The Doppel System for Controlled Testing of Sensor Network Apps

A Thesis

Submitted to the Faculty

of

Drexel University

by

Anbu Elancheziyan

in partial fulfillment of the

requirements for the degree

of

Doctor of Philosophy

March 2012

Dedications

To my family.

Acknowledgments

I would like to thank my thesis adviser Dr. Jaudelice Cavalcante de Oliveira for all her support through my doctoral studies at Drexel. Discussions with her have left me with numerous indelible memories on teaching, research and life.

I would like to thank all my thesis committee members for contributing their time, patience and knowledge. I would like to especially thank Dr. Timothy Kurzweg for his extensive support and encouragement, Dr. William Regli for the experience I gained collaborating with him, Dr. Nagarajan Kandasamy for offering his keen insights and cool enthusiasm, and Dr. Hande Benson for her generosity in imparting her wisdom and willingness to help.

I would like to extend my thanks to Jeffrey W. Wildman II with whom I have had many wonderful discussions and collaborated with for the former part of the thesis.

From Dr. Ali Shokoufandeh and Dr. Steven Weber not only have I enjoyed learning but also how to enable enjoyable learning. I would also like thank Dr. Gennady Friedman for the many engaging conversations we have had.

The current and former ECE department staff: Stacey, Tanita, Kathy, Chad, Dan, Delores and Wayne have been always welcoming and friendly.

My current and former lab members: Sukrit Dasgupta, Zhen Zhao, Josh Goldberg, Fernando Solano Donado, Fei Bao, Joydeep Tripathi and Peter Thai have made for a joyous journey. My friends from Drexel: Kashma Rai, Prathaban Mookiah, Sriraj Srinivasan and Zulfiya Orynbayeva have always provided great company and memorable moments.

My deepest gratitude goes to my immediate and extended family for the all the ways in which they have continued to support and inspire me despite life's challenges.

Table of Contents

List of Tables

List of Figures

Abstract

The Doppel System for Controlled Testing of Sensor Network Apps

Anbu Elancheziyan

Jaudelice Cavalcante de Oliveira, Ph.D.

Application software or apps for wireless sensor networks vary widely and are customized to run on resource-constrained sensor nodes. This places restrictions on how it can be tested especially when running on the target hardware and operating as a network. Any changes to the app for testing could result in non-trivial observer effects especially if the testing methodology requires use of the already scarce resources.

This leads us to the question of whether a system capable of testing sensor network apps while not using resources of the sensor nodes could be built. The objective of this research was to answer this question by designing the architecture and building the *Doppel* system which allows testing sensor network apps operating as a network.

We present an architecture that utilizes sensor nodes to provide the required sensory input and exercise control over the sensor nodes that are executing the app under test. In our architecture, each sensor node executing the app under test is paired with a modified sensor node called the control node. We showcase an implementation of the architecture using the MICAz sensor node platform and TinyOS operating system software. Evaluation results in a network setting are also presented. Our architecture provides the benefits of both hardware-based and software-based approaches to testing sensor network apps.

To manage the system efficiently when scaling up the system, there arises a need to find the optimal placement of base stations, what data each base station holds for operating the system and how the data needs to be routed. We modeled this optimization problem as the

well-known facility location problem, and provided a hybrid algorithm that uses simulated annealing and a standalone solver to solve the problem.

Chapter 1: Introduction

Wireless networks are now ubiquitous and among them wireless sensor networks have made a mark as unique instruments. They are ad hoc networks that are composed of inexpensive miniaturized communicating devices called wireless sensor nodes. Being an ad hoc network, there is no centralized controller or infrastructure and the task of coordination and communication is handled by the individual wireless sensor nodes. The nodes use distributed communication protocols to self organize and, while performing elementary operations at the node level, exhibit complex behaviors at the network level which are leveraged for various applications.

They are used for personal health monitoring [1], robust medical systems [2], emergency and disaster relief causes [3], civilian applications [4], wildlife habitat monitoring [5] and environmental causes [6]. There are also applicable in a variety of other scenarios because of their versatility [7] and are available commercially [8, 9]. The versatility stems from the fact that the nodes are programmable and the networks are robust and fault-tolerant even when the nodes themselves are inexpensive and prone to failure.

In a wireless sensor network (WSN), the individual nodes sense, process and communicate data from the environment they are deployed in to one or more end-user base stations using distributed algorithms [10, 11]. This allows a large number of inexpensive devices to monitor large areas and provide a macroscopic view [12] of the environment without the need for expensive or permanent infrastructures.

A wireless sensor node, the building block of a Wireless Sensor Network (WSN), is usually a combination of one or more sensors, a microcontroller, a communication module

such as a radio and a power source which is usually a battery pack. It is also designed to be small in size. For example, the MICA2 wireless node module [13] and the sensor board [14] that can be attached to it, shown in Figures 1.1 and 1.2, has a surface area around 60mm×35mm. The use of inexpensive and mass-produced miniature components allows deployment of large networks without significant cost and makes loss of nodes insignificant. However, the small and inexpensive components used place restrictions on the amount of resources available at each node [15].

This has led to the design of application software for sensor networks that use the least amount of resources and prioritize energy conservation to ensure that the network can be operational for as long as possible while meeting the requirements of the application [16, 17].

The availability of limited resources and conservative approach to power consumption restricts how the application software or app that is running on a sensor node can be analyzed and tested with different inputs when operating as a network. Any changes to the app to include testing functionality may adversely affect the functioning of the app at the individual nodes and potentially alter the complex behavior exhibited at the network level [18].

A system that provides a controlled testing environment with the least interference to the app under test would lead to more robustly built apps with predictable behavior at the network level.

1.1 Testing Sensor Network Apps

Consider a wireless sensor network where all nodes usually run the same app and measure physical phenomena in the environment such as light, heat, magnetism, etc. depending on the type of sensor. This data is then processed in the network and the result is collected and sent to a base station (usually a computer).

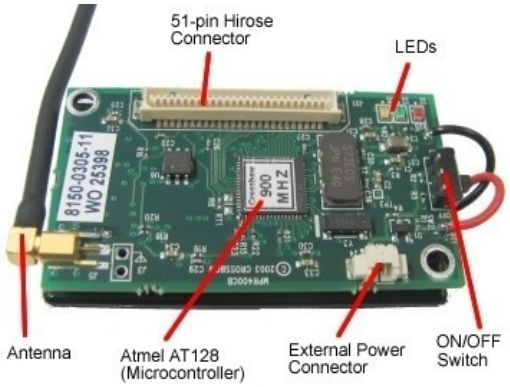


Figure 1.1: The MPR400 sensor node module without the sensor board. Batteries are under the PCB.

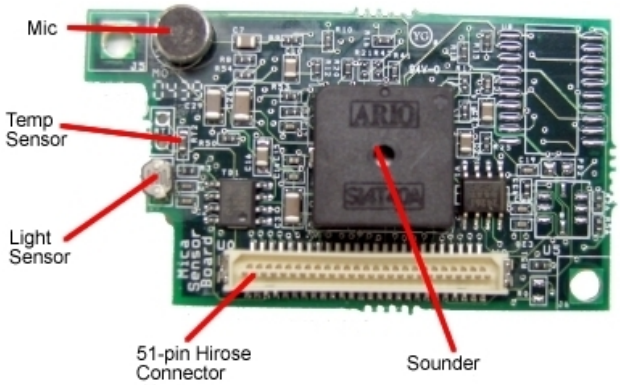


Figure 1.2: The MTS300 sensor board with various sensors. It connects to the MPR400 shown by a 51-pin hirose female connector attached on the bottom of the PCB.

Although the app runs on individual nodes, the sensor network can be collectively considered to be system running an app a that takes input from an environment (say x) and, after processing, produces an output (Figure 1.3).

When we make changes to app a such as tuning some parameters in the algorithm or substituting software components, there arises a need to evaluate the performance of the modified app b against the original app a.

This poses a problem as the environment that was once available to app a (environment x) is now not available to app b as the physical environment is constantly varying.

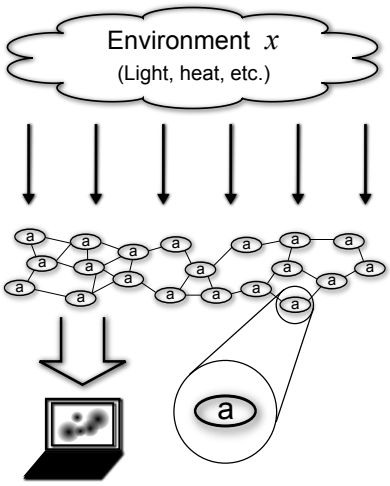


Figure 1.3: WSN running app a with input from environment x

When this happens, app b uses another environment (say y) as input (Figure1.4). This might lead to a biased comparison of apps b and a. While we can avoid this by recreating some physical quantities (such as light), it is difficult or expensive to do so for others such as heat, magnetism or GPS signals.

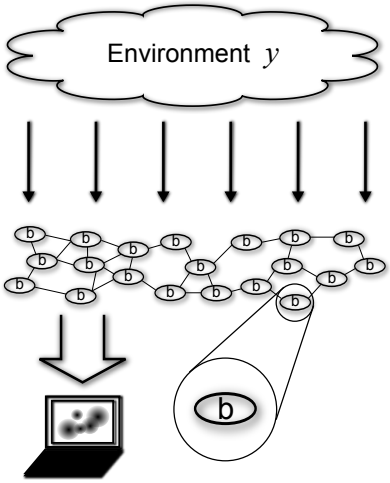


Figure 1.4: WSN running app b with input from environment y

We can overcome this by simulating the values that were obtained from environment x in a way that is transparent to app b to ensure that it is tested with the same environment as app a and the resulting behavior can be used for an unbiased comparison.

The current systems that can be used to simulate a pre-recorded environment's values at each sensor node can be categorized as either software-based or hardware-based.

Software-based systems achieve this by simulating (in software) the entire WSN or parts of it and, as a consequence, have lower fidelity and need the app to be modified to acquire data from software rather than hardware. Hardware-based systems use external hardware to generate the signals that would be obtained from a sensor. These signals would then be fed to the microcontroller on the node running the app.

In our example, when simulating values from environment x, with a software-based approach, we would use the values that were sensed in app a and feed them to app b. In a hardware-based approach, we would recreate the signals that were sent from the sensor (to the microcontroller) when app a was running and feed those signals to the microcontroller when app b is running.

Software-based systems

In a software-based system, the simplest way to ensure that the environment remains consistent is to simulate the entire wireless sensor network. This approach is cost-effective, requires lower effort compared to a hardware-based system and is easily scalable. However, the fidelity of the system is not on par with an app running on real hardware and the app (software) may not exactly match with what might be installed on a real node. Therefore, software-based systems trade fidelity for ease of use and programmability.

Hardware-based systems

In a hardware-based system, external hardware that generate the same signals as a sensor are used. These signals are fed to the microcontroller of the node running the app which cannot distinguish them from signals coming from a real sensor. This allows higher fidelity,

no storage restrictions for the data and minimal app modification. Therefore, hardware-based systems trade scalability and programmability for fidelity.

While software-based systems are cost-effective, scalable and simpler to operate, they do not provide high fidelity and might need extensive app modification. Hardware-based systems provide high fidelity and can work with minimally modified apps but cannot be easily scaled, not very cost-effective and are non-trivial to program, operate and automate.

The objective of this research is to design and evaluate a system for controlled testing of sensor network apps when operating as a network while combining the benefits of both hardware-based and software-based systems. We call this the *Doppel* system.

1.2 The Doppel System

The Doppel system provides the ease of use and programmability of a software-based system and the high fidelity of a hardware-based system while also allowing cost-effective scalability. We first discuss its architecture followed by a prototype implementation and its evaluation.

In order to understand how a hardware-based system works, we first look at the architecture of a sensor node. The app runs on a microcontroller which has built-in ADCs which convert the signals from the sensors into digital values for the app.

Therefore, if we can bypass the sensors and feed the signals directly to the ADCs of the microcontroller, the app is unaffected and is transparent to this change. A hardware system is required for this setup, but, we need one that is both easily programmable and cost-effective to scale. We can achieve this by using another microcontroller which can generate the required signals, has communication hardware, is programmable with existing tools and is readily available in scale.

A sensor node, essentially having all these features [19], can be purposed for this. We call this re-purposed node the control node. The control node can be paired with a sensor

node forming a Control Sensor node Pair (CSP). The control node can also control the on-off state of the sensor node, measure its power consumption and perform other control functions if necessary.

Therefore, as a hardware-based system, the Doppel architecture has the following characteristics:

- cost-effective (compared to a custom hardware solution),
- programmable similar to a sensor node (it is after all a sensor node in disguise), and
- operable as a network (built in communication hardware can form an identical wireless network).

The control nodes form an identical network which we call the control network. The sensor network can run app n (where $n = a$ or $n = b$). Each node in this sensor network is attached to a control node. The control network consisting of control nodes runs an app used to generate the simulated values and control the sensor nodes.

The system works as follows. First, the sensor network can run app a ($n = a$) and collect data as sensed values. These values are collected from each sensor node and stored at a base station. When the sensor network is running app b ($n = b$), a base station for the control network can send the sensed data for each sensor node to its corresponding control node. The control node is configured to convert these values into signals and feed them to the sensor node.

1.3 System Management

In a wireless sensor network, data is generated by the nodes in the network and is routed through the network to one or more base stations (otherwise called sinks) which collect the data and present it to the end user. These base stations are sometimes also used to send

out specific queries to the sensor nodes in network or even modify behavior of the nodes if required.

In the control network formed by the control nodes of the Doppel system, each control node uses data that it receives from a base station to provide stimuli to or control the sensor node it is attached to. Therefore, the flow of data is from the base stations to each control node in the network. Not only is this different from the usual direction of data flow of a sensor network, it is imperative that the data is delivered within the time frame that it is required. Otherwise, the control node cannot provide the required stimuli or control that the sensor node requires during that testing epoch which could invalidate the test results.

Therefore, we need to solve the problem of managing the disseminating of control packets to the appropriate control nodes in the network with strict time constraints. Additionally, the nodes have limited storage capabilities and cannot hold more than a limited number of packets. The wireless medium also poses challenges because of half-duplex constraints. This problem can be trivially solved by placing a base station at each control node. However, this is not viable for a large and geographically distributed system and is also not cost-effective.

Considering these factors, we can efficiently manage the system by finding the location of base stations that can be used for disseminating the control packets through the network. This requires knowing the network topology, the buffer capacities of nodes and the times each node requires its control packets. With this information, we can formulate an optimization problem that can be solved to find the location of the base stations and the routing information for the packets.

We modeled this optimization problem as the well-known facility location problem, and provided a hybrid algorithm that uses simulated annealing and a standalone solver to solve the problem.

1.4 Thesis Outline

In the next chapter, we first argue the case for a system which can accurately reproduce sensed input or stimuli for fair evaluation of wireless sensor network applications. It is shown, with a simple example, that consistent input is crucial in the evaluation of applications, and that the lack of such rigor may lead to wrong conclusions, and therefore a biased choice of what seems to be the best application. We then look at the details of some of the current software-based and hardware-based systems that provide this functionality.

Later, we introduce the Doppel system. The system allows providing artificial stimuli for testing sensor network apps that are operating as a network and also allows exercising some control functions. Being a hardware-based system, we first present the design requirements for such a system and propose the architecture for the Doppel system. In our architecture, each sensor node executing the app under test is paired with a modified sensor node called the control node.

We then showcase a prototype implementation of the architecture using the MICAz hardware platform and TinyOS operating system software. Evaluation results for the prototype in a network setting are then presented.

In Chapter 3, we first show how the control network can be managed efficiently by the strategic location of base stations. The problem is identifying the number and location of base stations required to run the system. We show that it is a type of facility location problem and then discuss how we can formulate it as an optimization problem. Using the network topology and the times each node requires data as the input, we can solve it to produce a list of nodes to which base stations must be connected. We also obtain information pertinent to routing the data.

In Chapter 4, we present a hybrid algorithm that combines simulated annealing with a

standalone solver to solve the optimization problem.

In the final chapter we summarize the work from all chapters and present suggestions for extending the work.

Chapter 2: Doppel: System Architecture and Prototype

In the last chapter we looked at how sensor network apps can be tested in an unbiased manner by using a controlled testing environment. In this chapter, we will first present a study on a simple app that quantifies the probability of error that can be introduced when a controlled testing environment is not available. We then look at the current methods of alleviating this problem which can be broadly classified into software-based and hardware-based systems. We then delve into the architecture of the Doppel system and the benefits it provides. We also showcase a prototype implementation of the architecture and evaluate its performance.

2.1 Introduction

Sensor network apps need to achieve acceptable performance while operating with limited energy and other scarce resources available on each node. An app is usually comprised of a set of algorithms, protocols, data structures, etc. Since there are multiple algorithms and protocols that provide similar functionality, versions of the app that serve the same purpose but operate differently can be built. A single app which can be tuned with various parameters could also be built. However, when testing different versions of apps (either with different internals or with different parameters), a controlled testing environment which provides identical inputs for the different apps under test is required for their fair evaluation. The apps could also be required to be tested with sensed data that is representative of the final deployment environment.

We can quantify the error an inconsistent testing environment can introduce when eval-

uating two apps to choose the better one. We embark on a test and evaluate two apps that are both tested with consistent inputs first and then with inconsistent inputs. With an inconsistent test environment, we see that there is a high probability of choosing the wrong app. This forms the basis of the next motivation section.

Once we show that a controlled testing environment is essential for unbiased evaluation, we present the currently available systems that can be classified as software-based or hardware-based and how they have unique benefits. The Doppel architecture is introduced later and we show how it can provide the benefits of both classes of systems even though it is a hardware-based system. We show a prototype implementation and how it can reproduce recorded stimuli and arbitrary stimuli for testing sensor network apps that are running as a network. The system is also evaluated for measuring the power consumption of individual sensor nodes and also simulating node failures in a network.

2.2 Motivation

The primary purpose of a wireless sensor network is to collect and process data. We can, therefore, consider the network as a collective computing machine that runs an app $\mathcal{A}$ that operates on an input or stimuli $\mathcal{I}$ and produces some output $\mathcal{O}$, which can be represented as:

$$\mathcal{A}(\mathcal{I}) = \mathcal{O} \tag{2.1}$$

Consider a scenario where we are to evaluate two slightly different apps, say $\mathcal{A}_1$ and $\mathcal{A}_2$, and pick a winner depending on some property of the output.

Also consider two separate experiments that have been designed to evaluate two apps $\mathcal{A}_1$ and $\mathcal{A}_2$. For *Experiment 1*, we have some input, $\mathcal{I}$, with high reproducibility, for evaluating

both apps and they produce the respective outputs $\mathcal{O}_1$ and $\mathcal{O}_2$, i.e.,

$$\mathcal{A}_1(\mathcal{I}) = \mathcal{O}_1 \tag{2.2}$$

$$\mathcal{A}_2(\mathcal{I}) = \mathcal{O}_2 \tag{2.3}$$

Using these outputs we can arrive at a conclusion through an appropriate selection function (represented as F) as

$$\mathcal{C}_1 = F(\mathcal{O}_1, \mathcal{O}_2) \tag{2.4}$$

For *Experiment 2*, we do not have the same level of reproducibility and have the original input, $\mathcal{I}$, and a less accurate reproduction of it, $\mathcal{I}'$. When we evaluate the apps with these two inputs we obtain the outputs $\mathcal{O}_1$ and $\mathcal{O}_2'$ respectively.

$$\mathcal{A}_1(\mathcal{I}) = \mathcal{O}_1 \tag{2.5}$$

$$\mathcal{A}_2(\mathcal{I}') = \mathcal{O}_2' \tag{2.6}$$

These outputs will lead us to conclusion $\mathcal{C}_2$ represented as

$$\mathcal{C}_2 = F(\mathcal{O}_1, \mathcal{O}_2') \tag{2.7}$$

While *Experiment 1* is clearly more accurate than *Experiment 2*, it is immaterial which experiment is executed when the conclusions are the same, i.e., when $\mathcal{C}_1 = \mathcal{C}_2$. The case $\mathcal{C}_1 \neq \mathcal{C}_2$, however, identifies apps that need highly reproducible inputs for fair evaluation. We now present examples of such apps and evaluate them with different inputs.

One of the simplest apps in sensor networks is to sample some physical phenomenon such as light intensity or temperature and calculate the average of the readings. We consider two such apps: *a*) random sampling and averaging, and, *b*) periodic sampling and averaging. The two sampling apps, say, $\mathcal{A}_r$ and $\mathcal{A}_p$ respectively, are to be fed with a time varying digital signal $\mathcal{I}$ and we wish to select the one with the lowest relative error compared to the original input signal. When the relative errors are represented as $\delta(\mathcal{O}_r)$ and $\delta(\mathcal{O}_p)$, the selection function F is given by

$$F = \begin{cases} \text{select } \mathcal{A}_r & \text{if } \delta(\mathcal{O}_r) \leq \delta(\mathcal{O}_p), \\ \text{select } \mathcal{A}_p & \text{otherwise} \end{cases} \tag{2.8}$$

For *Experiment 1*, we use the same input signal, $\mathcal{I}$, for both apps $\mathcal{A}_r$ and $\mathcal{A}_p$. For *Experiment 2*, we first obtain $\mathcal{O}_r$ using input signal $\mathcal{I}$ with app $\mathcal{A}_r$. We then use the input $\mathcal{I}'$, which has a finite amount of error, with app $\mathcal{A}_p$. The erroneous signal is obtained by introducing the same amount of relative error ϵ to each value of a time varying digital signal (illustrated in Figure 2.1). It is then fed to $\mathcal{A}_p$ and the output $\mathcal{O}'_p$ is obtained.

The two experiments with apps $\mathcal{A}_r$ and $\mathcal{A}_p$ were executed through computer simulation and a plot of the probability of arriving at different conclusions for different relative errors is given in Figure 2.2. Each data point was obtained with 95% confidence within $\pm 5\%$ error. For each run of the simulation, a signal lasting 6000 time units and assuming 10 random values (uniformly drawn between 0 and 1023) at random time instances within that period (also drawn from a uniform distribution) was used. This signal (Figure 2.1) was then used as the input for the two apps where 100 samples of the signal were acquired by each app to calculate their respective averages. A calculated error was then introduced into the signal to obtain the inaccurate signal for *Experiment 2*.

