AFRo:
An Adaptive Framework for WSN Routing

Takatosi Kanazawa

Faculty of Policy Management

Keio University

5322 Endo, Fujisawa Kanagawa 252-8520 JAPAN

Submitted in partial fulfillment of the requirements
for the degree of Bachelor

Advisors:

Professor Hideyuki Tokuda
Professor Jun Murai
Professor Osamu Nakamura
Associate Professor Hiroyuki Kusumoto
Associate Professor Kazu Takashio
Associate Professor Jin Mitsugi
Associate Professor Keisuke Uehara
Instructor Noriyuki Shigechika
Instructor Rodney D. Van Meter III
Instructor Jin Nakazawa

Abstract of Bachelor's Thesis

AFRo: An Adaptive Framework for WSN Routing

Wireless Sensor Networks (WSN) have gained widespread popularity during the past decade. Because changing the application software of a WSN node requires severe overhead of both time and man power, a reprogrammable application framework model has gained attention among WSN programming. However, traditional reprogrammable application frameworks does not enable dynamic changing of routing protocols among a WSN. WSN routing protocols are heavily application specific; i.e., a single routing protocol would not suffice every WSN application requirements. Hence, a reprogrammable application framework is unable to satisfy multiple application requirements.

In this thesis, we propose an Adaptive Framework for WSN Routing (AFRo), to solve by problems mentioned above. We have deployed AFRo as a TinyOS component, and evaluated it's performance on a TinyOS simulator.

AFRo enables dynamic selection of WSN routing protocols based on an application request, and enhances current WSNs to adapt to various application requests.

Takatosi Kanazawa

Faculty of Policy Management

Keio University

卒業論文要旨 2008年度（平成20年度）
AFRo:無線センサネットワークにおける動的経路制御切り替え機構

近年，無線センサネットワークに関する研究が盛んである．この中で，無線センサネットワークを用いたアプリケーション構築におけるセンサノードの更新，及びメンテナンスに要するコストの高さが大きな問題となっている．このような課題に対する解決として，ノードのアプリケーションを動的に変更可能なセンサノードの動的更新機構が注目されている．

一方，無線センサネットワークにおいて，ルーティングプロトコルはセンサネットワークのアプリケーションの多様な要求と密接に関係する．しかし，既存のセンサノードの動的更新機構は複数のアプリケーションをネットワーク上で動作させることによるネットワークの拡張性を目的としていたため，無線センサネットワークのアプリケーション以外の要素を変更することは想定されていない．センサネットワークのルーティングプロトコルを変更することは難しく，複数のアプリケーションに対して常に一つのルーティングプロトコルを使用する場合が多い．よって，センサネットワークにおける複数のアプリケーションの効率的な運用において，アプリケーション毎に最適なルーティングプロトコルを提供することが必要である．

本論文では，センサノードの動的更新機構上でのアプリケーションの変更に応じて最適なルーティングプロトコルを提供する機構，AFRo（An Adaptive Framework for WSN Routing）を提供する．AFRoを用いることにより，センサノードのアプリケーションの更新に際して最適なルーティングプロトコルが選択され，アプリケーション性能の最適化，またネットワーク資源の効率的な利用が達成される．

慶應義塾大学 総合政策学部

金澤 貴俊

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This chapter is an introduction for our thesis. First we describe our motivation for our research. We then propose our challenge and research goals regarding this subject. Finally, we describe structure of the thesis.

1.1 Motivation

Technological evolution has led to the creation of micro computer nodes capable of sensing environmental information. Such nodes could autonomously communicate with each other via wireless communication in order to form a ubiquitous network capable of sensing a large physical region. This kind of network is often referred to as a Wireless Sensor Network (WSN). Figure 1.1 illustrates a typical WSN deployed in a forest. The sensor nodes are connected by wireless communication as shown with the dotted lines.

WSNs requires no physical human interaction among sensing an environment. Therefore, although numerous applications of a WSN could be considered, WSNs are especially anticipated for use in areas where human interference is difficult, if not impossible. For example, Werner-Allen[21] has deployed a WSN among the surface of an active volcano, Volcan Reventador in northern Ecuador, where explosive events that ejected incandescent blocks, gas, and ashes has occurred on a daily basis. Mainwaring[15] has deployed a habitat monitoring WSN among Great Duck Island, Maine, placing nodes among the nests of the wild life creatures where direct human observance would cause severe precaution among the animals.

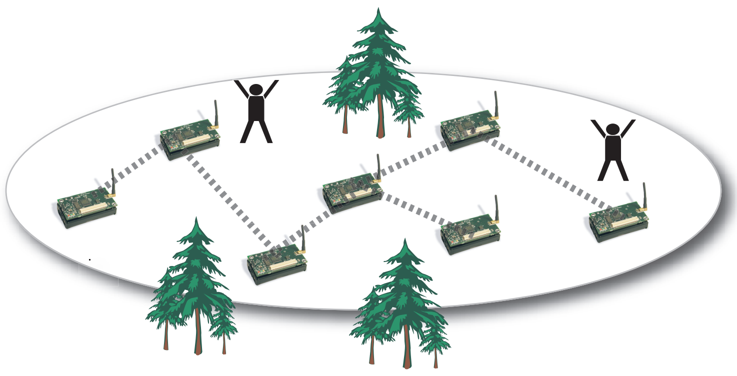


Figure 1.1: A Wireless Sensor Network

In order for theses nodes to form a WSN, each node must be programmed to fulfill the network's objective. For example, nodes would require a network stack to communicate with other nodes in a cooperative manner. A node deployed for precipitation monitoring would be required to be programmed with a humidity sensing application, and so on. It is tedious and stressful for WSN application programmers to require complete coverage, and pre-program the nodes among a WSN before deployment.

Also, note that a sensor network must deploy numerous sensor nodes in order to sense an area of interest; often hundreds and thousands of nodes are deployed in order for a precise and effective sensing. That combined with the above characteristic of a WSN would result in a high overhead for application maintenance or changing.

To overcome these constraint, an application layer component, namely a reprogrammable application framework model has been introduced for WSN application programming. A reprogrammable application framework enables dynamic reprogrammability of WSN nodes over the network, without any direct human interaction. Such frameworks include MATE[12], a TinyOS[7] component which provides custom made virtual machine like environment for applications, and ContikiOS[20], which enables runtime dynamic loading and unloading of application code. Using the reprogrammable application framework model has lead application programmers to easily maintain, and switch application code on demand.

However, although traditional reprogrammable application models offer dynamic functionality of WSN applications, dynamic reprogrammability of components exterior to the application layer, especially the routing functionalities of a WSN node has not been achieved. This is crucial for a WSN application, because a WSN application's performance is heavily reliant on the underlying data transfer methods. Usage of a non optimal routing protocols also lead to a faster consumption of the networks energy, hence leading to bad network utilization.

In this thesis, we identify and solve the above problem by implementing an Adaptive Framework for WSN Routing (AFRo).

1.2 Challenges and Research Goal

Numerous researches regarding WSNs have been done in the past decade. Noticeable among them are a reprogrammable application framework model of WSN application programming, which enables WSN application programmers to propagate their own code into the WSN over the network without direct interaction with the physical node.

On the other hand, WSN applications are heavily reliant on the network's data transfer method. For example, a precipitation tracking application would require running for a long time period, hence requiring an energy-efficient routing protocol to save data transfer energy consumption. On the other hand, a wild-life tracking application would require a reliable routing scheme to keep track of the rapidly moving animals. It is optimal for each application to select the network routing mechanism based on the application's requirement in order to achieve the best application performance and network utility.

Existing reprogrammable application frameworks are aimed towards achieving dynamic functionality among the WSN usage, i.e., the dynamic changing of WSN application code. Therefore, dynamic functionality beyond application functionality is not taken into consideration. This leads to current reprogrammable application frameworks unable to adapt to such application requests as mentioned above. This is difficult because tuning a data transmission component first requires the WSN to excessively shut down all data transfer due to network topology change and propagate the network with the new protocol. This two step required for changing a routing protocol is complex compared to an application switch, where only application code needs to be changed and the data transmission components intact.

As applications change on a reprogrammable application framework, the optimal routing protocol corresponding to the application's request must be chosen for the application's performance and network utility.

Our research goal is to provide WSN nodes with dynamic functionality among the data transfer layer, in order to meet the application changes done on the reprogrammable application framework. Figure 1.2 illustrates how AFRo would provide the diverse WSN applications with the optimal routing protocol.

