Simulating and Testing Mobile Wireless Sensor Networks

David J. Anthony †, William P. Bennett †, Mehmet C. Vuran †, Matthew B. Dwyer †,
Sebastian Elbaum †, Felipe Chavez-Ramirez ‡
†Department of Computer Science and Engineering
University of Nebraska-Lincoln, Lincoln, NE 68588
‡Whooping Crane Maintenance Trust, Wood River, NE 68883
†{danthony,wbennett,mcvuran,dwyer,elbaum}@cse.unl.edu
‡fchavez@whoopingcrane.org

ABSTRACT

Developing applications for wireless sensor networks (WSNs) can provide many challenges. Environmental conditions have a large impact on the behavior of an application, but it may not be feasible to replicate the conditions of the deployment environment while creating the application. Furthermore, long-term deployment of monitoring applications require extensive pre-deployment analysis of such applications since the sensors cannot be accessed after their deployment. Through a combination of simulation and software engineering practices, it is possible to rigorously test and validate the software for WSNs. In this paper, several methods for simulating distributed mobile WSNs and testing the software are provided. These methods are used in the development of a WSN that was deployed to track Whooping Cranes during their year long migration.

Categories and Subject Descriptors

C.2.1 [**Computer-Communications Networks**]: Network Architecture and Design—*Wireless Communications*; D.2.4 [**Software Engineering**]: Software/Program Verification—*Validation*

General Terms

Experimentation, Performance, Verification

Keywords

wireless sensor networks, testing, reliability

1. INTRODUCTION

Developing long-term applications for wireless sensor networks (WSNs) poses several challenges. With the improvement of microcontrollers for WSN motes, it is possible to develop sophisticated applications with several operation states within limited memory space. Moreover, the integration of higher complexity sensor modules such as GPS units further complicates the energy consumption profile of a sensor mote, which has traditionally been dominated by communication. Hence, evaluation and testing of the *whole* application code is required rather than communication stack-oriented solutions [2, 16].

Even when a fault is detected, it can be difficult to reproduce and debug. The long-term deployment of such applications may make accessing or communicating with the motes a challenge, such as in mobile, underwater, or underground sensor networks [4, 5]. The motes may exhibit faults from physical damage due to exposure to outdoor elements. The dynamic communication channels present in deployed WSNs are difficult to model and can lead to errors that are difficult to reproduce. These challenges call for simulation and validation tools that can (1) capture the complex interactions between the environment, the mote hardware, and its software, (2) provide valuable information about the performance metrics of interest, and (3) provide means to validate the software.

While the mentioned challenges apply to a large class of emerging WSN applications, in this paper, we focus on simulation and testing mobile wireless sensor networks. More specifically, we discuss our solutions for the development of an application to track migratory birds, i.e., Whooping Cranes, through their annual migration. Whooping Cranes are an endangered species of bird that are indigenous to North America. Collecting information on migration paths and environmental influences is of high importance to ecologists working to conserve the remaining bird population. To gather this information, motes are attached to the birds to collect sensor readings and transmit them to strategically placed information sinks.

Developing the Whooping Crane tracking program highlights several of the issues in developing WSN applications. The cranes are inaccessible except for brief windows in August and December. Therefore, the motes must be able to run unattended for almost a full year. In this time, the motes must cope with increasingly limited power supply. During their migration, the birds travel from Texas in the United States to Alberta, Canada. During this migration, part of the population dies at unpredictables times and locations, which means motes may never be recovered. The power constraints and the amount of data to be collected make the application complex and difficult to debug.

In this paper, we address several challenges for the development of the Whooping Crane project using a combination of simulation and run-time monitoring. The run-time monitoring is implemented by weaving in debugging code at compile time and monitoring the application state from within a simulated deployment. The use of a simulated environment allows for accelerated testing that exposes bugs that would otherwise manifest slowly.

The contributions of this paper are as follows: First, a simulation platform is developed to model mobility of whooping cranes and their effects on mote hardware. Second, a method and assessment of integrating run-time monitoring into the WSN development process is presented. Finally, comprehensive evaluations are presented to assess the effects of run-time monitoring solutions on simulation speed, complexity, and development costs compared to post-processing solutions. To the best of our knowledge, this is the first time run-time monitoring tools have been used in conjunction with mobile wireless sensor network applications.

The remainder of this paper is organized as follows: In Section 2, related work on mobile WSNs and verification techniques for WSNs are presented in detail. In Section 3, the motivation and background of the project is presented. The simulation and testing framework is described in Section 4. In Section 5, the empirical and simulation results are presented. Finally, conclusions and future work are discussed in Section 6.

2. RELATED WORK

Several approaches have been taken to develop mobile WSNs and testing and validating the correct operation of embedded systems and wireless sensor networks. In the following, we discuss these existing approaches.

2.1 Mobile WSNs

The recent success of WSN applications has motivated their usage in monitoring animals. Accordingly, WSNs that are characterized by the mobility of these animals have been developed [27]. Since the sensor motes are attached to these animals for a long duration, developing self-sustained solutions is of importance. As an example, within the Dairy Industry, animal lameness poses a large concern. To monitor and detect lameness in cattle, in [9], embedded sensors are implemented to monitor and record the lying patterns of dairy cows to benefit the study of lameness. An embedded accelerometer is used to measure the orientation of the hind leg of a dairy animal based on the relative direction of the gravitational field. A microcontroller with nano-watt power technology facilitates sensor sampling, data recording and time stamping, and interfacing with external hardware systems [9]. However, approaches to validate the developed application have not been reported.

In [8], a similar prediction system has been developed with herding cattle. A virtual fencing system is developed to minimize the cost of building fences to manage the grazing of cattle that can cost up to $20,000 per km. All or some of the cows are outfitted with the system, which has position and orientation sensors as well as sound and shock systems to provide feedback to the cattle. Accordingly, virtual fences can be created, which can be easily moved depending on the conditions of the fields. A directional sound system is used to indicate when a cow is approaching the virtual fence. If the animal does not respond to the sound cue, a small directional shock is applied. To do this, several algorithms are developed to model and predict the motions and reactions of the cattle. Additionally, by identifying leaders of the herd, the motes are placed only on a small percentage of the total herd. While the application results in a mobile WSN, the movements of the cows are limited in terms of both time and overall area.

