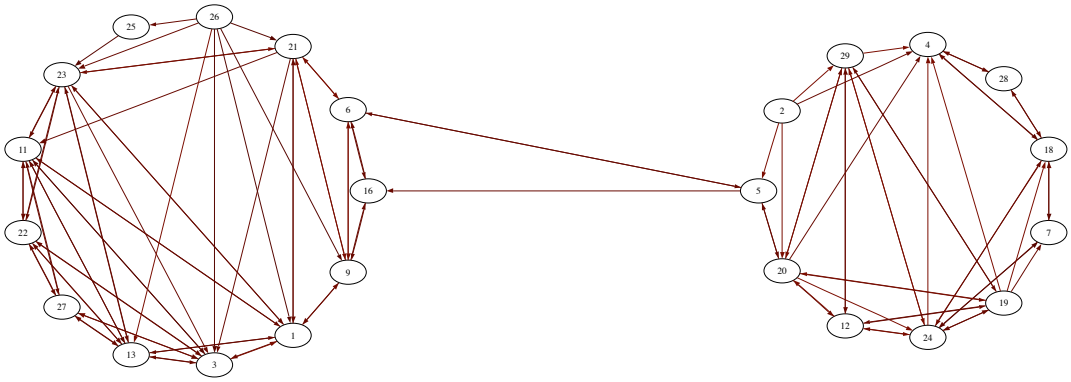


Figure 8: Connectivity graph of the nodes on our testbed showing links with better than 80% packet reception. This reveals two highly-connected subgraphs and a high percentage of asymmetry in link quality.

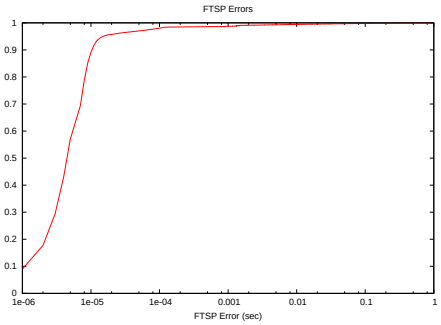


Figure 9: CDF of one hop FTSP error rates collected during all of our testbed experiments. FTSP provided the global clock for RFA evaluation.

The use of FTSP as the global clock impacts our experimental results in two ways. First, our results are pessimistic since their accuracy is limited by the accuracy of time stamping which is likely to be in the 5-10 μsec range. Secondly, this prevents us from making an objective comparison of accuracy between FTSP and RFA. In the future, we plan to use more controlled experimental setups to better evaluate the accuracy of RFA and compare it to other methods.

7.1.3 *Differences From the Simulator*

Sensor network devices in a real environment exhibit behavior not modeled by TOSSIM. Most significant differences for our algorithm are (1) communication latencies are unpredictable and difficult to measure accurately, (2) links are asymmetric and fluctuate over time, and (3) crystal differences cause node clocks to tick at different rates. All these effects make it difficult for a node to know exactly when another node fired, and unlikely that it will hear every firing message sent by its neighbors.

Paradoxically, our testbed is *better* than TOSSIM in one very significant way. The sensor node hardware has a high-precision clock and high-precision timer components. A TinyOS component written for our project allows nodes to both locally time stamp and set timers at μsec level resolution by multiplexing the `SysTime` and `MicroTimer` components onto one hardware counter. The simulator TOSSIM provides 4MHz local time stamping through access to its internal clock, but no equivalent of the `MicroTimer` component, forcing us to rely on the standard TinyOS `Timer` component with its millisecond resolution. This limits the resolution of the jumps a node can take and therefore the algorithm achieves only millisecond accuracy in simulation. As we were using the simulator primarily to explore the impact of parameter and topology, we felt the lack of precision this introduced was acceptable. The higher frequency `MicroTimer` component is the primary reason that group spread results from our testbed experiments are much better than the simulation, which at first would seem surprising.

7.2 Performance Evaluation

The table in Figure 10 summarizes the results from our experiments. We conducted four experiments with firing function constants (FFC) 100, 250, 500 1000.

In each of these experiments the network achieves synchronicity. In the case with a FFC of 100, the network is synchronized within 5 minutes, with a group spread of 131 μsec. As expected from our theoretical and simulation studies, the time to synchronize increases with larger firing function constants. The table also shows the 50th and 90th percentile group spread, and Figure 11 plots the group spread CDFs for each of the four FFCs. As we can see they are striking similar, aligning with our expectations from theory and simulations that group spread does not depend on the FFC value.

These results are very encouraging, given the caveats on our time stamping and the complexity of the testbed environment. However, there are also several features which are still unexplained and we plan to investigate these more in the future. Here we discuss two examples.