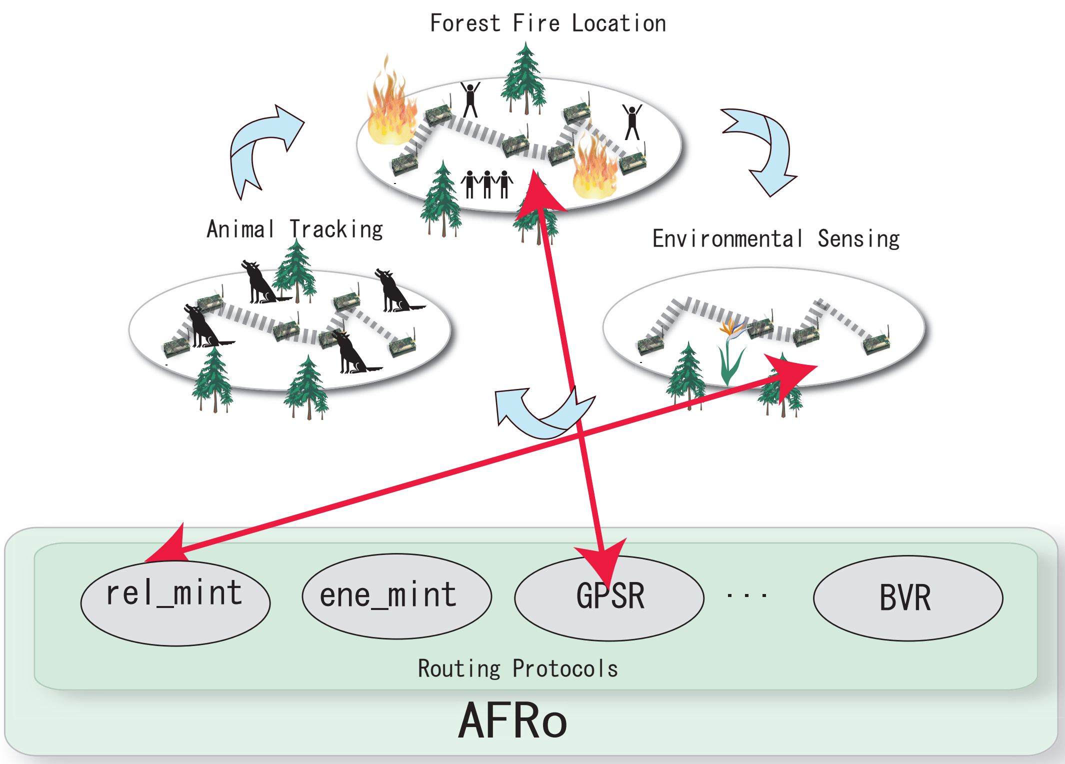


Figure 1.2: Research Goal

1.3 Structure of Thesis

The rest of the thesis is organized as follows. Chapter 2 describes the background materials in the area of WSN research, and related works for dynamic node configuration. In Chapter

3, we discuss the design and details of AFRo, and Chapter 4 explains the implementation of AFRo in both real life environment and a WSN simulator. In Chapter 5, we present the results and the analysis of several experiments, and the application performance achieved by using AFRo with traditional reprogrammable application frameworks. Finally, in Chapter 7, we present our conclusions and discuss future works regarding this subject.

Chapter 2

Background and Related Work

This chapter describes the summary and fundamental characteristics of WSN technologies. We then describe some of the major routing protocols used for multihop data transfer in a WSN. Finally, we describe some of the existing researches for routing protocol switching as related works.

2.1 WSN Technology

This section descries some of the fundamental properties of a WSN.

2.1.1 Overview

First of all, we will review some of the basic principles of a WSN. WSNs are networks in which are used for sensing our real life environments. WSN applications range from outdoor environmental monitoring to a more urban environment. Although the core function of sensing-and-sending does not change among the different environments, the diversity of target environments results in a drastic change of sensing hardware and software, network characteristics, and the algorithms used for data collection and transmission. We will address these diverse WSNs in our following chapters.

2.1.2 Hardware Platforms

A WSN is an interconnection of sensor nodes, in which each nodes functions individually, or as a hierachical component depending on the network's structure.

The sensing task is done through various sensor devices, which attached to the sensor node could measure physical conditions, such as temperature, humidity, pressures, etc. Sensordevices are categorized into 3 classes; active sensors, passive narrow beam sensors and passive omni directional sensors. While the former 2 tend to be more intelligent than the passive omni directional sensors, our research targets the latter type of sensor.

The sensed data is then sent to other sensor nodes through a transceiver. Radio frequency is one of the most frequently used transmission media, between a communication frequency of 433MHz and 2.4GHz. Optical communication and Infrared is also an alternative, but because optical communication requires communicating nodes to be in line of sight, and infared has its capacity limited to broadcasting data, radio frequency tends to be the most relevant among the mentioned.

With few exceptions, sensor nodes are deployed individually in an environment rather than wired to a connection cable, and are unable to rely on external power supply, and are required to be equipped with internal power resources. Batteries are therefore essential for sensor nodes in order for functioning. Two types of batteries are used among sensor nodes; rechargeable and non-rechargeable. It is obvious that sensor nodes equipped with rechargeable batteries are beneficial for a WSN in terms of maintenance and reliability. Some methods for recharging batteries are to supply each node with devices to convert energies such as solar power, heat and vibration, to battery power. However, because these devices tend to be expensive, and highly situational on the deployed environment, non-rechargeable batteries are still widely used among various WSNs.

The above mentioned hardware are all interconnected by a microcontroller device. A microcontroller processes tasks generated through events, and controls the other devices through its integrated circuit. Microcontrollers are programmable through various languages. For example, the Mica[3] family microcontrollers enables programming of sensor nodes by the nesC[10] language, while the SunSPOT[6] microcontrollers are programmable via Java[9].

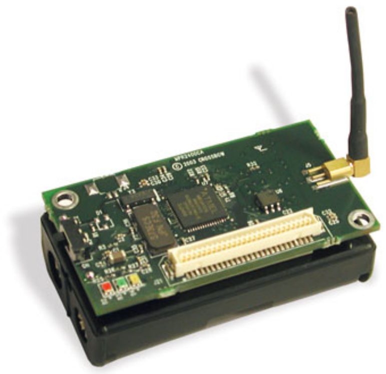
2.1.3 Applications

We present the various usage of a WSN with an existing deployment example. We classify the applications in three categories; environmental sensing, object tracking and guidance, and describe the applications in each category.

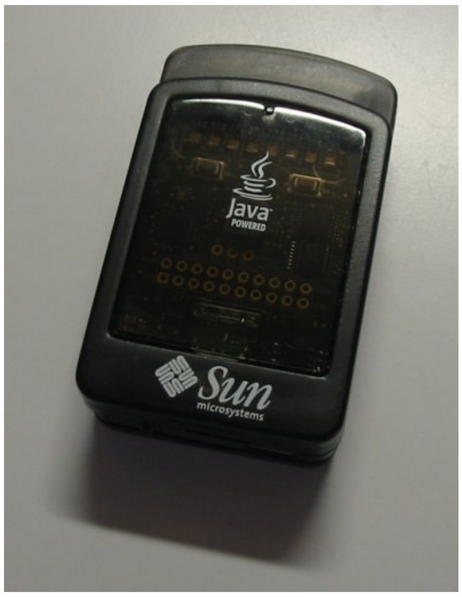
Environmental Sensing

Environmental sensing applications are deployed for humans to examine the environmental condition of a certain area of interest.

Talzi[19] et al., in their PermaSense project, has deployed a WSN among the Swiss Alps



(a) MICAz Mote



(b) Sunspot

Figure 2.1: Examples of Some Microcontroller Specifications

to collect permafrost related data. The PermaSense WSN consists of 10 sensor nodes which has been installed on Jungfraujoch at 3,500 m above sea level, and each node measures real-time parameters of temperature and conductivity values, which are indicative for rock moisture and its phase state in the near surface layer. Because the temperature of the deployed environment could fall below -30 degrees celsius with strong winds at times making it impossible to reach the nodes, the PermaSense WSN was initially deployed with the strong notion that WSN should be fully self-organizing and unattended for years, with several unique functionality to achieve this objective. At the time being, December 2008, the PermaSense project is successfully providing their WSN's gathered information through their online database[8].

Object Tracking

Object tracking, or event sensing applications are deployed to sense the location or occurrence of the event of a given, or unexpected events or objects.

Culler[2] has deployed a WSN to track the path of vehicles passing through the network. The nodes were initially deployed via an unmanned aerial vehicle. Magnetometer sensors were attached to the nodes in order to detect the proximity of the moving vehicles, and the nodes among the network would interact to estimate the path and velocity of the tracking vehicles.