In this paper, we are interested in developing long-term monitoring techniques for migratory birds using sensor motes. The resulting network is a highly dynamic and mobile network. Since the sensor motes cannot be accessed for at least 8-9 months after deployment, effective evaluation and validation techniques are necessary to measure the performance and robustness of the solutions.

2.2 Testing and Verifying WSNs

Even though testing and validating WSNs is in its infancy, recently, there have been several approaches developed. Several attempts have been made to run the mote software inside of a simulator and create emulated environmental phenomena to test the application [25]. In [20], static code analysis is utilized to guarantee program correctness. Run-time monitoring of motes has been attempted by running mote software inside of a simulator or on a mote [18, 23, 21, 17]. These approaches suffered from long execution times from exhaustively checking the program execution space, or missed potential faults by simulating an ideal mote environment.

To test the Crane Sensing application, two testing approaches are taken in this paper: (1) Log file analysis and (2) Aspect-oriented programming. Log file analysis separates program execution processes from fault detection methods. In this testing method, the program is modified to output data to some storage mechanism so that it can be analyzed by a separate process without interrupting the execution of the original program [6]. The analyzer program is a state machine that uses the outputs of the original application as inputs. The analyzer program decides that the application was correct if its state machine accepts the input, otherwise it rejects it.

As shown in [6], log file analysis can be a powerful tool to test software against formal specifications, but it does have some drawbacks. Developing the analyzer and guaranteeing its correctness can be difficult. The size of the of generated log files can also pose problems as shown in [26], where even relatively simple Java programs could produce logs containing several million lines.

Aspect oriented programming is designed to aid in the development and testing of programs by making it easier to address the cross-cutting concerns of a program. Cross-cutting concerns are properties of a system that are developed separately from each other, but must work together correctly in order for a program to function correctly [19]. To deal with these problems, functions called "aspects" are written separately from the target application. These aspects can monitor or even change the program behavior where cross-cutting concerns arise. Areas of code involved with the cross-cutting concerns are specified by a developer and then the aspects are "weaved" into the program at compile or run-time at these locations.

The AspeCt-oriented C Compiler (ACC) [13] brings aspect oriented programming support to the C language. ACC allows for compile-time weaving of aspects to monitor properties of the system at run-time. These aspects can cleanly capture many complex system properties. Unfortunately, run-time monitoring has been shown to inflict a severe performance penalty as demonstrated by [7]. To overcome this, several approaches have been proposed that reduce the number of states that must be analyzed by the run-time monitors [11, 18]. ACC has been used to build our mote application for a simulator to conduct run-time monitoring and property checking. The simulator uses data derived from prior crane tracking experiments to provide a realistic model and focus the debugging efforts on likely scenarios. To the best of our knowledge, this is the first effort to integrate ACC with WSN application development.

The evaluation and validation techniques generally rely on extensive simulation platforms. In this paper, we utilize TOSSIM, which simulates WSNs comprised of motes running TinyOS [16]. It is possible to run applications developed for TinyOS inside of TOSSIM with few or no modifications. TOSSIM simulates the network performance using a discrete event model. It offers high performance by compiling mote applications directly to the simulator's hardware platform. It further improves performance by omitting some underlying hardware details and behaviors.

In the crane tracking application, the primary interest is testing the higher level functionality of the system, so losing some information on the lower layers of the radio and hardware platform is acceptable. To this end, TOSSIM allows for the rapid simulation of many different deployment scenarios.

3. MOTIVATION AND BACKGROUND

In this section, we provide a background on the whooping crane migration and discuss the motivation for utilizing mobile WSNs to track them.

3.1 Whooping Cranes

Whooping Cranes are an endangered bird species that is indigenous to North America. The population of the cranes has declined to approximately 300, primarily due to habitat loss. The only remaining natural nesting grounds are the Aransas National Wildlife Refuge in Texas and the Wood Buffalo National Park in Alberta. The birds migrate from Texas to Alberta beginning in March, and make their return trip beginning in October. This migration process is of extreme interest to ecologists working to preserve the crane population because it accounts for a significant percentage of the bird deaths every year.

The environmental factors contributing to the bird deaths are largely unknown. The migration is an approximately 4,000 kilometer journey that stretches from southern Texas in the United States to Alberta, Canada. This large area means discovering where a crane has died is a rare event. Tracking the birds is difficult, not only because of the size of the geographic area, but because of the bird flight characteristics. The cranes tend to travel in sets of two or three and rarely congregate together at intermediate stopping points. Observing large numbers of the birds thus requires a large set of resources. The environmental factors that influence their decisions on nesting grounds, flight durations, and flight paths are not well understood.

In addition to migration environmental data, ecologists desire information on flight behaviors. It is known that the birds use a very economical method of flying that involves riding thermal updrafts in large circular motions to gain altitude, and then glide for several kilometers in the desired direction of travel until another updraft is encountered [14]. The cranes fly during the day and may rest for several days or weeks at intermediate locations. Most of the resting areas are opportunistically chosen, but there are some areas that are visited every year by groups of cranes.

3.2 Tracking Whooping Cranes

Prior efforts have been made to track the cranes and provide observations on their behavior and environmental conditions. The original efforts focused on attaching radio transmitters with a range of several kilometers to cranes and then following them in an airplane over the course of a migration [14]. Empirical observations were made on the environmental conditions and flight behaviors. More recent efforts have placed a tracking unit with a satellite transmitter and GPS receiver on the cranes. This allows for a more accurate and continuous set of observations to be made.

These tracking methods have significant drawbacks. Following the birds in an airplane is time-consuming and too expensive for large scale tracking efforts. The collected observations sometimes lack precision, which limit their scientific value. Also, the trackers sometimes lose contact with the birds for hours or days, which produce large holes in the dataset [14]. The use of satellite transmitters and GPS provides for better quality data, but these devices are too bulky and expensive to be deployed on a large scale. The current automated recorders are limited to providing information gathered from the GPS, and have no additional sensors for measuring other environmental data.