One observation from the CDF plot in Figure 11 is that there seems to be a limit on the achievable group spread of around 100 μsec. This could be related to FTSP accuracy, but it is also affected by clock skew. We expect clock skew to impact the accuracy of RFA because the M&S model upon which it is based assumes that nodes agree on a fixed firing period. However, two perfectly synchronized nodes will diverge on the very next firing by the amount of drift

FFC constant	*Time to sync (sec)*	*50th pct spread (μsec)*	*90th pct spread (μsec)*	*Mean group std dev*
100	284.3	131.0	4664.0	410.4
250	343.6	128.0	3605.0	572.2
500	678.1	154.0	30236.0	1327.8
1000	1164.4	132.0	193.0	63.6

Figure 10: Summary of Testbed Results. Four experiments were run on the 24 node testbed, with different firing function constants (where $FFC = 1/\epsilon$) As expected, the time to synchronize increases with FFC. The 50th percentile group spread is similar for all four experiments.

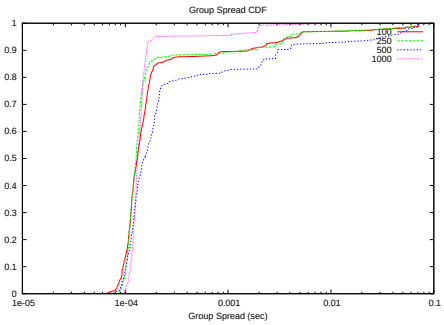


Figure 11: CDF of RFA Group Spread. The four different firing function constants in the testbed experiments produce similar levels of synchronicity.

that exists in their clocks — introducing an error that is on the order of the crystal accuracy. This causes the nodes to constantly readjust their phases. Clock skew also impacts the accuracy indirectly when nodes use the message delay (measured by the sender's clock) to adjust their own phase. In the future we plan to investigate the effects of clock skew more rigorously and look at alternate models of synchronization, such as synchronized clapping, where both frequency and phase are adjusted.

A second observation arises from looking at the group spread over time. We observed that even after tight groups form, occasionally disturbances still occur during the experiment. We refer to these as *dispersive events*. Figure 12 (a) shows an example of a dispersive event where the nodes fall out of phase and the system recovers from this impulse over the next few firing periods. Figure 12 (b) shows a different time course with the occurrence of two dispersive events which cause the group to spread over approximately 0.1 *sec* range. Note that these dispersive events are included in the data in Table 10 and impact the 90th percentile group spread. Analysis of the information collected during this experiment showed no clear reason for this spurious firing. It is unlikely to be caused by the algorithm, since the extent of dispersion is larger than the maximum jump possible given the FFC. What is encouraging is that the recovery from the 100 msec range dispersive events takes only a few rounds to achieve, thus showing that the system can recover quickly from perturbation. However, a more rigorous instrumentation and experimentation setup will be required to pinpoint the causes behind this behavior.

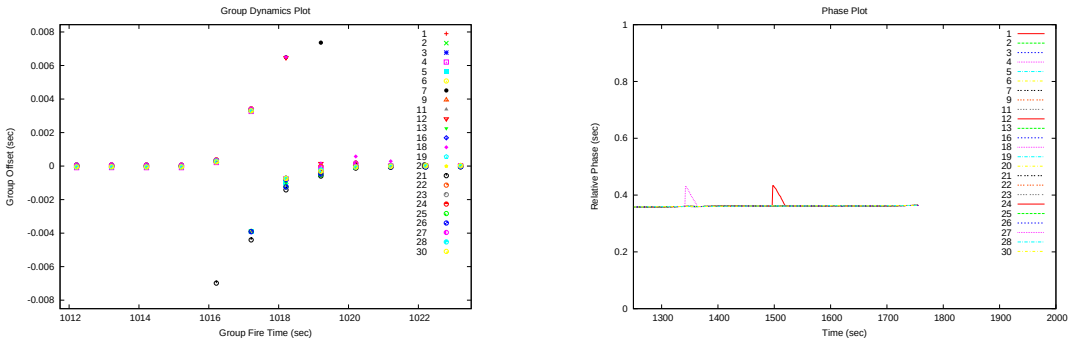
8. COMPARISON TO OTHER METHODS

Compared to algorithms such as RBS, TPSN and FTSP[2, 9, 3], the firefly-inspired algorithm represents a radically different approach. All of the nodes behave in a simple and identical manner. There are no special nodes, such as the root in TPSN or reference node in RBS, that need to elected. A node does not maintain any per-neighbor or per-link state; in fact it is completely agnostic to the identity of its neighbors. The algorithm remains the same even if the topology is multi-hop. There are no network-level datastructures, such as the spanning tree in TPSN, that must be re-established in case of topology change. As a result of these properties, the algorithm is *implicitly robust to the disappearance of nodes and links*. Lucarelli et al [8] have shown that the algorithm works on time-varying topologies; our testbed results show that the algorithm performs well even with asymmetric, lossy links. The inherent adaptive nature of such algorithms is one of the main attractions of biologically-inspired approaches.