The ZebraNet[23] project has deployed a WSN to observe and track the behavior of wild animals such as zebras and lions, which lives within a spacious territory. The WSN has been deployed at the Mpala research Center in Kenya, and the main interest of the project was to observe the behavior of individual animals, especially the interactions among different species and the impact of human development on the species. In ZebraNet, the animals of interest each carry sensor nodes, and an integrated GPS receiver is used to obtain estimates of their position and speed of movement. Light sensors are used to give an indication of their current habitant environment, and each node logs readings from their sensors on a three minute interval. Whenever a node enters the communication range of another node, the sensor readings and the identities of the sensor nodes are exchanged.

Guidance

In contrary to the above mentioned WSNs where a network is passively deployed, WSNs deployed for guidance purposes would actively note or perform certain actions depending on their environment.

The furniture parts and the construction tools are embedded with sensor nodes. These nodes contain various sensors; force sensors, gyroscope and accelerometers. The force sensor would sense the construction process involving joints, while the use of screwdrivers are notified by the gyroscope and hammer actions by the accelerometers. The sensor nodes form an ad-hoc network for detecting certain actions and sequences, and give visual feedback to the user via LEDs integrated into the furniture parts.

Table 2.2 summarizes the list of WSN application classification we have mentioned above.

Table 2.1: List of WSN Application Classifications

WSN Application Classification	Sample Application	Network Topology
Environmental Sensing	Volcano Activity Monitoring	Flooding
Object Tracking	Vehicle Location	Hierachial
Guidance	Furniture Construction Guidance	P2P

2.1.4 Routing Protocols

Compared to traditional computer networks such as the Internet, routing in a WSN is unique in a way that most of the routing protocols does not use an unique identifier, such as an IP[1] address, for data transmission. Instead, node hierarchy is often decided among various dynamic parameters, such as the wireless signal strength, remaining battery power, etc.

In this section, we address some of the frequently used routing protocols in a WSN.

MintRoute

MintRoute[22] is a link-state ad hoc routing protocol widely used among the TinyOS environment. In the MintRoute protocol, each node monitors a few neighbors and chooses the most optimal neighbor to be its parent for forwarding messages to the base station. The parent selection algorithm relies on the signal strength of each nodes; therefore, a parent of a specific node would change over time due to the dynamic changing nature of a WSN.

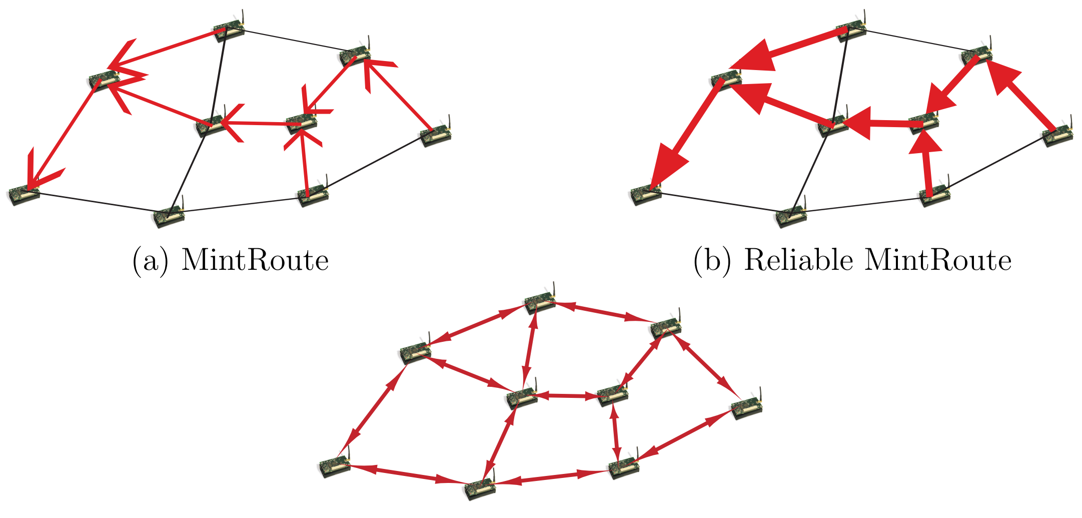
Reliable MintRoute

Reliable MintRoute is an alternative of the MintRoute protocol, with its emphasis on reliable data transfer. Compared to the original protocol, each node monitors their neighbor's signal strength of up to 16 neighbors, and periodically advertises its link quality to each neighbor. A cost function maps link quality and number of hops to a cost, and the nodes advertise their costs and try to minimize their total transmission cost.

Flooding

Flooding[16] is a primitive routing algorithm, where a node broadcasts its received data to its neighbors, unless a maximum number of hops for that packet are reached or the destination of the packets have arrived. Flooding could be implemented with less effort, and therefore used in many prototype systems. However, deficiencies such as implosion and overlapping of the packets, make it difficult for use in resource or time constraint environments.

Figure 2.2 illustrates the topology of our above mentioned protocols.



(a) MintRoute

(b) Reliable MintRoute

(c) Flooding

Figure 2.2: Figure of WSN Routing Protocols

2.1.5 Code Reprogrammability

WSNs are anticipated for deployment in location out of human reach; i.e., underwater, volcanoes, concrete walls, etc. Also, reconfiguration of a WSN node requires a tedious process of obtaining a node from such regions, and connecting the node to a programmable component. Hence, the node update cost among WSNs are significantly high compared to traditional computing environments. Under this constraint, a reprogrammable application framework has been proposed among node programming in a WSN. A reprogrammable application framework enables dynamic change of node context over a network, without physical human interaction. In figure 2.3. we shows how reprogramming a sensor node would enable WSNs to adapt to various user needs. In this example, an animal sensing application would be dynamically replaced with an application for forest fire evacuation, where a node sensing a fire would trigger the switching.

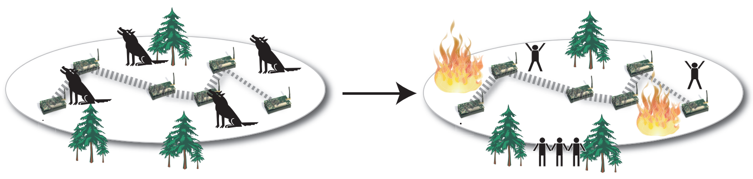


Figure 2.3: Reprogramming WSN Code

2.1.6 Application Requests

As described in the previous section, WSN application requirements towards the WSN data transfer are diverse. For example, a precipitation tracking application would require running for a long time period, hence requiring an energy-efficient routing protocol to save data transfer energy consumption. On the other hand, a wildlife tracking application would require an reliable routing scheme to keep track of the rapidly moving animals.

2.2 Research Issues

This section addresses research issues of our interest. Specifically, we focus on the dynamic changing of application requests on a WSN, and the problems which derive from this issue.

2.2.1 Mismatch between Application Requests and Routing Protocols

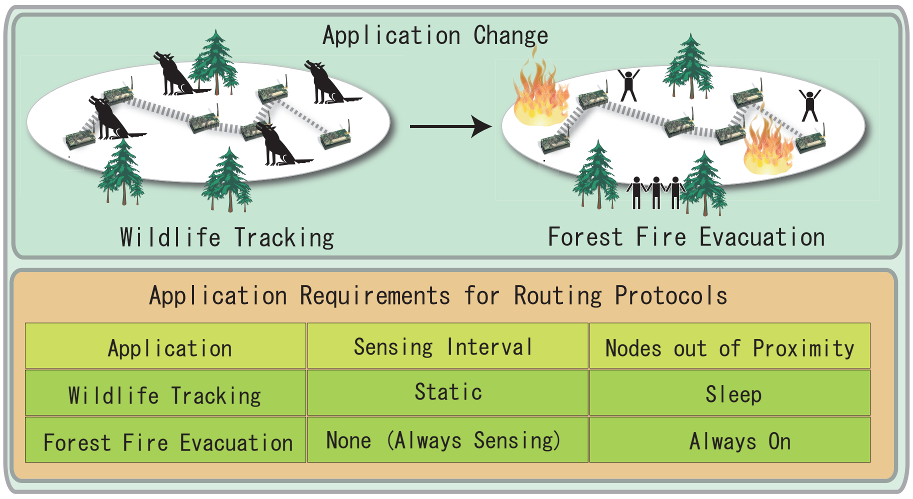
Consider a following WSN deployed in a suburban forest. Initially, this WSN was deployed for the purpose of precipitation monitoring along the forest. A precipitation monitoring application would be required to function for a long time span due to the nature of environmental sensing. Hence, the data transfer requirement for this application would be longevity, which correlates with energy efficient data transfer methods. On the other hand, a WSN deployed in such location could be also reprogrammed as a forest fire tracking application in case of emergency. Critical event tracking application would be require precise and rapid sensing of event occurence. Therefore , the data transfer requirement for this purpose would be the reliability of data transfer.

These two applications share the same physical network, yet they have two distinct application requests. As we mentioned in the prior section, it is ideal for each application requests to be served with the corresponding routing protocol.