3.3 Mobile WSNs for Whooping Crane Monitoring

Wireless sensor networks (WSNs) have been proposed to overcome the shortcomings in these data collection methods. WSNs can be customized with sensor packages tailored to the needs of the ecologists. By foregoing the use of expensive satellite transmitters in favor of shorter range, more commonly available components, the cost of the tracking devices can be brought down to more reasonable levels. The use of low power network solutions in the WSNs can extend the lifetime of the recording devices and enable the motes to communicate with each other, which leads to greater redundancy and robustness.

Using WSNs for monitoring Whooping Cranes faces three primary challenges. First, the average adult male Whooping Crane weighs around 7 kg, which means the motes must be extremely light weight. These weight restrictions preclude the use of any energy recovery mechanisms in the mote. Second, because the cranes travel in such a spread out manner, the communication opportunities between the cranes during the migration are limited. Finally, there are limited opportunities to retrieve information from the cranes. The cranes are only physically accessible in Texas and there is a small window of opportunity to establish radio contact with the birds in Canada.

3.3.1 Hardware

The first version of the sensor platform for tracking the cranes is constructed from off-the-shelf components from Crossbow Technologies. The mote consists of two main components. The first is an IRIS mote which contains the processor, radio, and flash memory. The second is a MTS-420 sensor board that contains the sensors and external GPS antenna.

The IRIS mote is a communication and processing module designed for wireless sensor networks. It is built around an Atmel ATmega1281 microcontroller with an IEEE 802.15.4 radio for networking. It offers a low 8μA power consumption in sleep mode. The IRIS contains flash memory for storing sensor measurements and an expansion port for connecting to a sensor board that contains a variety of sensors [1].

The MTS-420 sensor board is mated to the IRIS to provide the desired sensor data. The MTS-420 contains the sensors used to gather information on the cranes. These sensors are a GPS receiver, temperature, humidity, barometric pressure,

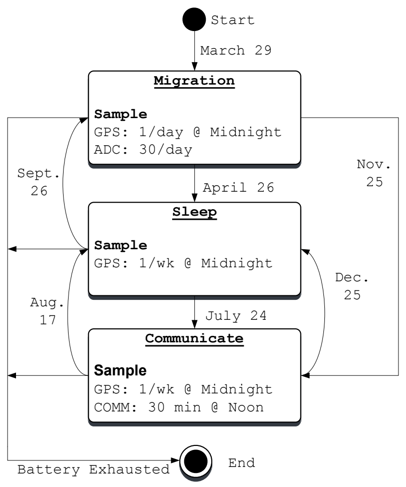


Figure 1: Migration process

and light sensor [1]. The use of these sensors allows ecologists to track the flight patterns of the cranes. Additionally, information on the flight characteristics and environmental conditions influencing migration patterns can be gathered through this sensor board.

3.3.2 Software

The crane tracking application is a non-trivial application that has a control flow that is dependent on both temporal and power constraints. The application has several operating modes that correspond to different periods in the crane's lifecycle. The modes utilize different resources to collect information that is of interest to ecologists. The different operating modes exhibit complex behaviors as they manage sampling different sensors and handling the system's resources appropriately.

At the highest level, the application changes operating modes depending on the stage of the migration the crane is in. The state diagram of the crane sensing application is shown in Fig. 1. The dates that determine the transitions between the states were developed by using observational data collected by ecologists during prior migrations. Initially, the motes start at a high sampling rate mode that takes GPS fixes every day and samples the other sensors at regular intervals throughout the day. The sensor stays in this mode during the time in which the cranes have been observed migration from Texas to Canada. Once enough time passes that the birds are assumed to be at their final roosting ground, they go into a power saving mode where the GPS is infrequently sampled. The motes exit this state midway through their Canada stay for a chance to communicate with ecologists that are conducting surveys of the nesting grounds. After this communication window ends, the motes once again enter deep sleep. The motes once again enter a high sampling rate state when they could potentially begin migrating and finish in a communicating state when they should be at their nesting grounds in Texas where ecologists can once again reach them. The transitions between the states and the operation of the sensors depends not only on time, but the power available to the mote.

The tracking program for the motes was built using MoteWorks, which is a derivative of TinyOS 1.1 created by Crossbow Technology [1]. For the purposes of simulating the code under TOSSIM, it was recompiled using the TinyOS 1.2 branch of the operating system because MoteWorks does not support TOSSIM.

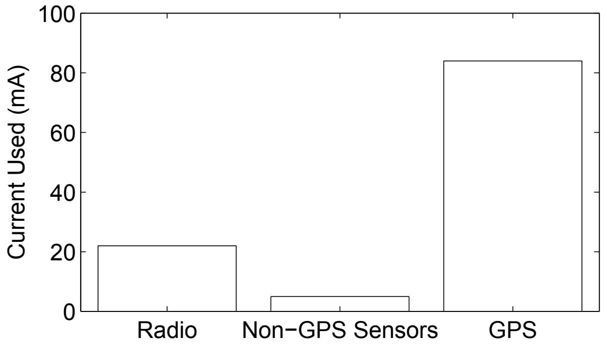


Figure 2: Power consumption of different mote components

The complexity of the crane tracking application adds to the difficulties in testing it. The mote utilizes many different sensors and components for monitoring the state of the environment that consume a great deal of power. In Fig. 2, the power consumption of three main components in the sensor board is shown in terms of the current used in mA. It can be observed that non-GPS sensors consume power equivalent to processing power and communication consumes 4 times higher power than processing. This tradeoff between communication and computation is well known [22] and is exploited for in-network processing solutions. However, it can also be observed in Fig. 2 that high profile sensors such as a GPS sensor, which is essential for this particular sensing application, consumes significantly higher power compared to other components. Furthermore, GPS operation generally lasts in the orders of seconds compared to milliseconds that is required for a typical communication. These significant differences result in a non-trivial energy consumption profile and requires the *whole* application code to be evaluated instead of only the communication stack. To improve energy efficiency, these components must be turned off when not in use or the motes will not be able to last for the entirety of the crane migration cycle.

