SENTRY-BASED SCHEME: TOWARD LONG-LIVED, ROBUST WIRELESS SENSOR NETWORKS

by

SHENGCHAO YU

A Dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Yanyong Zhang

and approved by

_____

_____

_____

_____

New Brunswick, New Jersey

January, 2012

ABSTRACT OF THE DISSERTATION

SENTRY-BASED SCHEME: TOWARD LONG-LIVED, ROBUST WIRELESS SENSOR NETWORKS

By SHENGCHAO YU

Dissertation Director:
Yanyong Zhang

Wireless sensor network has been gaining ground in applications involving remote surveillance and date collection. However, there are still barriers to overcome to deploy these applications in large scale, one of which is the limited network lifetime. Extending network lifetime is challenging in that wireless sensor networks are built out of error-prone and short-lived wireless sensor nodes. In order to extend the network lifetime beyond an individual node's lifetime, a common practice is to deploy a large array of sensor nodes, and have only a minimal set of nodes active performing duties while others stay in sleep mode to conserve energy. With this rationale, random node failures, either in active nodes or in redundant nodes, can seriously disrupt system operations. To address this issue, we proposed two node scheduling schemes to meet requirements from different applications: *R-Sentry*, a gang-based scheduling algorithm that attempts to bound the service loss time stemming from random node failures by coordinating the schedules among redundant nodes; and *P-Sentry*, a light-weight algorithm that keeps a certain number of redundant nodes always active while the others in deep sleep in an attempt to reduce the overhead caused by frequent wake-ups.

Like any other network, wireless sensor network is also subject to attacks from adversaries. The fact that, in the course of the network lifetime, only a minimum number of sensor nodes are active in *R-Sentry* leaves the system more vulnerable to attacks from malicious nodes. In the second part of the dissertation, we present two variants of *R-Sentry*: *Pre-emptive R-Sentry* and *Random Scheduling R-Sentry*, to address scheduling-related attacks on *R-Sentry*.

Acknowledgements

This dissertation would not have been possible without help and support from many people. Here I want to express my appreciation and thankfulness to all the people who have helped me along the way.

Particularly, I like to thank my defense committee for their time and consideration, and additionally, my advisor, professor Yanyong Zhang, for her guidance, and professor Yingying Chen for her encouragement. It's been a blessing to be around people as helpful as them. I also want to thank the fellow students in WINLAB I've worked with and spent time with. Their bright minds are always inspiring and sometimes reflective.

Finally, I want to thank my wife and my parents for their encouragement. Their unconditional support is the ultimate source of my strength and determination in completing the dissertation.

Dedication

To my grandparents and my parents ...

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Recent advances in wireless communication, micro-chip industry and embedded system design have enabled the development of relatively low-cost yet capable wireless micro-sensors that are small-sized and have short communication range. These sensor nodes, typically composed of micro-processor, radio transceiver, sensor board, memory chips and power source, form a peer-to-peer ad-hoc network, and collaborate among each other, providing services in the unit of a network. After the deployment, sensor nodes start continuously collecting data on the environment, processing the data if necessary, and forwarding the data to base stations that interface with the rest of the world.

This kind of network is by convention called *wireless sensor network* and is different from conventional ad-hoc wireless network in that [2]:

- The number of sensor nodes in a sensor network can be several orders of magnitude higher than in a traditional ad-hoc network.
- Sensor nodes are densely deployed.
- Sensor nodes are prone to pre-mature failure.
- The topology of a sensor network changes frequently.
- A sensor node mainly uses broadcast communication paradigm whereas most ad-hoc networks are based on point-to-point communications.
- Sensor nodes are limited in power, computational capacity, and memory.
- A sensor node may not have global identification.

Wireless sensor networks have been changing the way in which we interact with the physical world: instead of pulling data as a response to interrupts, sensor networks stream data to preconfigured analytical unit or decision-making unit, allowing necessary parsing and analysis to take place

before interrupts occur. With different types of sensors, we are able to collect data on various ambient conditions without being close to the spots of interest; and wireless networking makes the transportation of the collected data accessible remotely with little delay. These characteristics make wireless sensor network an attractive platform for pervasive computing and bring on a new class of applications involving event surveillance and data collection.

1.1 Wireless Sensor Network Applications

Sensor networks may integrate different types of sensors to collect different types of data, like temperature, humidity, sound, and etc.; and the collected data could be used for information purpose, further analysis, and even local control of remote actuators. Here we briefly look at some typical wireless sensor network applications.

Habitat monitoring

Wireless sensor network has its own advantages in habitat monitoring. The physical size of wireless sensor nodes can be so small that they could be deeply embedded into the environment and closely sense and measure the environment, which largely increases the fidelity of the individual measurement, thanks to the geographic proximity. Further, because of the low cost of a sensor node, we could deploy the sensors in batch mode without careful planning for precise deployment of sensors and recycling the nodes afterwards, which considerably reduces the operation cost. The following are two of the wireless sensor network applications in habitat monitoring.

- Forest fire detection: large number of thermal sensors could be randomly yet strategically spread over the forest by a plane, and form a wireless network within a small time window in an ad-hoc fashion. Periodically, sensor nodes sample ambient temperature. Whenever a fire happens, an extremely high temperature would be detected by the sensors near the fire spot and relevant data, like location and temperature, would be sent to the base station through the multi-hop network formed in the initial phase. The delay from the source node to the base station could be as short as less than a second, which allows timely actions and detailed planning for the rescue work.

- Intelligent environment: when we have sufficient information about the environment, it's just

natural to tune the environment to meet our needs , especially when the information are real-time and of certain accuracy. One of the example applications, intelligent light control, is described in [3]. A person feels most comfortable under a certain level of luminance in the office, while different people have different "best luminance level". By deploying a mobile wireless network of light sensors, the central light control could adjust the lights in the building separately based on the readings from the sensors to get all the people in the building their best luminance level as much as possible. [3] also talks about taking advantage of sunlight to reduce energy cost while still providing "best luminance level" with the help of wireless sensor networks.