Nevertheless, it is not yet clear whether such an algorithm will be competitive to algorithms such as TPSN and FTSP, in terms of accuracy and overhead, and much work remains to be done. In terms of accuracy, RFA achieves 100 μsec which is significantly less than the reported 10 μsec accuracy of FTSP, although as discussed before it is difficult to make a clear comparison because of the errors caused by using FTSP as our evaluation clock. We believe that the accuracy can be increased to tens of microseconds by eliminating errors in our evaluation methodology and by using a better optimized MAC-layer delay estimation (as used in FTSP [9]). However beyond that, the accuracy will still be limited by clock skew, as discussed in Section 7.2. We intend to investigate models that synchronize both phase and frequency, which would eliminate errors cause by clock skew. A second shortcoming of RFA is that the communication overhead is high. In particular, we choose $T = 1$ sec, unlike FTSP which has a time period of 30 seconds. Assuming one could compensate for clock skew, the main limit on T is the time taken to synchronize from startup. RFA takes approximately 200 time periods to synchronize on MoteLab, which is significantly more than the diameter of the network. On the other hand, the system recovers quickly from small dispersive events. One option would be to use a simple mechanism, such as an initial flood, to bring all nodes to within a small phase difference quickly. Then RFA could operate at a much lower frequency to tighten the accuracy and maintain synchronicity. A different option is to allow nodes to asynchronously backoff, or increase, their time period in multiples of T depending on whether they observe many out-of-phase firing events in their neighborhood. Thus nodes would self-adjust the overhead.

9. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a decentralized algorithm for synchronicity, based on a mathematical model of synchronicity achieved by biological systems. Our results show that even though the theoretical models make simplifying assumptions, this technique still works well and robustly in



(a) (b)

Figure 12: Dispersive events (a) This plot shows the group dynamics for several firing periods following the effect of one node falling out of sync. Nodes offsets against the average group firing time are plotted with respect to time. (b) This plot shows a different example, with FFC=1000. After the system has stabilized there are still two cases where nodes jump out of phase and are re-incorporated into the group.

the face of realistic radio effects and hardware limitations. In particular, we modified the algorithm to deal with communication latencies, but it still achieved synchronicity reliably and showed predictable behavior in relation to parameter choice. In the future we intend to study in detail how lossy links, clock skew and topology parameters affect the model. A detailed study of the robustness of the system will also help us better understand whether the adaptive and robust behavior of biological systems can be leveraged to design robust algorithms for sensor networks.

10. ACKNOWLEDGEMENTS

Nagpal is supported by a Microsoft Faculty Fellowship, Patel is supported by a NSF graduate Fellowship. We also thank Uri Braun, who worked on initial stages of the project.

11. REFERENCES

[1] Center for Information Technology Research in the Interest of Society. Smart buildings admit their faults. `http://www.citris.berkeley.edu/applications/disaster_response/smartbuil%dings.html`, 2002.

[2] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. In *Proc. OSDI '02*, Dec 2002.

[3] S. Ganeriwal, R. Kumar, and M. B. Srivastava. Timing-sync protocol for sensor networks. In *Proc. ACM SenSys '03*, Nov 2003.

[4] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System architecture directions for networked sensors. In *Proc. ASPLOS '00*, Nov 2000.

[5] Y. Hong, L. Cheow, and A. Scaglione. A simple method to reach detection consensus in massively distributed sensor networks. In *IEEE Intl Symposium on Information Theory, ISIT'04*, June 2004.

[6] Y. Hong and A. Scaglione. Time synchronization with pulse-coupled oscillators for uwb wireless ad hoc networks. In *IEEE Conf. on Ultra Wideband Systems and Technologies*, Nov 2003.

[7] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and scalable simulation of entire TinyOS applications. In *Proc. ACM SenSys '03*, Nov 2003.

[8] D. Lucarelli and I. Wang. Decentralized synchronization protocols with nearest neighbor communication. In *Proc. ACM SenSys '04*, Nov 2004.

[9] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi. The flooding time synchronization protocol. In *Proc. ACM SenSys '04*, Nov 2004.

[10] R. Mirollo and S. Strogatz. Synchronization of pulse-coupled biological oscillators. *SIAM*, 50(6):1645–1662, 1990.

[11] C. S. Peskin. *Mathematical Aspects of Heart Physiology*. New York University, New York, 1975.

[12] C. Schurgers, V. Tsiatsis, S. Ganeriwal, and M. Srivastava. Topology management for sensor networks: Exploiting latency and density. In *Proc. MobiHoc*, 2002.

[13] G. Simon et al. Sensor network-based countersniper system. In *Proc. ACM SenSys '04*, Nov 2004.

[14] N. Wakamiya and M. Murata. Scalable and robust scheme for data fusion in sensor networks. In *Intl Workshop on Biologically Inspired Approaches to Advanced Information Technology*, Jan 2004.

[15] 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'05)*, Jan 2005.

[16] G. Werner-Allen, P. Swieskowski, and M. Welsh. MoteLab: A wireless sensor network testbed. In *Proc. IPSN'05, Special Track on Platform Tools and Design Methods (SPOTS)*, April 2005.

[17] I. Wokoma et al. A weakly coupled adaptive gossip protocol for application level active networks. In *IEEE Intl Workshop on Policies for Distributed Systems and Networks*, 2002.

[18] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *Proc. IEEE INFOCOM '02*, June 2002.