4. SIMULATION AND TESTING PLATFORM

In this section, the whooping crane monitoring network (WCMN) simulation platform is explained in detail. First, the simulation platform and its components are explained in Section 4.1. This platform consists of several modules that model the migration path, GPS sensor performance, and mote hardware. These tools are used by running the application inside of TOSSIM. The execution of the TOSSIM simulator was guided by simulated sensor inputs created by another module that is based on prior tracking data. In Section 4.2, two validation approaches are described for testing and validating the WCMN software, where both log file analysis and run-time monitoring are utilized.

4.1 Simulation Platform

As discussed in [18], it is not feasible to exhaustively visit all possible states a WSN may be in because of the size of space makes run times prohibitively long. To best utilize the

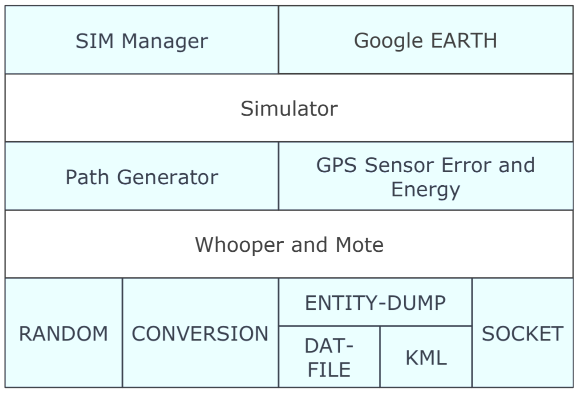


Figure 3: Independent modules of the simulator

given debugging time, a simulator was written that models the flight paths of cranes.

The data simulator is developed using data from prior crane tracking attempts [14]. Using this data, it is possible to generate likely flight paths of the cranes with coarse granularity. Additionally, the simulator models the time it takes to acquire a GPS fix for each sample. This is of vital importance, since our experiments indicate that the power consumed by the GPS is one of the largest factors in the mote lifetime, as shown in Fig. 2. The Time To First Fix (TTFF) is the amount of time it takes the GPS to determine its location after power on. This time varies from 30-210 seconds, and thus the power consumed by the GPS has a large variance. Other sensors exhibit a much more deterministic behavior and use less power, so their effect on the system is easier to determine and less important.

The simulator is developed with the python programming language, which utilizes Tython, a scripting language that augments TOSSIM and provides hooks for outside applications and scripts [10]. Using Tython and the developed simulation module together provides run-time inputs for the TOSSIM simulations.

The data simulator is developed using multiple modules, this allows for easy extension and reduces dependence between unique entities in the simulator. In Fig. 3 the modules of the simulator are shown.

Sim Manager module is the control point of the simulator and allows for manipulation of simulator input data and parameters. These parameters include the average daily distance and headings used by a Whooping Crane during migration and the average time it takes the GPS device to acquire a GPS fix. It also includes support to produce multiple replications of the simulation. *Google Earth module* is used to provide a display of the simulated paths and its correlated GPS sensor error for illustration purposes. *Simulator module* generates paths that the crane would take based on the input data and parameters. Additionally, there exists a module that simulates the errors of the GPS sensor device. *Generated Paths module* contains and controls multiple instances of the Whooper module to generates paths of each crane. This module also uses the underlying modules with each Whooper instance to times when individual cranes encounter each other during the simulation of the paths. *GPS Sensor Error and Energy module* contains and controls multiple instances of the Mote modules. The mote modules use the data generated by the Generated Paths module to apply accuracy error. This module uses inputs provided by the simulator module and the Generated Paths module bundled with underlying modules to apply accuracy error, fix rate, TTFF, and energy consumed during the migration period.

Whooper module is instantiated multiple times to represent each crane during the simulation. The whoopers generate paths that they travel over the specified period. This path is generated using random distance and bearing values provided by the module that instantiated the whooper. *Mote module* is also instantiated multiple times to represent each mote that is attached to each crane. Each mote is mapped to each simulated whooper.

Within each whooper and mote module, *random module* provides the parent modules with independent random variate streams needed for simulation and prevents correlation between simulated data. *Conversion module* provides data formatting functionality so that the parent modules can perform valid simulations. *Entity-Dump module* provides the parent module with multiple ways to output information about the simulator. *DAT-FILE module* supplies the user with output from data structures and variables of the parent module for post processing and analysis. *KML module* this module provides KML output of the simulation. The data created by this module is fed into Google Earth to show the simulation over the duration of time provided by the simulator module. *Socket module* provides the use of sockets to supply hooks into the program for real-time simulation values. In the future, this is intended to be used by python for dynamic simulation and testing of future software.

4.1.1 Input Data and Parameters

Input data and parameters are used to provide realistic outputs of the data simulation to be used as realistic inputs for TOSSIM. The input data and parameters that are needed for realistic output of the simulator include a distribution of distance and heading values, distribution of TTFF, duration of the migration, number of cranes to simulate, mortality rate, initial position, GPS accuracy error, the GPS fix rate and a battery model. To improve the performance metrics of the simulation, empirical data is used. The parameters and inputs can be easily modified to adhere to improved data and modification requirements.

Prior to this study, ecologists have collected data for previous crane tracking attempts. Accordingly, the average crane travels 71.2 miles a day with a deviation of 48.7 miles and has an average bearing of -27 degrees with a deviation of 22.5 degrees. Since the provided data has been collected using existing monitoring techniques, limited sample points exist. To compensate for this, without loss of generality, we assume the distribution of both headings and degrees come from a normal distribution, with a mean distance 71.2 miles and variance of 48.7 miles; a mean heading of -27 degrees and a variance of 22.5 degrees. Additionally, the ecologists provided the location that the captured whoopers would be released, resulting in the initial position. The prior studies performed by ecologists also provide that 6%-8% of the birds die during each migration. This statistic is also included in the simulation.