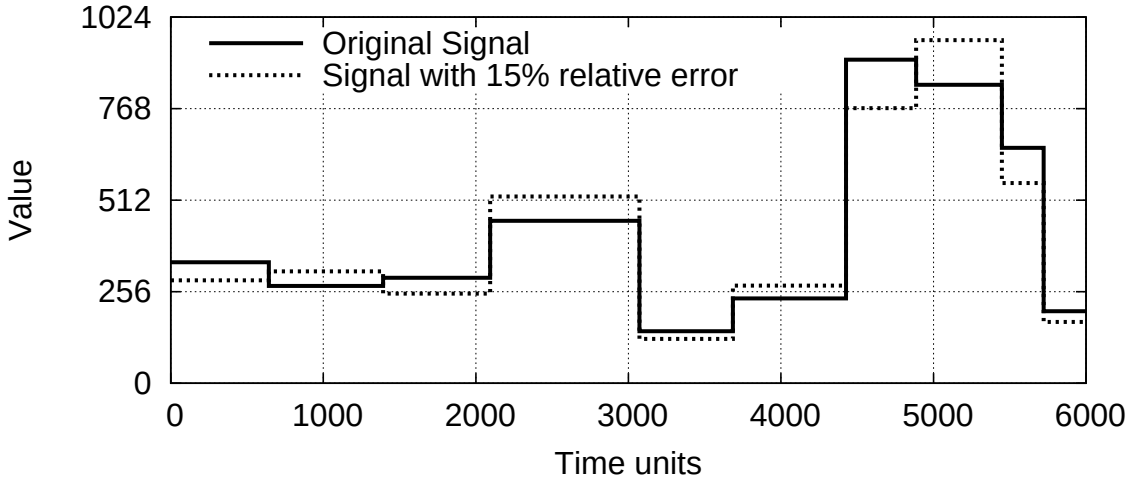


Figure 2.1: The original signal and a signal with 15% relative error.

For the given apps, $\mathcal{A}_p$ has lower relative error when identical inputs are used. But testing $\mathcal{A}_p$ using an input with relative error $\epsilon \geq 0.15$ leads to the conclusion that $\mathcal{A}_r$ has lower relative error with $> 50\%$ probability. Therefore, even with a 15% relative error, we have a high probability of arriving at the wrong conclusion. Figure 2.2 shows the probability of arriving at different conclusions for different values of relative error. The probability of different conclusions is non-zero even for a 0% relative error because both apps only sample the original signal and there may be cases when $\mathcal{A}_r$ has a lower relative error (relative to the original signal) compared to $\mathcal{A}_p$. From these results we can justify the need for a system which can produce stimuli (input for an app) with high reproducibility.

2.3 Related Work

While there exist many different systems that allow testing of sensor network apps, they can be broadly categorized into software-based systems which use an all software framework for simulating the operation or a hybrid system that uses real sensor nodes for part of the simulation. Hardware-based systems use external hardware that emulates the environment providing high fidelity. However, they are usually a custom hardware solution that cannot

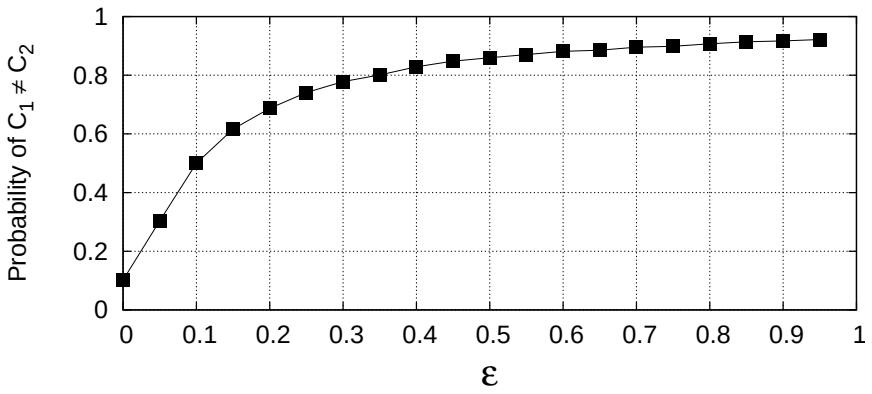


Figure 2.2: Probability of reaching different conclusions for various levels of relative error for the example app.

be scaled cost-effectively or do not have software toolchains that can be integrated with developing the sensor network app.

The most basic systems that can be used to test apps are generic simulation software frameworks such as GloMoSim [20] and OMNet++ [21]. These types of frameworks however cannot provide high fidelity as they allow testing app behavior and usually do not support simulating a real app that can run on sensor node hardware.

Frameworks that allow testing a real app's source code have also been developed. Examples are simulators such TOSSF [22] and TOSSIM [23] that can simulate execution of TinyOS sourcecode and COOJA [24] which can simulate execution of code for the Contiki operating system. While these can simulate network operation using the source code for apps, they lack fidelity when it comes to testing radio communications and transducer operations. Another class of software-based systems are instruction-level simulators that can use source code to approximate behavior on specific hardware platform such as ATEMU [25]. These too suffer from the drawbacks mentioned earlier.

A higher level of fidelity is achieved by hybrid hardware emulators such as EmStar [26], MULE [27], H-TOSSIM [28]. However, they cannot match the fidelity obtained when testing

with real hardware as parts of the system are still simulated.

The next level of fidelity is testing the app on real hardware. This, however, introduces the problem of providing previously sensed input or stimuli that meets both accuracy and repeatability constraints. In the case of traditional wired IP networks, a traffic generator can be used to address these requirements. In wireless sensor networks it is possible to introduce packets through the wireless medium from one or more nodes acting as a traffic generator. Such an approach, however, has two major disadvantages: *a*) it cannot capture the behavior of a node which is sensing but has turned off the communication hardware, and, *b*) it introduces interference into the wireless medium which may conflict with the normal operation of the app. These disadvantages can be overcome by providing data directly to the app without using the wireless medium via either software-based or hardware-based approaches.

Software-based approaches modify the app on each node and add an extra software module that either retrieves data from memory on the node similar to EnviroLog [29] or use a pseudo-random number generator (P-RNG) to generate random data or sample values from a stored data distribution such as Emuli [30]. In the former approach, if the data is stored in the RAM, the app cannot use all of the already scarce RAM, or, if it is stored in non-volatile memory, data read latencies can prevent normal operation unless it is buffered in the RAM. It also reduces the amount of non-volatile memory available for storing sensed data. In the latter approach, the P-RNG could strain the node's CPU affecting the energy consumption profile and altering regular app behavior especially if the CPU is already operating close to its maximum capacity. Also, the delay in acquiring data from the sensor can be unaccounted for (e.g., light sampling on a MICAz node with TinyOS requires approximately 13 milliseconds). Another approach to this problem is to program

the app to accept the data over a network interface such as MoteLab [31], ORBIT [32] and Kansei [33] which also requires app modification and a dedicated network.

In hardware-based approaches, we trade hardware cost for fidelity of stimuli and app tests. A thorough and comprehensive implementation of a system that can record and reproduce all the input (signals) for a sensor node is presented in EmPro [34]. Empro can record and playback analog and digital signals, profile and emulate battery performance, and generate radio signals with controllable interference to profile or benchmark different sensor node platforms. However, it requires a custom hardware setup with three different microcontrollers and other components including custom software and development tools for each node which can become expensive even for a small network. A related system that allows data logging and reprogramming is the Deployment Support Network (DSN) [35] which pairs every sensor node with a BTnode [36]. However, DSN is not designed for providing stimuli and requires BTnodes that run custom software.

We propose to adopt a combination of the latter two approaches which results in a network consisting of pairs of sensor nodes. In each pair, one acts as a sensor node running the app under test and the other acts as a control node that provides stimuli to the sensor node and allows control over its operation. To the best of our knowledge, this is the first architecture that combines the benefits of using custom hardware to minimize test app modification and increase fidelity and the benefits of using software-based in-app stimuli generation and control to enable ease of programming and automation.

2.4 Doppel Architecture

The proposed architecture allows testing sensor network apps by providing stimuli to and allowing control over the operation of each sensor node in the network. The stimuli is provided by another sensor node (hereafter referred to as the *control* node), which generates

the stimuli and routes it, and control signals to the sensor node through a wired interface. When testing an app on real hardware, the sensor nodes are usually deployed in a large area to match the desired topology and radio channels. Since nodes are usually not located close to each other, using a single control node to provide stimuli to multiple nodes may not be practical. We propose pairing every sensor node with a control node, creating a control sensor node pair (CSP). While this requires an equal number of control nodes, we argue that the benefits outweigh the costs and it is a highly flexible solution compared to custom hardware such as EmPro [34] which may cost many times as much. Besides, our architecture does not entail extensive hardware modifications and therefore the control nodes can be reclaimed to act as sensor nodes if the need arises. While the architecture does not prohibit a single control node from providing stimuli to multiple sensor nodes, this requires running wires from the control node to each sensor node and reduces the number of control and stimuli channels that it can provide. We now delve into the architecture of an CSP (shown in Figure 2.3) and then describe how it operates in a network.

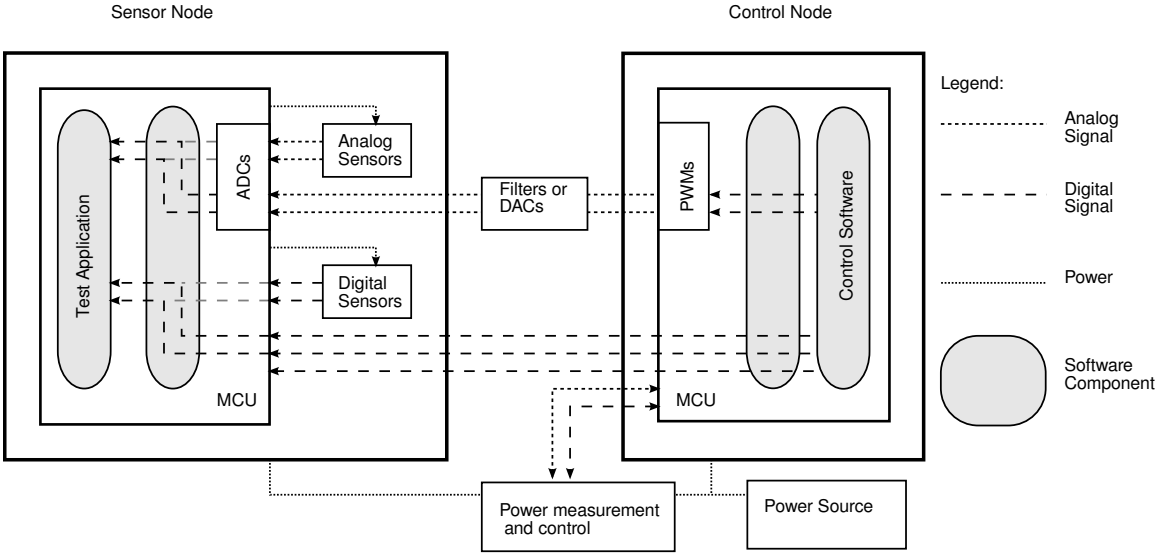


Figure 2.3: The architecture of a Control Sensor node Pair (CSP) showing the different analog and digital signals, power routing, and the software components.

2.4.1 The Control Sensor node Pair (CSP)

The sensor node of a CSP is loaded with the app to be tested and the control node is loaded with software that can generate the required stimuli. Before we generate stimuli, we need to be aware of how a sensor node senses and acquires data. The sensors or transducers are either analog or digital and typically produce analog (voltage) or digital signals respectively. These signals are read by a microcontroller (MCU), the brain of the sensor node (Figure 2.4). For versatility, sensor nodes are equipped with MCUs which can perform a multitude of tasks and can therefore directly read digital signals. They have on-board Analog-to-Digital converters to directly read analog voltages. Therefore, most sensors are wired to the MCU with minimal components which allows us to bypass the sensors and directly route stimuli signals to the MCU (Figure 2.5). However, if we tap into these pins, the original signal from the connected sensor could introduce errors. Therefore, to avoid hardware modifications, the stimuli signal can be routed to an unused pin of the MCU. Since sensor nodes are built to allow connecting additional sensors, some of the unused pins are available for the end user. This method also allows the original sensors to be powered, which may be critical when performing power consumption measurements. If no unused pins are available, the existing sensors can be disconnected, which should be trivial on a sensor node built with good modularity.

2.4.2 Stimuli generation

Since sensor nodes have general purpose MCUs, they are capable of generating different signals. We take advantage of this fact to convert a regular sensor node into a control node. While the MCU can trivially generate digital signals for parallel GPIO, UART, SPI or I2C, analog signals are more complex and can be generated by using one of two methods:

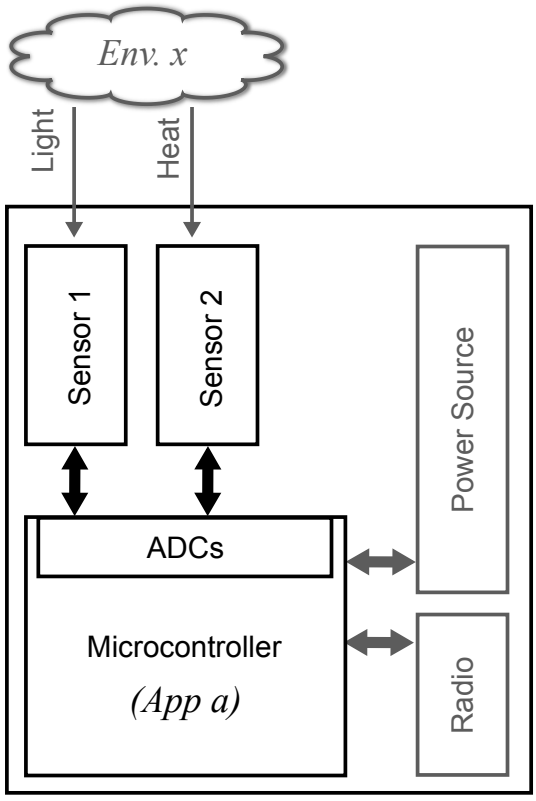


Figure 2.4: A block diagram of a sensor node that shows how physical phenomenon could be read from an Environment (Env. x) and then acquired by the app running on the microcontroller (App a).

generating a Pulse Width Modulated (PWM) wave or using a digital-to-analog converter. In the former method, the MCU generates a square wave without continuously engaging the CPU and the wave's duty cycle can be precisely controlled. The PWM wave can then be converted into an analog voltage by a simple low pass filter (such as an RC circuit). This method is very inexpensive and is useful for low frequency sampling, common in sensor network apps as the nodes themselves have low duty cycles to reduce power consumption. In case an app performs high speed sampling, the latter method of using a DAC is suitable as it has settling times on the order of nanoseconds [37]. If microsecond settling times are acceptable, we can use a DAC with an SPI interface such as MCP4921 [38], which needs a lower number of pins of the MCU. The low pass filter circuit using RC components has

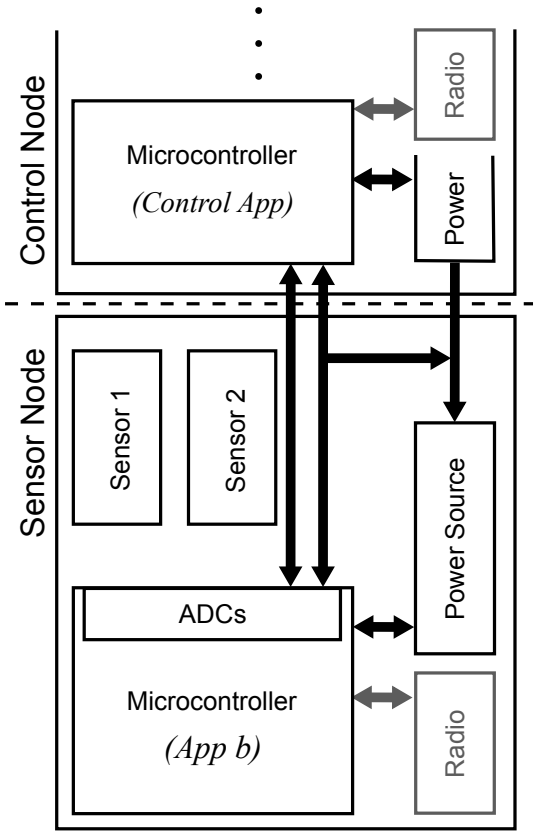


Figure 2.5: Block diagram of how a sensor node can be paired with a control node and stimuli can be provided by the control node by bypassing the sensors on the sensor node. Note that a different app (App *b*) could be running on the sensor node and we can provide the same values acquired by App *a* by generating them in the control node which is running a control app.

settling times in the millisecond range if a stable output voltage is required for some PWM frequencies.

An important factor to consider when generating analog signals is the resolution. The control node should be able to generate signals with at the least the same resolution as the ADC on the sensor node reading the analog voltage. Since it is essentially an identical MCU, the resolution of the PWM wave generator (effectively the resolution of the duty cycle) on the MCU is at least the resolution of the ADC. For example, the ATmega128L MCU [39] (part of the Crossbow MICAz [13] nodes) has ADCs with 10-bit resolution, the

same resolution as that of the PWM wave generator. If a DAC is used for stimuli, we can use higher sampling rates at the sensor node at the cost of using more pins of the control node. Since an MCU has multiple ADCs and PWM generators or even on-board DACs (such as the Texas Instruments MSP430), multiple signals to be used as stimuli can be generated.

2.4.3 Stimuli routing

All analog and digital signals from the control node are routed to unused pins of the sensor node, if available, or to the pins used for connecting the sensors after the sensors have been disconnected. When routing the signals to unused pins, the app needs to be modified to enable reading the signal from the appropriate pin. This is a minimal change that can be done at the device driver level, which typically involves modifying a variable or the mapping in a configuration file.

2.4.4 Power measurement and control

Having pairs of nodes allows the sensor node's power to be provided from the control node, which is in turn powered from the wall socket or a large battery. Control nodes may require more power due to the increased power consumption for generating signals. While it is optional, with a few simple electronic components to measure and control power, we can: 1) simulate node failures due to energy depletion during an app test, 2) run automated tests where node reboots may be required between different experiments, 3) measure the power consumption of the sensor node, and, 4) control power consumption to emulate different battery capacities and discharge curves. The ADC in the MCU on the control node combined with a shunt resistor can be used for measuring power consumption. This approach allows foregoing dedicated hardware although at the cost of reduced sampling

frequency [40]. Battery emulation can be implemented with a simplified version of a system such as B# [41, 42] which requires an MCU and a few electronic components that are already available on the control node.

2.4.5 Sensor node reprogramming

Another advantage of this architecture comes from connecting the programming lines of the sensor node to the control node. This allows sending the app binary over-the-air to the control node, which then reprograms the sensor node without the need for modifying the test app to include extra software such as Deluge [43] to enable this functionality.

2.4.6 Programming the control node

The data necessary to generate the appropriate stimuli signals need to be available on the control nodes. Since our control node is actually a sensor node in disguise, we can simply use the same tools or languages used to program the sensor node. Programming a control node only requires knowledge of how to access and control the signal generation capabilities of the MCU.

We can, therefore, program the control node to produce stimuli according to a pre-programmed sequence. This stimuli can be sensor readings taken in the target environment, which allows testing modifications to the app or optimization without redeploying the system in the target environment. The stimuli can also be randomly generated or follow arbitrary patterns. Either data can be stored on the non-volatile memory available on the control node without using any memory of the sensor node. We refer to the data required to generate stimuli henceforth as stimuli data.

2.4.7 Networking

Storing stimuli data on the control node may not be feasible when the app under test runs for extended periods of time. This can be easily overcome if we can send stimuli data as needed. This is possible when using the radio on the control node, another advantage of using a sensor node as a control node. Since the sensor network is usually a connected network and the nodes are paired, the control nodes can form a connected network as seen in Figure 2.6 and the stimuli data can be sent from a separate control base station to prevent interfering with the operation of the sensor network's own base station. When network partitioning is expected, control network base stations can be deployed at multiple locations without any effect on the test app. Topology control algorithms can also be used for automatically forming the control network.

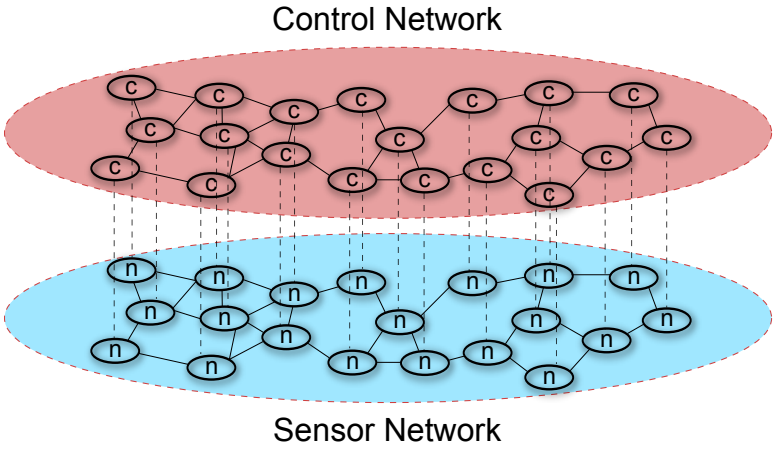


Figure 2.6: The parallel and identical control network that is composed only of control nodes. The control nodes are shown running an app c and the sensor network under test is shown running an app n

An example of how networking can be leverages is as follows. When node 18 running app a (consider $n = a$ in Figure 2.6) senses value 300 from its light sensor, this value is stored at the base station. When the sensor network runs app b (now $n = b$), the control network base station sends the value 300 to control node 18 (sensor node 18's corresponding control

node) and it generates the analog signals that the ADC on the sensor node's microcontroller would "sense" as value 300. However, app b is unaware that the signal it acquired is not from a real sensor as we only modify it at the lower layers to read from an unused ADC and not an ADC connected to the real sensor. The sensor is still powered on (ensuring power measurements are still valid) but not read from.

Sending stimuli data through the radio also allows observing the app's response to real-time changes in stimuli. To prevent radio interference, a source of data contention in DSN [35], all the control nodes' radios can be programmed to work on a different channel for all stimuli/control data traffic and also be used for time synchronization. A time synchronization protocol such as TinySeRSync [44] or FTSP [45] can be included in the control node software for accurate timing of stimuli. apps where communication among nodes do not depend on the time of the stimuli event may not require the control network to be continuously synchronized (while accounting for clock drift) and a simpler timing broadcast signal can be used. Apps that react depending on the timing of the signal would require a synchronized network for better accuracy. Also, if two different radio channels are used, the control nodes can be programmed to listen on the sensor nodes' channel to learn when the app starts and then switch over to a different channel to receive stimuli data.