Inventory control

Inventory control has emerged as one of the most popular and import applications of wireless sensor networks, mostly driven by the demand for a replacement of UPC bar code [4] system and the technology advance in micro-chip industry. When a wireless node is as small as a paper tag and costs as little as pennies, each item in the warehouse could have one attached. The sensor has information of the product stored on the chip and communicates with the base station through tiny-sized transponder or transceiver. Whenever an item enters into or leaves the warehouse, the base station reads the data on the sensor tag, update the database in the back end, and change the data on the sensor tag if necessary. The nice thing about this application is, all the processes happen remotely, and the location of an item could also be done through triangulation of wireless signals [5]. A more recognized name of the sensor tag in applications like inventory control is Radio Frequency IDentification, short as RFID [6]. RFID technology pushes the inventory control one step further from per-product tagging to per-item tagging.

Localization and tracking

Given the geographical closeness of the sensor to the events of interest, spatial information of the sensor is as important as the application data it collects. Location-related technologies based on wireless sensor networks have been emerging as the research hotspot lately.

- GPS-less localization: Radio signal reveals more than the data bits it carries: through Received Signal Strength Indicator (RSSI), we could have an estimation of the distance between the sender and the receiver. With this distance information, and the help from a reference

base station, of which the location is already known, the location of a sensor node could be obtained. The wireless sensor network-aided localization scheme provides a low-cost but function-rich alternative to satellite-based system like GPS (Global Positioning System). A lot of research work [7–9] has been devoted to improving the performance of the wireless sensor network-based localization. As the localization accuracy keeps going up and the cost of sensor system keeps going down, wireless sensor network-aided localization would be gaining more ground.

- Tracking: With each sensor node knowing its own location, either through GPS or through certain localization algorithm, tracking becomes a relatively easy task for wireless sensor networks: each sensor node could periodically update its location to the base station and each other; at the same time, the sensor nodes still collect data and transport it to the base station. The data, associated with spatial and temporal information, could be valuable on many occasions. ZebraNet [10] is one of the pioneering tracking applications using wireless sensor networks. In ZebraNet, each zebra under study wears a sensor node called *tracking collar*. The zebras, which appear as source nodes, and researchers, which appear as base stations, are both distributed in a large, wild area and moves around. The zebras and researches form an peer-to-peer wireless network and communicate in a ad-hoc fashion since there is no network infrastructure in the wild. One thing that is worth noting about ZebraNet is, a node has a good chance of being isolated from the network for an unpredictable period of time, because a zebra could be far away from the herd while communication range of the sensor node is limited. The issue is addressed by logging the data on the flash memory on the node and getting it transfered whenever it's within the range of other zebras; the compromise here is the relatively high latency. More details could be found in [10].

1.2 State of the Art in Wireless Sensor Networks

Wireless sensor network has been gaining ground in a lot of areas, like health care, sports, military surveillance, and etc.; and it is becoming an integral part of our daily lives. For wireless sensor networks to be widely adopted and better serve their purposes, there are some key issues we need to address. Let's take a brief look at the latest progresses in addressing these issues.

System platform

Berkeley-style mote has been a popular sensor platform in research community since its inception and has been deployed in some exploratory applications; and Crossbow Technology has been one of the pioneering suppliers of the commercial wireless sensor platform since 1990s. Following its MICA, MICA2 and MICAz series, Imote2 [11] represents their latest achievement in wireless sensor node platform. It's built around Marvell PXA271 XScale®processor and Marvell Wireless MMX DSP Coprocessor, with key components like memory of 256KB SRAM, 32MB FLASH, integrated 802.15.4 compliant TI CC2420 [12] radio and rich set of standard I/Os. It could connect with the sensor board through expansion slot to sense various ambient conditions. With all those components, the Imote2 node is sized as small as $36mm \times 48mm \times 9mm$. All the sensor platforms from Crossbow Technology run TinyOS [13] – an open-source operating system designed for embedded wireless sensor nodes. TinyOS's component-based architecture enables rapid implementation and innovation, while keeping the code size minimized to meet the requirement of stringent memory constraints.

Sun SPOT [14], short name for Sun Small Programmable Object Technology, is another noteworthy sensor platform widely adopted. It's the answer for wireless sensor networks from Sun Micro-System. A full, free-range Sun SPOT device is built by stacking a Sun SPOT processor board with a sensor board and battery. Each Sun SPOT has a 180MHz 32-bit ARM920T core processor with 512KB RAM and 4MB FLASH memory. Like Imote2, Sun SPOT processor board also has a TI CC2420 [12] radio with an integrated antenna on the board. The very feature that differentiates Sun SPOT from Berkeley-style mote is the Sun SPOT Java VM. Java VM executes directly out of FLASH memory, and all the device drivers are also written in Java. Given the user-friendliness of Java programming and huge base of Java code, Sun SPOT has its own edge over Berkeley-style mote when it comes to quick prototyping and code reuse.

Low-power listening radio

In most wireless sensor network applications, the events being monitored are discrete and sporadic, and often result in bursty traffic and low duty-cycle. In other words, most of the time, the radio listens to the idle channel without any traffic. This incurs huge energy waste, because measurements [15, 16] have shown that a radio in idle listening mode consumes the energy almost at the same rate as a radio in receiving mode. Reducing the listening power could be crucial in reducing

energy consumption across the network.

Take *TI CC1000* [17] as example, this single chip ultra low-power RF transceiver used in MICA2 mote [1] from CrossBow technology, has current draw in receiving mode at 7.4mA, but the current draw in power down mode goes as low as 0.2μA. Even though *TI CC1000* also supports a polling-based low-power listening mode with lower data rate, it still has a minimum 7.4mA current draw.

Energy-aware media access control (MAC) protocols

Energy efficiency has been a major factor that drives the design of MAC protocols in wireless sensor networks. Most recently, research efforts [18–21] in MAC protocol try to synchronize the schedules of sensor nodes involved in the communication session with the attempt for the nodes to keep the radios off for as long as possible to reduce energy consumption in idle listening, and minimize energy consumption caused by packet collisions. The trade-off for better energy efficiency in this category of MAC protocols is end-to-end delivery latency and scheduling fairness among the participating nodes.

