FIREFLY SYNCHRONIZATION IN WIRELESS SENSOR NETWORK

Thesis submitted in partial fulfillment of the requirements for the degree of B.Sc.
In
Electrical and Electronic Engineering

By
AlWaleed Ahmed Bahar

Under the supervision of
Dr. Hamid A. ALI

Department of Electrical and Electronic Engineering
University of Khartoum

July 2008
Dedication

To my dear father and my beloved mother..

To my brother and sisters..

To my grandmother and aunt..

To my friend Mos'ab Ibn Ouf soul, may his soul rest in peace.
Acknowledgment

Firstly and finally, all praise and gratefulness is due to Allah for his uncountable blessings, he would never be thanked enough.

I would like to thank my Supervisor Dr. Hamid A. Ali for his advice and guidance through the whole project.

I am deeply grateful to my partner Samawal for his friendship at first place, and then his partnership.

I would like to give my gratitude to everyone helped or even offered his help during the work.
(specially Mozaffar Omar)

My very sincere thanks to my whole 03 batch mates for the wonderful five years.
Abstract

Fireflies exhibit a fascinating phenomenon of spontaneous synchronization that occurs in nature. At dawn, they gather on trees and synchronize progressively without relying on a central entity. A recently presented algorithm called Reachback Firefly Algorithm was inspired from the phenomenon and based on a mathematical model that describes how fireflies spontaneously synchronize. This algorithm was produced as a new approach to achieve synchronicity in multi-hop ad-hoc wireless sensor networks and compared to other existing time synchronization protocols.

This approach is simulated with a MATLAB-based modified probabilistic wireless sensor network simulator for different network topologies and parameters choices.
المستخلص

تقدم اليراعات المضيئة (fireflies) واحدة من أروع الظواهر الطبيعية المتعلقة بعملية التزامن التلقائي. عند الفجر، تتجمع هذه الحشرات على الأشجار وتبشر عملية التزامن تدريجيا دون الاستناد إلى وحدة مركزية. هذه الظاهرة ظلت محل اهتمام العلماء لفترة طويلة من أجل تفسير حدوثها وتحليلها رياضيا.

قدمت مؤخرا خوارزمية مستندة من هذه الظاهرة تعرف بـ Reachback Firefly Algorithm تقوم على النموذج الرياضي المفسر لتلك الظاهرة. استخدمت هذه الخوارزمية كطريقة جديدة لتحقيق مفهوم التزامن في شبكات الحواسيب اللاسلكية، وتمت مقارنتها مع خوارزميات وبروتوكولات التزامن المختلفة.

لدراسة إمكانية تطبيق الخوارزمية المقترحة، تم استخدام برامج محاكي لبيئة شبكات الحواسيب اللاسلكية برمج باستخدام MATLAB. يتتيح خيارات متعددة للمتغيرات والبيانات.
# Table of Contents

<table>
<thead>
<tr>
<th>Chapter 1: Introduction</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Motivation and Objectives</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Thesis Layout</td>
<td>2</td>
</tr>
<tr>
<td><strong>Chapter 2: Ad-hoc Wireless Network</strong></td>
<td>3</td>
</tr>
<tr>
<td>2.1 Ad hoc computer network</td>
<td>3</td>
</tr>
<tr>
<td>2.2 Wireless Ad-hoc networks</td>
<td>3</td>
</tr>
<tr>
<td>2.2.1 Applications</td>
<td>4</td>
</tr>
<tr>
<td>2.3 Wireless sensor network</td>
<td>4</td>
</tr>
<tr>
<td>2.3.1 Applications</td>
<td>5</td>
</tr>
<tr>
<td>2.3.2 Characteristics</td>
<td>5</td>
</tr>
<tr>
<td>2.4 Energy Awareness</td>
<td>6</td>
</tr>
<tr>
<td><strong>Chapter 3: Synchronization in WSN</strong></td>
<td>7</td>
</tr>
<tr>
<td>3.1 Synchronization in Networks</td>
<td>7</td>
</tr>
<tr>
<td>3.1.1 Synchronization Definition</td>
<td>7</td>
</tr>
<tr>
<td>3.1.2 Synchronization in Telecommunications</td>
<td>7</td>
</tr>
<tr>
<td>3.1.3 Synchronization in Networks</td>
<td>7</td>
</tr>
<tr>
<td>3.1.4 Synchronicity and Synchronization</td>
<td>8</td>
</tr>
<tr>
<td>3.2 Synchronization in WSN</td>
<td>8</td>
</tr>
<tr>
<td>3.2.1 Approaches to Time Synchronization</td>
<td>8</td>
</tr>
<tr>
<td>3.2.2 Approaches to Time Synchronization</td>
<td>9</td>
</tr>
<tr>
<td>3.2.3 Comparing the Algorithms</td>
<td>10</td>
</tr>
<tr>
<td>3.3 Medium Access Control (MAC) Layer</td>
<td>10</td>
</tr>
<tr>
<td><strong>Chapter 4: Fireflies Synchronization</strong></td>
<td>11</td>
</tr>
<tr>
<td>4.1 Phenomena of self-organization</td>
<td>11</td>
</tr>
<tr>
<td>4.2 Mathematical Model (MaS Model)</td>
<td>12</td>
</tr>
<tr>
<td>4.3 Pulse-Coupled Oscillators</td>
<td>14</td>
</tr>
<tr>
<td><strong>Chapter 5: The Reachback Firefly Algorithm (RFA)</strong></td>
<td>15</td>
</tr>
<tr>
<td>5.1 From Theory to Practice</td>
<td>15</td>
</tr>
<tr>
<td>5.2 Reachback Firefly Algorithm</td>
<td>16</td>
</tr>
<tr>
<td>5.3 The Algorithm Explanation</td>
<td>17</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>5.3.1 Preemptive Message Staggering</td>
<td>19</td>
</tr>
<tr>
<td>5.3.2 Simplified Firing Function</td>
<td>19</td>
</tr>
<tr>
<td>Chapter 6: Simulation</td>
<td>20</td>
</tr>
<tr>
<td>6.1 Prowler: Probabilistic Wireless Network Simulator</td>
<td>20</td>
</tr>
<tr>
<td>6.2 Network topology and parameters</td>
<td>21</td>
</tr>
<tr>
<td>6.2.1 Network topology</td>
<td>21</td>
</tr>
<tr>
<td>6.2.2 Simulation Parameters</td>
<td>22</td>
</tr>
<tr>
<td>6.3 Firefly Algorithm Implementation</td>
<td>24</td>
</tr>
<tr>
<td>6.3.2 Parameters Setting</td>
<td>24</td>
</tr>
<tr>
<td>6.3.3 Simulation Results</td>
<td>25</td>
</tr>
<tr>
<td>6.4 Performance Optimization</td>
<td>26</td>
</tr>
<tr>
<td>6.4.1 Broadcasting Performance</td>
<td>27</td>
</tr>
<tr>
<td>6.4.2 Connection Links Formation</td>
<td>28</td>
</tr>
<tr>
<td>6.5 Analysis and Comments</td>
<td>29</td>
</tr>
<tr>
<td>6.5.1 Results Analysis</td>
<td>30</td>
</tr>
<tr>
<td>6.5.2 Simulator limitations</td>
<td>31</td>
</tr>
<tr>
<td>6.5.3 MATLAB Limitations</td>
<td>31</td>
</tr>
<tr>
<td>Chapter 7: Conclusion</td>
<td>32</td>
</tr>
<tr>
<td>7.1 Conclusion</td>
<td>32</td>
</tr>
<tr>
<td>7.2 Future Work</td>
<td>32</td>
</tr>
<tr>
<td>References</td>
<td>33</td>
</tr>
<tr>
<td>APPENDIX A : MATLAB CODE</td>
<td>34</td>
</tr>
</tbody>
</table>
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig 2.1</td>
<td>ad-hoc wireless network</td>
<td>3</td>
</tr>
<tr>
<td>Fig 4.1</td>
<td>Time evolution of the phase function</td>
<td>13</td>
</tr>
<tr>
<td>Fig 5.1</td>
<td>RFA mechanism</td>
<td>17</td>
</tr>
<tr>
<td>Fig 6.1</td>
<td>PROWLER interface</td>
<td>21</td>
</tr>
<tr>
<td>Fig 6.2</td>
<td>grid network topology (5 x 5) and parameters</td>
<td>22</td>
</tr>
<tr>
<td>Fig 6.3</td>
<td>MAC layer communication scheme</td>
<td>23</td>
</tr>
<tr>
<td>Fig 6.4</td>
<td>Flood algorithm simulation</td>
<td>25</td>
</tr>
<tr>
<td>Fig 6.5</td>
<td>Firefly algorithm simulation</td>
<td>26</td>
</tr>
<tr>
<td>Fig 6.6</td>
<td>Error Surface of the broadcasting</td>
<td>28</td>
</tr>
</tbody>
</table>