When implementing this architecture, an app is completely unaware that the sensor readings are artificial and that it is running in an isolated environment. The only input (other than the sensors and power) to the sensor node that is not artificial is the radio. If needed, RF generators and interference generators can be used as mentioned in EmPro [34] to simulate RF signals, although it would be expensive to outfit every node with a unit.
To summarize, the architecture offers very desirable features:

- introduces minimal interference to the app during the test

- can be implemented with minimal software modification to the sensor node for rerouting the sensor input,
- does not need expensive or custom external components to generate and communicate the signals to the sensor node,
- can sustain operations for the entire duration of the test by continuously receiving stimuli data through the radio, and,
- is programmable to automate tests with the same programming language and tools as the sensor node.

In the next section, we describe a prototype CSP (Control Sensor node Pair) built to test the stimuli and control capabilities of the architecture.

2.5 Prototype

The design goal of building a prototype was to use hardware and software that had open architectures and implement the stimuli system using the fewest and simplest components possible. We built the CSPs using MICAz [13] nodes from Crossbow Technologies Inc. and the open source TinyOS [46] operating system. TinyOS was used to build a simple test app and the software for the control node. A photograph of the prototype highlighting the pins relevant to stimuli generation and routing is shown in Figure 2.7.

The MICAz sensor node is powered by the ATmega128L MCU which has 8 ADCs with 10-bit resolution, 6 PWM outputs with a maximum resolution of 16-bits and 2 PWM outputs with 8-bit resolution. DACs can be substituted for the low-resolution PWM outputs when 8 analog outputs are needed. It has the CC2420 radio that can be operated on 16 different frequencies (channels 11–26) in the 2.4GHz ISM band. The sensors for the MICAz

are built on a separate board (MDA100CB) for modularity and connect to the MICAz node through a 51-pin small form-factor connector which provides access to all the pins of the MCU. The board provides two sensors, a light sensor that is a CdSe photocell, which changes its resistance depending on the light level, and a temperature sensor, which is a thermistor. The pins (or ports) of the MCU are named differently on the MDA100CB [14] and these names have been used here for clarity.

2.5.1 Sensor node modifications

To test the stimuli capabilities, we choose to replace an analog sensor and the CdSe photocell rather than the thermistor, because, for testing purposes, rapidly variable light sources are readily available compared to heat sources. The light and temperature sensors are read

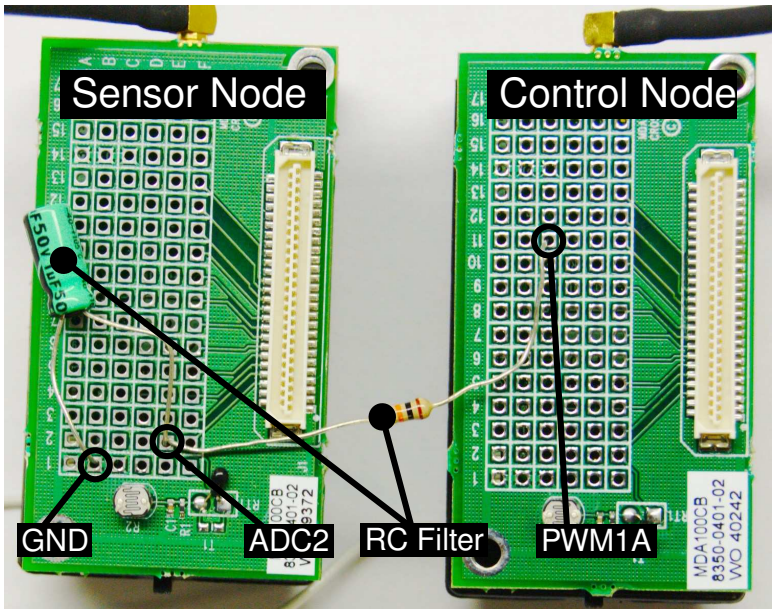


Figure 2.7: Photograph of a prototyped CSP showing the RC low-pass filter used and the pins relevant to stimuli generation and routing on a MDA100CB board. The control node is shown without the MIB520 programming board (which powers both of them) to provide better visibility. The wires for disabling/resetting the sensor node, measuring the power draw (with a small resistor) and setting equal ground potentials for both nodes are not shown.

through a single ADC, pin ADC1 on the MCU, which has a 10-bit resolution. To obtain a sample, the power (V_{cc}) for the appropriate sensor is turned on, and together with a $10\,K\Omega$ resistor, it forms a voltage divider and the voltage is read through pin ADC1. To enable the node to read from another ADC, `PhotoTempConfigC.nc`, a low-level layer of the software representation of the light sensor in TinyOS (`PhotoC`), was modified to enable reading the voltage through pin ADC2 of the MCU. The modification was done to allow transparent reading of the voltage through ADC2 while still powering the CdSe photocell, so that power consumption is not affected. The modification is trivial and does not affect other hardware components. Any test TinyOS app wishing to incorporate this change only needs to have the modified `PhotoTempConfigC.nc` file present with its source files before compilation. No other component of the app needs to be modified.

The RESET pin of the node was connected to the control node to allow it disable or reset the sensor node when needed. The power was provided from the control node.

2.5.2 Control node modifications

To generate analog voltages, we generated a PWM wave and used an RC low-pass filter circuit. An RC filter was used instead of a DAC since one of the goals was to use the least wiring and simple components. The MDA100CB board provides access to one of the pins of the MCU, which can output PWM: PWM1A. Although the MCU is capable of generating PWM waves of different resolutions and accuracies, we chose to generate a 10-bit resolution PWM wave which can be used for conversion to different analog voltages. The PWM wave is then fed through an RC low-pass filter circuit where $R = 10\,\mathrm{K}\Omega$ and $C = 1\mu\,\mathrm{F}$ for a time constant of 10 milliseconds. This allows faster response at the cost of reduced accuracy, as lesser ripple can be obtained only with large time constants. This analog voltage from the filter, variable from 0 to V_{cc} (3 V) with 10-bit resolution, is then routed

to ADC2 of the sensor node. To enable software control of the PWM wave, we developed a separate TinyOS component, `PWMControlC.nc` (see Appendix A) which abstracts the low-level implementation and provides a simple interface to a high-level app, enabling it to either turn on, turn off, or set the duty cycle of the PWM wave. The component also disables MCU sleeping, as TinyOS puts the MCU into a sleep state where it does not generate PWM. For stimuli routing, the control and sensor nodes were connected as shown in Figure 2.7.

Pin PW7 of the control node was wired to the sensor node's RESET pin to allow disabling or resetting the sensor node. We wrote a simple software component to allow either disabling the sensor node by setting the output low or resetting it by momentarily setting the pin low. The former functionality allows controlled node failures to study app behavior and the latter allows restarting the node when performing multiple automated tests.

Since the voltage produced on the control node is measured on the sensor node, the nodes need to have a common ground, which was established by shorting the ground pins (GND). The control node, powered by connecting it to a MIB520 programming board which draws power through the USB ports of a PC, also provided power to the sensor node. The PC could be substituted with a powered USB hub. This enabled us to evaluate the CSP without using large batteries.

The power to the sensor node was routed through a 51 Ω resistor (not shown in Figure 2.7) to allow measuring the power drawn by the sensor node. By measuring the voltage across the control node (which can be measured by itself) and measuring the voltage of the sensor node through an ADC (ADC2 of the control node) we can find the potential difference across the small resistor through which the sensor node draws current. This al-

lows us to calculate the real power drawn by the sensor node. The sensor node app can be modified to measure its own voltage but we cannot calculate the power drawn since the internal resistance of the node varies depending on the current activity of the node (such as sensing, sleeping, transmitting, etc.). Therefore, using the control node allows us to *a*) measure the exact power draw and *b*) obtain significantly more samples without affecting app behavior.

In the next section we evaluate the stimuli, sensor node control disabling/resetting, and the power measurement capabilities through experimentation.

2.6 Prototype Evaluation

To evaluate the stimuli system, we designed experiments to test the accuracy of stimuli generation and the operation of CSPs as a network. *Exp. 1* tests the accuracy of the system in recreating a pre-recorded set of values and *Exp. 2* tests the accuracy of generating an arbitrary set of values. *Exp. 3* tests the accuracy of the system in recreating values that are read as a stream by the sensor node (such as an accelerometer). To evaluate the control and measurement capabilities, we designed a different set of experiments and present their corresponding results. We first describe the setup used for the experiments (Figure 2.8) and discuss each experiment and its related results.

2.6.1 Stimuli system

To create a stimuli system, we built five prototype CSPs and placed them in locations with different light levels throughout a laboratory room.

An app that periodically samples light was built in TinyOS (using the nesC programming language) to serve as the test app and was used for both *Exp. 1* and *Exp. 2*. It is designed to sample light every 5 seconds and obtain 25 samples over a period of 125 seconds. To

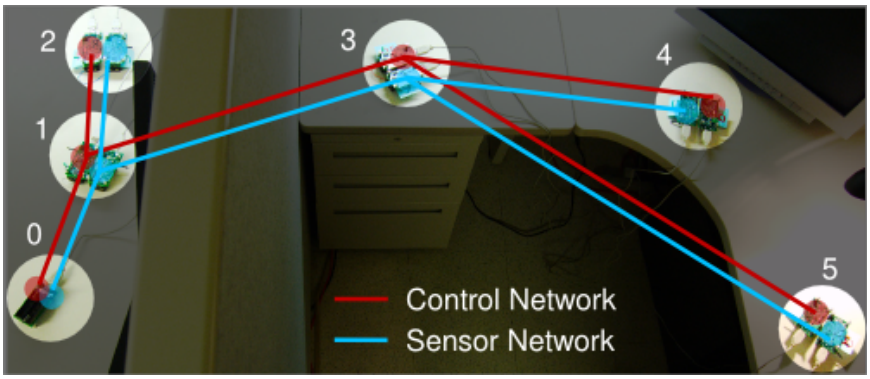


Figure 2.8: Photograph of the network of CSPs used for evaluation. They were photographed from above and are spotlighted for better visibility. CSP 1 is the root of two tree topologies and connects to the common base station (0).

provide stimuli for the sensor node, a custom TinyOS app for the control node was built. It can receive a set of time-value pairs over the radio and maintain the specified value for the specified time. This allows replicating a time varying digital signal similar to the one shown in Figure 2.1. This stimuli is then sampled at periodic intervals by the test app which is not aware that the stimuli is artificial.

Software for the base station was developed using the Java™ programming language to control the execution of the experiments by communicating with both the sensor and control nodes. Note that any programming language that can control the sensor network base station (such as Python™) can be used for controlling the control network. While we used static routing to create two similar but independent tree topologies, one with the sensor nodes and another with the control nodes (Figure 2.8), any topology control or routing protocol can be substituted. The base station communicates with the root nodes of both trees to send and receive data and control signals. The software on the sensor nodes, control nodes, and the base station were designed and deployed to enable automating the test process.

Exp. 1: Record and Replay

In this experiment, we test the accuracy of the stimuli system in replaying a pre-recorded set of light samples. This enables us to evaluate an app with a set of values (not necessarily light samples) recorded in another environment without deploying the nodes in that environment for every test. Since there are 5 nodes in the network, we need 5 sets of light samples. To acquire these, we used the existing control network of 5 nodes to periodically sample light every 5 seconds simultaneously for 125 seconds. They were at 5 different locations (Figure 2.8) and we manually varied the light levels at the 5 nodes to obtain a variety of values. Since all nodes have a 10-bit ADC, all collected values (25 samples $\times$ 5 nodes) were integers in the interval $[0, 1023]$.

The pre-recorded sample values are now considered as the stimuli data for the control network, the ground truth for this experiment. Once the system is reset and the stimuli data is disseminated through the control network, the base station first broadcasts a small control packet to the control network to start the PWM to allow settling time for the RC filter circuit. The settling time is the time difference between setting a certain duty cycle on the PWM and obtaining the corresponding analog voltage from the RC filter circuit, which is approximately 10 milliseconds for our filter. A control packet that notifies the test app to start is then broadcast to both the sensor and control nodes. This packet could be part of a time synchronization protocol if one is used. When the test app executes, the control nodes adjust the PWM period to generate the appropriate value of analog voltage to be sampled by the sensor node at the specified times. The control nodes are programmed to consider the time constant of the RC filter circuit and adjust their timing appropriately.

When the experiment is complete, the base station acquires the sensed values from each sensor node. Data from one of the nodes is shown in Figure 2.9, where the samples are

shown as points on the line. Although some sensed values are not equal to the stimuli provided (a sample is inset), the sensed values exhibit minimal deviation and the stimuli system has reproduced the original input with a high degree of accuracy.

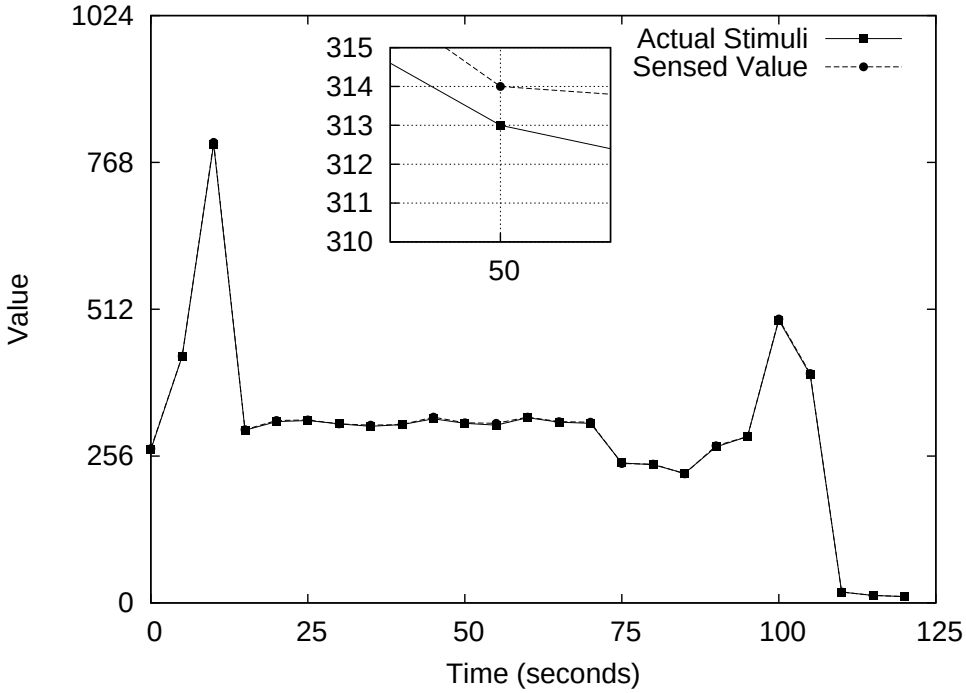


Figure 2.9: The sensed values for a sample run of the experiment overlaid with the given stimuli (ground truth data). The sensed values are not visible as they show minimal deviation from the stimuli.

To measure the accuracy of the system, we find the distribution of errors (the difference between the actual stimuli and the sensed value) for the values obtained from this node. Since the range of values sensed is an integer in the interval $[0, 1023]$, the errors are integers in the interval $[-1023, 1023]$. Figure 2.10 shows that the distribution of errors is clustered around 0, ranging from -1 to 3. The uneven distribution of errors is a result of using an inexpensive RC filter to convert the PWM wave to an analog voltage. The RC filter's output voltage is not perfectly smooth and contains ripples. Although they could be reduced by using a filter with a larger time constant, this would result in a larger settling time.

To evaluate the stimuli system over multiple runs of the same experiment using the

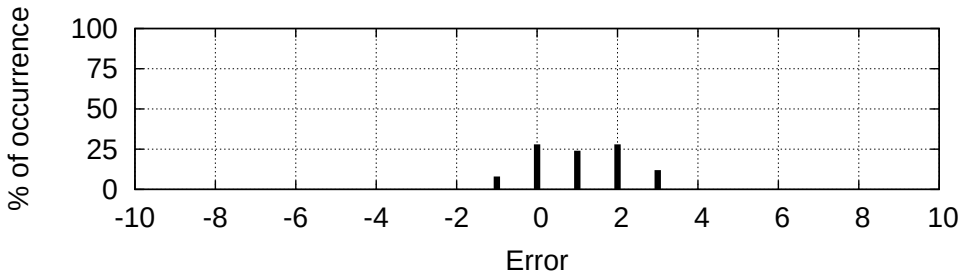


Figure 2.10: Distribution of errors for a sample run of the experiment showing the errors clustered around 0 and within -1 and 3.

same input, we repeated the experiment 100 times. For each node, the average value at each sampling instant is calculated over the 100 runs. This gives us a set of time-value pairs for each node. The values of 5 nodes are then averaged for every time instant to obtain the average value sensed as a network. Figure 2.11 shows this averaged light value sensed by the 5 nodes over a period of 125 seconds. Each sensed value is shown with error bars that indicate the maximum and minimum values that were sensed over 100 runs for all 5 nodes (5 nodes $\times$ 100 runs). The errors bars are not visible as there was minimal deviation, but the inset plot enlarges the difference between the stimuli and sensed values at 50 seconds for better visibility.

To better visualize and accurately quantify the distribution of errors, we use all the data over 100 runs without averaging, obtaining 12500 samples (5 nodes $\times$ 100 runs $\times$ 25 samples). Figure 2.12 shows the error distribution calculated from this data. The errors are again clustered around 0 and range from -2 to 4. Relative errors of the stimuli system were calculated using this data. Using the 12500 samples, we found that the mean relative error is 2.319% and, with 95% confidence, is in the interval $[2.245\%, 2.392\%]$.

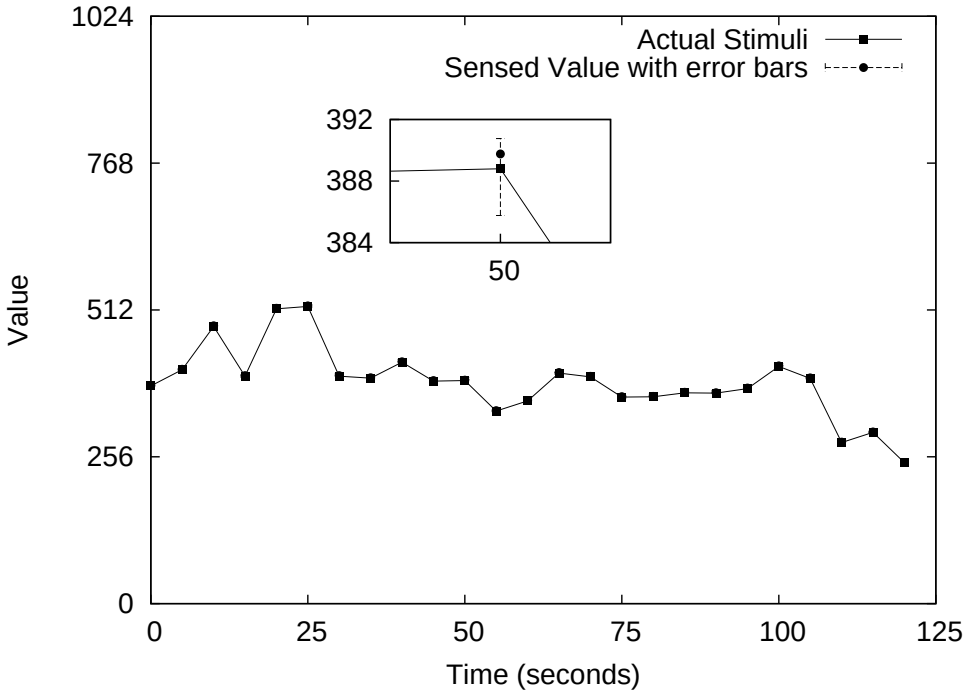


Figure 2.11: The average values of all 5 nodes in the network at different points in time (averaged over 100 runs). Inset plot with error bars shows that there are very low errors.

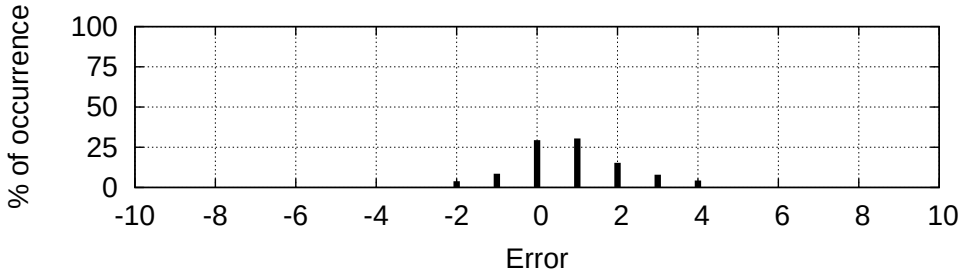


Figure 2.12: Distribution of errors with 12500 samples (5 nodes $\times$ 100 runs $\times$ 25 samples) showing errors clustered around zero and within -2 and 4.

Exp. 2: Pattern play

In this experiment, we evaluate the capability of the stimuli system to generate arbitrary stimuli. We chose to use samples of sine wave as the stimuli for the same test app. Similar to the previous experiment, as shown in Figure 2.13, we find that the sensed values have minimal error for a single experiment on a node. The inset plot enlarges the error at time

instant 45.

To quantify accuracy of reproducing an arbitrary data pattern, we again find the distribution of errors using the 12500 samples. The error distribution is shown in Figure 2.14 and, similar to the previous plots, is clustered around 0 but has a slightly different distribution. The errors range from -3 to 5, again due to the RC filter, but non-zero errors are lower (63.3%) compared to the previous experiment (70.7%). The mean relative error for this experiment was 1.137% and the 95% confidence interval is $[1.088\%, 1.186\%]$.