2.2.2 Reprogrammable Application Framework Restrictions

In the previous case, the two applications have two application requests; the request for energy-efficiency and the request for rapid data transmission. Therefore, on reprogramming the application, it is ideal for a reprogrammable application framework to also reprogram the routing protocol used by the application.

In a traditional reprogrammable application framework, such application requests towards the network's data transfer is not taken into consideration; i.e., traditional reprogrammable application frameworks are responsible for changing the context of the application, but the underlying routing protocol. Because WSN applications are heavily reliant on the node's routing architecture, traditional reprogrammable application frameworks are not able to optimize, and meet the application's request from the perspective of network data transmission. Figure 2.4 illustrates the diverse application requirements, and why traditional reprogrammable application frameworks are not sufficient to handle the various requests.



Application	Sensing Interval	Nodes out of Proximity
Wildlife Tracking	Static	Sleep
Forest Fire Evacuation	None (Always Sensing)	Always On

Figure 2.4: The Diverse Application Requests among Various Applications

2.3 Related Works

This section discusses some of the related works concerning the research issues mentioned above.

2.3.1 A Programmable Routing Framework for Autonomic Sensor Networks

He[11] has proposed a programmable routing framework for Autonomic Sensor Networks. In his research, He has divided WSN routing protocols into several programmable components, i.e., the structure of the neighbor lists, the data forwarding mechanisms, and etc. Combining these programmable components would allow programmers to develop a universal routing service to adapt to various WSN applications. The framework also embeds a shared library component which encapsulates commonly used routing method for code size reduction in a resource constraint environment such as a sensor node.

He's approach is flexible in a way that only certain portions of the runtime code is required to change in order to fulfill several application requests. However, the dynamic tuning of several particles suffers from severe overhead due to the checking required for the various components to consecutively function.

In AFRo, we challenge this approach by switching the entire routing protocols. Though AFRo may lack flexibility and extendibility compared to He's approach, we believe switching routing protocols as a whole would suffer less overhead and lead to a faster changeover of routing protocols.

2.3.2 TinyCubus

Marron[17] aims at providing an infrastructure to support various application needs. Marron's framework TinyCubus, is aimed for TinyOS based sensor networks, and consists of a data management framework, a cross-layer framework and a configuration engine. The data management framework enables dynamic selection and update of system and data management components. The cross-layer framework support data sharing and interaction between components in order to achieve cross-layer optimizations, and the configuration engine allows the distribution of application code. Routing in TinyCubus is done in a cross layer manner where routing decisions are made corresponding to the external features of the framework. However, TinyCubus uses a single routing architecture with tunable parameters.

As we have described in the previous section, the overhead a parameter changing approach for constructing a routing algorithm is intolerable in a case where rapid changing of routing protocols are required. Our approach of switching the entire routing protocol compared to the parameter changing method would be more rapid, and able to serve improptu or emergency application switching.

Chapter 3

An Adaptive Framework for WSN Routing

This chapter describes the detailed design of AFRo. First, we explain some scenarios in which AFRo would enable applications running on a reprogrammable application framework to meet the application requirement based on the switching of routing protocol. Next, we introduce the mechanism of how the optimal routing protocol is selected per application. We then introduce the basic system components of our proposed framework.

3.1 Assumption

In this section, we describe our proposed framework AFRo's overview and present the basic functionality. For our ease of explanation for this chapter, we assume a WSN were to be deployed in a deep suburban forest, and describe how AFRo would sufficiently change the routing protocol based on application requests. Our scheme is described in the figure below.

Our sample environment would consist of a WSN and a sink node. As we have described in chapter 2, a WSN interacts with exterior networks such as the Internet via a sink node. In this case, a sink node is used for propagating application code into a WSN. Suppose, there was a request to run three applications on different locations of the forest; namely an application to help hikers to evacuate from a sudden forest fire, an application to track the habitats of wild wolves, and an aplication to monitor humidity.

Now let us review that diverse applications have diverse application requests. In this case, a forest fire evacuation application would be required to assist hikers immediately to show the safe evacuation route, while the wolf tracking application would be required to precicely track the moving objects, and the humidity monitoring application would be required to function in an energy efficient manner. These diverse requests directly corresopnd to the routing protocol the application uses.

If this WSN were to not be programmed with AFRo on its nodes, all three regions with diverse applications would be running it's desired application, each with the same routing protocol. This is obviously inefficient both in terms of application's performance and network utility. For example, if the whole WSN were to be using/ a Reliable Mintroute protocol, altough the application request among the humidity monitoring application would suffice, other applications would not function in an optimal manner, and the network resources among regions using non-optimal routing protocol per application would be either wasted or lacking.

We have observed this trait of application performance and routing protocol by a simple experiment. In our experiment, we have run two light sensing applications, both using two distinct routing protocols, on a TOSSIM[13] network simulator. We have observed the total number of packets transmitted through the network for each application, with the increase of node numbers in the network. Figure3.1 shows our experimental results. The results indicate that the Reliable Mintroute routing protocol requires a significant amount of packet for data transmission compared to the Energy Efficient Mintroute.

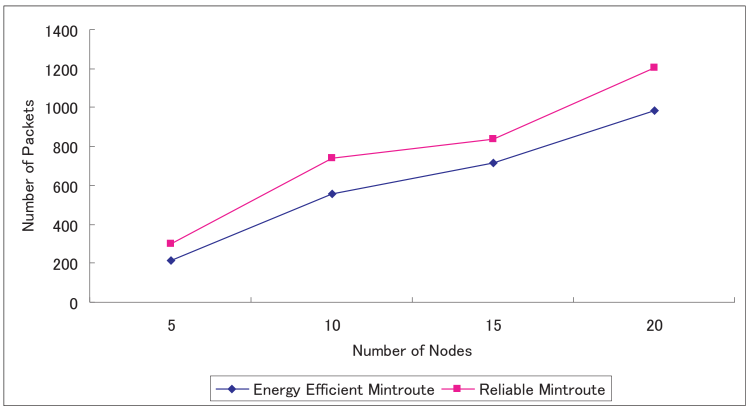


Figure 3.1: Application Performance based on Different Routing Protocols

From our prior example, obviously, the wolf tracking application would prefer to use the Energy Efficient Mintroute. Therefore, among application deployment in a WSN, we must first observe the application request for the application, and select the optimal routing protocol based on that application request. AFRo is built among this mission, and selects the most suited routing protocol per each application.

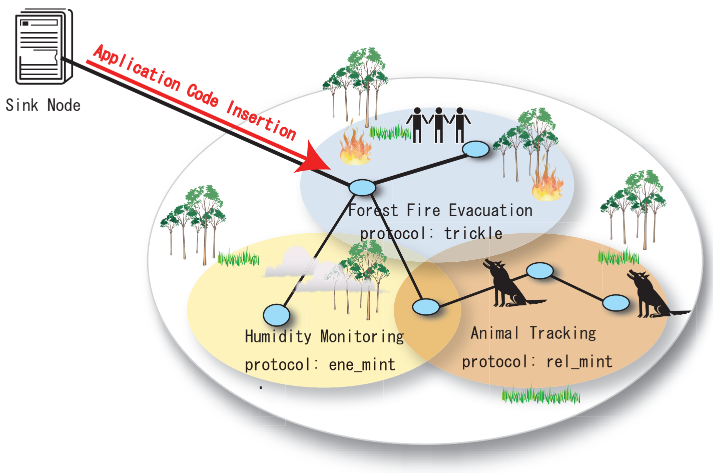


Figure 3.2: An Example AFRo Deployment

3.2 Why Use AFRo

In this section, we present two application scenarios using AFRo, and describe the merits of using our framework. First, we show how AFRo would fulfill diverse application requests regarding several applications. Then, we describe how AFRo would simplify the overhead required for routing protocol switching in a traditional WSN environment.

3.2.1 Application Requests among a WSN

Let us assume that a WSN deployed in a suburban forest for precipitation monitoring. A precipitation monitoring application would be required to function for a long time span due to the nature of environmental sensing. Therefore, the data transfer requirement for this application would be longevity, which correlates with energy efficient data transfer methods.

On the other hand, a WSN deployed in such location could be also used as a forest fire tracking application in case of emergency. Critical event tracking application would be require precise and rapid sensing of event occurence. Therefore , the data transfer requirement for this purpose would be the reliability of data transfer.

3.2.2 Overhead to Fulfil Several Application Requests

WSNs have been deployed in manufactural scenes for several purposes. Ruud[18] has described a cold chain management infrastructure using a WSN and has insisted that WSN under such purpose would require several complex relationship and self-organizing protocols for node communication. Under this environment, several nodes would be required to function individually, hence the application requirement towards the node's data transfer methods varies. The current model of this deployment would require every diverse functioning nodes to each be programmed with different data transfer mechanics based on the overlying application.