To model the distribution of the first fix times, several experiments were performed. Specific to the crane project, the GPS device will operate in the worst case TTFF in a mode commonly known as cold-start. The GPS is assumed to always be performing a cold-start because the limited battery prevents the GPS from being frequently used and the cranes can travel long distances in between GPS uses. This means the GPS will have limited information about the satellites in view every time it turns on, and will take a longer time to search for signals and compute a position fix. In our system the GPS device is enabled once per day. Therefore, the device cannot take advantage of location approximation and caching advantages of the GPS sensor. This results in the worst case TTFF. TTFF experiments were performed to analyze the off-the-shelf GPS sensor [3]. Accordingly, the average time it takes the sensor to acquire a fix is approximately 111s. This time deviates about 46 seconds throughout the tests performed. Using the TTFF experiment results, the data simulator generates realistic TTFF times. This duration is also used to compute the energy used by this device during the migration period. According to [23], the average accuracy of GPS sensors varies from device to device, but the average error is found to be 21.2m to 76.9m. For realistic simulation results, we use these bounds to define the range of error in the data simulator.

Since GPS performance varies from device to device, the need for analysis with a variable fix rate is important. Combining the known TTFF duration from prior results and trying multiple fix rates for a valid fix, the simulation platform is used to find the expected energy usage for each platform. In the application, we allow the GPS to be enabled over a 5 minute period. If a higher quality GPS sensor can acquire a first fix within a shorter amount of time, the lifetime can be improved. Since newer and better devices are more expensive, it is important to understand the difference in energy cost between devices. Accordingly, simulations are performed by varying the GPS fix rate to explore the costs-benefits of energy associated with acquiring a higher number of fixes at a higher rate.

Prior tracking attempts and patterns observed by ecologists studying whooping cranes have led to an understanding of time required for the migration from Aransas to Wood Buffalo is approximately 3 to 4 weeks [24]. For the worst case scenario, the duration provided to the simulator includes a 42 day migration period. Moreover, the number of cranes that can be monitored during a single migration is limited to 60 due to budget constraints, the intrusiveness of the devices, the possibility of the devices changing the cranes' behavior, and the complexity of trapping the birds.

4.1.2 Simulation Platform Architecture

The simulation platform is composed of two independent models. The first model consists of simulation of migration paths. Since high granular location information for Whooping Crane migration is not available, such a simulation is required. The generated paths serve as representation of the whooping cranes. The second model consists of the simulation of the GPS sensor characteristics. Realistic output of the data simulator is based entirely on the empirical data provided as input to each model. The second model relies on the paths generated by the first model and with additional inputs needed to generate realistic results.

Migration path model (MPM): MPM provides a stochastic crane path, based on a set of inputs that includes the number of days to simulate, number of cranes to simulate, and the μ (mean) and γ (standard deviation) of both the distances and headings from the provided empirical data. The MPM will create an instance of a whooper class, that contains a RandomLocationEntity class. This class contains the random number stream used to generate random headings and distances, and also a method to provide the MPM with the position for the requested step. The step for the results provided in this paper occurs every day for 42 days. The MPM creates multiple instances of the whooper class according to the specified number of cranes to simulate in a single migration. The random streams are adjusted during initialization to remove random number overlap. The generated paths created by MPM will be used as input for the simulation of GPS device error module.

GPS device error model (GEM): GEM provides error and energy consumption characteristics of the device during migration. Input used by GEM include MPM path, fix probability, percentage of deaths that occur, GPS device operational current, error distance μ and γ desired to simulate, and the μ and γ of TTFF. The GEM creates multiple instances of a mote class, based on the number of cranes, that represents the GPS devices. Each mote class includes an instance of RandomErrorEntity class. This class contains the random number stream used to generate random data based on the provided inputs. The GEM uses the same step as the previous model and adjusts the random streams in the same manner. The performance metrics that the GEM provides for each mote instance includes the rate at which the GPS acquires a fix, the number of fixes acquired during the time period, the number of cranes that died on their associated day of mortality, and the set of adjusted GPS fixes with applied error. All performance metrics are based on the provided inputs to the simulator as explained in Section 4.1.1.

4.2 Validation Platform

Using the data from the simulator it is possible to construct inputs for TOSSIM to model likely behaviors of the mote during a migration. This allows testing the software in an efficient manner. In addition to the simulated data, some random inputs are supplied to the program to uncover unlikely scenarios that are not observed in empirically gathered data, or conditions that are unlikely to happen. For example, it is unlikely that the voltage will fall to an extremely low level in the first days of operation. Hence, this would not be captured by the simulated data, but treating the voltage as a random variable can test this case.

Two different approaches are employed for software testing. The first uses log file analysis to check the motes behavior by using a simulator and manually inserted debugging statements. The second approach incorporates ACC in the simulator build process to conduct run-time monitoring of system properties. A richer set of properties are checked to validate the system software. The results and a comparison of the two techniques are discussed in Section 5

4.2.1 Log File Analysis

Log file analysis is used as a basis for measuring the effectiveness and complexity of using aspects for run-time monitoring. The log file analysis method uses debugging statements that are manually inserted to the program. These debugging statements output symbols that correspond to the mode the program is in, i.e. sleeping, sampling GPS, communicating, etc as shown in Fig. 1. The program is run in TOSSIM for several years of simulated time. The output of the simulation is recorded in log files. These log files are then post-processed using the following procedure.

Regular expressions are developed from the finite state machines that represent the ideal program execution behavior. A program then parses the output of the simulation and

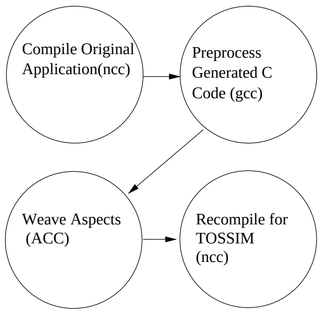

Compile Original Application(ncc)
Preprocess Generated C Code (gcc)
Weave Aspects (ACC)
Recompile for TOSSIM (ncc)

Figure 4: Compilation process with aspects

compares the state changes in the simulation to the allowed states that are represented by the regular expressions. If the output of the simulation matches the patterns in the regular expressions, the program is assumed to have performed only legal operations and passes the test. If the output of the simulation does not match the regular expressions, it fails and a fault is reported.