List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 6.1</td>
<td>Parameters Values</td>
<td>24</td>
</tr>
<tr>
<td>Table 6.2</td>
<td>Simulation results</td>
<td>26</td>
</tr>
<tr>
<td>Table 6.3</td>
<td>Results for Analysis</td>
<td>30</td>
</tr>
</tbody>
</table>
Abbreviations

**WSN**  Wireless Sensor Network
**MAC**  Medium Access Control
**SNR**  Signal to Noise Ratio
**RBS**  Reference Broadcast Synchronization
**TSPN** Timing-sync Protocol for Sensor Networks
**FTSP** Flooding Time Synchronization Protocol
**PCO**  Pulse Coupled Oscillators
**CSMA** Carrier Sense Multiple Access
**MaS**  Mirollo and Strogatz
**RFA**  Reachback Firefly Algorithm
**Prowler**  Probabilistic Wireless Network Simulator
1 Introduction

Since technology improvements in the last decade have made smaller and more inexpensive sensor nodes possible, sensor networks have become a big research field. In the beginning such a network was composed of small numbers of sensor nodes whereas these nodes were wired to a central processing unit. Nowadays sensor networks are mostly built-on wireless technology with a big number of sensing nodes with local processing. Hence, that progress enabled the use of sensor networks in a variety of applications. For example, the monitoring of a phenomenon could be a difficult challenge if the exact location where to place a sensing element is not known. Alternatively, distributed sensing allows a closer placement to the phenomenon and therefore a better Signal-to-noise Ratio (SNR).

However, in most cases the environment where such sensing nodes are in use is harsh and usually does not provide an infrastructure. Such a malicious environment challenges some design constraints like robustness, low power consumption, physical size, network discovery, lifetime and many others that vary from sensor to sensor. Thus, the sensors must rely on local, finite, and relatively small energy sources.

In order to be economically feasible, the devices generally must have a lifetime on the order of months to years. The major sources of energy waste are packet collisions, overhearing, control packet overhead, and idle listening whereas in many MAC-protocols such as IEEE 802.11 more than 50% are spent on idle listening. Several approaches have been proposed to improve energy efficiency focusing mostly on clustering mechanisms, routing algorithms, energy dissipation schemes, sleeping schedules, and so on.

Still a maximized network lifetime requires the use of a well-structured design methodology and must consider the tradeoffs between energy consumption, system performance, and operational fidelity.
1.1 Motivation and Objectives

Typical energy efficient implementations keep the nodes largely inactive for most of the time and become active only for a short time if something is detected. This results in a periodic sleep/listen approach which reduces the idle listening and is called sleep scheduling. A simple solution for this concept is based on coordination which means that an ensemble of nodes must agree on the same schedule and therefore will sleep and listen at the same time. However, clock synchronization in sensor networks, especially in multi-hop wireless ad-hoc networks, is an important necessity to share a common view of the local clock time. Without precise clock synchronization, the mobile devices do not wake up at the same time and thus the power management operation will not work well.

In this project, the biologically-inspired Firefly algorithm is used for synchronicity to achieve coordinated sleeping. Hence our approach is based on distributed synchronous clocking and is a type of internal synchronization. In contrast to centralized clock synchronization schemes, the distributed synchronization approach has the inherent advantage for complete scalability and graceful degradation. This approach is appropriate for ad-hoc sensor networks where the topology is not known or might change.

1.2 Thesis Layout

This thesis is organized as follows:

Chapter 2: Introduces the ad-hoc wireless sensor network and energy awareness.

Chapter 3: Defines synchronization concept in WSNs and preview the existing protocols used to achieve time synchronization.

Chapter 4: Presents the firefly synchronization phenomenon and it's related mathematical model.

Chapter 5: Explains the firefly-based RFA algorithm.

Chapter 6: The simulation of the firefly algorithm using prowler simulator and results analysis.

Chapter 7: Produces the conclusion and recommendations for future work.
2 Ad-hoc Wireless Network

2.1 Ad hoc computer network

Ad hoc is a network connection method which is most often associated with wireless devices. The connection is established for the duration of one session and requires no base station. Instead, devices discover others within range to form a network for those computers. Devices may search for target nodes that are out of range by flooding the network with broadcasts that are forwarded by each node. Connections are possible over multiple nodes (multi-hop ad hoc network). Routing protocols then provide stable connections even if nodes are moving around.