3.3 AFRo Design

WSN applications are required to select the most relative data transmission mechanism based on the application requirement; yet, the current WSN programming architecture makes this a difficult task.

In this section, first we discuss the merits of using AFRo as a routing layer framework. Next we will describe the mechanism for selecting routing protocols via various application, and explain how the switching is done in our framework. We then discuss how the application layer framework would interact with AFRo, and finally we will present the system architecture of our proposed framework.

3.3.1 Application Selection of Routing Protocols

Because of the diversity of WSN application requests, it is impossible to handle all application request among a single routing protocol. However, it is possible to extract the features among these requests, and classify them among a certain criteria. In this section, we discuss the various WSN application requests, and classify them among two vectors; the service quality of an application, and the networking topology under which the application runs.

3.3.2 Service Quality

An ideal WSN application would function under the prospect of data transfer that i) the data transferred among the network is reliable, i.e., less packet losses, rapid data transfer, and fault-tolerance, and ii) it functions in an energy efficient manner for longevity of the application itself. Ironically, these two concepts conflict with each other, because stressing reliability among a WSN correlates with more packet transferred among the network, hence faster energy consumption; which leads to the fact that a WSN application could not be reliable and energy-efficient at the same time. To make things worse, because WSN applications are distinct from the underlying data transfer methods, it is possible for applications to select the irrelevant data transfer method. For example, a precipitation monitoring application required to function for a long time period might use a routing protocol which excels in rapid packet transfer, sacrificing energy-efficiency which disagrees with the initial deployment principle. WSN applications are therefore required to select the routing protocol that relatively matches it's desired service quality.

3.3.3 Network Topology

The network topology in which WSN applications communicate also has sufficient impact on the application's performance. A centralized networking topology consists of a single sink node and numerous sensing nodes. The sensing node's objective is to sense environmental data and transfer it to the sink node, where it aggregates all data gathered by the sensing nodes and provide an interface for users to access the data. In a centralized networking topology, while each sensing nodes cooperate with each other in a multi-hop manner to send data to the sink node, there is no peer to peer connection involved. A node does not establish nor recognize data transfer channels between other nodes; instead, it forwards the data to its neighboring nodes based on the routing algorithm. Because the sensing of data and data transfer is distinct, plain environmental sensing applications which requires no complex transaction among each nodes well suits this topology. However, a node could also be programmed to transfer data in a peer to pee manner. Under this topology, a node would establish data transfer channels between distinct nodes other than the sink node. A P2P networking topology enables complex sensing mechanisms based on inter-node communication. For example, applications that are deployed for tracking objects are required to dynamically sense the location of the object, hence requiring P2P communication among the nodes.

3.3.4 Classification of WSN Routing Protocols based on Application Requests

Under the classification of routing protocols mentioned above, we have categorized the various WSN applications with the relevant routing protocols in figure 3.3.

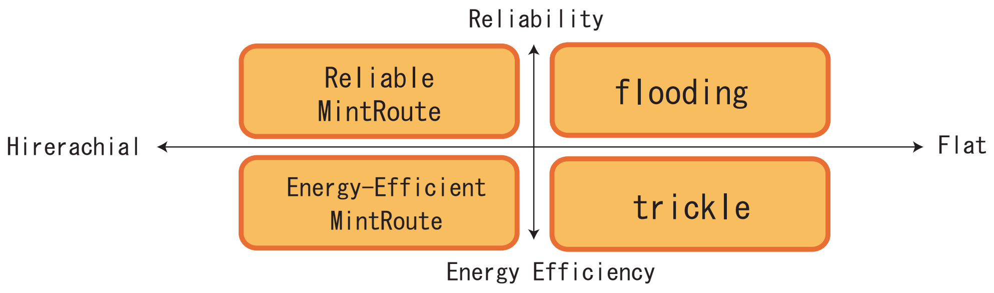


Figure 3.3: Routing Protocol Clasification

The vertical axis describes the network topology in which the routing protocol would construct; i.e., either hierachial or P2P. The horizontal axis indicates the service quality, described as either reliable or energy efficient.

3.3.5 Selection of Routing Protocols for Applications

AFRo enables dynamic switching of routing protocols per application. Following the classification of routing protocols per applications in section 2, AFRo interprets application requests as queues, and refers to the inner table which hold corresponding information between an application request and the routing protocol. For example, if a WSN deployed in a forest were initially programmed with an animal tracking application, AFRo would select the reliable MintRoute protocol because applications in the form of tracking would be required to precisely locate the location of the object, rather than an energy-efficient tracking without precision. If the WSN application were to be reprogrammed with a precipitation monitoring application after the tracking obligation has finished, AFRo would dynamically change the routing protocol to Energy Efficient Mintroute, because precipitation monitoring requires the long-term sensing of an environmental data, hence the energy-efficient protocol is selected in this scenario.

3.3.6 Application Interaction

AFRo is built as a TinyOS component. More specifically, it is a component of the MATE Virtual Machine, which is dynamically reprogrammable environment of WSN applications. Whenever an application is loaded onto a sensor node, MATE triggers a reset timer which notifies that the node has been loaded with a new application code. AFRo takes advantage of this feature; i.e., whenever the reset timer is triggered, AFRo evaluates the application's request from our latter shown component via referring to an application request table. AFRo then connects the functions required for routing with that of the optimal routing protocol's function.

Note that the switching of routing protocol is done seamlessly without any user interaction with the node's network capabilities. MATE application programmers would specify their application requests at the beginning of the code, and AFRo would automatically wire the optimal routing functions with the application.

This feature is unique compared to our related works mentioned in chapter two, where each routing mission would require complex tuning of parameters by the application programmers. Although complex tuning allows application programmers to be more dominant and control the network's capability in detail, AFRo's automatic switching requires no maintenance nor painful runtime configuration. If we were to consider future WSNs where WSNs were to be more casually deployed in our everyday environment unlike most of the WSN that are deployed in the research field today, the automatic configuration method would be far more desirable due to the simple deployment and running of application code.

3.3.7 System Architecture

The basic system architecture of AFRo is described in figure 3.4 We will discuss the core AFRo modules in the following section.

Reprogrammable Framework Interface (RFI)

The RFI serves as an interface between AFRo and the overlying application layer. Current reprogrammable application frameworks statically wire a single routing protocol to use throughout the framework. To achieve dynamic switching of routing protocols in AFRo, we need to forward the reprogrammable application framework wiring vector to a manually defined module which masquerades itself as a routing module, but functions as a gateway for the routing protocol switching architecture. RFI also receives application parameters from the upper layer, and transmits these parameters used for protocol selection in the Application Requirement Conversion Module.

Application Request Conversion Module (ARC)

The ARC converts the application requests fetch from the RFI to the corresponding routing protocol, which then transmits this information to the RPS. An inner conversion table is defined in the ARC for conversion.

Routing Protocol Switching Module (RPS)

The RPS recieves the application's desired routing protocol data from the ARC, and provides the pointer to the desired routing protocl to the RFI. Note that keeps a list of routing protocol binaries on its memory. Because most WSN node programming frameworks provide an interface for routing mechanisms, we could not directly use the existing routing codes for each application; rather, the routing protocols we use in AFRo are required to be slightly altered to meet the routing interface of the reprogrammable application framework.