4.2.2 ACC

ACC is incorporated into the build process of the application to provide run-time monitoring. Aspects describing different properties of the system are written and woven into a simulator executable. This process allows the system properties to be monitored at run-time, as opposed to log file analysis which is conducted after the program execution. When erroneous behavior is detected it is possible to not only report the violation, but to give information of the state of the system when the violation happens. This constrasts with the log file analysis, where only the violation is reported.

To weave aspects, the compilation process for the motes has to be altered as shown in Fig. 4. The original application is written in nesC, which is a dialect of C [12]. The nesC application is compiled using ncc which is an extension of gcc that compiles nesC code for TinyOS. As part of its compilation process, ncc produces a C file containing all of the application code. This application source code is then processed using gcc to make it conform to ACC specifications. This transformed source code is then passed into ACC, where aspects written in C are woven into the source file. This final source file is then passed back into ncc to produce the final simulator executable. To the best of our knowledge, this is the first time ACC has been used in conjunction with wireless sensor networks.

To test the system, seven aspects are defined to monitor different properties. The properties are focused around preserving energy and checking the order of system events. Others are written to illustrate the power of the aspects and their capabilities. These aspects are shown below.

- *timers_fired* tracks when all systems timers are fired.
- *voltage_sensor* checks that the voltage level is checked before sampling non-GPS sensors.
- *no_stalls* ensures the mote does not spend too much time in any particular state.
- *gps_off* monitors when the GPS is turned on and off to make sure it is never left on after it is done being used.
- *state_change* is equivalent to the log file analysis that was performed. It makes sure the system follows a valid set of state transitions.
- *trace* generated an execution trace of the entire program.
- *voltage_gps* made sure the system power level was checked before turning on the GPS.

5. RESULTS

To test the impact of aspects on application behavior, each aspect is woven into the application at compile time into the TOSSIM Framework that can be run on a PC [16]. A TOSSIM application simulates TinyOS and its execution at its lowest level. This provides high fidelity simulation of TinyOS applications, but lacks real environment simulation [16]. The simulator did not possess the capability to simulate the variable voltage levels of the mote power sources and the sensor inputs. To adapt to this limitation, new models are developed that provide realistic voltages and GPS sensor inputs that can modify the state of the application.

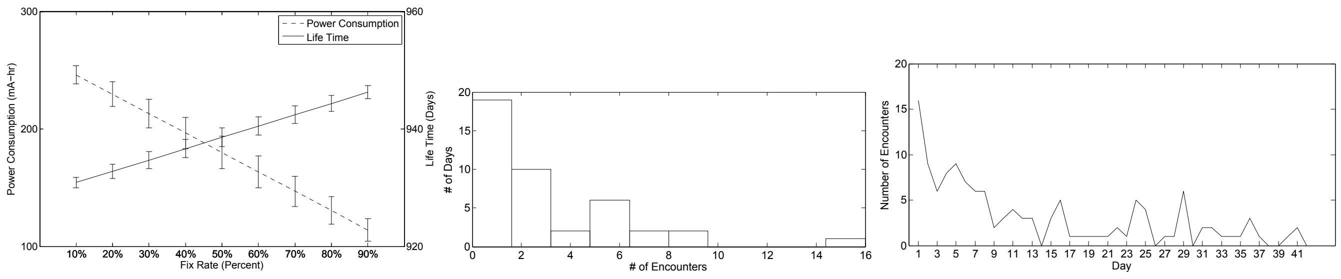
The simulations are performed for a longer period of execution time (3 years) than that motes are expected to run in the field. This provided extensive coverage of different execution scenarios very quickly, resulting in the unmodified application taking only 15 minutes to run and using only 0.03s of CPU time.

The results of each simulation provide a comparison of performance between the original application and each individual aspect. The performance metrics of the simulations include CPU time of the simulation, size of the executable, the output generated from each simulation, and faults that existed in the application.

5.1 Simulation Results

The simulation was performed for 100 replications using the input parameters discussed in Section 4.1.1. The number of cranes is set to 60 and the simulation uses the step of once a day over the longest expected duration of 42 days. The mean and standard deviation of path distance in each day are selected from prior crane paths, where these values are found as 71.2 miles and 48.2 miles, respectively. Similarly, the mean of the heading is found to be -27 degrees with a standard deviation of 22.5 degrees. For the GPS sensor, maximum error of 76.9 m is used to represent the worst case scenario and the mean and deviation of TTFF are set as 111s and 42.9s according to empirical experiments. For the migration, the initial location is selected as the Aransas National Wildlife Refuge (28.2336191,-96.9002659) and the final location is chosen based on the location of the Wood Buffalo Wildlife Refuge (59.390833, -112.986389). Percent mortality is chosen from the previous crane study as 6%. The GPS fix rate is varied from 10% to 90% and 100 replications are performed for each data point. Finally, the minimum distance for communication is selected as 5 miles based on the assumption that cranes will likely interact with each other if the area is small enough.

To ensure the simulator is working properly, it is important to test the generated data against the available empirical data. To test the reliability of the generated paths, we applied a chi squared goodness of fit test to our generated data set, to compare it to a normal distribution. The resulting test in MATLAB concluded the generated data is in fact fit for the simulator.



(a) (b) (c)

Figure 5: (a) TTFF and lifetime as a function of the GPS fix rate, (b) Histogram and (c) number of encounters during one migration (60 cranes, 42 days).

After the MPM simulation completes, the generated paths are then used as inputs for the GEM simulation. Outputs to be evaluated include percentage of deaths that occur, GPS device operational current, and the mean and variance of TTFF. Each of these outputs are generated for 100 replications and also for an adjusted GPS fix rate, incrementing by 10% for 9 iterations. The average number of deaths (4.09) and average distance error (57m) were constant for each replication. This is due to the fact that the number of cranes will not increase or decrease because of a performance increase of a GPS device. The error distance correctly falls within the bounds reported in [15].

In Fig. 5(a), the average TTFF duration and the corresponding estimated lifetime is shown as a function of the GPS fix rate. For every fix that is not acquired, the timeout duration, i.e., 300s is reached. For every invalid GPS fix the system will accumulate a 300 second penalty against energy. This can be seen in the variation of 16 days of operation in the lifetime as the GPS fix rate changes. The lifetime of the application can be significantly impacted by the GPS fix rate. This motivates the importance of higher accuracy GPS sensors on motes.