2.2 Wireless Ad-hoc networks

A wireless ad hoc network is a decentralized wireless network. The network is ad hoc because each node is willing to forward data for other nodes, and so the determination of which nodes forward data is made dynamically based on the network connectivity. This is in contrast to wired networks in which routers perform the task of routing. It is also in contrast to managed wireless networks, in which a special node known as an access point manages communication among other nodes.

Figur 2.1 : ad-hoc wireless network
2.2.1 Applications

The decentralized nature of wireless ad hoc networks makes them suitable for a variety of applications where central nodes can't be relied on, and may improve the scalability of wireless ad hoc networks compared to wireless managed networks, though theoretical and practical limits to the overall capacity of such networks have been identified. Minimal configuration and quick deployment make ad hoc networks suitable for emergency situations like natural disasters or military conflicts. The presence of a dynamic and adaptive routing protocol will enable ad hoc networks to be formed quickly.

Wireless ad hoc networks can be further classified by their application:

- mobile ad hoc networks
- wireless mesh networks
- wireless sensor networks

The last type is concerned in this project.

2.3 Wireless sensor network

A wireless sensor network (WSN) is a wireless network consisting of spatially distributed autonomous devices using sensors to cooperatively monitor physical or environmental conditions, such as temperature, sound, vibration, pressure, motion or pollutants, at different locations. The development of wireless sensor networks was originally motivated by military applications such as battlefield surveillance. However, wireless sensor networks are now used in many civilian application areas, including environment and habitat monitoring, healthcare applications, home automation, and traffic control. In addition to one or more sensors, each node in a sensor network is typically equipped with a radio transceiver or other wireless communications device, a small microcontroller, and an energy source, usually a battery. The envisaged size of a single sensor node can vary from shoebox-sized nodes down to devices the size of grain of dust, although functioning 'motes' of genuine microscopic dimensions have yet to be created. The cost of sensor nodes is similarly variable, ranging from hundreds of dollars to a few cents, depending on the size of the sensor network and the complexity required of individual sensor nodes. Size and cost constraints on sensor nodes result in corresponding constraints on
resources such as energy, memory, computational speed and bandwidth. A sensor network normally constitutes a wireless ad-hoc network, meaning that each sensor supports a multi-hop routing algorithm (several nodes may forward data packets to the base station).

2.3.1 Applications

The applications for WSNs are many and varied, but typically involve some kind of monitoring, tracking, and controlling. Specific applications for WSNs include habitat monitoring, object tracking, nuclear reactor control, fire detection, and traffic monitoring. In a typical application, a WSN is scattered in a region where it is meant to collect data through its sensor nodes.

2.3.2 Characteristics

Unique characteristics of a WSN include:

- Limited power they can harvest or store
- Ability to withstand harsh environmental conditions
- Ability to cope with node failures
- Mobility of nodes
- Dynamic network topology
- Communication failures
- Heterogeneity of nodes
- Large scale of deployment
- Unattended operation

Sensor nodes can be imagined as small computers, extremely basic in terms of their interfaces and their components. They usually consist of a processing unit with limited computational power and limited memory, sensors (including specific conditioning circuitry), a communication device (usually radio transceivers or alternatively optical), and a power source usually in the form of a battery.
2.4 Energy Awareness

Energy is the scarcest resource of WSN nodes, and it determines the lifetime of the networks. WSNs are meant to be deployed in large numbers in various environments, including remote and hostile regions, with ad-hoc communications as key. Sometimes often more than 50 percent of energy is used for idle listening. Data transmission is also a large energy consuming operation. For this reason, algorithmic research in WSN mostly focuses on the study and design of energy aware algorithms for idle listening and data transmission from the sensor nodes to the base stations. Data transmission is usually multi-hop (from node to node, towards the base stations), due to the polynomial growth in the energy-cost of radio transmission with respect to the transmission distance. In order to reduce power consumption caused by idle listening, it is necessary to turn off the transceiver module as much as possible. To achieve that purpose, this project produces a mechanism based on biologically inspired synchronization approach.
3 Synchronization in WSN

3.1 Synchronization in Networks

3.1.1 Synchronization Definition

Synchronization (or Sync) is a problem in timekeeping which requires the coordination of events to operate a system in unison. The familiar conductor of an orchestra serves to keep the orchestra in time. Systems operating with all their parts in synchrony are said to be synchronous or in sync. Some systems may be only approximately synchronized, or plesiochronous. For some applications relative offsets between events need to be determined, for others only the order of the event is important.

3.1.2 Synchronization in Telecommunications

Many services running on modern digital telecommunications networks require accurate synchronization for correct operation. For example, if switches do not operate with the same rate clocks then slips will cause degrading performance. Telecommunications networks rely on the use of highly accurate Primary Reference Clocks which are distributed network wide using Synchronization Links and Synchronization Supply Units.

3.1.3 Synchronization in Networks

Network synchronization is a generic concept that depicts a way of distributing common time and frequency references to all the nodes of a network, to align their respective time and frequency scales. The role of network synchronization, especially in the context of frequency accuracy, has grown in importance since the late 1970s, when digital transmission and switching were introduced. Today it strongly influences the quality of most operator services.

To prevent loss of information when digital signals are being transported over a link, the receiving end must either operate at the same average frequency as the transmitting end (digital exchanges) or with spare bandwidth capacity (transport network equipment). This applies to every network node involved in transport between the transmitter and receiver. The task of keeping entities operating at the same average frequency (and within proper quality boundaries) is referred to as
network synchronization. If synchronization is poor—that is, if the network nodes do not operate at the same average frequency—impairments, such as slips and bit errors, will occur.

### 3.1.4 Synchronicity and Synchronization

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 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 and in multi-hop routing protocols to save energy.

### 3.2 Synchronization in WSN

Wireless sensor network applications, similarly to other distributed systems, often require a scalable time synchronization service enabling data consistency and coordination.

#### 3.2.1 Approaches to Time Synchronization

Time synchronization algorithms providing a mechanism to synchronize the local clocks of the nodes in the network have been extensively studied in the past. Two of the most prominent examples of existing time synchronization protocols developed
for the wireless sensor network domain are the Reference Broadcast Synchronization (RBS) algorithm and the Timing-sync Protocol for Sensor Networks (TPSN).