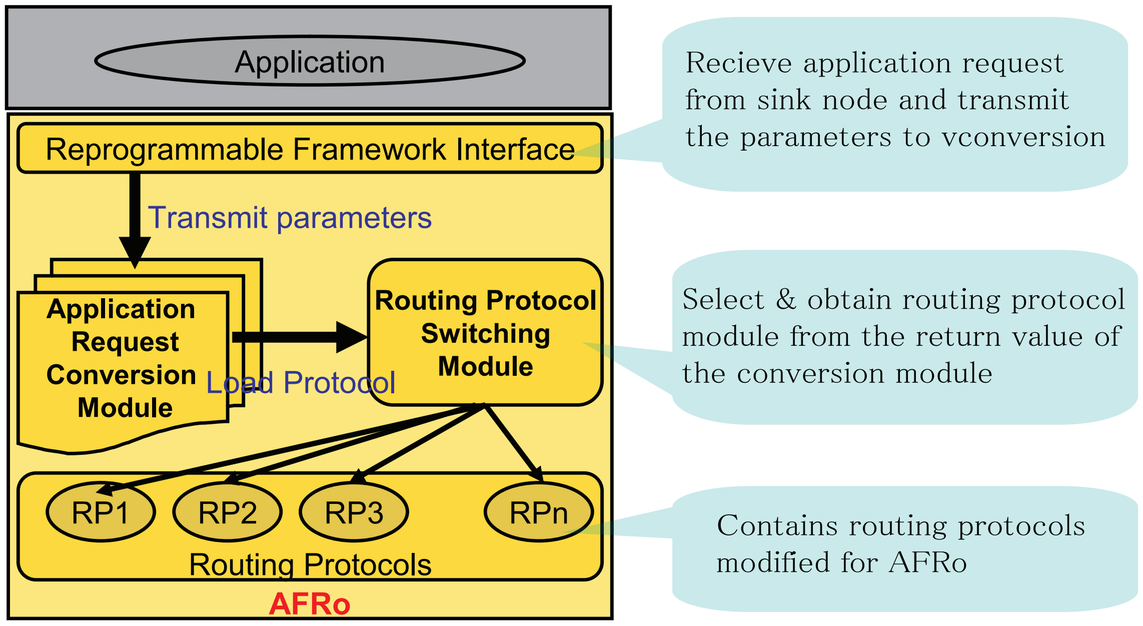

Application
Reprogrammable Framework Interface
Transmit parameters
Application Request Conversion Module
Load Protocol
Routing Protocol Switching Module
RP1
RP2
RP3
RPn
Routing Protocols
AFRo
Recieve application request from sink node and transmit the parameters to vconversion
Select & obtain routing protocol module from the return value of the conversion module
Contains routing protocols modified for AFRo

Figure 3.4: AFRo System Architecture

Chapter 4

Implementation of AFRo

This chapter describes the implementation of our research. First, we overview the basic components of AFRo as a TinyOS component. Next, we describe the Mate Virtual Machine implementation used for AFRo. Finally, we explain the implemental details of the three major components of AFRo.

4.1 Overview

Our AFRo implementation is deployed as a TinyOS component, which runs on a tailored MATE Virtual Machine environment. In this section, first we describe the MATE Virtual Machine tuning in required to run our proposed framework. We will show the basic operation codes(opcodes) used for our reseach, and illustrate the basic TinyOS components used to construct the Virtual Machine used in our framework. Next, we will describe our routing protocol switching function which is implemented as a TinyOS component. Finally, we will show how the above features are wired into our whole framework.

4.2 MATE Virtual Machine Implementation

In this section, we give a detailed explanation of the MATE Virtual Machine we use for our AFRo implementation.

4.2.1 Opcodes

As we have shown in the previous section, the MATE Virtual Machine is a runtime environment for MATE applciations written in its specific language. Although several applications could be run on its default configuration, the initial **send()** routine is wired with a default routing protocol(MintRoute), which therefore is not capable of switching protocols based on various applications. Our research targets dynamic switching of routing protocols based on application switching; therefore we must construct a custom made Virtual Machine -+ environment to meet our interests.

For this matter, first we have defined the opcodes used for constructing a MATE Virtual Machine applications. List 4.1 describes our opcode configuration file.

The opcodes defined in our configuration will be the primary functions used for constructing an AFRo application. For example, the **bclear** and **bfull** opcodes defined at line 12 and 13, are used for clearing the application's buffer and checking wheter the buffer is full or not. Also, for sensing environmental values, we will use two pre-defined functions, namely **light** for light for light readings, and **temp** for temperature readings. AFRo's core functionality is called using the **send** opcode, where application programmers would call by specifying the application's request as an argument.

List 4.1: Mate Configuration used for AFRo

```
 1
 2 <VM NAME="BombillaMicaVM" DESC="AFRo tailored vm"
 3 DIR="../../../../apps/BombillaMicaNishiki">
 4 <SEARCH PATH="../opcodes">
 5 <SEARCH PATH="../contexts">
 6 <SEARCH PATH="../languages">
 7 <SEARCH PATH="../extensions">
 8
 9 <LOAD FILE="../sensorboards/micasb.vmsf">
10 <LANGUAGE NAME="tinyscript">
11
12 <FUNCTION NAME="bclear">
13 <FUNCTION NAME="bfull">
14 <FUNCTION NAME="bsize">
15 <FUNCTION NAME="bufsorta">
16 <FUNCTION NAME="bufsortd">
17 <FUNCTION NAME="eqtype">
18 <FUNCTION NAME="err">
19 <FUNCTION NAME="id">
20 <FUNCTION NAME="int">
21 <FUNCTION NAME="led">
22 <FUNCTION NAME="rand">
23 <FUNCTION NAME="send">
24 <FUNCTION NAME="sleep">
25 <FUNCTION NAME="uart">
26
27 <CONTEXT NAME="Trigger">
28 <CONTEXT NAME="Timer0">
29 <CONTEXT NAME="Timer1">
30 <CONTEXT NAME="Once">
31 <CONTEXT NAME="Reboot">
32 <CONTEXT NAME="Broadcast">
```

4.2.2 MATE Application Code

Using the opcodes defined in the previous section, we will describe the MATE written applications we use for our evaluation. List 4.2 describes the MATE application code of an environmental sensing application which periodically senses temperature.

List 4.2: Light Sensing Application on MATE

```
buffer buf;
bclear(buf);

for i=0 until bfull(buf)
  buf[] = light();
next i
send(ENVIRONMENTAL);
```

4.3 AFRo Components

In this section, we present the nesC written components of our Framework.

4.3.1 Reprogrammable Framework Interface

RFI is a nesC component used as a gateway component for AFRo to interact with the application layer. The getRoutingProtocol function fetches the application's requested routing protocol, and pushes it's value onto the MATE stack. List 4.3 describes the TinyOS implementation of the RFI. The RFI defines the application request and the corresponding routing protocol from line 10 to 13. The routing protocol modules are then defined as a TinyOS module at line 19 to 22, which is then wired to the RFI itself at line 37 to 39.

List 4.3: Reprogrammable Framework Interface

```
1 includes Mate;
2
3 configuration OPsend_afro {
4   provides interface MateBytecode;
}

implementation {

  enum {
    APP_ENE_HIE = unique("ene_mint"),
    APP_ENE_FLAT = unique("trickle"),
    APP_REL_HIE = unique("rel_mint"),
    APP_REL_FLAT = unique("flooding")
  }

  components OPsend_afro, MStacksProxy, MQueueProxy;
  components MErrorProxy, MContextSynchProxy, MVirusProxy;
  components MTypeManagerProxy;
  MultiHopRouter_ene_mint as multihopM_ene_hie,
  MultiHopRouter_trickle as multihopM_ene_flat,
  MultiHopRouter_rel_mint as multihopM_rel_hie,
  MultiHopRouter_flood as multihopM_rel_flat;

  components GenericComm as Comm, MateEngine as VM;

  MateBytecode = OPsend_afroM;

  getRoutingProtocol(APPLICATION\_REQUEST);

  OPsend_afroM.Queue -> MQueueProxy;
  OPsend_afroM.Synch -> MContextSynchProxy;
  OPsend_afroM.Error -> MErrorProxy;
  OPsend_afroM.TypeCheck -> MTypesProxy;
  OPsend_afroM.Type  -> MTypeManagerProxy;
  OPsend_afroM.Stacks       -> MStacksProxy;
  OPsend_afroM.EngineStatus -> VM;
  OPsend_afroM.SendAdHoc -> Router.Send[AM_MATEROUTEMSG];
  OPsend_afroM.sendDone <- Comm.sendDone;
  OPsend_afroM <- VM.SubControl;
  VM.SubControl -> Router;
}
```

4.3.2 Application Request Conversion Module

In this section, we will describe the implementation of ARC. List 4.4 describes the TinyOS implementation of ARC. First we define the interconnection of routing protocols and application requests by the **enum** control sequence. We use the **unique()** function to create a static value throughout the component. Then, we load our routing protocols through the **components** declaration. Each routing protocol is defined as an individual component in its specific file under the $TOSROOT/tos/lib/ directory.

List 4.4: Application Request Conversion Module

```
implementation{
  // define routing protocol and application
  // request table
  enum {
    APP_ENE_HIE = unique("ene_mint"),
    APP_ENE_FLAT = unique("trickle"),
    APP_REL_HIE = unique("rel_mint"),
    APP_REL_FLAT = unique("flooding")
  }

  // define multihop components
  components ...
  MultiHopRouter_ene_mint as multihopM_ene_hie,
  MultiHopRouter_trickle as multihopM_ene_flat,
  MultiHopRouter_rel_mint as multihopM_rel_hie,
  MultiHopRouter_flood as multihopM_rel_flat;

  ...
}
```

4.3.3 Routing Protocol Switching Module

The RPC enables dynamic switching of routing protocols based on application requests. We have implemented the RPC as a switch statement using the application's as a parameter.