Routing protocols

Like in MAC protocols, energy efficiency is also a major concern in the design of routing protocols for wireless sensor networks. The majority of routing protocols in ad-hoc networks are flooding-based, and there is no exception for ad-hoc wireless sensor networks [22]. The major problem with the flooding-based protocols is, it brings on significant amount of packet traffic, more than what is actually needed. To reduce the traffic and thus the energy consumption in packet transmission, we need to get less nodes involved in the flooding process. Some routing protocols achieve this goal by forming clusters [23–25] among nodes to have only cluster heads participate in the flooding; some by utilizing geographic information [26–29] to reduce redundant nodes from the same geographic area; and some by adjusting radio transmission ranges [30, 31] to have optimal topology for the network.

Other important aspects that need to be addressed include *scalability*, *in-network processing* [32], *congestion control* [33–35], *security* [36–39] and etc. [2] gives a detailed survey on applications, design, communication architecture of Wireless Sensor Networks. Our thesis work focuses on how to achieve robustness and energy-efficiency in the wireless sensor networks even at the presence of random sensor failures.

1.3 Problem Statement

Wireless sensor network applications, though diverse in nature, share two common building blocks: (i) data collection and delivery component; and (ii) data analysis and actuation component. The former is realized through networked sensor nodes, while the latter often resides on facilities with abundant computing resources, like the base station. As more and more applications adopt wireless sensor networks [10,40–44], it becomes critically important to ensure that the sensor network will be able to deliver as much spatio-temporal information as possible, i.e. sensor network must guarantee both coverage (thorough data collection) and connectivity (reliable data delivery) over a significant period of time. Although initial solutions have been proposed to provide continuous coverage and connectivity, there is a severe problem–the frequent failings of sensor nodes–that has been ignored.

There are several reasons for the short lifetime of sensor nodes. First, many sensor applications require these devices to operate on batteries, which typically have a maximum longevity of a couple of years; second, sensor network are often deployed in open and harsh environments that are ofter subject to extreme environmental conditions and stay unattended after deployment. These factors could quickly wear out sensor nodes, and make them fail prematurely; third, in order to cut down energy consumption and reduce size, which are crucial to many applications, most of today's sensor nodes are very densely built. Higher chip integration densities and lower voltages (to reduce power consumption) cause the circuits to be more susceptible to bit flips [45]. Regardless of how these failures happen, they can disrupt the operations of the entire network: either there will be areas within the network field that are left unmonitored, or the data can not be delivered to data sinks. As many applications depend on the uninterrupted delivery of data to draw meaningful conclusions or to take timely actions, failures to maintain coverage and connectivity will put the very motivation of deploying the sensor network at risk.

To minimize the negative impact caused by premature failures of individual nodes, we normally deploy in the target area much more sensors than actually needed when nodes don't fail prematurely. As long as the number of the working nodes in the network is above certain threshold, the network as a whole has a good chance of providing smooth coverage and connectivity to the application. However, this method limits the network's function time to the lifetime of the individual sensors. To extend the network lifetime beyond the lifetime of short-lived sensors, one common practice is to

keep only minimum number of nodes actively functioning at any moment, while keeping the other nodes, which appears redundant here, in sleep mode with radio turned off. When the current set of active nodes could not fulfill the duty for whatever reason, some of the redundant nodes would turn active and start performing. This process goes on until we run out of nodes, and could prolong the network lifetime by n-fold, where n is theoretically only bounded by the redundancy degree of the nodes. Having talked about the benefit of the prolonged network lifetime, this shift-based scheme however leaves the network more fragile to random node failures, because any failures from the active nodes would seriously disrupt the coverage and connectivity of the network. Here we are facing a dilemma: longer network lifetime contradicts with good quality of service. Most previous work [27, 46–49] fails to address this dilemma, mainly because they didn't take into account the random node failures in the design. The main contributions of the thesis are:

- **P**ersistent **Sentry**-based node protection (short as *P-Sentry*): Getting the active node sentry nodes to protect the network against the active node's failure just appears natural for a robust sensor network system. *P-Sentry* scheme intends to provide each active node with a certain number of nodes that constantly monitor the healthiness of the active node. Whenever the target active node dies, its sentries could take over to perform the duty. This protection mechanism effectively maintains the network service at certain level even when the nodes could die before its battery power drains out.

- **R**otary **Sentry**-based node scheduling (short as *R-Sentry*): Timely recovery of network service is as important as extended network lifetime in a fault-tolerant wireless sensor network. *R-Sentry* goes one-step further than *P-Sentry* by sleeping the sentries to conserve more energy. The sentry nodes wake up at proper times to make sure any failure to the target active node could be caught in time, with the coordination from the target active node. We also introduce the concept of *gang* to make sure the sentries are able to fully cover the active node.

- R-Sentry with **pre-emptive** scheduling policy : *R-Sentry* is designed to address random failings of sensor nodes. However when sensor module is the only component that fails on a failing sensor, regular *R-Sentry* shows its vulnerability. The situation could be even worse when some of the sensor nodes intentionally stop reporting sensor readings. *Pre-emptive R-Sentry* aims at minimizing the aforementioned failing sensors by setting an quota on the

continuous active time for each node.

- **Random Scheduling** R-Sentry : attacks like not sending sensor readings are passive, in the sense that malicious nodes don't change wake-up schedules of redundant nodes. When more and more active nodes start sending tampered schedules to redundant nodes, redundant nodes' schedules get distorted and *R-Sentry*'s performance deteriorates. To tackle this type of proactive attacks, We propose *Random Scheduling R-Sentry*, which introduces randomness into *R-Sentry*.

The rest of the thesis is organized as follows. Chapter 2 gives a background introduction and reviews previous research efforts in energy-efficiency and fault-tolerance in Wireless Sensor Networks. Chapter 3 presents the network model and the assumptions we use in our thesis. Following that, we discuss the *P-Sentry* and *R-Sentry* in Chapter 4 and Chapter 5 respectively, and the simulation results are presented in Chapter 6. *Pre-emptive R-Sentry* and *Random Scheduling R-Sentry* are then discussed in Chapter 7. We conclude the thesis with a summary in Chapter 8.

Chapter 2

Background and Related Work

2.1 Background

One-time dense deployment of sensor nodes and self-organizing afterwards has been one of the characteristics of the wireless sensor networks in a broad range of applications. For node scheduling schemes to work effectively in extending the network lifetime, there has to be more sensor nodes available than actually needed; and more importantly, the gain from sleeping redundant nodes has to be significantly large to justify the methodology of shift-based scheduling.