In the RBS, a reference message is broadcasted. The receivers record their local time when receiving the reference broadcast and exchange the recorded times with each other. The main advantage of RBS is that it eliminates transmitter-side non-determinism. The disadvantage of the approach is that additional message exchange is necessary to communicate the local time-stamps between the nodes. To our best knowledge the algorithm has not been extended to large multi-hop networks. The TPSN algorithm first creates a spanning tree of the network and then performs pairwise synchronization along the edges. Each node gets synchronized by exchanging two synchronization messages with its reference node one level higher in the hierarchy. The TPSN achieves two times better performance than RBS by time-stamping the radio messages in the MAC layer of the radio stack and by relying on a two-way message exchange. The shortcoming of TPSN is that it does not estimate the clock drift of nodes, which limits its accuracy, and does not handle dynamic topology changes.

To overcome the inadequacies of the previous protocols, a very well designed protocol know as Flooding Time Synchronization Protocol is produced and so soon became highly employed to achieve synchronization.

### 3.2.2 Flooding Time Synchronization Protocol (FTSP)

The goal of the FTSP is to achieve a network wide synchronization of the local clocks of the participating nodes. Each node is assumed to have a local clock exhibiting the typical timing errors of crystals and can communicate over an unreliable but error corrected wireless link to its neighbors. The FTSP synchronizes the time of a sender to possibly multiple receivers utilizing a single radio message time-stamped at both the sender and the receiver sides. MAC layer time-stamping can eliminate many of the errors. However, accurate time-synchronization at discrete points in time is a partial solution only. Compensation for the clock drift of the nodes is inevitable to achieve high precision in-between synchronization points and to keep the communication overhead low. Linear regression is used in FTSP to compensate for clock drift. Typical WSN operate in areas larger than the broadcast range of a single node; therefore, the FTSP provides multi-hop synchronization. The root of the
network - a single, dynamically (re)elected node - maintains the global time and all other nodes synchronize their clocks to that of the root. The nodes form an ad-hoc structure to transfer the global time from the root to all the nodes. This is robust against node and link failures and dynamic topology changes.

3.2.3 Comparing the Algorithms

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. The dynamics of these protocols in terms of robustness to topology changes and node population have not been widely studied.

3.3 Medium Access Control (MAC) Layer

The Media Access Control (MAC) data communication protocol sub-layer, also known as the Medium Access Control, is a sublayer of the data link layer specified in the seven-layer OSI model (layer 2). It provides addressing and channel access control mechanisms that make it possible for several terminals or network nodes to communicate within a multipoint network, typically a local area network (LAN) or metropolitan area network (MAN). The MAC sub-layer acts as an interface between the Logical Link Control (LLC) sublayer and the network's physical layer. The MAC layer emulates a full-duplex logical communication channel in a multipoint network. This channel may provide unicast, multicast or broadcast communication service. The task of energy reduction is assumed to be assigned to this layer. Some MAC protocols have already incorporated such a concept (e.g. S-MAC, T-MAC, etc.).
4 Fireflies Synchronization

In certain parts of South-East Asia alongside riverbanks, male fireflies gather on trees at dawn, and start emitting flashes regularly. Over time synchronization emerges from a random situation, which makes it seem as though the whole tree is flashing in perfect synchrony. This phenomenon forms an amazing spectacle, and has intrigued scientists for several hundred years. Over the years, two fundamental questions have been studied: Why do fireflies synchronize? And how do they synchronize? The first question led to many discussions among biologists.

In all species of fireflies, emissions of light serves as a means of communication that helps female fireflies distinguish males of its own species: the response of male fireflies to emissions from females is different in each species. However it is not clear why in certain species of fireflies, males synchronize. Several hypothesis exist: Either it could accentuate the males rhythm or serve as a noise-reduction mechanism that helps them identify females. This phenomenon could also enable small groups of males to attract more females, and act as a cooperative scheme.

4.1 Phenomena of self-organization

Early hypotheses had difficulties explaining the firefly synchronization phenomenon. For example, Laurent in 1917 dismissed what he saw and attributed the phenomenon
to the blinking of his eyelids. Others argued that synchrony was provoked by a single stimulus received by all fireflies on the tree. However the presence of a leading firefly or a single external factor is easily dismissed by the fact that not all fireflies can see each other and fireflies gather on trees and progressively synchronize. The lack of a proper explanation until the 1960s is mostly due to a lack of experimental data. Among early hypotheses, Richmond stated in 1930 what came very close to the actual process: “Suppose that in each insect there is equipment that functions thus: when the normal time to flash is nearly attained, incident light on the insect hastens the occurrence of the event. In other words, if one of the insects is almost ready to flash and sees other insects flash, then it flashes sooner than otherwise. On the foregoing hypothesis, it follows that there may be a tendency for the insects to fall in step and flash synchronously.” This statement identifies that synchronization among fireflies is a self-organized process, and fireflies influence each other: they emit flashes periodically, and in return are receptive to the flashes of other males. The internal clock of a firefly, which dictates when a flash is emitted, is modeled as an oscillator, and the phase of this oscillator is modified upon reception of an external flash. In general this type of oscillator is termed relaxation oscillators, which are not represented by a typical sinusoidal form but rather by a series of pulses.

Different models inspired by such a biological synchronization were proposed in the last decades. The most important one are the phase-advance respectively the phase-delay model from Buck, the Pulse-coupled Biological Oscillators (PCO) model from Mirollo and Strogatz (also called MaS model), and the RFA model. Other models are similar, but make different assumptions on the coupling strength and the propagation delay in wireless communication.

4.2 Mathematical Model (MaS Model)

In the Mirollo and Strogatz (MaS) 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 4.1(a,b)). The amount of adjustment is determined by the function $f(t)$, which is called the firing function, and the parameter, 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

$$t'' = f^{-1}(f(t') + \varepsilon) \quad (4.1)$$

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 $\varepsilon$ 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 $t = t'$.

![Figure 4.1: Time evolution of the phase function](image)

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. 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).
4.3 Pulse-Coupled Oscillators

Fireflies can simply be abstracted as oscillators that emit pulses of light periodically. This type of oscillators is referred to as "Pulse Coupled Oscillators", and is also used to study biological systems such as neurons and earthquakes. Mirollo and Strogatz analyzed spontaneous synchronization phenomena and also derived a theoretical framework based on pulse-coupled oscillators for the convergence of synchrony.