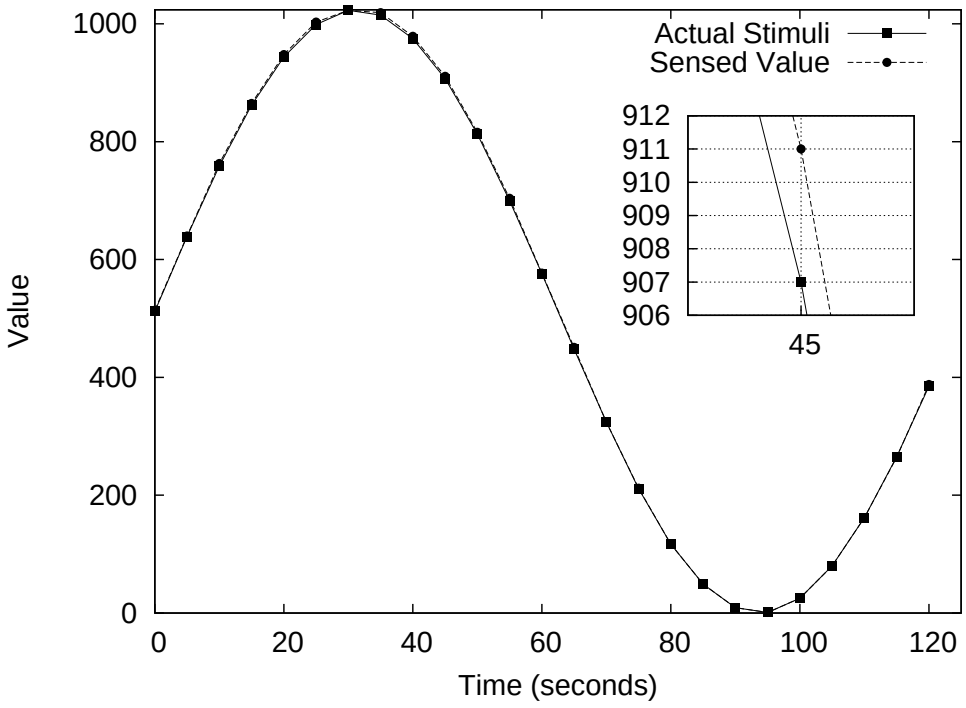


Figure 2.13: The sensed values of a node overlaid with the given artificial stimuli. The errors are again too small to be visible and one of them (at time=45) is enlarged for improved visibility (inset).

The experiments show that the stimuli system is capable of artificially reproducing both pre-recorded and arbitrary data, with high accuracy, while using a simple RC filter in a networked environment.

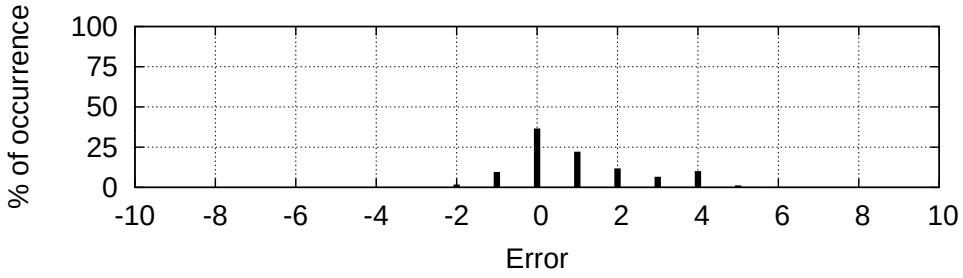


Figure 2.14: Distribution of errors for 100 runs of the Exp. 2: showing errors clustered around 0 and having a slightly larger error range compared to Exp. 1.

Exp. 3: Replay stream data

Some sensors provide data as a stream. To evaluate the system's ability to replay such data, we designed an experiment to replay the data obtained from an accelerometer using the same external hardware used in the previous experiments.

The data was obtained from a node with an accelerometer sensor board that was secured at the end of a flexible wooden ruler which was set to oscillate. This resulted in a damped oscillation and, by sampling the relevant axis of the accelerometer, we obtained the data as a stream. TinyOS provides software components for reading streamed data. The sensor board used was the Crossbow MTS310 and the sampling frequency was 50 Hz. This data was considered the stimuli data and is shown in Figure 2.15 labeled 'Actual Stimuli'.

Once the data was collected, a CSP was set up similar to the previous experiments to test the efficacy of the system in replaying the collected stream and determining the accuracy. The sensed values from one run of the experiment are shown in Figure 2.15. Since the values are very close to the actual stimuli, there is a lot of overlap. We see that the node was idle until about 15 seconds into the experiment, started a damped oscillation and then came to rest around the 50 second mark. Although the values overlap, we do note that the error is more visible compared to the previous two experiments. TinyOS software

for the experiment did not require any extra components other than what was developed for the previous experiments together with other standard components to enable sampling the accelerometer.

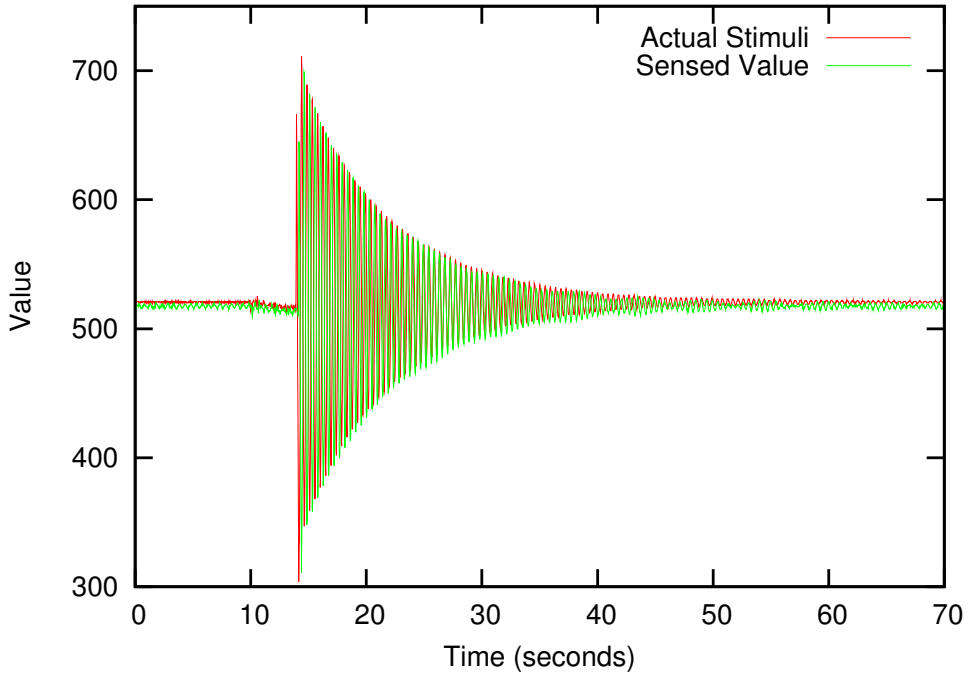


Figure 2.15: The data stream collected from the accelerometer overlaid with the given artificial stimuli. Compared to the last two experiments, errors are noticeable, although the sensed values almost overlay the actual stimuli.

To determine the accuracy of the system, we repeated the experiment 10 times. Each run of the experiment yielded 3500 data points for a total of 35,000 data points. The distribution of absolute errors is larger compared to the previous experiments and has about 85% of the errors distributed in the range $[-30, 25]$.

The distribution of absolute errors is shown in Figure 2.16. We note that the errors are clustered around 0 and are close to 0% beyond the range $[-30, 25]$. We calculated the relative errors for the experiment and the mean relative error is 0.042% and with 95% confidence is in the interval [0.041%, 0.043%]. We see that the simple low-pass filter actually provides fairly accurate stimuli even for a high-frequency sampling app and the experiment

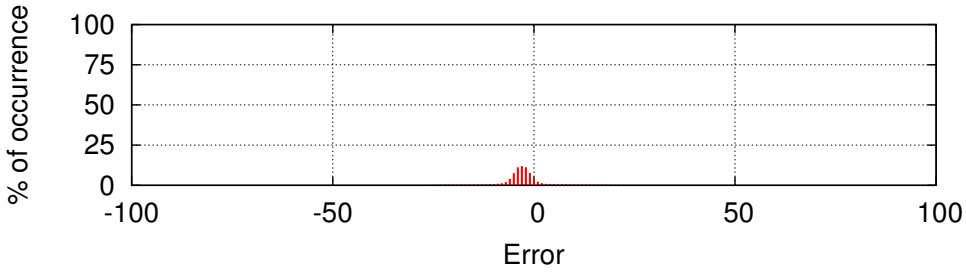


Figure 2.16: Distribution of errors for 10 runs of Exp. 3 showing errors clustered around 0.

needed no change to the hardware involved. The addition of a DAC or more complicated hardware interconnects can further increase the accuracy.

2.6.2 Control

To test the disabling and resetting capabilities, we built a simple TinyOS software component that allows the control app to perform either function with ease. The software, by default, has the RESET pin of the sensor node set high (since it is an active low pin) and then switches the state when needed by toggling pin PW7 on the control node. Note that the test app needs at least 2 seconds before it starts executing. This is normal and is the time required for the power up and internal set up of the TinyOS system. This can be observed in Figure 2.17 (explained in the following section).

2.6.3 Power measurement

To evaluate the power measurement capabilities we compare the measurements obtained from the control node with that obtained from a multimeter. To enable measuring different power draws we needed apps that varied in power consumption. For this, we used or modified existing TinyOS apps or built our own that repeatedly performed a single task.

Table 2.1 shows the summary of results. Note that the null app was programmed to not perform any task and 1 byte packets were used for all radio related operations. The voltage

Table 2.1: Voltages measured by a multimeter and the control node for different apps.

App	Multimeter measurement (Volts)	ADC measurement (Volts)
Null app	3.24	3.233–3.245
Self voltage measurement	3.23	3.107–3.248
Continuous light sensing	3.07	3.033–3.096
3 LEDs on	2.92	2.888–2.929
Transmitting packets	2.19	2.132–2.332
Idle listening	2.18	2.171–2.180
Receiving packets	2.17	2.103–2.187

measured by a multimeter across the sensor node when each app was running is shown in column 2. Column 3 shows the range of voltages measured by the ADC on the control node (after conversion) from 100 samples, each 125 milliseconds apart, for each app. Note that as the app executes on the sensor node there are subtle variations in the power drawn and while the multimeter usually displays the average on a time-varying signal, the ADC on the control node can provide us with more accurate readouts.

The table shows that the measurements from the control node closely match those from the multimeter. The table also shows the efficiency of the TinyOS operating system as we see that the apps use only the power that they need. The null and self voltage measuring app use the least power (higher voltage means lower power consumption) whereas the radio uses the most power.

In order to study the effect of measuring power in an app, we designed a simple experiment that compares the power consumption of an app that does not or does measure its voltage when executing.

For this experiment, we chose the *Blink* app, provided as a sample app with TinyOS. The Blink app repeats a binary sequence from 0 to 7 using the 3 LEDs and changes state

every 250 milliseconds. We first measured the power consumption of this app using the control node for 2 sequences. We then added a component to the app that measures its own voltage every 250 milliseconds and again measured its power consumption.

Figure 2.17 shows a comparison of these measurements against time. The app is started around Time = 1 s and we notice that the voltage decreases dramatically as the node boots up signifying that there is significant power consumption. The pattern observed from Time = 3 s to Time = 7 s and repeated again from Time = 7 s to Time = 11 s are the two binary sequences. The node is off around Time = 11 s.

The plot confirms that when an app measures its own voltage its power consumption changes. Therefore, the use of a control node allows measuring power consumption with lower observer effects.

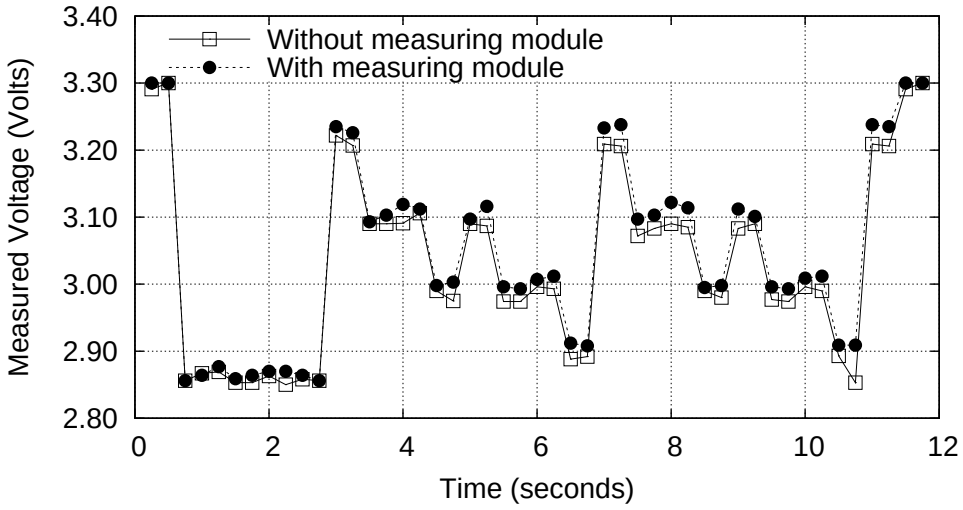


Figure 2.17: Voltage measurements against time for the *Blink* app when measuring its own voltage and otherwise.

2.7 Conclusion

In this chapter we presented the architecture of the Doppel system which uses an identical network of sensor nodes called control nodes to allow controlled testing of a sensor network

running an app. In the control and sensor node pair (the CSP), the control node can provide the stimuli and exercise control over the other node which runs the test app. This architecture combines the benefits of using hardware-based stimuli for fidelity and software-based programmability for ease of control.

A prototype implementation of the Doppel architecture for generating analog stimuli was also presented. The prototype was evaluated as a networked system, presenting experimental results quantifying the accuracy of the system when reproducing pre-recorded data, generating arbitrary signals and reproducing high-frequency sampled data using very simple external hardware components.

The results show that the architecture is viable as an artificial stimuli system with high accuracy. We also showed how the system can be used for emulating node failures and accurate power measurement. Because the control system design and operation is not dependent on the sensor network and app under test, the architecture can be extended for testing systems that require repeated recreation of distributed state. Ad hoc and other wireless mesh networks that have nodes that are distributed in space and are not easily interfaced with a centralized test network can also benefit from the system.

In the next chapter, we will look at how the control network can be managed efficiently for large networks that have long-running experiments.

Chapter 3: System Management

In this chapter we look at efficiently managing dissemination of data through the control network by strategic location of base stations. This is required because of the need to deliver the appropriate control node data to the corresponding control node through the control network, i.e., using control nodes to relay packets, while making sure the packets are delivered on time and with certainty as the loss of a packet could potentially invalidate the test.

We first explain this problem with a simple example. We then show how we can simplify the problem and formulate it as an optimization problem that takes into account the constraints imposed by the network. We model this optimization problem as the well-known facility location problem. We will overview the facility location problem and our formulation. Solving the problem, we can find out where to locate the base stations, what data each base station should hold, and how to route the data to each control node.

3.1 Efficient data dissemination

The control network is composed of control nodes that require data to either produce the stimuli or perform some control action. Since the control network is simultaneously operating with the sensor network under test, each control node cannot hold the data required for the entire duration of the test especially if the network is large and the test is long-running. This is due to the limited storage available on each control node.

We can trivially solve the problem by locating a base station at each node in the network. However, each base station is a point of control for the network and for a large network,

it becomes unwieldy. We also cannot always allocate one base station for a fixed number of nodes as, depending on the problem, the base stations may not be able to meet all the requirements.

Therefore, the data for each control node needs to be transported to the corresponding node through the control network while making sure it is delivered before the time the data is required (for stimuli generation or control). We also need to find routing information to deliver the packets while taking into consideration the buffer capacity of the nodes and the half-duplex nature of the wireless network.

Therefore, given the network topology, the times that each node requires control data and the buffer capacity of each node, solving the problem efficiently will yield the location of the base stations, the data at each base station and the routing information. We will now look at an example of a small 3-node network that illustrates the complexity of the problem.

A 3-node Example

Let us consider a simple network of 3 nodes as shown in Figure 3.1. In this network, all 3 nodes are connected to each other and can transfer packets among them. The nodes also require their own control packets at specified times as shown in the figure. The $d_i^t = 1$ means that node i requires a control packet (or has a demand) at time t. If a packet is not required, then $d_i^t = 0$ for node i at time t (not shown in the figure). We also assume that all nodes have a buffer capacity of 2, i.e., they can hold only two packets in their buffer in any time slot. The raises the question as to where base stations need to be located (which nodes they are connected to) and how we can route the packets. Note that we assume that a base stations can directly add packets to a node buffer and time is slotted.

The solution is to place a base station at node 0. The base station can supply the

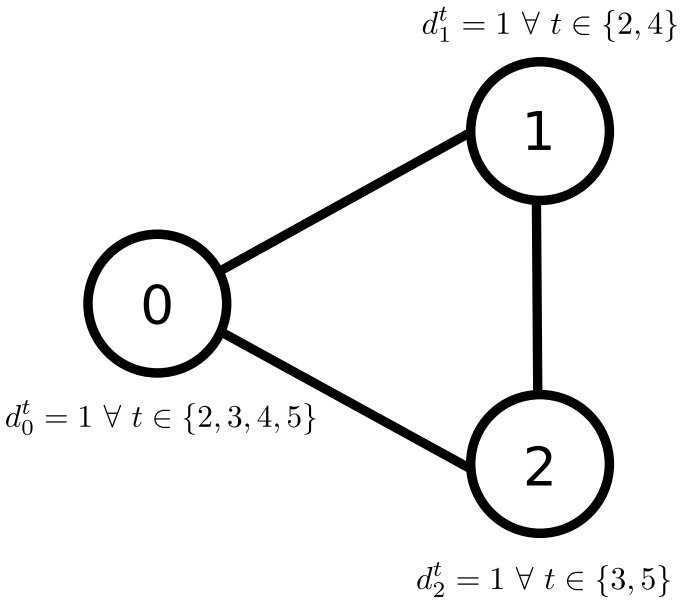


Figure 3.1: A network of 3 nodes (0, 1 and 2) with bidirectional links. When $d_i^t = 1$, it means that node i requires a control packet at time t.

demand of node 0 for all time slots starting at time $t = 1$, i.e., inject the required packets into node 0's buffer. We assume that the packets take one time slot to appear in the node's buffer. At times $t = 1, 2, 3, 4$ it supplies demands of node 0. At times $t = 0$ and $t = 2$ it supplies demands of node 1 (injects them into node 0's buffer) and at times $t = 1$ and $t = 3$, it supplies demands of node 2.

At time $t = 1$, node 0 transmits the packet to node 1 and it appears in node 1's buffer at time $t = 2$. At time $t = 2$, node 0 transmits the packet for node 2 which appears in node 2's buffer at time $t = 3$. This repeats for times $t = 3$ and $t = 4$. Therefore, node 0 alternates transmitting packets between node 1 and node 2. This is because it cannot transmit packets to both node 1 and node 2 at the same time as the packets for the nodes are not identical. The nodes have their demands met as they have the required packets in their buffer at the specified times.

However, when we have only one base station, we cannot simply connect it to any node

in the network. For example, let us assume that the base station is connected to node 1 instead of node 0 as mentioned before. This means that node 1's demands are all met by the base station. However, since it needs to get a packet transmitted at times $t = 1, 2, 3, 4$ to node 0's demand at those times, it cannot transmit to node 2 at times $t = 2$ and $t = 4$ to meet node 2's demand. A similar case arises when the base station is placed at node 2.

Therefore, with one base station, we see that it can be placed only at node 0 in order to satisfy the demands of all nodes. While we can have base stations at nodes 1 and 2 together with the one at node 0, it would be redundant.

Now, let us consider the same problem except that node 1 has a demand at time $t = 3$ in addition to times $t = 2, 4$. This can be seen in Figure 3.2. While there is only a minor change in the problem, the previous solution of having one base station attached to node 0 cannot satisfy the demands of all nodes.

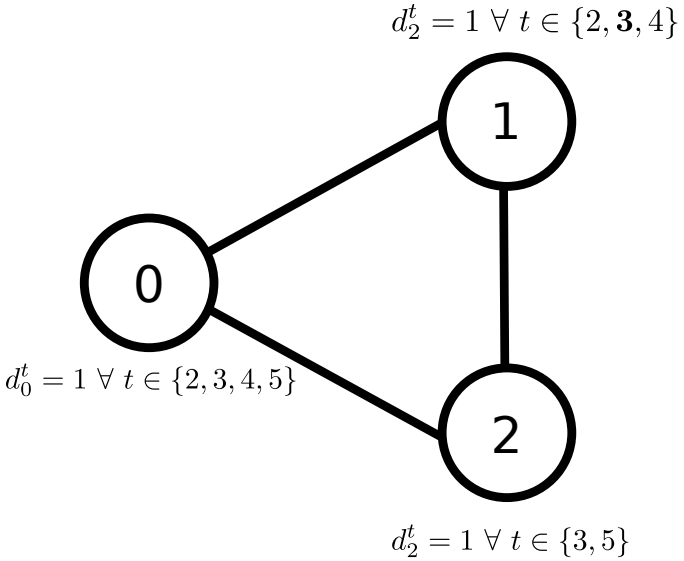


Figure 3.2: The same network shown in Figure 3.1 except that node 1 requires a control packet at time $t = 3$.

The crux of the problem is, at time $t = 3$ both nodes 2 and 3 have demands and when the packets originate at node 0 as the base station is located there, it cannot transmit to

both of them at time $t = 2$. In order to solve this, a second base station needs to placed at either node 1 or node 2. The two base stations can be also be at nodes 1 and 2 and one or both of them can transmit packets to node 0.

Thus, we see that even for a simple 3-node network we cannot just assign a base station to an arbitrary node. The network topology, buffer capacity and the demands of all nodes have to be taken into account. In order to find the solution for an arbitrary network, we can formulate it as an optimization problem. The input variables would be the network topology, buffer capacity of the nodes and the demand times of each node. Solving the problem will yield the least number of base station, where they are to be located and the complete routing information. We model this optimization problem as the well-known facility location problem [48]. We will now overview the facility location problem and our formulation.

3.2 Facility Location Problems

Facility location problems (FLPs) have been extensively studied for their use in supply chain management. In the context of supply chain management, consider the problem of shipping products from factories (facilities) to retail outlets with or without intermediate warehouses. We can select locations where the facilities should be located from a set of candidate locations in order to minimize the transport costs of goods and the storage costs to warehouses and retail outlets. Additionally, there may be time constraints and multiple products involved with each retail location needing a different combination of products [49, 50].

While there are many different types of facility location problems, we will only consider discrete facility location problems where the facilities will be located at pre-selected locations [51] unlike non-discrete FLPs where they can be located anywhere in a given area.