	Currents	
	value	units
Micro Processor (Atmega128L)		
current (full operation)	6	mA
current sleep	8	μA
Radio (CC1000)		
current in receive	8	mA
current in transmit	12	mA
current in sleep	2	μA
Logger (flash memory)		
write	15	mA
read	4	mA
sleep	2	μA
Sensor Board		
current (full operation)	5	mA
current sleep	5	μA

Table 2.1: Power profile of MICA2 mote [1]

The majority of the on-going research in low-power sensor network, including our thesis work, assumes that a node in sleep mode put the radio, the micro-processor and all the other components in power down mode except a wake-up timer. When the timer fires at the scheduled time, the radio and the micro-processor are woken up. If we look at the power profile of MICA2 mote in table 2.1, we'll find that, for every major component on MICA2, the current draw in active mode is about three

orders of magnitude higher than in sleep mode. The same thing could also be said about the latest generation of mote–Imote2, where the current draw is 390μA in deep sleep mode while 66mA in active mode, according to its official data sheet [11]. This observation has led to a series of research efforts in turning redundant nodes off to conserve energy, hoping for a longer network lifetime.

The key issue previous works try to address is, how long a redundant should sleep to avoid the degrading of network service due to nodes oversleeping. Most of the previous work either fail to take into account the pre-mature failing of sensor nodes, or couldn't recover the network service in time. In the rest of the chapter, we will review some of the related work in the area of energy-efficiency and fault-tolerance in wireless sensor networks.

2.2 Literature Review

The quality of service (short fo QoS) in the presence of random node failures serves as an indicator of the capability of fault-tolerance. A wireless sensor network has two major responsibilities: networking and sensing, which are respectively measured by connectivity and coverage when it comes to the network QoS.

Connectivity

Fault-tolerant networking/routing has been studied extensively under the context of wireless networks. Two broad classes of solutions have been proposed for this purpose. The first class employ multiple routing paths or alternative routing paths to achieve resilience against node failures, such as the techniques proposed in [22, 50–55]. These techniques usually assume that the underlying networks have enough active nodes to dynamically form alternative routing paths. In the scenarios where there are not enough active nodes, the second class of techniques could be employed to adapt the transmission power of each node to control the network topology [56–62].

Local network repair has been studied under the context of sensor networks, and several strategies have been proposed, such as GAF [26], AFECA [63], and ASCENT [64]. GAF [26] identifies those nodes that are equivalent in terms of routing functionality, and then turns them off to conserve energy. In AFECA [63], a simple adaptive scheme is used to determine the node sleep interval for those nodes that are turned off. ASCENT [64] shares the same goal as GAF: it has similar rationale of making redundant nodes sleep, turn them back on and join the routing when necessary later. In

order not to degrade connectivity significantly, sleeping nodes need to wake up relatively frequently. In addition, a fixed sleep interval is used for every node in the network. The authors also proved that the upper bound of the improvement on the lifetime is the ratio of a node's sleep interval over its awake time.

Coverage

Besides information delivery, data collection is the other critical task of sensor networks. Though many studies have been conducted on specifying coverage models [27, 48, 65–74] or selecting a minimum set of nodes to satisfy these models [44, 46, 48, 49], few of them focused on making the resulting systems failure resilient. In PEAS [46], a random probing technique was proposed for redundant nodes to check whether there is an active node within its vicinity.

In the context of fault-tolerant wireless sensor networks, energy efficiency, connectivity, and coverage have to be taken into account together to have a realistic solution in real-world network settings. In the rest of the chapter, we will review some of the recent research works in network QoS (connectivity and coverage), energy conservation, and fault-tolerance in wireless sensor networks.

2.2.1 ASCENT: Adaptive Self-Configuring sEnsor Networks Topologies

ASCENT [64] intends to keep the node density at an appropriate level to avoid disconnected link between the source and the sink, and the excessive collisions due to overcrowding.

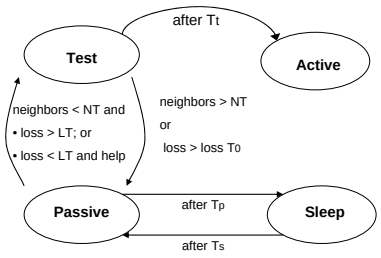


Figure 2.1: State transitions in ASCENT

In ASCENT, a node could be in one of the four states:sleep, passive, test, active, as illustrated in figure 2.1. A node in sleep state becomes passive after T_s time units. The node in passive state doesn't transmit, only passively listens for T_p time units to gather information on the state of the network without causing interference. Based upon the state information, which includes number of working neighbors (nodes that are in test or active state) and data loss rate, the passive node

decides its next state. If before T_p expires, the number of working neighbors is less than neighbor threshold (NT), and either the data loss (DL) rate is higher than the loss threshold (LT) or DL is below the LT but the node received a help message from an active neighbor, it switch to test state. Otherwise, it goes back to sleep state. If the node transitions into test state, it continues gathering state information for T_t time units. Then, the node in test state makes a decision to become active or go back to passive state, again based on the network conditions. In addition, this transition process is aided by the sink node that sends out help message to signal passive nodes to join the network when it experiences high message loss rate. Every node other than the source and the sink goes through the state transition process until it runs out of energy.

ASCENT is a connectivity-oriented self-organizing scheme and provides a number of system parameters for system tune-up; but it has its own drawbacks:

1. The traffic pattern has to be known to all the nodes; otherwise the node is not able to count the data loss rate, which is vital to ASCENT.

2. That fact that a sleep node has to go through two states and each state transition requires information collection for certain amount of time makes ASCENT inertial to system dynamics. Though the time window for information gathering could be reduced to cater for the dynamics, it is still not dynamically adjustable and could cause state thrashing because of small system fluctuation, which could possibly leave the nodes in either test or passive state most of the time.

3. Network coverage is not taken into account in the design at all.

2.2.2 PEAS: A Robust Energy Conserving Protocol

PEAS [46] is a light-weight yet effective protocol compared to ASCENT. In PEAS, a node in sleep mode wakes up and probes for active nodes within a certain range; if no active node replies for the probing, the node becomes active; otherwise, it goes back to sleep for a randomized period of time. Unlike in ASCENT, the collection of state information is not necessary and the time in probing state is extremely short.