When coupled to others, an oscillator is receptive to the pulses of its neighbors. Coupling between nodes is considered instantaneous, and when a node $j$ $(1 \leq j \leq N)$ fires at $t = \tau_j$, i.e. $\Phi_j (\tau_j) = \Phi_{th}$, all nodes adjust their phase as follows:

1. $\Phi_j (\tau_j) = 0$
2. $\Phi_i (\tau_j) = \Phi_i (\tau_j) + \Delta \Phi (\Phi_i (\tau_j))$ for $i \neq j$

By appropriate selection of $\Delta \Phi$, a system of $N$ identical oscillators forming a fully-meshed network is able to synchronize their firing instants within a few periods. The phase increment $\Delta \Phi$ is determined by the Phase Response Curve (PRC). For their mathematical demonstration, Mirollo and Strogatz derive that synchronization is obtained whenever the firing map $x_i(t) = f(\Phi_i (t))$ is concave up and the return map $\Phi_i (t) + \Delta \Phi (\Phi_i (t)) = g(x_i(t) + \varepsilon)$, where $\varepsilon$ is the amplitude increment, is its inverse.

The resulting operation $\Phi_i (t) + \Delta \Phi (\Phi_i (t)) = g(f(\Phi_i (t)))$ yields the PRC, and is a piecewise linear function:

$$\Phi_i (\tau_j) + \Delta \Phi (\Phi_i (\tau_j)) = \min (\alpha \cdot \Phi_i (\tau_j) + \beta, 1)$$

with:

$$\alpha = \exp(b \cdot \varepsilon)$$
$$\beta = \frac{\exp(b \cdot \varepsilon) - 1}{\exp(b) - 1}$$

where $b$ is the dissipation factor. Both factors $\alpha$ and $\beta$ determine the coupling between oscillators and are identical for all. The threshold $\Phi_{th}$ is normalized to 1.

Interestingly the synchronization scheme relies on the instant of arrival of a pulse, and receivers adjusting their phases when detecting this pulse. Interference in the typical way is not observed, and two pulses emitted simultaneously can superimpose constructively. This helps a faraway receiver to detect the superimposed pulses, and to synchronize with the rest of the network.
Chapter 5

The Reachback Firefly Algorithm (RFA)

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

5.2 Reachback Firefly Algorithm

The RFA was introduced in 3rd international conference on Embedded Networked Sensor Systems, November 2005, and is based on the MaS model, but with the difference that it is more appropriate for the implementation in wireless networks. Some assumptions resulting from the theoretical MaS model are listed below and make it difficult for a practical application.

1. The oscillators have identical dynamics (e.g. same period).
2. Nodes can instantaneously fire.
3. Every firing event must be observed immediately (no loss).
4. All computations are performed perfectly and instantaneously.

The most problematic one is the third assumption, because in reality the firing is implemented as a broadcast message which usually has an unpredictable delay mainly caused by the channel contention prior to the transmission. The other assumptions are not so problematic. For example, the nodes are based on oscillators which generally have a small but measurable drift. Next, the limits in floating point arithmetic result in inexact computations and finally the loss of messages in wireless networks due to influence problems or the varying link quality can not be avoided. For this reason, the RFA model controls the unpredictable message delay via MAC-layer time-stamping, so that the receiver has knowledge about the MAC delay and consequently is able to determine the correct firing time. In addition, the RFA model is based on an explicitly added random transmission delay between 0 and constant D at the application level, called message staggering delay. The message staggering delay should avoid message collisions, in the case if many synchronized nodes want to send simultaneously. In contrast to the MaS model, the RFA model does not immediately react on an observed firing event. Instead it stores the corrected timestamp of all received firing events in a queue. Then, if the phase
of a node reaches the end, it computes the new start phase based on the content of the queue. The computation is the same as in the MaS model. As a result, a node seems to react instantaneously, but to the data from the last period. The algorithm for calculating the starting phase is showed below.

**Algorithm 1** Calculation of the start phase.

1. `overall_phasejump = 0`
2. while queue not empty do
3.   `tStamp = next element from queue {get next timestamp}`
4.   `phasejump = Δ(tStamp + overall_phasejump)`
5.   `overall_phasejump = overall_phasejump + phasejump`
6. while end
7. set current phase to `overall_phasejump`

5.3 The Algorithm Explanation

In order to make it easier to understand the algorithm working scheme, an the following example illustrates how does it work through figure 5.1.

![Fig 5.1 RFA mechanism](image)

Example: 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 time-stamping 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.
5.3.1 Preemptive Message Staggering

Carrier Sense Multiple Access (CSMA) schemes attempt to avoid channel collisions by causing nodes to backoff 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 << T) before processing the queue so that delayed messages from synchronized nodes are received.

5.3.2 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 (4.1) 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.

\[
\Delta(t') = \epsilon t' \quad (5.1)
\]
6 Simulation

In order to simulate our algorithm, a MATLAB based wireless sensor network simulator called PROWLER is used.

6.1 Prowler: Probabilistic Wireless Network Simulator

The probabilistic wireless network simulator (Prowler) is an event-driven simulator that can be set to operate in either deterministic mode (to produce replicable results while testing the application) or in probabilistic mode that simulates the nondeterministic nature of the communication channel and the low-level communication protocol of the motes. It can incorporate arbitrary number of motes, on arbitrary (possibly dynamic) topology, and it was designed so that it can easily be embedded into optimization algorithms. The simulator runs under MATLAB, thus it provides a fast and easy way to prototype applications, and has nice visualization capabilities. The graphical user interface of Prowler is shown in the figure (6.1). The network simulator models the important aspects of all levels of the communication channel and the application. The nondeterministic nature of the radio propagation is characterized by a probabilistic radio channel model. A simplified, but accurate model is used to describe the operation of the MAC layer. The applications interact with the MAC layer through a set of events and actions.
Fig 6.1: PROWLER interface

What do the colors mean?
- Small red dots indicate motes with pending transmission.
- Big red dots indicate transmitting motes.
- Small green dots indicate receiving motes.
- Green LED is toggled when a message is received successfully.
- Yellow LED is toggled when a collided message is received.

6.2 Network topology and parameters

6.2.1 Network topology

The network topology chosen for this experiment is a mesh topology, also known as peer-to-peer topology, improves reliability and scalability by multipath routing and therefore can be ad-hoc, self-organizing, and self-healing. Moreover, messages can also be routed through multiple hops which make this topology appropriate for wireless sensor networks. Other applications that benefit from this may be industrial control and monitoring, or asset and inventory tracking.
This topology is represented by a grid of nodes in two dimensions (Fig 6.2). One of the motes initiates the transmission, and all the receiving motes retransmit the first received message with a probability of $p$ (probabilistic flood). The transmission signal strength $s$ can be set within a range. (The values $p$ and $s$ are the same on each mote.) The goal is to compare the performance of the FTSP protocol to our firefly synchronization method.