There are also some qualifiers used to identify the common traits in FLPs. A dynamic FLP has time constraints. A capacitated FLP has facilities whose output is limited, limitations on the quantities moved in transport or stored in warehouses and retail locations, etc. A transshipment FLP has products that are shipped through intermediate locations such as warehouses. Similarly, a multi-echelon FLP has multiple intermediate points of shipment or facilities such as national warehouses and local warehouses. A multi-commodity FLP has multiple non-identical products that are shipped through the network.

Our problem of disseminating control packets through the network can be modeled as a FLP. Each node in the network is analogous to a retail outlet if it requires any data at any time, a warehouse if it relays packets, and a facility if it is attached to a base station. It also involves time, limited buffer capacities, multiple-types of packets (one type per node since packets are not interchangeable) and packets that traverse the network through multiple nodes. Therefore, it could be qualified as a dynamic (or multi-period), capacitated, multi-commodity, multi-echelon facility location problem. Our objective, however, is to minimize the number of base stations, which are analogous to the facilities, whereas the common objective is to reduce total costs.

Another difference from traditional formulations is that in our setting, we could have data flowing from any node in the network to any other node in the network and all entities in the network are identical. While there are studies of hierarchical FLPs where there are many layers of entities such as factories, warehouses, distribution centers, retail locations, etc. [52], they are considered distinct and have well-defined product flow paths which are also unidirectional (from factories to end-points). The literature also contains FLP formulations that consider reverse logistics [53] and closed-loop supply chain management [54], which take into account product returns and recycling, but the return paths themselves are also

considered distinct in these problems. Other approaches in the literature, such as using network flow techniques [55] or dynamic transshipment methods [56] also do not account for all constraints.

In the context of networks, FLPs have been used in the context of content distribution networks [57] in wired networks. However, these also consider unidirectional flows and, our network being wireless, we also have half-duplex constraints where each node cannot transmit and receive at the same time instant. While commodity transport restrictions have also been studied, where only a certain quantity of commodity may be moved at any time [58], these different from the half-duplex constraints.

In the context of wireless networks, facility location formulations have been used for service placement [59, 60], clustering [61], etc., but these also do not include all constraints, such as time, buffer capacities, trans-shipments, etc. Protocols for real-time data delivery in sensor networks have also been proposed [62–64], and while they can have high on-time delivery ratios, they do not guarantee on-time delivery. Next, we present our formulation.

3.3 Problem formulation

First, we present assumptions for the problem. We then present the formulation starting with the input and decision variables, and then the objective and constraints. We refer to the nodes that require base station to be attached to them as *injector* nodes.

Assumptions:

- The network consists of stationary nodes with a fixed topology.
- Links are bi-directional.
- Time is slotted and a packet can be successfully transmitted in one slot.

- A node can either transmit or receive a single packet in a time slot and simultaneous transmission between node pairs not sharing a common node is possible.
- If a node is an injector, multiple packets can be added or removed from its buffer in any time slot without affecting its transmission or reception.
- All packets go through a node's buffer when injected into the network, when stored before relaying or when consumed by a node.
- A packet injected at time t appears in the node's buffer at time $t + 1$.
- A node needs to have a packet in its buffer before it can be transmitted.
- A node needs to have a packet in its buffer before it can be used for stimuli or control.
- When a packet is transmitted at time t, it appears in the receiving node's buffer and is removed from the transmitting node's buffer at time $t + 1$.
- All nodes' buffers are empty at time $t = 0$

We model the network as a graph $\mathcal{G}(\mathcal{N}, \mathcal{L})$ which has a set of nodes, $\mathcal{N}$, and a set of links, $\mathcal{L}$, and operates over a set of time slots, $\mathcal{T}$. Transporting node-specific packets is modeled as a multi-commodity problem which means that node 0 requires commodity 0 (referred to as packet type 0). Therefore, the number of packet types equals the number of nodes, i.e., the set of packet types is identical to $\mathcal{N}$. Now we will present the inputs to the problem and the formulation. We will also provide a simple example to illustrate the use of the variables in the problem.

Input Variables:

$\mathcal{N}$ = Set of nodes (and packet types) in the network

$\mathcal{L}$ = Set of links(i, j) in the network, $(i, j) \in \mathcal{L}$ if link (i, j) exists between nodes $i \in \mathcal{N}$ and $j \in \mathcal{N}, i \neq j$

$\mathcal{T}$ = Set of time slots

q = The last slot in set of time slots, $\mathcal{T}$

$$d_i^t = \begin{cases} 1 & \text{if node } i \text{ needs a packet at time } t, \forall i \in \mathcal{N}, \\ & \forall t \in \mathcal{T} \\ 0 & \text{if not,} \end{cases}$$

B = Buffer capacity of a node (identical for all)

Decision Variables:

$$Z_i = \begin{cases} 1 & \text{if node } i \text{ is an injector,} \\ 0 & \text{if not,} \end{cases}$$

$$Y_{i,j}^{k,t} = \begin{cases} 1 & \text{if link } (i,j) \text{ carries packet type } k \\ & \text{in time slot } t, \\ 0 & \text{if not,} \end{cases}$$

$$W_i^{k,t} = \begin{cases} 1 & \text{if node } i \text{ injects packet type } k \\ & \text{in time slot } t, \\ 0 & \text{if not,} \end{cases}$$

$H_i^{k,t}$ = number of packets of packet type k at node i in time slot t

Example:

Consider a simple network shown in Fig. 3.3. Here $\mathcal{N} = \{0, 1\}$. The bidirectional links are given by $\mathcal{L} = \{(0, 1), (1, 0)\}$. Let us say the node 0 requires a packet at time 2 and node 1 at time 3, the set of time slots are $\mathcal{T} = \{0, 1, 2, 3, 4\}$. Note that there is always 1 time slot after the last demand. The last time slot is 4 in this case and $q = \{4\}$. The demands are represented as $d_0^2 = 1$ and $d_1^3 = 1$. This means that node 0 has a demand at time slot $t = 2$ and node 1 has a demand at time slot $t = 3$. Since there are no demands at all other time slots, $d_i^t = 0 \; \forall \; (i \neq 0 \text{ and } t \neq 2), \text{and } (i \neq 1 \text{ and } t \neq 3)$. We assume that the buffer capacity of both nodes is 2, i.e., $B = 2$.

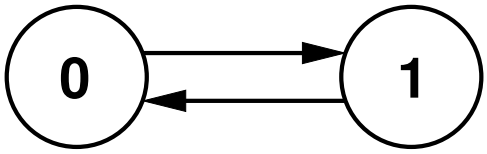


Figure 3.3: A network of 2 nodes with bidirectional links.

For this problem, we can find a solution manually. A simple solution is that the base station is located at node 0 and control packets for both nodes are injected at node 0 at time slot $t = 1$. This is represented as $W_0^{0,1} = 1$ and $W_0^{1,1} = 1$ which means that at node 0, packet types 0 (packet for node 0) and 1 (packet for node 1) are injected at time 1. The injected packets appear in the buffer at the next time slot. Since there are only two demands for the entire duration, only two of the W variables indexed by node, packet type and time slot will be set to 1. All other W variables will be set to 0.

When both the injections happen at node 0 at time slot $t = 1$, the two packets appear in node 0's buffer in the next time slot $t = 2$. Therefore, the buffer values are as follows: $H_0^{0,2} = 1$, $H_0^{1,2} = 1$. This means that the buffer at node 0's has 1 unit of packet type 0 and 1 unit of packet type 1, both at time slot $t = 2$.

Node 0 also has a demand at time slot $t = 2$, represented as $d_1^2 = 1$. The packet is consumed (and is removed) from its buffer at time 3, i.e., the packet needs to exist in the node's buffer at the same time slot as the demand and is consumed in the next time slot. This is represented by $H_0^{0,3} = 0$ which means that node 0 has 0 packets of type 0 at time slot $t = 3$.

At time slot $t = 2$, node 0 also transmits the packet type for node 1 to node 1. This is represented as $Y_{0,1}^{1,2} = 1$. Because of the transmission at time slot $t = 2$, the buffer at node 0 now holds 0 packets of type 1 at time slot $t = 3$ which can be represented as $H_0^{1,3} = 0$. Since the transmitted packet appears in node 1's buffer at time slot 3, it is represented as $H_1^{1,3} = 1$.

This packet that was just transmitted by node 0 to node 1 is consumed by node 1 at time slot 3 (per the demand at node 1, $d_1^3 = 1$). Therefore the buffer at node 1 holds 0 packets of type 1 at time slot 4 which is represented as $H_1^{1,4} = 0$. In this example, although we looked at all the variables that changed through time, we did not look at all variables, i.e., with all different applicable indices. These variables, though are all defined for the problem, are set to 0 and remain set to 0 through all time slots.

We looked at the input and decision variables and will now look into the objective and constraints required to solve the problem in a general context, not specific to the example above.

Objective and Constraints:

$$\min \sum_{i \in \mathcal{N}} Z_i \tag{3.1}$$

subject to

$$H_i^{i,t} \geq d_i^t \qquad \forall i \in \mathcal{N}, \quad \forall t \in \mathcal{T} \tag{3.2}$$

$$H_i^{k,t} \geq Y_{i,j}^{k,t} \qquad \forall (i,j) \in \mathcal{L}, \quad \forall i,k \in \mathcal{N}, \quad \forall t \in \mathcal{T} \backslash q \tag{3.3}$$

$$H_i^{i,t+1} - H_i^{i,t} = W_i^{i,t} - d_i^t + \sum_{j \in \mathcal{N}} Y_{j,i}^{i,t} - \sum_{j \in \mathcal{N}} Y_{i,j}^{i,t} \quad \forall i \in \mathcal{N}, \quad \forall t \in \mathcal{T} \backslash q \tag{3.4}$$

$$H_i^{k,t+1} - H_i^{k,t} = W_i^{k,t} + \sum_{j \in \mathcal{N}} Y_{j,i}^{k,t} - \sum_{j \in \mathcal{N}} Y_{i,j}^{k,t} \qquad \forall i,k \in \mathcal{N}, \quad k \neq i, \quad \forall t \in \mathcal{T} \backslash q \tag{3.5}$$

$$\sum_{k \in \mathcal{N}} \sum_{j \in \mathcal{N}} Y_{j,i}^{k,t} + \sum_{k \in \mathcal{N}} \sum_{j \in \mathcal{N}} Y_{i,j}^{k,t} \leq 1 \qquad \forall i \in \mathcal{N}, \quad \forall t \in \mathcal{T} \tag{3.6}$$

$$\sum_{t \in \mathcal{T}} d_k^t = \sum_{t \in \mathcal{T}} \sum_{i \in \mathcal{N}} W_i^{k,t} \qquad \forall k \in \mathcal{N} \tag{3.7}$$

$$\sum_{k \in \mathcal{N}} H_i^{k,t} \leq B \qquad \forall i \in \mathcal{N}, \quad \forall t \in \mathcal{T} \tag{3.8}$$

$$\sum_{k \in \mathcal{N}} H_i^{k,t} = 0 \qquad \forall i \in \mathcal{N}, \quad t = 0 \tag{3.9}$$

$$W_i^{k,t} \leq Z_i \qquad \forall i,k \in \mathcal{N}, \quad \forall t \in \mathcal{T} \tag{3.10}$$

The term in $\min \sum_{i \in \mathcal{N}} Z_i$ (3.1) denotes the objective. It is a sum of the variable Z, indexed by i which is the id of nodes. When the constraints are met, the nodes that become injectors, i.e., the nodes which will have base stations attached to them will have Z_i set to 1 where i is the id of the node. Therefore, when we minimize (3.1), we minimize the number of injectors (base stations).

Inequality (3.2) ensures that a node's data is in its own buffer before the specific time it is needed, i.e., if a node i has a demand at time slot t, it has a packet of its own type (i) in its buffer at time t ($H_i^{i,t}$). The constraint enforces that if $d_i^t = 1$, then $H_i^{i,t} \geq 1$. This is true for all nodes at all time slots.

Inequality (3.3) ensures that a packet is in the buffer before transmission. Therefore, if a transmission is to occur from node i to node j with a packet type k at time slot t, the buffer at node i has that packet type (k) at time slot t. It is similar to the previous inequality except that it applies for all links in the network, all nodes, all packet types and all time slots.

Equation (3.4) considers nodes and their own packet types and it ensures that the change in buffer capacity for a particular node satisfying its own demand can be attributed to the injection from the node itself, the demand and the incoming and outgoing transmissions. The left-hand side of the equation denotes the change in buffer capacity at a node in a particular time slot t (except the last time slot represented by q). The right-hand side shows that a packet can be injected, used up through a demand, received from a another node or transmitted to another node.

Equation (3.5) is similar to equation (3.4) except it is for nodes that act as relays for data demanded by other nodes. Note that the equation considers changes in the buffer of a node of packets not of the node's own type ($i \neq k$). While the left-hand side is similar to the previous equation, the right hand side does not include the d term. This is because a node will never consume a packet that is not of its own type and therefore that will not effect a change in its buffer for that particular packet type.

Inequality (3.6) ensures that every node transmits or receives one packet or is idle in a time slot. This is the half-duplex constraint. The first term sums all receptions of a node

over all packet types. The second term sums all transmissions leaving a node over all packet types. The ≤ 1 ensures that only one of the sums is true, i.e., one packet of any packet type is either transmitted or received, or the transceiver is idle (sum is 0).

Equation (3.7) ensures all demands are satisfied. For every packet type, the sum of demands over all time slots (the left-hand side) should be equal to the sum of all injections at all nodes (this is because a particular node's demand can be satisfied by injection from multiple points in the network) and over all time slots (the right-hand side).

Equation (3.8) ensures that a node does not exceed its buffer capacity B. This is to be ensured at every node and at every time slot.

Equation (3.9) ensures that the buffer at every node (summed over all packet types) at time slot $t = 0$ is empty.

The last inequality (3.10) determines the nodes that will be the injectors (facilities), i.e., if at any point in time a node injects a packet ($W_i^{k,t} = 1$), the corresponding node becomes an injector ($Z_i = 1$).

In the next section, we will generate small instances of the problem and provide solutions.

3.4 Solving small problem instances

Facility location problems are NP-hard and in this section we provide results from solving simple instances of the problem. Larger instances will be presented later.

We first generated small problem instances, i.e., networks with 5, 8, 10 and 12 nodes. The nodes were distributed uniformly over a 100×100 area with a communication range to produce a connected network. We assigned buffer capacities of 10 packets per node. The nodes were also assigned 10 demands each for all the network sizes. The number of time slots was set to 250. The demands were uniformly distributed (periodic) over the time slots for all nodes. We also added an additional number of time slots equal to the number of

nodes at the beginning.

The problems were solved using the Gurobi 4.0.1 optimizer [65] on a computer with Intel® Core™2 Quad processor and 8 GB of RAM. We used the same formulation presented earlier for all problems without any modifications for speedup. In the solution for all four problems, the objective was 1 i.e., only one node needed to be an injector.

The time for solving each instance is shown in Fig. 3.4. We can see that the time for solving the problem exponentially increases with increase in the network size. This is unsurprising as they are NP-hard problems. The large amount of time consumed for the problems was a result of the basic formulation used. We note that we are using linear programming here as a tool to solve our problem, and therefore we will not focus in providing a more efficient LP formulation.

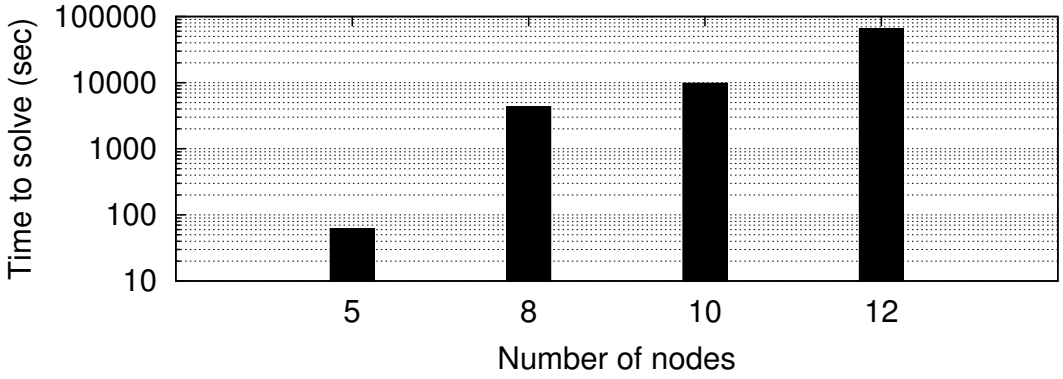


Figure 3.4: Time to solve problems of different network sizes.

To find the efficiency of the solution, we also looked at the number of transmissions required for disseminating the packets. The result is shown in Fig. 3.5 and we can see a linear increase in transmissions as the network size increases. This is not unexpected as the 5-node network has a total of 50 demands while the 12-node network has a total of 120 demands.

The plot in Figure 3.4 shows that it takes almost 20 hours for solving the 12-node

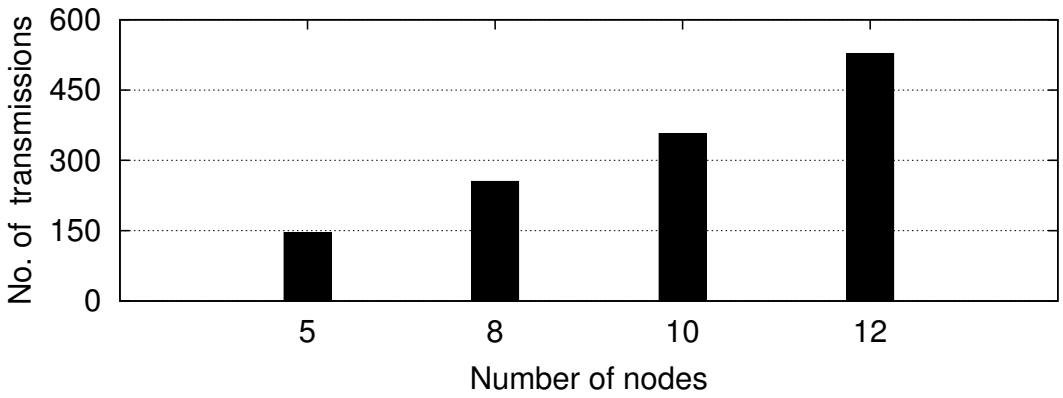


Figure 3.5: Number of transmissions required by the solution for different network sizes.

network problem. This shows that larger networks will require large amounts of time for solving with the basic formulation we provided (we again note that linear programming is being used here as a tool and therefore we do not concentrate on finding the most efficient formulation). In the next chapter we develop an algorithm that can expedite solving the problem while leveraging the same formulation presented here.

3.5 Conclusion

In this chapter we looked at how we can efficiently manage the dissemination of data through a control network by strategic placement of base stations. The trivial solution is to place base stations at every control node and this can be avoided. By taking into account the network topology, the times each control node requires data, buffer constraints, half-duplex constraints, etc. we can find the right locations of base stations.

We illustrate this with an example which showed how even a minor variation in the input can change the base stations required and the routing information. We formulate this problem as an uncapacitated facility location problem, and pay special attention to constraints that stem from the wireless nature of the communications.

In the next chapter, we will look at how we can expedite solving the problem using the same formulation provided here.

Chapter 4: A Hybrid Algorithm

In this chapter we will present a hybrid algorithm that can solve the problem by trading speed for optimality while using an identical formulation of the problem with the solver. We first present the design criteria for the algorithm and look at some facility location problems that have been solved with mathematical techniques or heuristic methods and how we can design an algorithm for solving our problem by hybridizing a metaheuristic and an exact algorithm. We then present the algorithm, illustrating it with pseudocode.

The algorithm is a hybrid solution combining simulated annealing which is a metaheuristic algorithm and a standalone optimization problem solver which is an exact algorithm. Later, we discuss the implementation of the algorithm and evaluate it. We compare the algorithm with a standalone solver using the same problem instances and the same formulation for the solver that were used in the last chapter. Then, we show results from solving problems which have their parameters varied to test their sensitivity. We also look at problems with a grid network topology. Later, we test the effectiveness of the algorithm by using contrived problems that have known optimal values to check how close the objective values found by the algorithm are to the optimal values. We also compare the relative time it takes to solve all problems that were discussed.

4.1 Algorithm Design Criteria

Before we set out to develop an algorithm, we set forth the following criteria:
The algorithm:

- should be faster than a standalone solver for non-trivial problem sizes using the same

formulation,

- should not produce solutions that violate any constraints,
- should be able to find a feasible solution quickly,
- could be aborted at any time to yield the current best solution, and,
- could be run for longer periods of time for improved solution even if there are diminishing returns.

We will now see what solutions are available and how we can design our algorithm making sure all criteria are satisfied.

4.2 Related Work

As seen in the previous chapter, there are few readily available models that can be applied for our problem. Consequently, there are no existing solutions for our model. However, there are both exact algorithms and heuristics that have been used for solving facility location problems.

Some facility location problems that share a lot of characteristics with our problem have been solved with exact methods [66, 67] and also heuristic methods [68] but the models lack non-distinct entities, half-duplex constraints, etc.

Solutions for FLPs in the context of wireless sensor networks can be found for location of multiple sinks and efficient data collection [69–71] but they consider unidirectional data flows, distinct nodes and sinks, no time constraints, etc., and are not applicable.

When considering all constraints and formulated as an optimization problem, it is NP-complete and a heuristic is required to expedite solving similar to p-median problems [72]. Among the metaheuristics used for solving FLPs, we considered genetic algorithms, tabu

search and simulated annealing. Due to the large number of constraints, a simple implementation of any of them by generating random values for all variables could result in a lot of unsatisfied constraints and infeasible solutions. Similarly, using two different levels of metaheuristics (simulated annealing) for finding the facilities and then satisfying all demands [73] would also not be applicable as the heuristics to satisfy all demands would generate many infeasible solutions.

In order to solve this issue, we can use a hybrid algorithm that combines a metaheuristic with an exact algorithm. This is also a common approach to solving facility location problems [74, 75]. Therefore, we can generate random values for the objective variables (Z) using the metaheuristic method and use the exact algorithm to check the feasibility of those values. While we can just use the Gurobi Optimizer for the exact algorithm, we needed to select the metaheuristic to use. This is known as integrative combination as we integrate the exact algorithm into the metaheuristic algorithm [74]. Among metaheuristics, genetic algorithms require a pool of solutions to start solving and since finding each solution involves using the standalone solver, it would require a substantial amount of time even before it starts. Between tabu search and simulated annealing, the former searches around the initial solution space before moving on to the next whereas the latter jumps around the solution space more randomly [76]. Therefore, with simulated annealing, we can sample a larger part of solution space within a few iterations even when the initial solution is not very desirable.