Chapter 5

Performance Evaluation

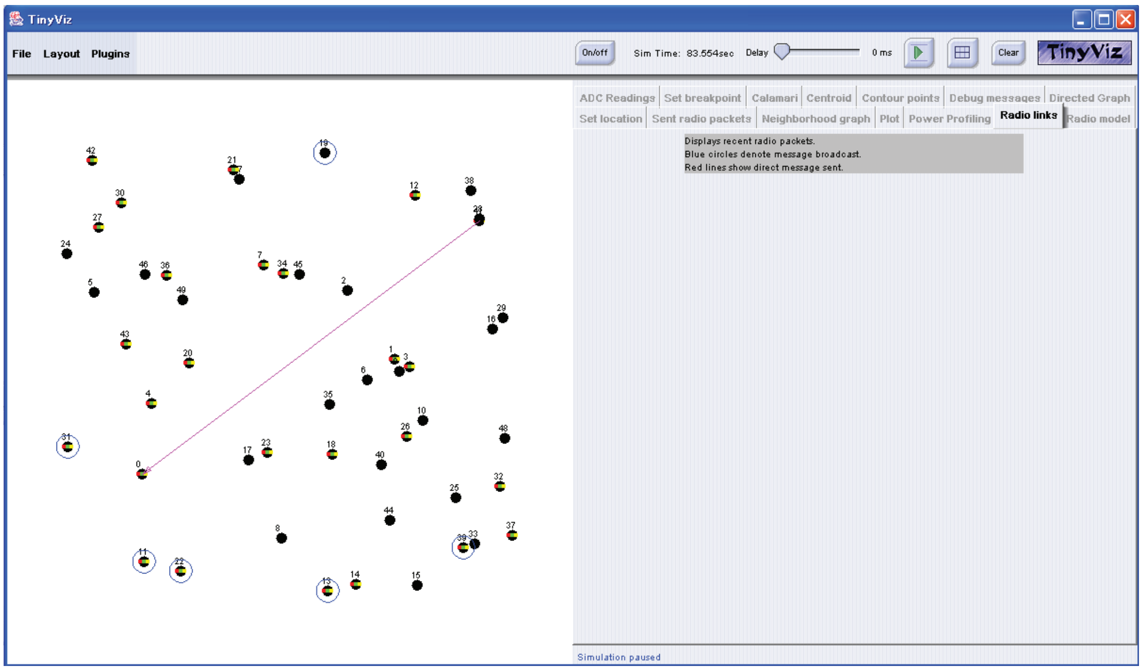
In this chapter, we show the experimental results of AFRo and detailed simulation results. In our evaluation, we test to observe the routing protocol switching overhead of AFRo based on several application requests. We then address some future evaluation methods in consideration.

5.1 Evaluation Environment

We have evaluated our AFRo framework under the TinyOS Simulator (TOSSIM)[13]. Unlike other network simulators such as the NS2[5], TOSSIM simulates TinyOS codes as if it were running on a native hardware. We have run TOSSIM on the below environment under Cygwin[4]. We describe our sample configuration used for evaluation in the table below.

Table 5.1: Evaluation Environment

Environment	Version
Operating System	WindowsXP SP3
TinyOS Version	1.1.15
GCC Version	avr-gcc (GCC) 3.3-tinyos
Java Version	1.4.1_02



Displays recent radio packets.
Blue circles denote message broadcast.
Red lines show direct message sent.

Figure 5.1: TOSSIM Screen Shot

5.2 Protocol Switching Overhead

We have evaluated AFRo by the switching overhead of protocols. We have defined the number of packets required for routing protocol propagation, and the propagation time required as a metric for our evaluation. The WSN parameters of our evaluation environment is listed in table 5.2. Each node on the network is coded with MATE and AFRo. Our evaluation was first done by running a humidity monitoring application. This application senses the environment 's humidity information on a 1 minute interval. The default routing protocol under use is Reliable MintRoute. After 5 minutes of running the application, we performed a routing protocol switch, by dynamically reprogramming the node with the same application, but with the Energy Efficient Mintroute protocol. We have measured the overall packet number and time required for this evaluation, under several quantity of nodes.

Table 5.2: Packet Overhead for Routing Protocol Switch

Node Quantity	Number of Packets Required for Routing Protocol Switch
10	44
20	263
30	449
40	657
50	823

Figure 5.2 shows that the number of packets required for routing protocol shows a linear increase with a gradient of approximately 20 to that of the number of the deployed nodes. Our results indicate that the packet overhead is predictable given the number of nodes in a WSN.

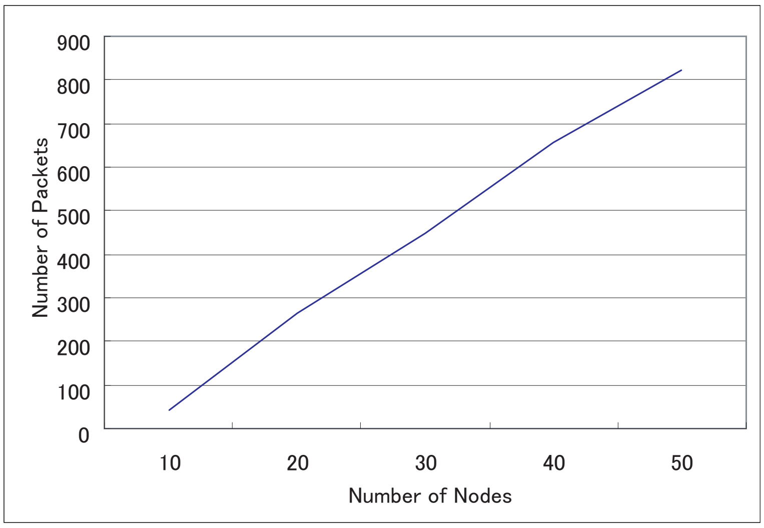


Figure 5.2: Packet Overhead Graph

5.3 Switching Time

The time required for switching a routing protocol in a WSN is critical for applications deployed in purpose of an emergency situation; i.e., fire location tracking, intruder detection. We have therefore evaluate the quantity of packets transferred among the network on routing protocol switching among AFRo. Table 5.3 describes the mean time length for AFRo to perform a routing protocol switch.

Table 5.3: Switching Time Overhead

Node Quantity	Switching Time (sec)
10	20
20	70
30	80
40	84
50	83

We have observed that the overhead for routing protocol data propagation caps at approx-

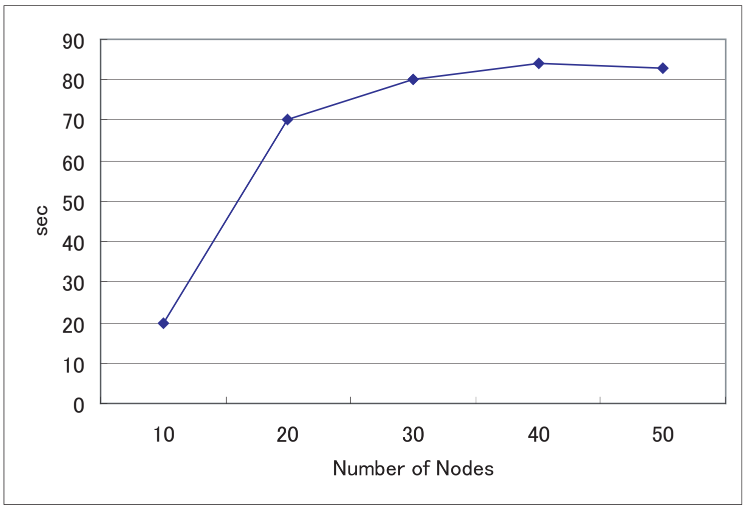


Figure 5.3: Protocol Switching Time Overhead

imately 80 seconds. That being said, we could assume that switching to application-suited routing protocol via AFRo is beneficial in situations, where an application's task is done in a large time interval. For example, application deployed for environmental sensing, which senses environmental information on an one hour interval would not notice the protocol switching time overhead due to the long delay in application's task processing. On the other hand, the routing protocol switching benefit would be lost among mission critical applications and applications required to process information in a short time interval, because the switching time overhead would greatly exceed the application's task execution.

5.4 Future Evaluations

In this thesis, we have evaluated AFRo from a quantitative approach. We have evaluated the overhead required for AFRo to perform a routing protocol switch from the terms of the number of packets and required switching time.

Our experimental results show that AFRo does not impose severe overhead to the deployed node. With that into consideration, for future evaluations, we will perform a qualitative evaluation based on a real-life case study. We will simulate a sensor network deployed in a forest and perform switching of routing protocols based on application switches. We will observe whether the routing protocols switch would qualify the appliation's request, and present a best-case and worst-case model among AFRo usage.