![Simulation Diagram](image)

**Fig 6.2:** grid network topology (5 x 5) and parameters

### 6.2.2 Simulation Parameters

The implementation of the naturally inspired synchronization algorithm has several important parameters which determine the behavior of the fireflies. The most important ones are described below. Figure 6.2 shows part of these parameters.

**Probability of retransmitting $P$:** takes value between 0.1 and 1 and determine the probability of retransmitting the packet by any receiving node.

**Signal Strength $S$:** The ideal received signal strength is $P_{\text{rec id}} = P_{\text{transmit}} \cdot f(x)$

$f(x)$ is the ideal propagation function. The ideal propagation function defines the signal strength vs. the distance between the transmitter and
receiver, where $x$ is the distance.

\[ f(x) = \frac{1}{1 + x^2} \]

**Starting Mote:** The starting node can be selected. It's better to choose one of the four corner nodes to be the starting node. (node no. 1 is at the bottom left corner)

**X_Number, Y_Number:** are used to change the topology.

**Distance $x$:** the distance between a node and its neighbor node.

**Radio transmission parameters:**

**Fading Effect:** The fading is modeled by adding noise components to the received signal strength. The received signal strength is

\[ P_{\text{rec}} = P_{\text{rec id}} * (1 + \alpha(x)) * (1 + \beta(t)) \]

where $\alpha$ and $\beta$ are random variables with normal distributions $N(0, s_{\alpha})$ and $N(0, s_{\beta})$, respectively.

**Transmission Error ($P_{\text{error}}$):** models all other sources/effects that may lead to an unsuccessful transmission.

**Reception Limit:** The signal from node $k$ is successfully received if

\[ \text{received powr}_k > \text{reception limit} \]

during the time of the reception.

**MAC-layer model:** The MAC-layer model uses 5 parameters:

- $\text{min\_waiting\_time}$
- $\text{var\_waiting\_time}$
- $\text{in\_backoff\_time}$
- $\text{var\_backoff\_time}$
- $\text{packet\_length}$

The waiting time between channel request and channel idle check is between $\text{min\_waiting\_time}$ and $\text{min\_waiting\_time} + \text{var\_waiting\_time}$.

Similarly, the backoff time after an unsuccessful idle check is between $\text{min\_backoff\_time}$ and $\text{min\_backoff\_time} + \text{var\_backoff\_time}$. The length of a single transmission is $\text{packet\_length}$.

![Fig 6.3: MAC layer communication scheme](image)

All the MAC-layer parameters are measured in bit-times, where one bit-time is the length of one transmitted bit in the radio channel, one bit-time equals to $1/40000 \text{ sec.}$
6.3 Firefly Algorithm Implementation

In order to achieve the objective of comparing between several synchronization algorithms, the RFA algorithm is implemented into MATLAB code and then included in the Prowler simulator.

6.3.2 Parameters Setting

Before running the simulator, the general parameters values must be predetermined and unified for both experiments of synchronization algorithms. Table 6.1 contains the assigned values of parameters.

<table>
<thead>
<tr>
<th>Table 6.1: Parameters Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>$P$</td>
</tr>
<tr>
<td>$S$</td>
</tr>
<tr>
<td>starting node</td>
</tr>
<tr>
<td>$x_{\text{number}}$</td>
</tr>
<tr>
<td>$y_{\text{number}}$</td>
</tr>
<tr>
<td>distance</td>
</tr>
<tr>
<td>$s_{\alpha}$</td>
</tr>
<tr>
<td>$s_{\beta}$</td>
</tr>
<tr>
<td>$p_{\text{error}}$</td>
</tr>
<tr>
<td>reception limit</td>
</tr>
<tr>
<td>min_waiting_time</td>
</tr>
<tr>
<td>rand_waiting_time</td>
</tr>
<tr>
<td>min_backoff_time</td>
</tr>
<tr>
<td>var_backoff_time</td>
</tr>
<tr>
<td>packet_length</td>
</tr>
</tbody>
</table>
6.3.3 Simulation Results

Firstly, the Flood algorithm simulation was run with previous parameters as shown in figure 6.3. After that, the Firefly algorithm simulated with the same parameters (figure 6.4). The results from the two experiments are denoted in table 6.2.

Fig 6.3 : Flood algorithm simulation
Fig 6.4 : Firefly algorithm simulation

Table 6.2 : Simulation results

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Firefly</th>
<th>FTSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>time to sync.</td>
<td>8.3 sec</td>
<td>8.1 sec</td>
</tr>
<tr>
<td>sent messages</td>
<td>44</td>
<td>48</td>
</tr>
<tr>
<td>received messages</td>
<td>311</td>
<td>421</td>
</tr>
<tr>
<td>received collided messages</td>
<td>291</td>
<td>254</td>
</tr>
<tr>
<td>sending nodes</td>
<td>44</td>
<td>48</td>
</tr>
<tr>
<td>receiving nodes</td>
<td>97</td>
<td>100</td>
</tr>
</tbody>
</table>

6.4 Performance Optimization

The presented algorithm is supposed to be used for optimizing the performance of the sensors network. To achieve that goal, the next two experiments were deployed.
6.4.1 Broadcasting Performance

This experiment is a broadcast in a network of 100 nodes (10 x10 grid). One of the nodes initiates the transmission, and all the receiving nodes retransmit the first received message with a probability of $p$. The transmission signal strength $s$ can be set within a range. (The values $p$ and $s$ are the same on each mote.).

The goal is to maximize the overall performance of the network by finding the optimal $p$ and $s$ parameters. The performance metric is composed from the number of receiving nodes in the network (the more nodes receive the better) and the consumed power (the less power is used the better):

$$\begin{align*}
E_2 &= \lambda_1 (100 - N_{rec})^2 + \lambda_2 N_{tr} S
\end{align*} \quad (6.1)
$$

where the first term is the error when not all the motes receive the message, while the second is the estimation of the total consumed power, and :

$E_2$: error surface (cost function).

$N_{rec}$: number of receiving nodes.

$N_{tr}$: number of sending nodes.

$S$: signal power.

$\lambda_1, \lambda_2$: scaling factors.

The exhaustive search method was used to find the minimum of the error surface. The surface was evaluated in 10x12 points in the regions of $P = [0.1, 1]$, $S = [0.1, 3]$; in each point 10 experiments were run. The generated error surface is shown in figure (6.5). The best performance was found when $P = 0.3$ and $S = 1.0$.