In our hybrid algorithm we combine simulated annealing [77, 78] with the Gurobi Optimizer standalone solver. We will delve into the algorithm in the following section.

4.3 The Hybrid Algorithm

In this section we first give an overview of the algorithm and then, with the aid of pseudocode listings, delve into the details of its operation. The speedup in the algorithm is achieved by

taking advantage of the fact that a feasibility check and partial solve is faster compared to solving the complete problem.

The proposed algorithm uses simulated annealing to generate partial solutions and check the feasibility at each iteration using the solver. At each iteration the simulated annealing algorithm generates a bit string of length $|\mathcal{N}|$, one bit for each Z_i, and we determine if Z (i.e., $Z_i \; \forall \; i$) is feasible using the solver. The energy at each iteration is the number of 1s in the bit string (the number of facilities) when the Z is feasible or $|\mathcal{N}| \times 1000$ if infeasible. The energy is related to the simulated annealing algorithm which will be reviewed later.

This approach allows us to get a feasible solution at the end of a fixed number of iterations. If we resort to generating random values for all variables, it may lead to a large number of iterations of infeasible solutions and may not provide us with us a feasible solution even after a number of iterations.

Now we delve into the pseudocode of the algorithm. We have split the algorithm into many sub-routines (algorithms) for clarity. For brevity and convenience, we use the term Gurobi Optimizer interchangeably with solver.

Algorithm 1 lists the main part of the algorithm which sets up the optimization problem and initializes some variables. We will now go into a line-by-line description of the algorithm. We first create an instance of the problem. The function 'CreateProblemInstance' is a combination of multiple functions available through the Gurobi Optimizer to create an optimization problem instance. This returns an instance of the problem which contains all the input variables and constraints that is ready to be solved by the Gurobi Optimizer. The next two lines create new pseudo-random number generators (P-RNG) that are stored in the respective variables. These can be initialized with different seeds. The next two lines create the *ExploredZ* set data structure that holds all the explored values of Z. It is initialized

with 2 members. These two members denote the infeasible solution ($Z_i = 0 \ \forall \ i \in \mathcal{N}$) and the always feasible solution ($Z_i = 1$). We do this to prevent the solver from using these values in any iteration. The next line initializes a variable called $InfeasibleEnergy$ which is the energy used by the simulated annealing algorithm when the generated Z is an infeasible solution. The next line obtains three parameters for simulated annealing that depend on the size of the problem ($|\mathcal{N}|$). This function is listed as *Algorithm 2* and will be explained later. In the next 2 lines we just combine the parameters needed by the 'SASolve' routine (*Algorithm 3*) and then call the routine with the problem instance and the parameters needed. The parameter $T_{\max}$ is the starting temperature of the annealing algorithm and $T_{\min}$ is the ending temperature. The *steps* parameter is the number of steps or iterations used in the annealing. Using these three parameters, we can find the factor of reduction of the temperature at each iteration. The RNG_{Anneal} is the P-RNG used by the simulated annealing algorithm to accept or reject a solution at each iteration.

The variables $RNG_{\text{Flip}}, ExploredZ$ and $InfeasibleEnergy$ are declared as global variables although they are only accessed by the routines 'FlipBits', 'GetNextState' and 'CalculateEnergy' respectively.

Algorithm 1 Main(ProblemInputs)

1: $PI \leftarrow$ CreateProblemInstance(ProblemInputs)
2: $RNG_{\text{Flip}} \leftarrow$ CreateNewP-RNG()
3: $RNG_{\text{Anneal}} \leftarrow$ CreateNewP-RNG()
4: $ExploredZ \leftarrow \phi \ \cup \ (Z_i = 0 \quad \forall \ i \in |\mathcal{N}|)$
5: $ExploredZ \leftarrow ExploredZ \cup \ (Z_i = 1 \quad \forall \ i \in |\mathcal{N}|)$
6: $InfeasibleEnergy \leftarrow |\mathcal{N}| \times 1000$
7: $T_{\max}, T_{\min}, steps \leftarrow$ GetSAParameters($|\mathcal{N}|$)
8: $AnnealParams \leftarrow (T_{\max}, T_{\min}, steps, RNG_{\text{Anneal}})$
9: SASolve($PI, AnnealParams$)

Now we look into how the parameters for simulated annealing are determined in the

routine 'GetSAParameters' (*Algorithm 2*). The first step finds the respective δ_c and *steps* for a particular N which is passed as a parameter and is equal to $|\mathcal{N}|$ (number of nodes and the length of bit string). The lookup table used is shown in Table 4.1. The parameter δ_c represents the difference between the number of 1 bits in the first iteration of the bit string generation algorithm ('FlipBits') and the second iteration. The *steps* represents the number of steps or iterations required for the bit string to have at most 20% of 1 bits. We get the maximum of both these values out of 10000 runs. We developed a script for automating this.

In the next line we find the temperature the annealing starts at to ensure at least a 60% acceptance rate by the simulated annealing algorithm. We set the minimum temperature to 0.01 in the next line and then return the parameters. Some of these parameters will be more clear when we discuss the 'SimulatedAnneal' function later.

Algorithm 2 GetSAParameters(N)

1: $\delta_c \leftarrow \text{LookupTable}_{\delta_c}(N)$
2: $steps \leftarrow \text{LookupTable}_{steps}(N)$
3: $T_{\max} \leftarrow \lceil \frac{\delta_c}{\log \frac{1}{0.6}} \rceil$
4: $T_{\min} \leftarrow 0.01$
5: **return** $(T_{\max}, T_{\min}, steps)$

Table 4.1: Lookup Table for δ_c and *steps*

$\lvert\mathcal{N}\rvert$	δ_c	*steps*
5	5	20
8	6	55
10	6	45
12	6	60
15	7	55
20	7	75
25	8	95

The 'SASolve' routine (*Algorithm 3*) exists to ensure simulated annealing starts with a feasible solution. We first set *success* which is a binary variable representing the feasibility

of the solution (Z). Then we enter a while loop where we initialize Z to all 1s and then call another routine 'GetNextState' to obtain a modified Z which is not all 1s or 0s or not an already found Z. We then call the Gurobi Optimizer in the next line (routine 'GurobiResetAndSolve') to check if it is a feasible solution. This routine resets the problem instance PI to an unsolved state and checks if Z is a feasible solution. It returns 2 variables: one denoting feasibility (stored in *success*) and when feasible, the value of the objective (stored in *Objective*). The objective is the number of bits that are 1 in Z. This loop therefore continues until the first feasible solution is found i.e., until *success* is set to **true**. Note that the 'GetNextState' routing uses the variable *ExploredZ* to store all the infeasible solutions that were found.

Algorithm 3 SASolve($PI, AnnealParams$)

```
1: success ← false
2: while success is false do
3:     Z_i ← 1 ∀ i
4:     Z ← GetNextState(Z)
5:     success, Objective ← GurobiResetAndSolve(PI, Z)
6: end while
7: SimulatedAnneal(AnnealParams, GetNextState, CalculateEnergy)
8: return Z
```

We will not look into how the 'GetNextState' routine (*Algorithm 4*) works. The routing exists to generate the (next) state for each iteration of the simulated annealing process. Taking Z as a parameter we call a routine to flip the bits in Z ('FlipBits'). We continue this until we find a Z that has not already been found. The data structure *ExploredZ* contains the previously seen values and also includes Zs which are all 1s and all 0s (which were added in the 'Main' routine). Therefore, 'GetNextState' ensures a previously unseen Z is generated each time it is called. However, we makes sure that the Z generated is a modified version of the Z passed as a parameter. The key is the 'FlipBits' routine (*Algorithm 5*). This

routine uses the RNG_{Flip} P-RNG to generate random numbers. It goes through each bit of Z and if it is a 1, it flips it to a zero with probability 0.5 and vice-versa with probability 0.25. The probabilities are set to decrease the number of 1s in the solution and therefore minimize the objective.

Algorithm 4 GetNextState(Z)

```
1: repeat
2:    Z ← FlipBits(Z)
3: until Z ∉ ExploredZ
4: ExploredZ ← ExploredZ ∪ Z
5: return Z
```

Algorithm 5 FlipBits(Z)

```
1: r ← GetNextFloat(RNG_Flip)
2: for i = 1 to |N| do
3:    if Z_i = 1 and r < 0.50 then
4:       Z_i ← 0
5:    else if Z_i = 0 and r < 0.25 then
6:       Z_i ← 1
7:    end if
8: end for
9: return Z
```

The last routine 'CalculateEnergy' is a routine needed by the 'SimulatedAnneal' routine. We will first give an overview of the latter. The 'SimulatedAnneal' routine is a simple simulated annealing routine which takes some of the parameters needed for annealing and two other routines as parameters and operates as follows. It first calculates the energy of the system using the routine passed to it ('CalculateEnergy') at the first iteration. For the next iteration, it calls the 'GetNextState' routine to obtain a new value of of the solution (Z in our case) and then calls the 'CalculateEnergy' routine to calculate the energy. If the energy has increased and the difference is δ_E, it accepts the solution as the current solution with the probability $e^{\delta_E/T}$ where T is the current temperature. It also accepts the solution

if there is a decrease in energy. It then decreases the temperature based on a reduction factor (calculated from the maximum and minimum temperatures and the number of steps) and continues. During this time, it also keeps track of the lowest energy found and the corresponding solution. The routine also exits immediately if the energy (value of the objective) is found to be 1 as this is the minimum objective.

Since we have already discussed the 'GetNextState' routine, we will now describe the 'CalculateEnergy' routing (*Algorithm 6*). The routine is called exclusively by the simulated annealing algorithm and takes the value of Z that was generated using the 'GetNextState' routine as a parameter.

We start by assuming that the Z is an infeasible solution and therefore setting the energy equal to $InfeasibleEnergy$ (initialized in the 'Main' routine). Then, we just call the Gurobi Optimizer with the problem instance PI and the Z to check feasibility (while also solving). If it is feasible, we set the energy to the objective value returned by the solver (equal to the number of 1s in Z). We then return the energy.

Algorithm 6 CalculateEnergy(Z)

```
1: Energy ← InfeasibleEnergy
2: success, Objective ← GurobiResetAndSolve(PI, Z)
3: if success is true then
4:     Energy ← Objective
5: end if
6: return Energy
```

The 'GurobiResetAndSolve' routine performs another minor operation for increasing the solving speed. Since we the know the value of Z, we can determine the values of some variables. After the routine resets the problem instance to an unsolved state, it sets values for these variables before calling the final solving routine of Gurobi. We will now look into how the values of some variables can be inferred from Z.

Determining variable values from Z

We will first define two sets F (Equation 4.1) and F' (Equation 4.2) which represent the set of nodes that are facilities and set of nodes that are non-facilities respectively.

$$F = \{i : Z_i = 1, \quad \forall\ i\ \in\ \mathcal{N}\} \tag{4.1}$$

$$F' = \{i : Z_i = 0, \quad \forall\ i\ \in\ \mathcal{N}\} \tag{4.2}$$

We know that if a node is a facility, it produces its own commodity and holds it in its buffer. Since a packet that a node consumes will exist in its buffer one time slot before the required time slot and at the required time slot, we can set $W_i^{i,t} = 1$ for the time slots $t+1$ and t (Equations 4.3 and 4.4).

$$W_i^{i,t} = 1 \qquad \text{if} \qquad d_i^{t+1} = 1 \qquad \forall\ i \in F \qquad \forall\ t \in \mathcal{T} \tag{4.3}$$

$$H_i^{i,t} = 1 \qquad \text{if} \qquad d_i^{t} = 1 \qquad \forall\ i \in F \qquad \forall\ t \in \mathcal{T} \tag{4.4}$$

However, when a node is not a facility, it does not produce any commodities at any time slot (Equation 4.5).

$$W_i^{k,t} = 0 \qquad \forall\ i \in F' \qquad \forall\ k \in \mathcal{N} \qquad \forall\ t \in \mathcal{T} \tag{4.5}$$

Similar to equations 4.3 and 4.4, we can set the values for the buffers of nodes that are not facilities when they have demands in the next time slot. However, unlike the nodes

that are facilities, there may be more than 1 packet in the buffer. Therefore, we set a lower-bound of 1 for both time slots (Equations 4.6 and 4.7).

$$H_i^{i,t} \geq 1 \qquad \text{if} \qquad d_i^{t+1} = 1 \qquad \forall\, i \in F' \qquad \forall\, t \in \mathcal{T} \tag{4.6}$$

$$H_i^{i,t} \geq 1 \qquad \text{if} \qquad d_i^{t} = 1 \qquad \forall\, i \in F' \qquad \forall\, t \in \mathcal{T} \tag{4.7}$$

We can also assume that the commodity of a node that is a facility will not be transmitted or received by any node in the network. Therefore we can set the variable Y accordingly (Equation 4.8). This also means that the commodity of a node that is a facility will never be stored in a node that is not a facility (Equation 4.9).

$$Y_{i,j}^{k,t} = 0 \qquad \forall\, k \in F \qquad \forall\, i,j \in \mathcal{N} \qquad \forall\, t \in \mathcal{T} \tag{4.8}$$

$$H_i^{k,t} = 0 \qquad \forall\, i \in F' \qquad \forall\, k \in F \qquad \forall\, t \in \mathcal{T} \tag{4.9}$$

In this section we discussed how the algorithm combines simulated annealing with a solver. In the next section we will discuss the implementation of the algorithm and later present results from solving various problems.

4.4 Implementation

The routines discussed in the previous section that did not involve the Gurobi Optimizer were implemented using the Python programming language (version 2.7) and the calls to the Gurobi Optimizer (version 4.6) were made using Python bindings provided by Gurobi. The values in the lookup tables (Table 4.1) were also generated by scripts developed in

Python. We also used an open source Python implementation of simulated annealing [79] and modified it to terminate annealing when the energy is equal to 1 which is an optimal solution.

We used a computer with Intel® Core™2 Quad processor and 8 GB of RAM to implement the algorithms and solve the problems which we will present next.

4.5 Results

To test our algorithm, we generated three different types of network topologies. The first topology has nodes distributed randomly over an area while forming a connected network. This is the common topology of resource constrained wireless networks. The second type of network uses a grid (array) of nodes while maintaining connectivity through a limited percentage of all available links. This type of topology is common in testbeds where the nodes are arranged as an array. The third type of network uses contrived topologies with either partially or fully known solutions to test the solving efficiency of the algorithm. We will now go into the details of the problems with the three topologies, each with variations, and discuss the results obtained from solving them.

4.5.1 Random node locations

For testing the solver on a random topology, we generated networks with nodes distributed randomly over a 100×100 area with a communication range to produce a connected network.

While using the same formulation of the problems, we looked at the algorithm's time compared to the the standalone solver. We generated topologies with 5, 8, 10 and 12 nodes. The number of time slots was set to 250 together with an additional number of slots equal to the number of nodes. There were 10 demands per node and the demands were uniformly distributed over the time slots (periodic) for all nodes. These are the same problems that

were solved in the last chapter. We solved the problems using the standalone Gurobi Optimizer and then the proposed algorithm where the same formulation of the problem was used in both. Since the algorithm involves randomness, we used results from 10 runs of the solving the same problem, utilizing random number generators with different seeds.

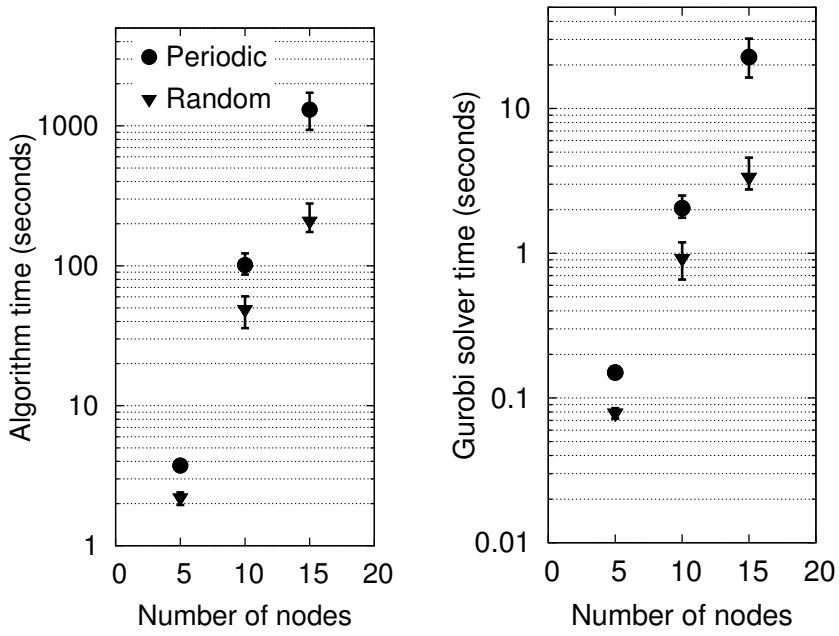
We found that the time for the algorithm varies from 1/10 (5-node network) to 1/150 (10-node network) of the time used by the standalone solver when considering the maximum and minimum times and using the same simple formulation. Additionally, we found that the proposed algorithm found the same objectives as the standalone solver for all instances (the objective was 1 for all problems).

Having observed the speed of the proposed algorithm, we generated more random topologies with variations on the different parameters of the problem. We will henceforth refer to the former set of problems as random long-running problems.

We will first look into a set of problems similar to the aforementioned problems but with fewer number of time slots. We reduced the number of time slots to 30 with the same 10 demands per node. We generated topologies with 5, 10 and 15 nodes with uniformly distributed (periodic) demands. However, we also generated problems with random demands. For example, for a network with 5 nodes, we have a total of 50 demands (10 per node). These demands were distributed randomly over the 150 slots. After assigning the demands, two extra slots were added at the beginning and one at the end for each node (slots at which there are no demands) to ensure feasibility. The buffer sizes per node were set to the size of the network. We will now look at the results from solving these problems (Figure 4.1).

The plot on the left (Figure 4.1a) shows the time taken by the proposed algorithm. Since the proposed algorithm uses the solver, we also show the average time taken by the solver per iteration when being used by the algorithm (averaged over all steps of simulated annealing)

in the plot on the right. Therefore Figure 4.1b shows the average time for a feasibility check and partial solve. Both plots show the time for three network sizes, each with both periodic and random demands. The data points also include error bars showing the maximum and minimum times obtained from 10 solves of the same problem with different random number generator seeds similar to the previous problem instances. We see that the problems with random demands are always solved faster than the problems with periodic demands. This is because random demands usually end up with some nodes having continuous demands which results in a tight timing constraints leading to faster checks on feasibility and lower solving times. As expected, the algorithm time increases with network size. We can also see that there is not much variance as indicated by the short error bars.



(a) Algorithm time **(b)** Gurobi solver time

Figure 4.1: Algorithm time and iteration time to solve problems of random short-running networks.

In Figure 4.2 we can see the average objective values found by the algorithm for the problems. Since these problems were not solved using the standalone solver, we do not

know the optimal values. We can see that the objective values increase with network size and for network size 5 and 10, the objective values are the same for network with both periodic and random demands. For network size 15, we see that the average value is 4 for periodic demands and about 4.75 for random demands. The discrepancy is mostly due to randomness as 8 out of the 10 runs of the algorithm found an objective of 4 and 2 runs found 5. For all the other instances of the problems, all 10 runs of the algorithm found the same objective (as seen by integer average values).

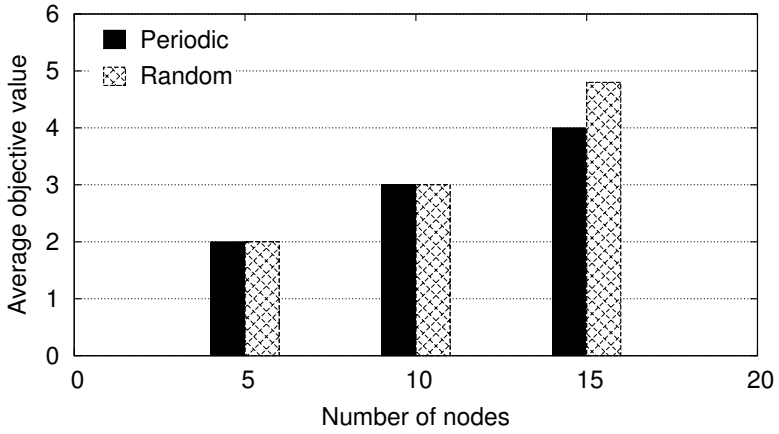
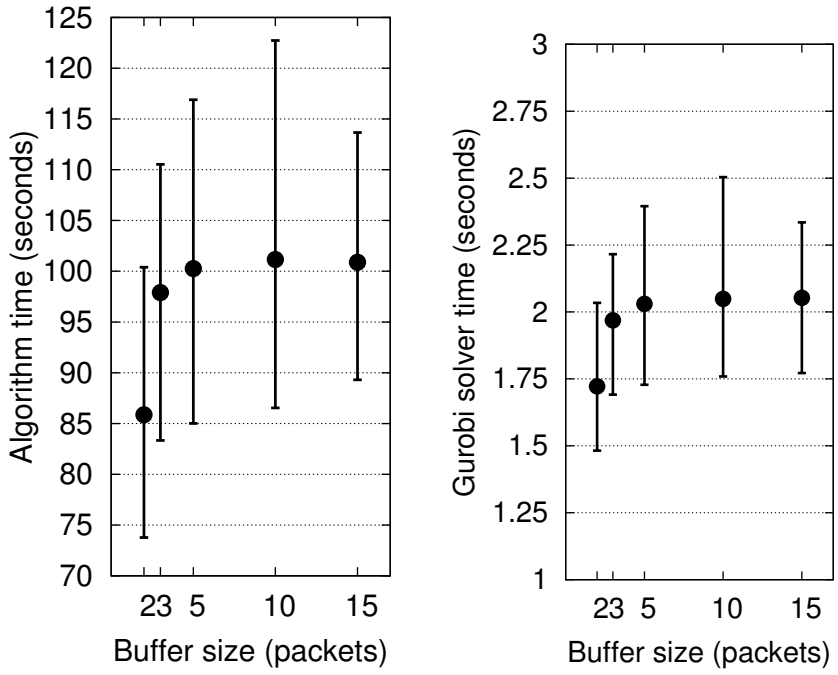


Figure 4.2: The average objective value of problems of random short-running networks.