PEAS makes sure a redundant node becomes active to fill the service hole once it detects there is a hole within its probing range; and more importantly, PEAS tries to keep the aggregate probing

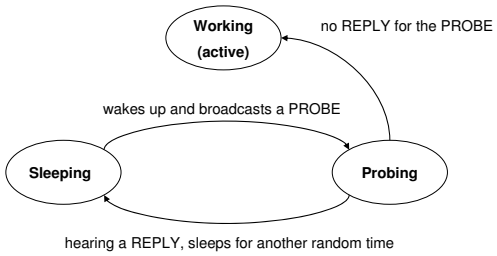


Figure 2.2: State transitions in PEAS

rate $\bar{\lambda}$ of all the sleeping neighbors of each working node at about a desired rate λ_d and distribute the wake-ups evenly over time as much as possible. In other words, roughly every $1/\lambda_d$ seconds, one of the active node's redundant nodes would wake up to probe. This makes sure that the failure of the working node could be detected and the ensuing service hole could be filled within a short time.

To achieve the desired probing rate, every time an active node receives a probing message, it replies with λ_d and the actual probing rate $\widehat{\lambda}$ that it has observed, based on which the probing node calculates its new probing rate as $\lambda^{new} = \lambda^{old}\frac{\lambda_d}{\widehat{\lambda}}$; then the probing node generates its new sleep interval t_s, following the probability density function $f(t_s) = \lambda^{new}e^{-\lambda^{new}t_s}$. The rationale behind the above algorithm is, the exponentially distributed intervals between successive wake-ups observe a Poisson process of wake-up events, thus the probings from different sleeping neighbors still construct a Poisson process. In this process, the authors use the actual probing rate $\widehat{\lambda}$ to estimate the aggregate probing rate $\bar{\lambda}$.

In addition, PEAS uses probing range to control the number of active nodes on duty. The shorter the probing range, the more the active nodes, since the probing range decides how far any two working nodes are separated.

PEAS requires little state bookkeeping and doesn't depend on any deterministic conditions; random sensor failure theoretically couldn't stop PEAS from functioning as designed. However, in experiments PEAS doesn't perform as good as it is designed to do, mainly because of the inaccurate approximation of the theoretical value from the measured value in probing rate:

1. The desired probing rate is unlikely to be achieved, because the measured probing rate is not always an accurate estimate of the actual aggregate probing rate. It takes a large number

of samples to observe a statistical property in reality, which is unlikely to happen in a dynamic environment like the sensor network with relatively frequent node failing and topology change.

2. Even the desired probing rate is achieved, still one node is usually not enough to recover the service hole left by a working node's death. PEAS fails to address the issue specifically.

2.2.3 Differentiated Surveillance

In [44, 49], Differentiated Surveillance, DiffSurv for short, takes a round-based approach, i.e. the time is divided into fixed-length slots and each node is scheduled to work for a certain amount of time in each slot. DiffSurv tries to schedule the node only once - during the initialization period of the deployment, and does little adjustment in response to network dynamics.

Specifically, the tuple of $(T, Ref, T_{front}, T_{end})$ makes a node's schedule, where T is the length of each round, Ref is a random time reference point within $[0, T)$, T_front is the duration of active time before the reference point Ref, and T_end is the duration of active time after the reference point. Two things that need to be pointed out about the tuple are:

1. The tuple is determined during the initialization period and doesn't change during the whole working phase, unless a rescheduling is required for fault-tolerance purpose.

2. Time duration T_{front} and T_{end} should be in the interval of $[0, T)$, and the sum of T_{front} and T_{end} should be less than or equal to T, which is obvious since that a node can't be active more than the duration of the round within any specific round.

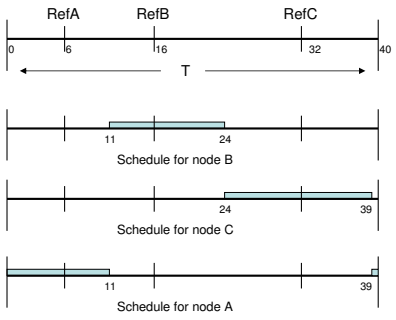


Figure 2.3: Schedule calculation for a single grid point

During the initialization period, each node randomly picks a reference point Ref among $[0, T)$ and broadcasts a beacon message with its location and Ref included. After each of the nodes has received the beacon message from all of their neighbors, a node would know each of its neighbors' coverage, which is represented as the grid points covered under grid coverage model 3. For each grid point, a node would sort the reference points of itself and its neighbor nodes in ascending order and then calculate its T_{front} and T_{end} for that specific grid point. Figure 2.3 shows how node A, B and C calculate their schedules to have grid point X always covered, where grid point X is covered by the three and only these three nodes. The shaded segment represents the active time for the node. For node B, the calculation is relatively straightforward: $Tb_{front} = (RefB - RefA)/2$ and $Tb_{end} = (RefC - RefA)/2$. Following the same rule, there are $Ta_{end} = (RefB - RefA)/2$ and $Tc_{front} = (RefC - RefA)/2$. For nodes that has the smallest and the biggest reference point, node A and C respectively in the example, Ta_{front} is calculated as $(T + RefA - RefC)/2$ and $Tc_{end} = (T + RefA - RefC)/2$. Once the tuple is determined, the schedule of the node is set and no more scheduling is required. The same schedule will be followed in each round.

This scheduling algorithm makes sure the grid point of interest is always covered by a active node in the course of the whole working phase and is optimal in terms of energy conservation. Each node runs the same scheduling algorithm for each grid point it covers and then integrates the resulted schedules into one schedule. DiffSurv also gives an option of having differentiated surveillance that enables multiple nodes cover each grid point. With this round-based deterministic approach, differentiated service could be easily achieved without incurring much extra overhead in communication and computation.

The DiffSurv is essentially designed for relatively stable networks. On the occasions of frequent random node failures, DiffSurv suffers from prohibitive overhead, because it has to reschedule all the nodes to guarantee coverage if any of the node dies, and rescheduling requires all the nodes to be up and a new round of initialization. A wireless sensor network is usually deployed in a highly dynamic environment, a deterministic approach is unlikely to handle the network dynamics efficiently.