Some of the other points at the bottom of the canyon-shaped surface would give almost equally good results, while outside that region the performance drastically decreases.
6.4.2 Connection Links Formation

The goal is to find the optimal value of $P$ in order to achieve optimal performance. The performance metric is composed of the necessary time to connect the nodes, and the consumed total power. The nodes are considered to be connected if more than 90% of them are connected. The time to connect the nodes ($T$) and the total number of sent messages ($M$) are combined to a performance metric:

$$E_1 = T + M \lambda$$  \hspace{1cm} (6.2)

Where the scaling factor $\lambda = 1/200$.

The cost function $E_1$ was calculated with 50 motes placed in a 5 x 10 grid. Twelve experiments were made for different values of $P$ in the range, $P = [0.01, 0.9]$. The results are shown in Figure (6.6). Where the upper curve shows the mean run time ($T$), the second plot is the mean total message number ($M$) and the third one is for the error surface ($E_1$).
The results showed that the optimum value was $P = 0.1$. With this setting the algorithm connects the nodes in approximately four seconds by sending 350 messages on average.

![Graphs showing mean run time, mean message number, average cost function, and number of experiments.](image)

**Fig 6.6**: results of second experiment

### 6.5 Analysis and Comments

After running the simulation for so many times (more than 200 times), a full view is now available to discuss the output results, simulator and Matlab limitation.
6.5.1 Results Analysis

Before analyzing the results, one more calculation should be done to proceed with the whole view. This was done using equation (6.1) with to calculate both terms of the formula (arbitrary variables \( \lambda_1=1/100, \lambda_2=1/10 \)). Table 6.3 contains the final outcome.

<table>
<thead>
<tr>
<th>Performance Metrics</th>
<th>Firefly</th>
<th>FTSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receiving Error</td>
<td>0.09</td>
<td>0</td>
</tr>
<tr>
<td>Power Consumption</td>
<td>4.4</td>
<td>4.8</td>
</tr>
</tbody>
</table>

- From the table above, it's easy to figure out that the firefly protocol is less than FTSP protocol in accuracy and better in power consumption. These conclusions are matching the inferences of other related works (mentioned in references).
- Compared to algorithms such as RBS, TPSN and FTSP, 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 needs to be 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.
- Although the gotten results are not very accurate, it still give a strong base for comparison objective and provided the necessary graphs which used to determine optimum values of some parameters. Those values could be used in case of identical circumstances.
6.5.2 Simulator limitations
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.
Another important point is that choosing to simulate along with animation will lead to longer sync time, sometimes nearly double time needed. The previous experiments were tested with no animation option.
Despite these limitations, the simulator is useful for exploring the partial parameters space of the algorithm and produce results in a simple manner.

6.5.3 MATLAB Limitations
The MATLAB is not considered to be specialized in networking or telecommunications scope, but still modeling, simulation and algorithms development part of its typical uses. So, when it comes to the level of programming the sensors of the wireless network, MATLAB language is not a choice up to now. The most well-known simulation and programming environment for WSNs is TinyOS, as described further in future work section.
7 Conclusion

7.1 Conclusion

In this thesis an alternative synchronization approach based on the natural behavior of fireflies was introduced. The results show that the synchronization approach rules are particularly simple and well suited for a deployment in ad hoc networks.

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.

Due to the fact that in reality many sensor networks eventually form unidirectional communication paths, this algorithm is not really applicable for sensor networks which have to support high dependability and availability. Otherwise if such a network is only used to gather data where it is not dramatic if some messages are lost (e.g., meteorological station for temperature measurement), then the use of such a synchronization scheme could be a choice.

7.2 Future Work

It's strongly recommended for future work to simulate the presented algorithm using a more powerful simulator called TOSSIM.

TinyOS is a free and open source component-based operating system and platform targeting WSNs. It is an embedded operating system written in the nesC programming language as a set of cooperating tasks and processes. TOSSIM is the TinyOS simulator. This simulator provides a large number of parameters which is highly important to study synchronization algorithms and protocols. It might face the researcher some difficulties in installing the system because it's setup process passes through many steps which may be hard on the normal pc users. The simulator is available on the system official website: http://www.tinyos.net.

It's so expected that the firefly algorithm will be integrated with FTSP or TPSN to produce efficient, scalable and a comprehensive solution to the problem of time synchronization in sensor networks.
References


- Robert Leidenfrost, "Establishing Wireless Time-Triggered Communication using a Firefly Clock Synchronization Approach".


- Miklós Maróti, Branislav Kusy, Gyula Simon, Ákos Lédeczi, "The Flooding Time Synchronization Protocol".

- http://www.isis.vanderbilt.edu/Projects/nest/prowler/

- http://www.wikipedia.org
function application(S)
    S;
    S;  persistent app_data
    S;  global ID t
    S;  [t, event, ID, data]=get_event(S);
    S;  [topology, mote_IDs]=prowler('GetTopologyInfo');
    S;  ix=find(mote_IDs==ID);
    S;  if ~strcmp(event, 'Init_Application')
        S;      try memory=app_data{ix}; catch memory=[]; end,
    S;  end
    S;

    SENDER_ID=sim_params('get_app', 'Start_Mote');
    if isempty(SENDER_ID), SENDER_ID=1; end
    switch event
        case 'Init_Application'
            S;      signal_strength=1;

            S;      memory=struct('signal_strength', signal_strength, 'parent', -inf, 'hops', inf);

            if ID==SENDER_ID
                S;      Set_Clock(1000)
            end
            PrintMessage('')
        case 'Packet_Sent'

case 'Packet_Received'
    p = sim_params('get_app', 'P');
    if isempty(p); p = .3; end

    if memory.parent < 0
        memory.parent = data.data.ID;
        memory.hops = data.data.hops + 1;
        if rand < p
            Send_Packet(radiostream(struct('time', t, 'ID', ID, 'hops', memory.hops),
            memory.signal_strength));
        end
    end
    PrintMessage([num2str(memory.parent) '/' num2str(memory.hops)])
    DrawLine('Arrow', memory.parent, ID, 'color', [0 0 0])
    if memory.hops < 3
        LED('red on')
    elseif memory.hops < 6
        LED('green on')
    else
        LED('yellow on')
    end
end

case 'Collided_Packet_Received'
    % this is for debug purposes only

case 'Clock_Tick'
    if ID == SENDER_ID
        memory.parent = 0;
        memory.hops = 0;
        Send_Packet(radiostream(struct('time', t, 'ID', ID, 'hops', 0),
        memory.signal_strength));
    end

case 'GuiInfoRequest'