Next we test the sensitivity of the buffer size variable. We generated problems with 10 nodes, 30 slots (with 3 extra slots as mentioned earlier) and 10 demands per node (periodic). The buffer sizes were set to 2, 3, 5, 10 and 15 and each problem was solved. For buffer size 10, we actually used results from the previous set of problems. In Figure 4.3, similar to Figure 4.1, we can see the times taken by the algorithm and average time of the solver for an iteration. From Figure 4.3a we see that the algorithm time increases as the buffer size increases until 5 and then plateaus. This is because a low buffer size problem is more constrained with lower feasibility check times. As expected it increases with increase in buffer size and, as expected, will not will not impose any overhead after a certain number

which we see to be 5 in this case. This is indeed expected behavior and we also notice the same behavior in the Gurobi solver time as seen in Figure 4.3b. Although the times seem to vary a lot (seen from the error bars), this is due to the change in scale as well as a linear scale. The objective values for all the problems solved were 3 for all 50 runs of the algorithm.



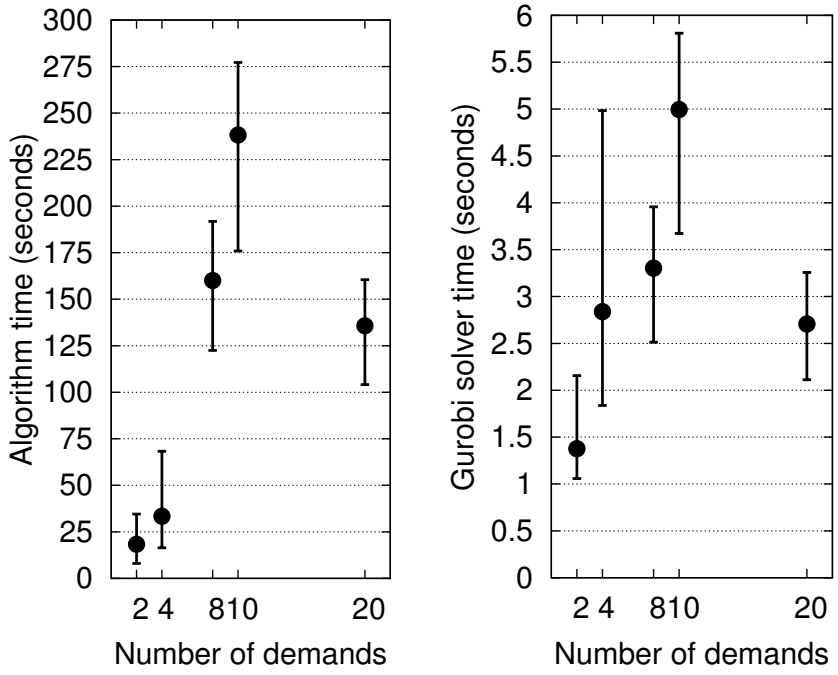
(a) Algorithm time **(b)** Gurobi solver time

Figure 4.3: Time to solve the same random short-running network of nodes with different buffer sizes.

After varying the buffer sizes, we varied the ratio of time slots to number of demands per node. We generated problems with the same parameters used for the previous problem except we fixed the buffer size at 10 and the number of time slots to 40 (+3 extra). The demands were set to 2, 4, 8, 10 and 20 per node (factors of 40 to ensure uniform distribution). The results from solving the problems can be seen in Figure 4.4.

From Figure 4.4a we can see that the time increases with number of demands and then decreases when the demand is 20. The increase in time can be attributed to finding the

routing information for more demands as they increase. However, after a certain number of demands, the problem is too constrained that feasibility checks are faster. In Figure 4.4b we can see there is a variation in the solver times but the scale shows that it is very minimal. For these set of problems, the objective however varied. It was 1 when the demands were 2 and 4, 2 when the demands were 8 and 10 and 4 when the demand was 20. This is not unexpected as higher demands would be requiring more facilities for service.



(a) Algorithm time **(b)** Gurobi solver time

Figure 4.4: Time to solve the same random short-running network of nodes with different number of demands per node.

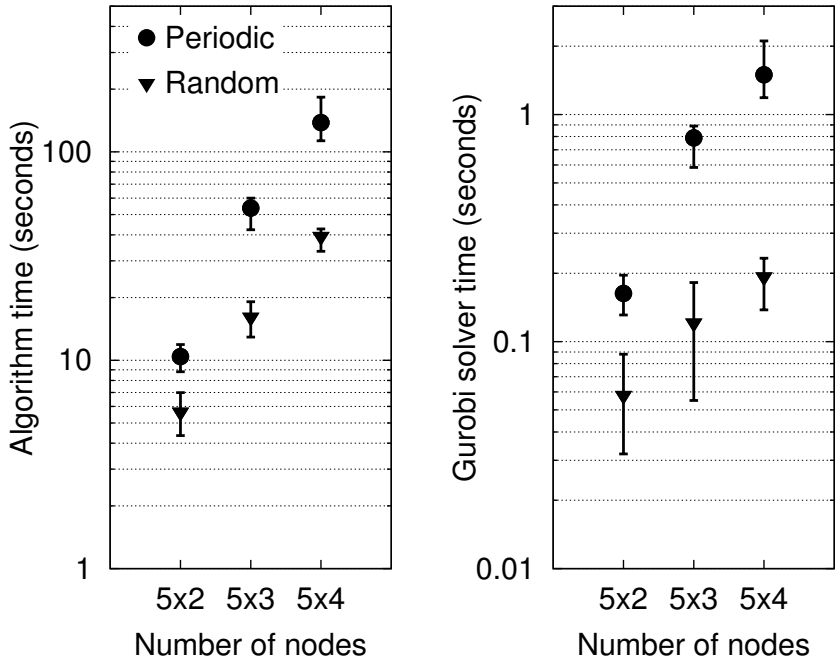
In this section we looked at problems having random topologies with long-running periodic demands and short-running periodic and random demands. We also looked at how the number of nodes in the network, the buffer size and the ratio between the number of time slots to the number of demands relate to the solving time of the proposed algorithm.

4.5.2 Nodes on a grid

A lot of testbed networks have nodes that are located in a grid pattern. In this section, we test the proposed algorithm on topologies that have nodes arranged in a grid (array pattern). We consider topologies with the number of nodes similar in scale to the topologies in the previous section. We generated topologies with nodes in 5×2 (10 nodes), 5×3 and 5×4 grid configurations. We also used only 49% of the links (including all diagonal links on the grid) which were selected randomly while also checking for a connected network. We set the number of time slots to 30 (+3 extra), number of demands per node to 10 and buffer sizes per node equal to the size of the network. Except for the topology, all other parameters are the same as the earlier problems.

We can see the results in Figure 4.5 which shows that, similar to the times in Figure 4.1, the problems with periodic demands need more time to be solved compared to problems with random demands for the same reason. However, we notice that the times taken are generally an order of magnitude less (compared with the same sized network with randomly distributed nodes). The key difference we found between the topologies is the number of links in the network. In a grid topology there are fewer links (almost half or fewer) which produces a more constrained problem to enable faster feasibility checks. Similar to the other topology, we see that the time increases with the increase in number of nodes as expected. The times required by the Gurobi solver (Figure 4.5b) is also similar to times in Figure 4.1b.

Looking at the objective values found by the algorithm in Figure 4.6, we can see that it varies more compared to the earlier problems and especially more when there are random demands. For 10-node networks, we see that for periodic demands it is always 3 whereas for random demands it is between 4 and 5. The 20-node network has a large variance in the objective (9–13) because one of the runs had an objective of 13 but we can see that



(a) Algorithm time **(b)** Gurobi solver time

Figure 4.5: Time to solve problems of networks with nodes on a grid.

the average is lower (around 10). We can also see that the difference between the average objective values for the periodic and random instances increases with number of nodes.

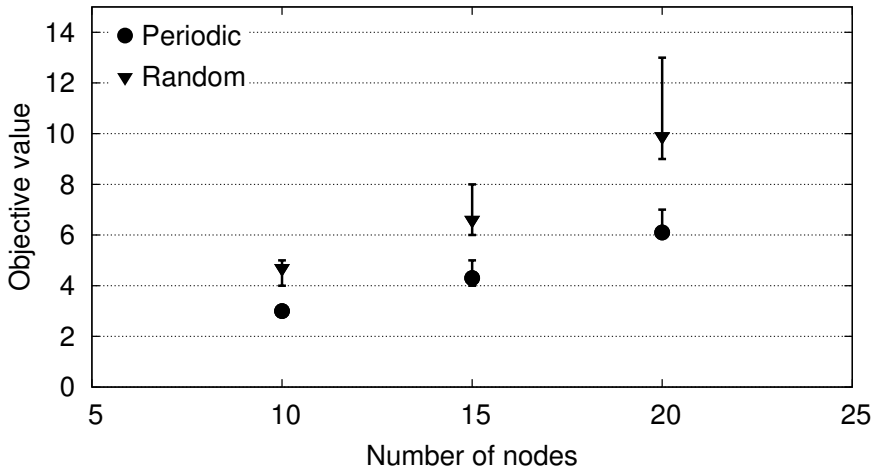


Figure 4.6: The objective value of problems of networks with nodes on a grid.

4.5.3 Networks with unique optimal values

After considering common topologies, we wanted to check how close the objective values found by the proposed algorithm were to the optimal objective values. Since we had the optimal values only for the random long-running problems, we created problems for which we knew the optimal objective values. We designed problems with appropriate topologies and demands such that there were a fixed number of nodes acting as facilities and each is connected to an independent set of nodes which would be acting as its clients (being served from the facility). For example, a network designed to have 3 facilities with four connected clients can be seen in Figure 4.7. Note that the clients are not directly connected to each other. While the nodes that would be facilities are connected to each other to make a connected network, these links are not used.

We coerce the nodes to be facilities and clients by setting the buffer sizes and demands appropriately. We set the buffer size to be 2 per node. At the node acting as a facility, one unit of the buffer will be used for meeting its own demands at every time slot and the other unit will be used to serve each client in succession. We set the number of time slots to 4, one each for serving the four clients while each facility also has a demand at each slot (and as before, three extra slots were added). Let us see how this creates a set of clients being served by one facility. Considering only one facility and its associated set of four clients, the facility injects a packet for the first client in its buffer at time slot 1 which is transmitted in the same time slot to the client, reaching it at time slot 2 which is when it has its demand. The second client has its demand at time slot 3 which would be transmitted at time slot 2 from the facility and so on for the next two clients. Therefore, each facility would be serving its clients at times 2, 3, 4 and 5, one at a time. The facility also has demands at the same four time slots and uses the second unit of the buffer for satisfying its demand. The

second unit of buffer at the clients are never used. Setting the buffer sizes and demands as mentioned will lead to a unique optimal solution including unique values for each variable.

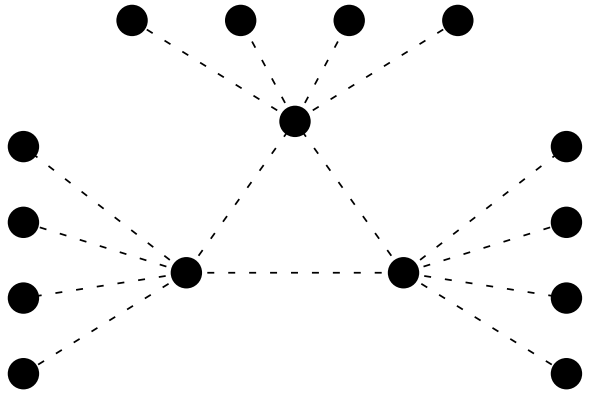


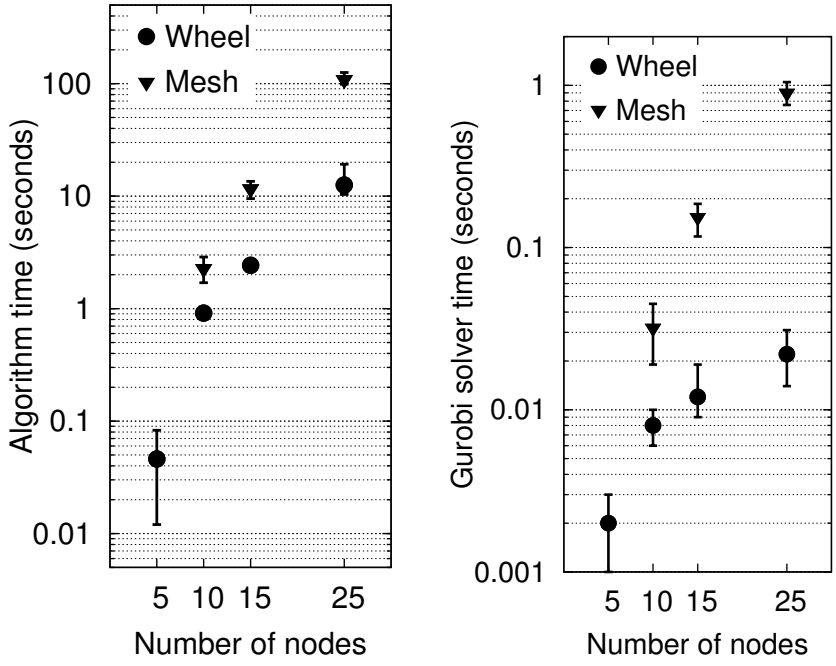
Figure 4.7: Network topology designed for a 3-facility network with a total of 15 nodes (3 facilities as the corners of a triangle at the center, each with 4 clients).

Since this contrived problem has a unique optimal solution as mentioned above, any more facilities will be sub-optimal and any less facilities will make it infeasible. Since there is no other solution, a solver set on the aforementioned problem will find the unique optimal solution (objective) and the unique values of other variables as appropriate. This contrived problem therefore allows us to use the proposed algorithm to check if it can find a unique optimal solution. We created four topologies to have 1, 2, 3 and 5 facilities (with corresponding optimal objective values) wherein each facility in all four networks had four connected clients. We refer to these problems as n-facility problems.

We call the topology shown in Figure 4.7 a *wheel* configuration. However, we also generated a slightly modified configuration of the problem. For this configuration, we connect all facilities to all clients i.e., each client can be served from any one of the facilities. For example, to transform the network in Figure 4.7, the leftmost four nodes connected to one of the facilities in the bottom left corner of the triangle will be connected to the other two facilities (two other corners of the triangle). There will be an additional 24 links, 8 from each facility to the eight clients that they are not connected to. We call this the *mesh*

configuration and while the optimal objective value is still unique (equal to the number of facilities), the solution is not unique anymore at every time instant there are 3 clients that have demands and 3 facilities that can serve these demands but no restriction on which facility can serve which client.

From solving both configurations of the problems, we can expect that the algorithm will be faster with the more restricted *wheel* configuration and Figure 4.8 shows that it is indeed the case. Note that with the 5-node network, there is 1 facility and 4 clients and therefore, both configurations are the same. We see that the time increases with network size as seem in earlier sets of problems (Figure 4.8a). We also see from Figure 4.8b that the time required by the solver increases slower for the *wheel* configuration which can again be attributed to the highly constricted nature of the problem.



(a) Algorithm time **(b)** Gurobi solver time

Figure 4.8: Time to solve problems of n-facility networks.

Moving on to the objective values found by the algorithm, we can see from Figure 4.9 that the optimal objective value is not always found. Note that the data points have been

nudged very slightly in their x-axis to enable comparing both configurations easier. For a 1-facility network (5 nodes), we see that the algorithm always found 1. For a 2-facility network with a total of 10 nodes, we see that the objective varies between 2 and 4, and 2 and 5 for the two configurations. The objectives for the 5-facility network are in the range 8–18 and 8–11 for the two configurations. We see that the solution has 32% of the nodes as facilities as opposed to the 20%. This difference only comes into play as we increase the network size and could be improved by running the algorithm longer.

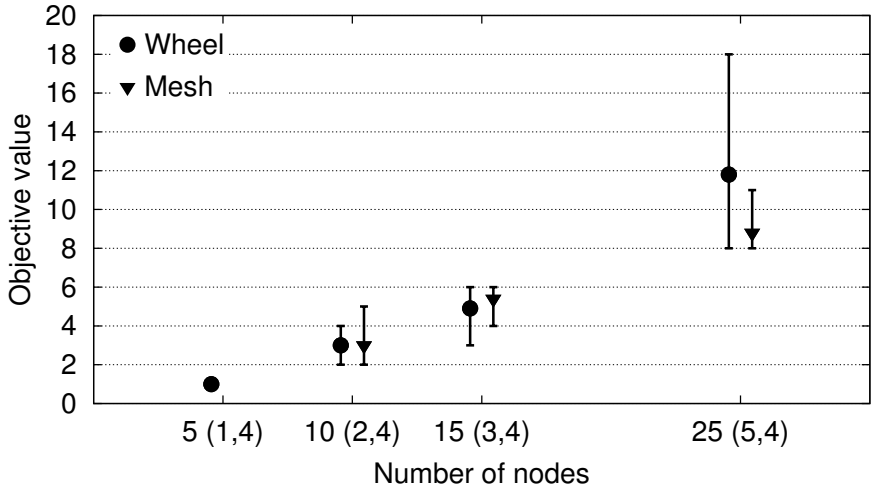


Figure 4.9: The objective value of problems of n-facility networks. The x-axis labels are in this format: Total number of nodes (number of facilities, number of clients per facility).

4.6 Conclusion

In this chapter we presented a hybrid algorithm that uses simulated annealing and a standalone solver (Gurobi Optimizer) to expedite solving the problem of locating base stations for efficiently disseminating control data packets. We detailed the pseudocode for the algorithm which leverages the speed of a feasibility check to expedite the solving, therefore, trading optimality for speed.

We presented the results from using the algorithm to solve problems with different

topologies. We looked at random, grid and contrived topologies and the methods used to generate them. We also looked at the sensitivity of the algorithm to number of nodes, buffer size and ratio of time slots to demands. The proposed hybrid algorithm trades speeds for optimality and was shown to run faster. We emphasize that LP is merely used as a tool in this thesis and we do not claim that our formulation is the most efficient, which also implies that the running time of a more efficient formulation could be very different than the results provided in this document. The same goes for the difference between the speed of the hybrid algorithm versus the solver, which would likely be less prominent with a more efficient formulation. Nevertheless, the proposed algorithm allows us to solve the problem for larger networks and larger demands in a reasonable amount of time.

Chapter 5: Conclusion

Wireless sensor networks with their programmability, low-cost nodes and robust networks have found use in a variety of applications. The nodes are programmed individually and exhibit complex behavior at the network level. However, testing the software application or app that is on the node when it runs as a network is challenging as any test functionality included in the app could interfere with the normal functioning of the app due to the limited resources available on the node. While many solutions have been proposed before, they either do not provide high fidelity or cannot be scaled cost-effectively.

In this thesis we designed and built a hardware-based system that allows controlled testing of sensor network apps as a network. The system pairs each sensor node with another sensor node which we call the control node which can provide the required input for the sensor node as stimuli and also perform control functions. We call our system *Doppel*.

First we presented the architecture of the Doppel system in Chapter 2 which uses a network of identical sensor network which are called control nodes to allow controlled testing of a sensor network running an app. In the control and sensor node pair (the CSP), the control node can provide the stimuli and exercise control over the other node which runs the test app. This architecture combines the benefits of using hardware-based stimuli for fidelity and software-based programmability for ease of control.

Next we presented a prototype implementation of the Doppel architecture for generating analog stimuli. The prototype was evaluated as a networked system, presenting experimental results quantifying the accuracy of the system when reproducing pre-recorded data, generating arbitrary signals and reproducing high-frequency sampled data using very sim-

ple external hardware components. The results show that the architecture is viable as an artificial stimuli system with high accuracy. The control subsystem was also showcased for emulating node failures and accurate power measurement of individual nodes in the network.

In the Doppel system, efficiently managing the dissemination of packets through the control network requires strategic placement of base stations. By taking into account the network topology, the times each control node requires data, buffer constraints, half-duplex constraints, etc. we can find the right locations of base stations. In Chapter 3, we discuss the problem and formulated it as a facility location problem.

In Chapter 4 we looked at some of the solutions available for facility location problems and how our formulation could require a long time for solving. We then proposed a hybridized algorithm that combines a metaheuristic, simulated annealing, with an exact algorithm, Gurobi Optimizer, a standalone solver. We presented the results from using the algorithm to solve problems with different topologies and also looked at the sensitivity of the algorithm to the number of nodes, buffer sizes and ratio of time slots to demands.

5.1 Summary of Contributions:

The major contributions of this research include:

- Designing the architecture of the Doppel system which allows automated testing of sensor networks apps and consists of two major subsystems, one to provide simulated values and another to exercise control over the sensor nodes. The system allows creating a controlled environment for the duration of the test.

- Implementing the Doppel architecture using real sensor node hardware. The MICAz sensor node platform and TinyOS software toolchain was used for developing software

for both subsystems.

- Designing tests and developing software to evaluate the performance of each subsystem in the Doppel system. The tests show that both systems are viable and offer high accuracy.

- Identifying that the control network can efficiently manage data dissemination with base stations at strategic locations and formulating an optimization problem that can be solved to find the locations for a given dissemination problem.

- Developing a hybrid algorithm that integrates an exact algorithm into a metaheuristic to solve the NP-hard optimization problem and evaluating it with different network topologies and sensitivity to parameters.

5.2 Extending the System

While the Doppel system was designed for testing sensor network apps that require input from the environment to be tested, the architecture can be extended for increased utility and other applications.

One of the basic improvements to the system would include interfacing DACs to all or part of the nodes in the system to reproduce high precision and high frequency analog signals. This is useful especially for apps that use a large amount of high precision streamed sensor data. Since each sensor node is paired with a control node, by using appropriate interface and wired connections, we can easily reprogram individual sensor nodes over the air through the control network. The control network would allow reprogramming either individual nodes or the entire sensor network to enable debugging network level problems much faster than traditional means. This allows for a system similar to Deployment Support Network [35]. Currently, the Doppel system only allows control over the reset and on/off

functionality of a sensor node. With appropriate external hardware, we can control the voltage and current the sensor node receives when it is powered by the control node. Since a lot of sensor network apps self-measure power and change behavior accordingly, this allows testing network behavior of the app by varying the power supplied to individual nodes in the network. A setup for a single node was demonstrated in EmPro [34].