2.2.4 Geography-informed Energy Conservation

[26] is another node scheduling algorithm for ad-hoc wireless networks that introduce a geographical adaptive fidelity (GAF) algorithm to conserve energy without sacrificing the network connectivity.

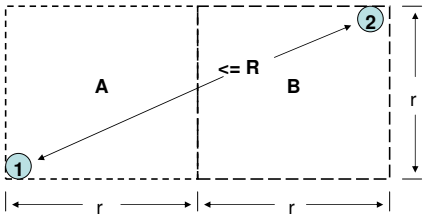


Figure 2.4: Virtual grids in GAF

The basic idea of GAF is to group nodes that are equivalent in terms of communication and only keep one node from each group active. Since GAF assumes the availability of each node's location, it divides the while network field into virtual grids such that all the nodes within one virtual grid could directly communicate with each other, and any node in a virtual grid could directly communicate with any node in the adjacent virtual grids. This virtually guarantees the routing equivalence among the nodes within one grid. As long as there is at least one node active in each virtual grid, the whole network is connected. Here the size of the grid, r, is derived based on the node's radio range, R. Take the two adjacent virtual grids as example in figure 2.4, if we pick the grid size so that the node 1 located at the bottom left of grid A is able to communicate with the node 2 located at top right of grid B, the equivalence is achieved; so there should be:

$$r^2 + (2r)^2 \leq R^2 \tag{2.1}$$

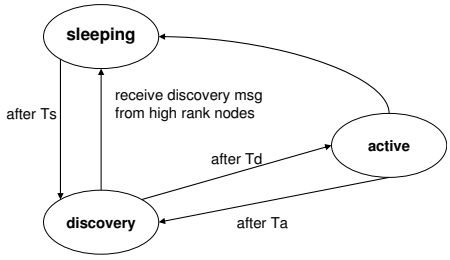


Figure 2.5: State transitions in GAF

After the virtual grids is set up, the nodes within the same virtual grid would negotiate to elect one active node routing. In GAF, nodes could be in one of the three states: sleeping, discovery, and

active. Figure 2.5 is the state transition flow in GAF. When a node enters into discovery state, it starts a timer for T_d. If the node has not received any discovery message from other nodes in the grid, it broadcasts its own discovery message and becomes active; otherwise it goes into sleeping state. Nodes in active or discovery states could also transit to sleeping state directory if it finds out other equivalent node will be active serving the network.

GAF targets at proving an energy-efficient routing protocol and leaves good network coverage as a by-product of good connectivity, which is not always the case in sensor networks. Also, GAF fails to address the impact of random sensor failures, which is critical in wireless sensor networks.

2.2.5 SPAN

SPAN [27] is another distributed node scheduling algorithm for ad-hoc wireless network that aims to preserve the network connectivity with prolonged network lifetime. In SPAN, active nodes, denoted as *coordinators*, form a network *forwarding backbone*, which roughly makes sure every populated radio range in the entire network contains at least one coordinator.

Every node in SPAN periodically broadcasts HELLO message that contain the node's status (coordinator or non-coordinator), its current coordinator, and it's current neighbors. The node in SPAN independently makes a decision if it should becomes coordinator or not based on the local information it collects from HELLO messages. The eligibility rule to be a coordinator is, two of its neighbors can not reach each other either directly or via one or two coordinators. The nodes in SPAN switch state every now and then between coordinator (i.e. active mode) and non-coordinator (i.e. sleep mode). Each coordinator periodically checks if it should withdraw as a coordinator. If it detects it's not eligible as a coordinator, it becomes non-coordinator. Also, to rotate the coordinating duty among all the nodes, a coordinator could voluntarily mark itself as a tentative coordinator after it serves coordinator for a certain period. A tentative coordinator still forwards packets, but it marks itself as non-coordinator in its HELLO message. To have balanced remaining energy on the nodes and minimal number of coordinators in the network, SPAN adopts a tie-break rules among neighboring nodes when they send out HELLO messages announcing their status switching into coordinator: the more pairs of neighbor nodes the node could connect, and the more remaining energy the node has, the better chance the node has to be a coordinator.

SPAN tries to achieve four goals by following the aforementioned rules:

1. Sufficient coordinators to provide network connectivity.

2. Balancing the workload across all the nodes.

3. Minimizing the number of coordinators to have maximal network lifetime.

4. Realizing the algorithm in a distributed manner, only local information is used.

SPAN does show its effectiveness in preserving network capacity in packet forwarding and conserving node energy. It however leaves the network coverage out of the design, which is essential for wireless sensor networks. Random node failures is another major fact that SPAN fails to take into consideration in its design.

2.2.6 Integrated Coverage and Connectivity Configuration

[48] presents a Coverage Configuration Protocol (CCP) that could provide different degrees of coverage to meet requirements from wide range of applications; it further integrates CCP with SPAN [27] to ensure the network connectivity.

[48] did a geometric analysis of the relationship between coverage and connectivity and provided some insightful observations that could help address network coverage and connectivity under a unified framework:

1. For a convex sensor network field where each grid point is covered by at least one sensor node, which is noted as 1-covered in CCP, the network is connected if the communication range R_c is at least two times the sensing range R_s, i.e. $R_c \geq 2R_s$.

2. For a K_s-covered convex sensor network field, it is K_s connected if $R_c \geq 2R_s$. In other words, at least K_s nodes need to be removed to disconnect the network.

3. For a K_s-covered convex sensor network field, its interior connectivity is $2K_s$ if $R_c \geq 2R_s$.

The above observations lead to a K_s-coverage eligibility algorithm, which each node uses to decide its state: SLEEP, ACTIVE or LISTEN. Each node periodically enters into LISTEN state to collect beacons (HELLO messages) from its neighbor nodes and reevaluates its eligibility to determine the next state. Specific rules for state transition in CCP are:

- In **SLEEP** state, the node wakes up and turn into LISTEN state when the sleep timer T_s expires.

- In **LISTEN** state, whenever a beacon message of HELLO, WITHDRAW, or JOIN type is received, the node evaluates its eligibility. If it's eligible to be active, a join timer T_j starts, otherwise it starts T_s timer and goes to SLEEP state. Before T_j expires, if the node becomes ineligible (e.g. due to a JOIN message from a neighbor node), it cancels T_j; otherwise, it broadcasts a JOIN message and enters into ACTIVE state. When the listen timer expires, the nodes goes into sleep state.