We are also interested in the number of cranes that would encounter another crane during the migration. This is accomplished by computing the distance between each combination of cranes for each day of the migration. Since cranes generally roam around their layover points up to 5 miles, we assume if the cranes land within a 5 mile distance, a communication chance occurs. This analysis is performed on a random simulation selected out of 100 replications, that had a fix rate probability of 1/2. Since their exists no correlation between the ability to achieve a GPS fix and the ability to communicate, this process is considered to be valid for analysis purposes.

The results are shown in Fig. 5(b) and Fig. 5(c), where the histogram for encounters with the same bird and the number of encounters in a day are shown, respectively. It is interesting to note that there is non-negligible chance that a mote has multiple opportunities to encounter another particular mote. Additionally, as shown in Fig. 5(c), the greatest opportunity to communicate occurs during the earlier part of the migration and that the window for communication becomes random throughout the later part of the migration. These results motivate the need for probabilistic solutions that exploit the opportunities to communicate to develop *networking* solutions for Whooping Crane monitoring.

5.2 Validation Results

The process of using simulated data for run-time and log file analysis is effective for finding software defects. The effectiveness of these techniques must be weighed against the cost in execution and development time. In this section these results are shown.

5.2.1 Execution Times

In Table 1, the impacts of aspects on the execution time and executable size are shown. The results of the execution time show that run-time monitoring can have an extremely negative impact on performance. Simple properties such as checking to make sure the GPS is always turned off once it is no longer needed can increase the execution time by several orders of magnitude. Monitoring when the system timers are fired show the need for focusing the instrumentation on specific points of the program, rather than attempting to reconstruct the complete program execution flow.

The simulations that used a higher magnitude of CPU include *timers_fired* and *trace*. This result is not surprising considering the nature of the aspects. The *timers_fired* aspect inserts a print statement after every timerFired method in the application. Since most micro-controllers feature robust timer systems, the timer fired method is called continuously throughout execution. This leads to large overhead.

The largest consumer of CPU during the simulations is the *trace* aspect simulation. Using tracing during a program execution is extremely helpful for debugging, but consumes significantly more CPU. This overhead is created from printing out after every call in execution. As shown in Table 1, the percent increase is magnitudes higher than other aspects.

The impact of certain aspects on the execution time can sometimes be surprising. Monitoring the *state_change* has one of the lowest overheads, but has some of the farthest reaching implications in terms of overall program correctness. On the other hand, the *gps_off* aspect is used in a much more narrow scope, but incurs a much greater performance penalty.

In relation to run-time monitoring of actual embedded devices, the only aspects that would not be feasible to run on a device are the *trace* and *timers_fired* aspects. The additional overhead from running the aspects on a real mote could severely deplete the limited battery power. Several of the heavier aspects could inadvertently affect the correctness of the program as they could prevent the mote from responding to events in a timely fashion.

5.2.2 Executable Size

Another important performance metric is the impact of

Aspect	CPU Time(s)	CPU % Increase	Executable Size (bytes)	Size % Increase
Unmodified	0.0327	0	258607	0
timers_fired	0.57	1642	283665	9.7
voltage_sensor	0.06	83.3	284231	9.9
no_stalls	0.0382	16.7	279134	7.9
gps_off	0.1009	208.3	281077	8.7
state_change	0.0463	41.7	286067	10.6
trace	31069	94933258	721783	179.1
voltage_gps	0.0518	58.3	283048	9.5

Table 1: Overhead of aspect in terms of CPU time and executable size.

the aspects on the size of the produced executable. While this does not impact the functionality of the simulator process, if this process is to be used on actual mote hardware there will be much more severe constraints on the size of the program that can be used. Single aspects were woven into the application and the size of the resulting executable was recorded as shown in Table 1.

The results reveal that the aspects have a very acceptable impact on the executable size. With the exception of the *trace* aspect, all aspects add less than 11% to the size of the executable. Therefore, if a corresponding change is made to the hardware build process, it is very likely the program will still fit in the memory on the mote. It is interesting to note that even though the aspects may have a small impact on the size of the system image, this does not directly correlate to the effect on the execution time. Therefore, caution must be taken in assuming that infrequently woven aspects will have a non-trivial effect on performance.

5.2.3 Implementation Cost

One of the largest benefits of using the aspect oriented approach is the speed in which the property checking could be implemented by a user. Implementing the four properties using the off-line analysis took approximately 10 hours. Most of this time was spent developing and testing the regular expressions used in the analyzer. The resulting complexity of the code means that any changes to the execution flow of the crane tracking application will incur significant costs from restructuring the analyzer.

In contrast, developing the single aspect that monitored and checked all of the state changes took less than 4 hours. The resulting aspect is much easier to maintain than the off-line analyzer because state machine is more clearly represented in the aspect.

5.2.4 Violations

The testing revealed several bugs in the application. The log file analysis and state checking aspect were able to catch several instances where the mote did not transition to a correct state or got caught in a state and never transitioned to a new one. Had these violations been included in the actual deployment, it would have resulted in the motes running out of power in days or weeks and failing to complete their mission.

The aspect oriented approach revealed additional faults that did not affect the correctness of the program that the log file analysis did not detect. For instance, the GPS receiver was being commanded to turn on when it was already on. The log file analysis did not catch this error because it occurred within a discrete state, but the aspect that checked the GPS power on sequence caught it immediately.

Finally, the use of the simulator to accelerate the mote testing process proved invaluable. The violations resulting from incorrect state transitions had a dependency on the power level of the mote. In general, finding these types of violations on real hardware can be a time consuming process, since many tests would be needed to try all possible power level combinations. It was also possible for the violations to manifest themselves only after weeks or months of run time, which is also difficult to test in a controlled fashion on real hardware.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we develop effective testing and debugging techniques for wireless sensor networks. These techniques are used to test a deployed wildlife tracking application. The power of the run-time analysis and low overhead incurred by some aspects makes this a powerful tool in the process of deploying the software. The use of a simulator to accelerate the testing process was beneficial because it exposed bugs that are slow to manifest and depend on specific power conditions.