Appendix A  Matlab Code
if ~isempty(memory)
    disp(sprintf('Memory Dump of mote ID# %d:\n',ID)); disp(memory)
else
    disp(sprintf('No memory dump available for node %d:\n',ID));
end

case 'Application_Stopped'
    % this event is called when simulation is stopped/suspended

case 'Application_Finished'
    % this event is called when simulation is finished

otherwise
    error(['Bad event name for application: ' event])
end

S;
S;        app_data{ix}=memory;
S;

function b=Send_Packet(data);
global ID t
radio=prowler('GetRadioName');
b=feval(radio, 'Send_Packet', ID, data, t);

function b=Set_Clock(alarm_time);
global ID
prowler('InsertEvents2Q', make_event(alarm_time, 'Clock_Tick', ID));

function PrintMessage(msg)
global ID
prowler('TextMessage', ID, msg)

function LED(msg)
global ID
prowler('LED', ID, msg)

function DrawLine(command, varargin)
switch lower(command)
    case 'line'
        prowler('DrawLine', varargin{:})
    case 'arrow'
        prowler('DrawArrow', varargin{:})
    case 'delete'
        prowler('DrawDelete', varargin{:})
    otherwise
        error('Bad command for DrawLine.')
end

function param=params;

    param(1).name='P';           param(1).default=0.5;
    param(2).name='Start_Mote';  param(2).default=1;
    param(3).name='X_Number';    param(3).default=10;
    param(4).name='Y_Number';    param(4).default=10;
    param(5).name='Distance';    param(5).default=1;

function [topology,mote_IDs]=topology(varargin);

    ix=1;t=[];
    Nx =sim_params('get_app', 'X_Number'); if isempty(Nx),  Nx=10; end
    Ny =sim_params('get_app', 'Y_Number'); if isempty(Ny),  Ny=10; end
    dist=sim_params('get_app', 'Distance'); if isempty(dist), dist=1; end
\begin{verbatim}
X=1:dist:(Nx-1)*dist+1;
Y=1:dist:(Ny-1)*dist+1;
for i=X
    for j=Y
        t=[t; i,j];
    end
end
topology=t;
mote_IDs=1:Nx*Ny;

function x=animation
    persistent anim_data
    if isempty(anim_data)
        small=5; medium=20; large=50;
    end
    % Event_name Animated Color/{on/off/toggle} Size
    anim_def={...}
        {'Init_Application', 1, [0 0 0], small}, ...
        {'Packet_Sent', 0, [1 0 0], small}, ...
        {'Packet_Received', 0, [1 0 0], small}, ...
        {'Collided_Packet_Received', 0, [1 0 0], small}, ...
        {'Clock_Tick', 0, [0 0 0], small}, ...
        {'Channel_Request', 0, [0 0 0], small}, ...
        {'Channel_Idle_Check', 1, [1 0 0], small}, ...
        {'Packet_Receive_Start', 0, [0 1 0], small}, ...
        {'Packet_Receive_End', 0, [0 0 0], small}, ...
        {'Packet_Transmit_Start', 1, [1 0 0], medium}, ...
        {'Packet_Transmit_End', 1, [0 1 0], small}};

    for i=1:length(anim_def)
        a=anim_def{i};
        if i==1
            anim_data=struct('event', a{1}, 'animated', a{2}, 'color', a{3}, 'size', a{4});
\end{verbatim}
else
    anim_data(i)=struct('event', a{1}, 'animated', a{2}, 'color', a{3}, 'size', a{4});
end
end
end
x=anim_data;

function varargout=simstats

global global_event_Q

[topology, mote_IDs, void]=prowler('GetTopologyInfo');
N=length(mote_IDs);

for i=1:N
    node_stat(i)=struct(...
        'Sent_Messages', 0, ...
        'Received_Messages', 0, ...
        'Received_Collided_Messages', 0, ...
        'Send_Times', [], ...
        'Receive_Times', [], ...
        'Collide_Times', []);
end

L=length(global_event_Q);

for i=1:L
    t=global_event_Q(i).time;
    e=global_event_Q(i).event;
    id=global_event_Q(i).ID;
    ix=find(mote_IDs==id);
    switch e
    case 'Packet_Transmit_Start'
node_stat(ix).Sent_Messages = node_stat(ix).Sent_Messages + 1;
node_stat(ix).Send_Times = [node_stat(ix).Send_Times, t];

case 'Packet_Received'
    node_stat(ix).Received_Messages = node_stat(ix).Received_Messages + 1;
    node_stat(ix).Receive_Times = [node_stat(ix).Receive_Times, t];
case 'Collided_Packet_Received'
    node_stat(ix).Received_Collided_Messages = node_stat(ix).Received_Collided_Messages + 1;
    node_stat(ix).Collide_Times = [node_stat(ix).Collide_Times, t];
otherwise
    % not handled; can be extended
end
end

sys_stat = struct(...
    'Sent_Messages', 0, ...,
    'Received_Messages', 0, ...,
    'Received_Collided_Messages', 0, ...,
    'First_Send_Time', inf, ...,
    'First_Receive_Time', inf, ...,
    'Last_Send_Time', -inf, ...,
    'Last_Receive_Time', -inf, ...,
    'Sending_Nodes', 0, ...,
    'Receiving_Nodes', 0);

for i=1:N
    sys_stat.Sent_Messages = ... + node_stat(i).Sent_Messages;
    sys_stat.Received_Messages = ... + node_stat(i).Received_Messages;
    sys_stat.Received_Collided_Messages = ...

end
sys_stat.Received_Collided_Messages +
node_stat(i).Received_Collided_Messages;

sys_stat.First_Send_Time = ...
    min([sys_stat.First_Send_Time, node_stat(i).Send_Times]);
sys_stat.First.Receive.Time = ...
    min([sys_stat.First.Receive.Time, node_stat(i).Receive_Times]);

sys_stat.Last_Sent_Time = ...
    max([sys_stat.Last_Sent_Time, node_stat(i).Send_Times]);
sys_stat.Last.Receive.Time = ...
    max([sys_stat.Last.Receive.Time, node_stat(i).Receive_Times]);

sys_stat.Sending_Nodes = ...
    sys_stat.Sending_Nodes + (node_stat(i).Sent_Messages > 0);
sys_stat.Receiving_Nodes = ...
    sys_stat.Receiving_Nodes + (node_stat(i).Received_Messages > 0);
end

if nargout==0
    disp(sys_stat)
else
    varargout={sys_stat, node_stat};
end