- In **ACTIVE** state, when a node receives a HELLO message, it updates its sensing neighbor table and runs the eligibility algorithm. If it determines itself is ineligible, it starts a withdraw timer T_w. If the node becomes eligible before T_w expires, it cancel the withdraw timer. If T_w expires, the node broadcasts a WITHDRAW message and enters into SLEEP state with sleep timer turned on.

In scenarios where the condition $R_c \geq 2R_s$ doesn't hold, SPAN is incorporated into CCP to ensure the connectivity. Both SPAN and CCP have their own eligibility criteria, to guarantee bother coverage and connectivity under the condition of $R_c < 2R_s$, the eligibility rule could be simply put as:

- An inactive node will be eligible if it meets the eligibility criteria of either CCP or SPAN.

- An active node will be ineligible if it can not meet the eligibility criteria of neither CCP or SPAN.

The integrated framework from [48] address in detail the relationship between connectivity and coverage in scenarios where communication range and sensing range hold different relationships. It follows the idea of picking a subset of nodes to be active to conserve energy. Still, [48] leaves the random node failure out of the discussion. Given the $O(N^3)$ computational complexity of the CCP eligibility algorithm and the dynamics of sensor networks, it's critical to address the impact of node failures.

2.2.7 Upper Bounds on Network Lifetime Extensions

The majority of cooperative network protocols in extending the sensor network's lifetime, including the ones presented in this section, assume a cell-based sensing model, in which a sensor node has uniform communication capability in all the directions. In order to evaluate the potential and the effectiveness of those cooperative network protocols, [75] investigates the upper bounds on the network lifetime extension for the cell-based energy conservation techniques.

[75] assumes a homogeneous environment where all the nodes are uniformly and independently deployed across the network field, workload is balanced across the area, and the each node has the same sensing and transmitting capacity and initial energy. It gives the definition of network lifetime for ad-hoc networks and sensor networks, with emphasis on connectivity and coverage respectively.

Definition 1 (Ad-Hoc Networks Lifetime). Let $G(t) = (V(t), E(t))$ be the communication graph of the ad-hoc network at time t, where $V(t)$ is the set of alive nodes at time t. Assume that $G(0)$ is connected, and denote with $n(t)$ the cardinality of $V(t)$, with $n = n(0)$. The network lifetime is defined as the minimum between t_1 and t_2, where t_1 is the time it takes for the cardinality of the largest connected component of $G(t)$ to drop below $c_1 \cdot n(t)$, t_2 is the time it takes for $n(t)$ to drop below $c_2 \cdot n$, and $0 \leq c_1, c_2 \leq 1$.

Definition 2 (Sensor Networks Lifetime). Let $G(t) = (V(t), E(t))$ be the communication graph of the sensor network at time t, where $V(t)$ is the set of alive nodes at time t. Assume that $G(0)$ is connected and covers the deployment region $R = [0, l]^d$, and denote with $n(t)$ the cardinality of $V(t)$, with $n = n(0)$. The network lifetime is defined as the minimum between t_1, t_2 and t_3, where t_1 is the time it takes for the cardinality of the largest connected component of $G(t)$ to drop below $c_1 \cdot n(t)$, $t2$ is the time it takes for $n(t)$ to drop below $c_2 \cdot n$, t_3 is the time it takes for the volume covered to drop below $c_3 \cdot l^d$, and $0 \leq c_1, c_2, c_3 \leq 1$.

These generalized definitions greatly facilitate the analysis on the network lifetime for different scenarios in real world applications. Particularly for ad-hoc network, [75] presents the analysis that leads to the theorem:

> In the region $R = [0, l]^d$, for $d = 1, 2, 3$, and assume that $r^d n \geq d \cdot k_d l^d l n(l)$, where r is the communication range, d is the dimension of the region, l is the length of each

dimension, and $k_d = 2^d d^{d/2}$. If a cooperative protocol is used to alternatively turn off the "routing equivalent" nodes, then the probability $P(NL_l \geq hT) \rightarrow$ as $l \rightarrow \infty$, where NL_l is the random variable denoting network lifetime and $h = d(1-\varepsilon)ln(l)$, for any constant ε such that $0 < \varepsilon < 1$.

The theorem gives a lower bound to the lifetime of an ad-hoc network that is assumed to be asymptotically almost surely connected. [75] does extensive simulation to investigate the trade-off between node density and lifetime and the effect on the network lifetime from the connectivity requirement. The contribution of [75] is, it generalizes the Ad hoc/Sensor network scenarios regarding network lifetime and provides guidance with certain simplifications for the design of cooperative protocols like GAF [26].

The contribution of [75] lies in its guidance and insight in designing and optimizing the GAF-like cooperative protocols.

2.2.8 Bounding the Lifetime of Sensor Networks via Optimal Role Assignments

[76] takes on the issue of network lifetime optimization in sensor networks by converting the problem into a network flow problem. Given that there has been proven methodology for network flow issue that utilizes linear programming to get optimized objective value with linear constraints, obtaining upper-bound of the network lifetime becomes a tractable issue with the help of linear programming.

In the collaborative network model presented in [76], each sensor node takes a role, which is composed of one or more of the following sub-roles:

- **Sensor**: the node observes the source via a sensor, digitizes the information, post-processes it and produces data which must now be relayed to the base station.

- **Relay**: the node simply forwards the received data onward without any processing.

- **Aggregator**: the node receives two or more raw data streams and then aggregates them into a single stream.

In a role, at most one sensor sub-role can be assumed. Several relay sub-roles can be assumed only when data streams being relayed originated from distinct sensor nodes. Similarly, a role can

have several aggregation sub-roles only if those sub-roles aggregate data from distinct originating sensors. Two key concepts that the optimization framework is based upon are:

- **Role Assignment:** An assignment of roles to nodes in a network constitutes a role assignment.