Chapter 6

Conclusions and Future Work

In our final chapter, we summarize our evaluation results and refer to future works regarding our framework. We have proposed AFRo, an Adaptive Framework for WSN Routing. While traditional reprogrammable application frameworks were not capable of changing routing protocol based on application requests, AFRo adapts to the current running application and offers the most optimal routing protocol to the applciation. Our future works involve the propagation mechanism of routing protocols and dynamic construction of routing protocols in our framework.

6.1 Summary

In this thesis, we have proposed AFRo to dynamically adapt routing protocols to applications based on the application's request. AFRo enables dynamic selection of WSN routing protocols based on an application request, and enhances current WSNs to adapt to various application requests.

In AFRo, the routing protocols are classified by two metrics; network topology and service quality. Network topology is defined by either the routing protocol takes a flat network topology or a hierarchical topology. The service quality is defined by the protocol's reliability. From these classification, we have embedded four protocols, Reliable MintRoute, Energy-Efficient MintRoute, Flooding and Trickle. Applications are dynamically provided by the optimal routing protocol by specifying it's request towards the routing protocol.

We have implemented our framework as a TinyOS component, and used the MATE Virtual Machine for our running environment.

Our evaluation shows that the number of packets required for AFRo to propagate a routing protocol into the network is linear to that of the number of deployed nodes. We have also shown that the propagation of routing protocol would be complete within 80 seconds among any network scale. The results have proven that AFRo is usable regardless of the network scale and if the application could tolerate an 80 second delay among routing protocol propagation.

6.2 Future Works

In this section, we propose two works in progress for our framework.

6.2.1 Routing Data Propagation Algorithm

In this thesis, we have focused on the switching mechanism of routing protocol based on application requests. For routing protocol code propagation, we have used MATE's standard code propagation protocol, Trickle[14] as the propagation protocol. Trickle uses a "polite gossip" policy, where each node broadcasts a code summary to local neighbors but stay quiet if they have recently already received the same code. Trickle's broadcasting technique is efficient in certain environments where each node has rich hardware and energy resources. However, Trickle assumes all sensor nodes are powered on at the time of code propagation; i.e., in an environment where both node powered on and off coexists, the network would not be able to keep the consistency with the network's transmitted data. This could lead to ineffective sensing of the application, or even the falldown of the whole network.

In order to avoid such schemes, we must deploy a routing protocol propagating mechanism which enables safe propagation of application code in a safe and consistent manner.

6.2.2 Dynamic Construction of Routing Protocol

As we have mentioned in Chapter 2, He and Marron has deployed a framework where programmers would select various parameters for their applications and routing protocol to ahieve flexibility, with the cost of complex protocol propagation and speed.

In our framework, we have switched the entire routing protocol code in request from the application. Although our approach tend to adapt to rapid and impromptu application changes, the application requirements AFRo is able to handle is constrained by the four routing protocols AFRo posesses. As our future work, we must consider both using a preset routing protocol and using a routing framework which the application progammers could tune parameters and cope with more complex application requests.

6.3 Future Evaluations

In this thesis, we have evaluated AFRo from a quantitative approach. We have evaluated the overhead required for AFRo to perform a routing protocol switch from the terms of the number of packets and required switching time.

Our experimental results show that AFRo does not impose severe overhead to the deployed node. With that into consideration, for future evaluations, we will perform a qualitative evaluation based on a real-life case study. We will simulate a sensor network deployed in a forest and perform switching of routing protocols based on application switches. We will observe whether the routing protocols switch would qualify the appliation's request, and present a best-case and worst-case model among AFRo usage.

Acknowledgments

First and foremost, I would like to thank my advisor, Professor Hideyuki Tokuda, for his technical and professional advice, guidance, and encouragement.

I would like to thank Professors Jun Murai, Hiroyuki Kusumoto, Osamu Nakamura, Kazu Takashio, and Instructors Noriyuki Shigechika, Rodney D. Van Meter, Keisuke Uehara, Jin Mitsugi and Jin Nakazawa for their valuable and technical comments on this thesis.

I am extremely thankful for Dr. Hiroto Aida, Dr. Masato Saito and Masato Mori for their great support and advices. I would also like to thank the numerous members of Tokuda and Murai Laboratory for their great support .

Finally, I would like to thank my beloved family for the consecutive support of my daily life. I can no other answer make, but, thanks, and thanks.

January 27, 2009

Takatosi Kanazawa

References

Published Papers Related to this Thesis

- Takatosi Kanazawa, Hiroto Aida, Kazunori Takashio, and Hideyuki Tokuda
  "AFRo: An Adaptive Framework for WSN Routing, " Asian Workshop on Ubiquitous and Embedded Computing (AWUEC),
  Aug. 2008.
- Takatosi Kanazawa, Hiroto Aida, Kazunori Takashio, and Hideyuki Tokuda
  "AFRo: 無線センサネットワークにおける動的なルーティングプロトコルの変更機構, " ユビキタス・センサネットワーク研究会 (USN),
  Jan. 2009.

Bibliography

[1] *RFC791 - Internet Protocol. Retrived from http://www.faqs.org/rfcs/rfc791.html.* 1981.

[2] *29 Palms Fixed/Mobile Experiment Home Page. Retrived from http://robotics.eecs.berkeley.edu/ pister/29Palms0103/.* 2008.

[3] *Crossbow technology inc. Retrived from http://www.xbow.com.* 2008.

[4] *Cygwin website. Retrived from http://www.cygwin.com/.* 2008.

[5] *The Network Simulator website. http://www.isi.edu/nsnam/ns/.* 2008.

[6] *SunSpot World website. Retrived from http://www.sunspotworld.com/.* 2008.

[7] *TinyOS website. Retrived from http://www.tinyos.net/.* 2008.

[8] *PermaSense Online Database Browser. Retrived from http://tik42x.ee.ethz.ch:22001/.* 2009.

[9] Technical Advisor, Ken Arnold, Tim Lindholm, Frank Yellin, Frank Yellin, The Java Team, Mary Campione, Kathy Walrath, Patrick Chan, Rosanna Lee, Jonni Kanerva, James Gosling, James Gosling, James Gosling, James Gosling, Bill Joy, Bill Joy, Bill Joy, Guy Steele, Guy Steele, Gilad Bracha, and Gilad Bracha. The java language specification second edition, 2000.

[10] David Gay, Matt Welsh, Philip Levis, Eric Brewer, Robert Von Behren, and David Culler. The nesc language: A holistic approach to networked embedded systems. pages 1–11, 2003.

[11] Yu He, C. S. Raghavendra, S. Berson, and B. Braden. A programmable routing framework for autonomic sensor networks. In *Autonomic Computing Workshop*, pages 60–68, 2003.

[12] P. Levis and D. Culler. Mate: A tiny virtual machine for sensor networks. In *International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, USA*, Oct. 2002.

[13] Philip Levis, Nelson Lee, Matt Welsh, and David E. Culler. Tossim: accurate and scalable simulation of entire tinyos applications. In *SenSys*, pages 126–137, 2003.

[14] Philip Levis, Neil Patel, Scott Shenker, and David Culler. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *In Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*, pages 15–28, 2004.

[15] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. ACM International. Workshop on Wireless Sensor Networks, 2002.

[16] 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.

[17] Pedro J. Marrón, Andreas Lachenmann, Daniel Minder, Jörg Hähner, Robert Sauter, and Kurt Rothermel. Tinycubus: A flexible and adaptive framework for sensor networks. In *Proceedings of the Second European Workshop on Wireless Sensor Networks (EWSN 2005)*, pages 278–289, January 2005.

[18] Ruud Riem-Vis. Cold chain management using an ultra low power wireless sensor network. In *SenSys. Baltimore, USA*, November 2004.

[19] Igor Talzi, Andreas Hasler, Stephan Gruber, and Christian F. Tschudin. Permasense: investigating permafrost with a wsn in the swiss alps. In *EmNets*, pages 8–12, 2007.

[20] Voigt. Adam dunkels, bjorn gronvall, thiemo voigt. *Annual IEEE International Conference on Local Computer Networks*, pages 455–462, 16-18 Nov. 2004.

[21] Geoffrey Werner-Allen, Konrad Lorincz, Matt Welsh, Omar Marcillo, Jeff Johnson, Mario Ruiz, and Jonathan Lees. Deploying a wireless sensor network on an active volcano. *IEEE Internet Computing*, 10(2):18–25, March 2006.

[22] Alec Woo, Terence Tong, and David Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 14–27, New York, NY, USA, 2003. ACM Press.

[23] Pei Zhang, Christopher Sadler, Stephen Lyon, and Margaret Martonosi. Hardware Design Experiences in ZebraNet. In *Proceedings of the 2nd international conference on Embedded networked sensor systems (Sensys)*, Baltimore, MD, November 2004. ACM.