Similar to the sensors, the RF transceiver on the sensor node also uses digital signals to communicate with the microcontroller and therefore the app. Therefore, with the right hardware setup that can isolate and tap into the interface between the RF chip and the microcontroller, the system will be able to recreate node communications. While this would be of lower fidelity, it will be more cost-effective compared to outfitting each sensor node with an RF signal/interference generator. Such a system designed with network connectivity can also allow bridging test beds that are geographically isolated and enable testing on very large scales.

The optimization model and algorithm for solving the problem can also be used for some networks that have actuators in the nodes such as Wireless Sensor and Actor Networks [80]. This will allow time critical data to be delivered to the node initiating the actuation. The model can be extended to use continuous time, include wireless interference from other nodes in the network, use different buffer capacities for each node in the case of heterogeneous nodes, consider node mobility, etc. The model can also be relaxed of some constraints and used in other contexts. Relaxing buffer and half-duplex constraints will result in a model useful for data dissemination for wired hosts in a network.

There exist numerous embedded devices that are deployed over a large area and work as a network either through a wired or wireless network. Developing a way for the Doppel system's control node to interface with standardized or non-standardized test ports on these

devices would allow a major leap in its use. The Doppel system can then be used for testing any of these systems that require distributed but controlled testing conditions and could be useful in testing home automation devices to smart grids.

Bibliography

[1] Aleksandar Milenković, Chris Otto, and Emil Jovanov. Wireless sensor networks for personal health monitoring: Issues and an implementation. *Comput. Commun.*, 29:2521–2533, August 2006.

[2] Fei Hu, Meng Jiang, Laura Celentano, and Yang Xiao. Robust medical ad hoc sensor networks (MASN) with wavelet-based ECG data mining. *Ad Hoc Networks*, 6(7):986 – 1012, 2008.

[3] K. Lorincz, D.J. Malan, T.R.F. Fulford-Jones, A. Nawoj, A. Clavel, V. Shnayder, G. Mainland, M. Welsh, and S. Moulton. Sensor networks for emergency response: challenges and opportunities. *Pervasive Computing, IEEE*, 3(4):16 – 23, oct.-dec. 2004.

[4] Jeongyeup Paek, K. Chintalapudi, R. Govindan, J. Caffrey, and S. Masri. A wireless sensor network for structural health monitoring: Performance and experience. In *Embedded Networked Sensors, 2005. EmNetS-II. The Second IEEE Workshop on*, pages 1 – 10, may 2005.

[5] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, WSNA '02, pages 88–97, New York, NY, USA, 2002. ACM.

[6] Jane K. Hart and Kirk Martinez. Environmental sensor networks: A revolution in the earth system science? *Earth-Science Reviews*, 78(34):177 – 191, 2006.

[7] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393 – 422, 2002.

[8] National Instruments Wireless Sensor Networks. `http://www.ni.com/wsn`.

[9] Spinwave Systems. `http://www.spinwavesystems.com`.

[10] Deborah Estrin, Ramesh Govindan, John Heidemann, and Satish Kumar. Next century challenges: scalable coordination in sensor networks. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, MobiCom '99, pages 263–270, New York, NY, USA, 1999. ACM.

[11] T.H. Meng and V. Rodoplu. Distributed network protocols for wireless communication. In *Circuits and Systems, 1998. ISCAS '98. Proceedings of the 1998 IEEE International Symposium on*, volume 4, pages 600 –603 vol.4, may-3 jun 1998.

[12] David E. Culler. Toward the sensor network macroscope. In *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, MobiHoc '05, pages 1–1, New York, NY, USA, 2005. ACM.

[13] Crossbow Technology, Inc., MICAz 2.4 GHz Wireless Module. `http://www.xbow.com`.

[14] Crossbow Technology, Inc., MTS/MDA Sensor Board Users Manual. `http://www.xbow.com`.

[15] D. Puccinelli and M. Haenggi. Wireless sensor networks: applications and challenges of ubiquitous sensing. *Circuits and Systems Magazine, IEEE*, 5(3):19 – 31, 2005.

[16] M. Bhardwaj, T. Garnett, and A.P. Chandrakasan. Upper bounds on the lifetime of sensor networks. In *Communications, 2001. ICC 2001. IEEE International Conference on*, volume 3, pages 785 –790 vol.3, 2001.

[17] W.R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on*, page 10 pp. vol.2, jan. 2000.

[18] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker. Complex Behavior at Scale: An Experimental Study of Low-Power Wireless Sensor Networks. Technical report, UCLA Computer Science Department, 2002.

[19] J.L. Hill and D.E. Culler. Mica: a wireless platform for deeply embedded networks. *Micro, IEEE*, 22(6):12 – 24, nov/dec 2002.

[20] Xiang Zeng, Rajive Bagrodia, and Mario Gerla. Glomosim: a library for parallel simulation of large-scale wireless networks. In *Proceedings of the twelfth workshop on Parallel and distributed simulation*, PADS '98, pages 154–161, Washington, DC, USA, 1998. IEEE Computer Society.

[21] OMNeT++ Network Simulation Framework. `http://www.omnetpp.org`.

[22] L.F. Perrone and D.M. Nicol. A scalable simulator for tinyos applications. In *Simulation Conference, 2002. Proceedings of the Winter*, volume 1, pages 679 – 687 vol.1, dec. 2002.

[23] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. TOSSIM: accurate and scalable simulation of entire TinyOS applications. In *SenSys '03*, pages 126–137, New York, NY, USA, 2003. ACM.

[24] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level sensor network simulation with cooja. In *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, pages 641 –648, nov. 2006.

[25] J. Polley, D. Blazakis, J. McGee, D. Rusk, and J.S. Baras. Atemu: a fine-grained sensor network simulator. In *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on*, pages 145 – 152, oct. 2004.

[26] Lewis Girod, Thanos Stathopoulos, Nithya Ramanathan, Jeremy Elson, Deborah Estrin, Eric Osterweil, and Tom Schoellhammer. A system for simulation, emulation, and deployment of heterogeneous sensor networks. In *SenSys '04*, pages 201–213, New York, NY, USA, 2004. ACM.

[27] D. Watson and M. Nesterenko. MULE: Hybrid Simulator for Testing and Debugging Wireless Sensor Networks. Technical report, Department of Computer Science, Kent State University, 2004.

[28] W. Li, X. Zhang, W. Tan, and X. Zhou. H-tossim: Extending tossim with physical nodes. *Wireless Sensor Nework*, 1(4):324 – 333, 2009.

[29] L. Luo, T. He, G. Zhou, L. Gu, T. F. Abdelzaher, and J. A. Stankovic. Achieving repeatability of asynchronous events in wireless sensor networks with envirolog. In *INFOCOM 2006*, pages 1–14, April 2006.

[30] Thomas Clouser, Richie Thomas, and Mikhail Nesterenko. Emuli: Emulated Stimuli for Wireless Sensor Network Experimentation. Technical report, Kent State University, 2007.

[31] Geoffrey Werner-Allen, Patrick Swieskowski, and Matt Welsh. MoteLab: a wireless sensor network testbed. In *IPSN '05*, page 68, Piscataway, NJ, USA, 2005. IEEE Press.

[32] M. Ott, I. Seskar, R. Siraccusa, and M. Singh. Orbit testbed software architecture: supporting experiments as a service. In *Testbeds and Research Infrastructures for the Development of Networks and Communities, 2005. Tridentcom 2005. First International Conference on*, pages 136 – 145, feb. 2005.

[33] Anish Arora, Emre Ertin, Rajiv Ramnath, Mikhail Nesterenko, and William Leal. Kansei: A high-fidelity sensing testbed. *IEEE Internet Computing*, 10(2):35–47, 2006.

[34] Chulsung Park and P.H. Chou. EmPro: an Environment/Energy Emulation and Profiling Platform for Wireless Sensor Networks. *SECON '06*, 1:158–167, Sept. 2006.

[35] Matthias Dyer, Jan Beutel, Thomas Kalt, Patrice Oehen, Lothar Thiele, Kevin Martin, and Philipp Blum. Deployment Support Network - A Toolkit for the Development of WSNs. In *EWSN 2007*, pages 195–211. Springer Berlin / Heidelberg, 2007.

[36] Jan Beutel, Matthias Dyer, Martin Hinz, Lennart Meier, and Matthias Ringwald. Next-generation prototyping of sensor networks. In *SenSys '04*, pages 291–292, New York, NY, USA, 2004. ACM.

[37] National Semiconductor, DAC0808 Digital-to-Analog converter datasheet. `http://www.national.com/mpf/DA/DAC0808.html`.

[38] Microchip Technology Inc., MCP4921 Digital-to-Analog converter datasheet. `http://ww1.microchip.com/downloads/en/DeviceDoc/21897a.pdf`.

[39] Atmel Corporation, ATmega128L microcontroller datasheet. `http://www.atmel.com/atmel/acrobat/doc2467.pdf`.

[40] Xiaofan Jiang, Prabal Dutta, David Culler, and Ion Stoica. Micro power meter for energy monitoring of wireless sensor networks at scale. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 186–195, New York, NY, USA, 2007. ACM.

[41] C. Park, J. Liu, and P.H. Chou. B#: a battery emulator and power-profiling instrument. *Design & Test of Computers, IEEE*, 22(2):150–159, March-April 2005.

[42] Chulsung Park, K. Lahiri, and A. Raghunathan. Battery discharge characteristics of wireless sensor nodes: an experimental analysis. In *Sensor and Ad Hoc Communications and Networks, 2005. IEEE SECON 2005. 2005 Second Annual IEEE Communications Society Conference on*, pages 430–440, Sept., 2005.

[43] Jonathan W. Hui and David Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *SenSys '04*, pages 81–94, New York, NY, USA, 2004. ACM.

[44] Kun Sun, Peng Ning, and Cliff Wang. Tinysersync: secure and resilient time syn-

chronization in wireless sensor networks. In *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, pages 264–277, New York, NY, USA, 2006. ACM.

[45] Miklós Maróti, Branislav Kusy, Gyula Simon, and Ákos Lédeczi. The flooding time synchronization protocol. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 39–49, New York, NY, USA, 2004. ACM.

[46] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. *SIGPLAN Not.*, 35(11):93–104, 2000.

[47] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesc language: A holistic approach to networked embedded systems. In *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, PLDI '03, pages 1–11, New York, NY, USA, 2003. ACM.

[48] Zvi Drezner and Horst W. Hamacheri. *Facility location: applications and theory.* Springer, New York, 2002.

[49] Margaret L. Brandeau and Samuel S. Chiu. An overview of representative problems in location research. *Management Science*, 35(6):pp. 645–674, 1989.

[50] M.T. Melo, S. Nickel, and F. Saldanha da Gama. Facility location and supply chain management a review. *European Journal of Operational Research*, 196(2):401 – 412, 2009.

[51] M. S. Daskin. *Network and Discrete Location: Models, Algorithms, and Applications.* Wiley, New York, 1995.

[52] Gven Sahin and Haldun Sural. A review of hierarchical facility location models. *Computers & Operations Research*, 34(8):2310 – 2331, 2007.

[53] R. Dekker, L.N. van Wassenhove, and K. Indurfurth. *Reverse Logistics.* Springer, New York, 2004.

[54] B. Lebreton. *Strategic Closed-Loop Supply Chain Management.* Springer, New York, 2007.

[55] Jay Aronson. A survey of dynamic network flows. *Annals of Operations Research*,

20:1–66, 1989.

[56] Yale T. Herer and Michal Tzur. The dynamic transshipment problem. *Naval Research Logistics (NRL)*, 48(5):386–408, 2001.

[57] Lili Qiu, V.N. Padmanabhan, and G.M. Voelker. On the placement of web server replicas. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1587 –1596 vol.3, 2001.

[58] Daniela Ambrosino and Maria Grazia Scutell. Distribution network design: New problems and related models. *European Journal of Operational Research*, 165(3):610 – 624, 2005.

[59] Denis Krivitski, Assaf Schuster, and Ran Wolff. A local facility location algorithm for large-scale distributed systems. *Journal of Grid Computing*, 5:361–378, 2007.

[60] G. Wittenburg and J. Schiller. A survey of current directions in service placement in mobile ad-hoc networks. In *Pervasive Computing and Communications, 2008. PerCom 2008. Sixth Annual IEEE International Conference on*, pages 548 –553, march 2008.

[61] Furuta Takehiro, Sasaki Mihiro, Ishizaki Fumio, Suzuki Atsuo, and Miyazawa Hajime. A new clustering model of wireless sensor networks using facility location theory. *Journal of the Operations Research Society of Japan*, 52(4):366–376, 2009-12.

[62] Tian He, J.A. Stankovic, Chenyang Lu, and T. Abdelzaher. Speed: a stateless protocol for real-time communication in sensor networks. In *Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on*, pages 46 – 55, may 2003.

[63] Chenyang Lu, B.M. Blum, T.F. Abdelzaher, J.A. Stankovic, and Tian He. Rap: a real-time communication architecture for large-scale wireless sensor networks. In *Real-Time and Embedded Technology and Applications Symposium, 2002. Proceedings. Eighth IEEE*, pages 55 – 66, 2002.

[64] H. Li, P. Shenoy, and K. Ramamritham. Scheduling messages with deadlines in multihop real-time sensor networks. In *Real Time and Embedded Technology and Applications Symposium, 2005. RTAS 2005. 11th IEEE*, pages 415 – 425, march 2005.

[65] The website for Gurobi Optimization. `http://www.gurobi.com`.

[66] M.T. Melo, S. Nickel, and F. Saldanha da Gama. Dynamic multi-commodity capaci-

tated facility location: a mathematical modeling framework for strategic supply chain planning. *Computers & Operations Research*, 33(1):181 – 208, 2006.

[67] Cem Canel, Basheer M. Khumawala, Japhett Law, and Anthony Loh. An algorithm for the capacitated, multi-commodity multi-period facility location problem. *Computers & Operations Research*, 28(5):411 – 427, 2001.

[68] Hasan Pirkul and Vaidyanathan Jayaraman. A multi-commodity, multi-plant, capacitated facility location problem: formulation and efficient heuristic solution. *Computers & Operations Research*, 25(10):869 – 878, 1998.

[69] Zoltan Vincze, Rolland Vida, and Attila Vidacs. Deploying multiple sinks in multi-hop wireless sensor networks. In *Pervasive Services, IEEE International Conference on*, pages 55 –63, july 2007.

[70] Zoltán Vincze, Dorottya Vass, Rolland Vida, Attila Vidács, and András Telcs. Adaptive sink mobility in event-driven multi-hop wireless sensor networks. In *Proceedings of the first international conference on Integrated internet ad hoc and sensor networks*, InterSense '06, New York, NY, USA, 2006. ACM.

[71] Jonathan Berry, William E. Hart, Cynthia A. Phillips, James G. Uber, and Jean-Paul Watson. Sensor placement in municipal water networks with temporal integer programming models. *Journal of Water Resources Planning and Management*, 132(4):218–224, 2006.

[72] Nenad Mladenovi, Jack Brimberg, Pierre Hansen, and Jos A. Moreno-Prez. The p-median problem: A survey of metaheuristic approaches. *European Journal of Operational Research*, 179(3):927 – 939, 2007.

[73] Ren Peng, Xu Rui-hua, and Qin Jin. Bi-level simulated annealing algorithm for facility location problem. In *Information Management, Innovation Management and Industrial Engineering, 2008. ICIII '08. International Conference on*, volume 3, pages 17 –22, dec. 2008.

[74] Jakob Puchinger and Gnther Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In Jos Mira and Jos lvarez, editors, *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, volume 3562 of *Lecture Notes in Computer Science*, pages 113–124. Springer Berlin / Heidelberg, 2005.

[75] L. Jourdan, M. Basseur, and E.-G. Talbi. Hybridizing exact methods and metaheuristics: A taxonomy. *European Journal of Operational Research*, 199(3):620 – 629, 2009.

[76] R. Battiti and G. Tecchiolli. Simulated annealing and tabu search in the long run: A comparison on qap tasks. *Computers & Mathematics with Applications*, 28(6):1 – 8, 1994.

[77] V. J. Rayward-Smith, I. H. Osman, C. R. Reeves, and G. D. Smith. *Modern Heuristic Search Methods*. Wiley, 1996.

[78] Steven Skiena. *The Algorithm Design Manual*. Springer, 2008.

[79] Richard Wagner. Python Simulated Annealing Module. `http://www-personal.umich.edu/~wagnerr/PythonAnneal.html`.

[80] Ian F. Akyildiz and Ismail H. Kasimoglu. Wireless sensor and actor networks: research challenges. *Ad Hoc Networks*, 2(4):351 – 367, 2004.

Appendix A: Appendix for Chapter 2

The code to generate PWM signals from TinyOS is an important part of the stimuli system and we list the TinyOS source code of the `PWMControlC` component here.

The `PWMControlC` component is a combination of three files, each with their unique function. The `PWMControlC.nc` (Listing A.1) file is the main file which can be used as a component in a TinyOS application (app). However, the file itself does not contain most of the functionality. The functionality is split into two files, a module and an interface. The module file, `PWMControlP.nc` (Listing A.3), is the file which has all the functionality. The interface file, `PWMControl.nc` (Listing A.2) only contains the method signatures that can be used to interact with the `PWMControlP.nc` file. The component file, `PWMControlC.nc` has the configuration that combines the former two files to create a module.

We will first look at the component file `PWMControlC.nc`. Line 2 shows that it provides only one interface (`PWMControl`) to interact with it. Lines 6–8 show that four components are required and the module `PWMControlP` is one of them. Lines 10–16 show how the various modules are wired. An important piece of code is Line 16 which is used to disable the sleeping of the microcontroller. Without this, the PWM will not be activated. Note that we have two PWM generators that are provided, PWM1A and PWM1B. However, we will only be displaying the code and functionality of PWM1A as the other is similar. Next we look at the source code of the interface.

Listing A.1: PWMControlC.nc

```
1  configuration PWMControlC {
2    provides interface PWMControl;
```

```
3 }
4
5 implementation {
6   components PWMControlP, MainC;
7   components HplAtm128GeneralIOC, HplAtm128Timer1C;
8   components McuSleepC;
9
10  PWMControl = PWMControlP;
11  PWMControlP.Boot         -> MainC.Boot;
12  PWMControlP.PortPWM1A -> HplAtm128GeneralIOC.PortB5;
13  PWMControlP.PortPWM1B -> HplAtm128GeneralIOC.PortB6;
14  PWMControlP.Compare1A -> HplAtm128Timer1C.Compare[0];
15  PWMControlP.Compare1B -> HplAtm128Timer1C.Compare[1];
16  McuSleepC.McuPowerOverride -> PWMControlP.McuPowerOverride;
17 }
```

The code for the PWM control interface is given in Listing A.2. We see that it provides six methods, three for each PWM channel (PWM1A & PWM1B). The `setPWM1ADuty` sets the duty cycle for PWM. Although it has 16 bits of precision, only 10 bits (LSB) are used as the PWM only has 10-bit resolution. The other two functions are self-explanatory. Next we look at the `PWMControlP` module which contains all the functionality.

Listing A.2: PWMControl.nc

```
1 interface PWMControl {
2   async command void setPWM1ADuty(uint16_t val);
3   async command void PWM1AOn();
4   async command void PWM1AOff();
```

```
5     async command void setPWM1BDuty(uint16_t val);
6     async command void PWM1BOn();
7     async command void PWM1BOff();
8   }
```

The lines 1–11 provide the required interface to the `PWMControlC` component and the interface to disable the microcontroller from sleeping. Lines 14–21 set the appropriate registers of the ATmega128L MCU to generate the PWM and lines 22–24 set the frequency. The next two lines make the PWM1A pin an output pin. Lines 29–35 set the duty cycle if it is non-zero. Setting the MCU sleep state is done in lines 38–40.

Listing A.3: PWMControlP.nc

```
1  module PWMControlP {
2    provides {
3      interface PWMControl;
4      interface McuPowerOverride;
5    }
6    uses {
7      interface Boot;
8      interface GeneralIO as PortPWM1A;
9      interface HplAtm128Compare<uint16_t> as Compare1A;
10   }
11 }
12 implementation {
13   event void Boot.booted() {
14     SET_BIT(TCCR1A, COM1A1);
15     CLR_BIT(TCCR1A, COM1A0);
```

```
16      SET_BIT(TCCR1A, COM1B1);
17      CLR_BIT(TCCR1A, COM1B0);
18      CLR_BIT(TCCR1B, WGM13);
19      SET_BIT(TCCR1B, WGM12);
20      SET_BIT(TCCR1A, WGM11);
21      SET_BIT(TCCR1A, WGM10);
22      CLR_BIT(TCCR1B, CS12);
23      CLR_BIT(TCCR1B, CS11);
24      SET_BIT(TCCR1B, CS10);
25      call PortPWM1A.clr();
26      call PortPWM1A.makeOutput();
27    }
28    async command void PWMControl.setPWM1ADuty(uint16_t val) {
29      if((val & 0x3FF) == 0x000)
30      {
31        CLR_BIT(TCCR1A, COM1A1);
32        call PortPWM1A.clr();
33      } else {
34        SET_BIT(TCCR1A, COM1A1);
35        call Compare1A.set(val & 0x3FF);
36      }
37    }
38    async command mcu_power_t McuPowerOverride.lowestState() {
39      return ATM128_POWER_IDLE;
40    }
41    async event void Compare1A.fired() { }
42  }
```

Vita

Name: Anbu Elancheziyan

Education:

- Ph.D., Computer Engineering, Drexel University, Philadelphia, 2006–present.
- M.S., Computer Engineering, Drexel University, Philadelphia, 2003–2006.
- B.E., Electrical and Electronics Engineering, University of Madras, India, 1999–2003.

Selected Publications:

- A. Elancheziyan, J. C. de Oliveira, and S. Weber, *"A New System for Controlled Testing of Sensor Network Applications: Architecture, Prototype and Experimental Evaluation"*, Volume 10, Issue 6, August 2012, Pages 1101–1114, Ad hoc Networks.
- Z. Zhao, A. Elancheziyan, and J. C. de Oliveira, *"Sleeping Policy Cost Analysis for Sensor Nodes Collecting Heterogeneous Data"*, in the Proceedings of the Conference on Information Sciences and Systems (CISS 09), Princeton, NJ, Mar 18–20, 2009.
- A. Elancheziyan, J. C. de Oliveira, F. S. Cohen, and F. Reisman, *"Integrating Sensor Networks in Undergraduate Curriculum: A Marriage between Theory and Practice"*, in the Proceedings of the ASEE Annual Conference 2008, Pittsburgh, PA, Jun 22–25, 2008.