The results of this work have been encouraging and suggests several avenues for improvement in the future. Currently all of the testing has been done using TOSSIM. As mentioned in Section 4, TOSSIM does not fully capture the behavior of a mote hardware. The developed run-time monitoring platform can be adapted to run on a real mote, albeit with a reduced fault reporting capability. This may still prove useful in tracking down program faults that appear in hardware, but not the simulator.

The use of simulated data made it possible to test the application in a reasonable amount of time. The time it took to test the various properties of the system was very reasonable compared to [18], where exhaustive searches of the program's execution space were attempted. These testing techniques are proving beneficial as the project is beginning to use custom hardware platforms that require drivers for hardware components that have not been used with TinyOS. The advanced testing methods will also prove valuable as more complicated power and sensor management techniques are explored that can be difficult to debug and verify.

7. REFERENCES

[1] Crossbow technology. http://www.xbow.com.

[2] The network simulator - ns-2. http://www.isi.edu/nsnam/ns/index.html.

[3] u-blox ANTARIS 4 GPS Module. http://www.u-blox.com/, Aug. 2010.

[4] I. F. Akyildiz, D. Pompili, and T. Melodia. Underwater acoustic sensor networks: Research challenges. *Ad Hoc Networks Journal (Elsevier)*, 3(3):257–279, March 2005.

[5] I. F. Akyildiz and E. P. Stuntebeck. Wireless underground sensor networks: Research challenges. *Ad*

Hoc Networks Journal (Elsevier), 4:669–686, July 2006.

[6] J. H. Andrews and Y. Zhang. Broad-spectrum studies of log file analysis. In *Proc. ACM ICSE '00*, pages 105–114, Limerick, Ireland, June 2000. ACM.

[7] E. Bodden. J-LO - A tool for runtime-checking temporal assertions. http://www.bodden.de/pubs/bodden05jlo.pdf, 2005. Diploma Thesis, RWTH Aachen University.

[8] N. Correll, M. Schager, and D. Rus. Social Control of Herd Animals by Integration of Artificially Controlled Congeners. In *Proc. SAB '08*, pages 437–447, Osaka, Japan, July 2008.

[9] M. Darr and W. Epperson. Application note: Embedded sensor technology for real time determination of animal lying time. *Comput. Electron. Agric.*, 66(1):106–111, 2009.

[10] M. Demmer, P. Levis, A. Joki, E. Brewer, and D. Culler. Tython: a Dynamic Simulation Environment for Sensor Networks. Technical Report UCB/CSD-05-1372, EECS Department, University of California, Berkeley, 2005.

[11] M. B. Dwyer, A. Kinneer, and S. Elbaum. Adaptive Online Program Analysis. In *Proc. ACM ICSE '07*, pages 220–229, Minneapolis, MN, May 2007.

[12] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. In *Proc. ACM PLDI '03*, pages 1–11, San Diego, CA, June 2003. ACM.

[13] W. Gong and H.-A. Jacobsen. *AspeCt-oriented C Language Specification*. Middleware Systems Research Group, University of Toronto, Toronto, CA, 0.8 edition, January 2008.

[14] E. Kuyt. Aerial radio-tracking of Whooping Cranes migrating between Wood Buffalo National Park and Aransas National Wildlife Refuge, 1981-1984. Technical report, Canadian Wildlife Service, 1992.

[15] M. Lehtinen, A. Happonen, and J. Ikonen. Accuracy and time to first fix using consumer-grade GPS receivers. In *Proc. IEEE SoftCom '08*, pages 334 –340, Dubrovnik, Croatia, Sep. 2008.

[16] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: Accurate and scalable simulation of entire tinyos applications. In *Proc. of ACM SenSys '03*, pages 126–137, November 2003.

[17] J. Li, Y. Wu, K. Kapitanova, J. A. Stankovic, K. Whitehouse, and S. H. Son. Run time assurance of application-level requirements in wireless sensor networks. In *Proc. ACM SenSys '09*, pages 367–368, Berkeley, CA, Nov. 2009.

[18] P. Li and J. Regehr. T-check: bug finding for sensor networks. In *Proc. IEEE IPSN '10*, pages 174–185, Stockholm, Sweden, April 2010.

[19] K. Mens, C. V. Lopes, B. Tekinerdogan, and G. Kiczales. Aspect-Oriented Programming Workshop Report. In *Proc. AITO ECOOP '97*, pages 483–496, Jyvaskyla, Finland, June 1998. Springer-Verlag.

[20] N. T. M. Nguyen and M. L. Soffa. Program representations for testing wireless sensor network applications. In *Proc. DOSTA '07: Workshop on Domain specific approaches to software test automation*, pages 20–26, Dubrovnik, Croatia, Sep. 2007.

[21] M. Okola and K. Whitehouse. Unit Testing for Wireless Sensor Networks. In *Proc. Workshop on Software Engineering for Sensor Network Application*, Cape Town, South Africa, May 2010.

[22] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43:51–58, May 2000.

[23] O. Sokolsky, U. Sammapun, J. Regehr, and I. Lee. Runtime Verification for Wireless Sensor Network Applications. In B. Finkbeiner, K. Havelund, G. Rosu, and O. Sokolsky, editors, *Proc. Dagstuhl Seminar 07011*, number 07011 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, August 2008. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[24] J. Unrau. Whooping cranes sighted in record numbers. http://www.theglobeandmail.com/news/technology /science/article804129.ece, 12 2007. Retrieved 2007-12-17.

[25] M. Woehrle, C. Plessl, J. Beutel, and L. Thiele. Increasing the reliability of wireless sensor networks with a distributed testing framework. In *Proc. ACM EmNets '07*, pages 93–97, Cork, Ireland, June 2007.

[26] J. Yang, D. Evans, D. Bhardwaj, T. Bhat, and M. Das. Perracotta: mining temporal API rules from imperfect traces. In *Proc. ACM ICSE '06*, pages 282–291, Shanghai, China, May 2006.

[27] P. Zhang, C. Sadler, S. Lyon, and M. Martonosi. Hardware design experiences in ZebraNet. In *Proc. ACM SenSys '04*, Baltimore, MD, Nov. 2004.