- **Feasible Role Assignment (FRA).** A role assignment is termed feasible if it:

    1. Results in data being relayed from the minimum specified number of sensors to the base station, and,

    2. Has no redundancy i.e. no sub-role in any node can be deleted while still obeying the first property

Through the whole lifetime of the network, a collection of *FRA*s are deployed sequentially achieve the coverage and connectivity. The collection of *FRA* is referred to as **Feasible Collaborative Strategy** if all nodes have non-negative residual energy after its execution. The problem of optimizing lifetime now becomes maximizing the sum of each *FRA*'s lifetime in a feasible collaborative strategy. If we take a collaborative strategy as a $|F|$-tuple where i^{th} element specifies the time for which the corresponding *FRA* is sustained, we will have the basic linear programming problem in determining optimal collaborative strategy, as shown in Table 2.2.

Objective
$\max\ t = \sum_{i=1}^{\|F\|} t_i$
where t_i corresponds to the time for which FRA f_i is sustained.
Constraints
$t_j \geq 0 : 1 \leq j \leq \|F\|$
$\sum_{j=1}^{\|F\|} p(i, f_j) t_j \leq e_i : 1 \leq i \leq N$

Table 2.2: Linear program for determining optimal collaborative strategy

For scenarios with aggregating and full coverage, additional constraints are added to the basic linear programming model to reflect the new conditions.

By transforming the network lifetime issue to network flow issue, [76] greatly simplifies the analysis of a sensor network's theoretical lifetime upper-bound and introduces a new way of analyzing the sensor network's coverage and connectivity.

2.2.9 Optimal Geographical Density Control (OGDC)

OGDC [47] is another round-based node scheduling algorithm, which selects active nodes based on their geo-locations to improve network lifetime. OGDC is based on three theorems of which [47] provides with proof and are known to and assumed by a substantial percentage of works in sensor networks:

1. Assuming the network field is a convex set, the condition that radio transmission range of the sensor node is at least two times the sensing range is both necessary and sufficient to ensure that complete coverage of a convex region implies connectivity in an arbitrary network.

2. Suppose the size of a sensing disk is sufficiently smaller than that of a convex region R. If one or more disks are placed within the region R, and at least one of those disks intersect another disk, and all crossings in the region R are covered, then R is completely covered.

3. If all sensor nodes (i) completely cover a region R and (ii) have the same sensing range, then minimizing the number of working nodes is equivalent to minimizing the overlap of sensing areas of all the working nodes.

With the aforementioned theorems, OGDC starts each round with a selection process that decides what nodes should server as working nodes and what nodes should go to sleep to conserve energy. In the selection process, sensor nodes utilize several timers and performs geometric computation to pick the nodes with the optimal location among the candidate nodes as working nodes, in an attempt to reduce the number of working nodes and thus to maximize the network lifetime. After the selection phase is the steady state phase, when all the nodes remain in the same state until the start of the next round. This process goes on until there are no enough living nodes left to meet the requirement from the applications.

To certain extent, OGDC achieves (i) fairness among all the nodes in terms of a node's probability of being a working node by using random timers in selection phase; (ii) optimization of network lifetime by choosing best located nodes in terms of coverage as working nodes. However, OGDC is unlikely to perform as effectively as it's designed for the following reasons:

1. Round-based approach requires tight synchronization and little disruption within each individual round. Whenever certain working nodes die in steady state phase, there will be network

holes that couldn't be filled until start of the next round.

2. Energy saving through picking best located nodes based upon geometric computation doesn't necessarily outweigh the energy consumption in exercising the computation. The complexity of the computation is another issue that needs to be taken into account.

2.2.10 Analysis on the Redundancy of Wireless Sensor Networks

[69] has some interesting observations on the minimum and maximum number of 1-hop neighboring sensor nodes that are required to provide complete redundancy in network coverage. [69] also gives average partial redundancy and tight upper and lower bounds of the probability of complete redundancy.

Given the randomness nature of sensor nodes deployment, the analysis in [69] shows it's expensive and sometimes impossible to get a full coverage of the network. Table 2.3 gives the relationship derived from [69]'s analysis among the number of 1-hop neighbors of a sensor node, the probability of complete redundancy and the percentage of the redundant area.

# of neighbors	Prob. of complete redundancy	% of redundant area ($\geq$)
4	9.56%- 22.24%	86.24%
5	31.22%- 37.01%	91.62%
6	49.74%- 52.13%	94.90%
7	64.29%- 65.21%	96.89%
8	75.15%- 75.49%	98.11%
9	82.97%- 83.09%	98.85%
10	88.48%- 88.52%	99.30%
11	$\approx$ 92.28%	99.57%
12	$\approx$ 94.87%	99.74%
13	$\approx$ 96.62%	99.84%
14	$\approx$ 97.78%	99.90%

Table 2.3: Redundancy with different numbers of neighbors

The table shows that if we ask for a node's 90% sensing area to be covered by its neighbors, 5 neighbors are sufficient. The analytical results shed some light on the redundancy degree of sensor nodes in designing energy-efficient and fault-tolerant sensor networks .

2.2.11 LEACH

LEACH [74], short for *Low-Energy Adaptive Clustering Hierarchy*, observes that one-hop direction communication consumes more energy than multi-hop communication with *first order radio model*. Instead of each node transmitting data to the base station directly, LEACH organizes the sensor nodes in the region into clusters, each of which has a cluster head. The member nodes in a cluster cooperate in a TDMA fashion, where each node collects and transmits data to the cluster head in the allocated time slot. A cluster head doesn't simply forward the data to the base station, it instead first processes the data, then transmits the data directly to the base station.

Like most cluster-based algorithm, LEACH adopts a round-based approach to ensure that each node has an fair chance to communicate with the cluster head. Each round consists of cluster set-up phase and steady-state phase. In set-up phase, clusters are formed in a self-elected fashion and the cluster head makes schedule for each node in the cluster it manages; In steady-state phase, cluster member nodes collect and transmit data to the head while cluster head processes and transmit data to base station.

LEACH is a relatively simple clustering protocol without the need of a routing protocol; and the simulation results shows it performs fairly well in certain scenarios.

2.2.12 Flexible Power Scheduling for Sensor Networks

[77] presents *Flexible Power Scheduling* (FPS), a distributed on-demand power-management protocol that aims to reduce power consumption while supporting fluctuating demand. FPS combines coarse-grained scheduling at the routing layer to schedule radio mode (on/off) and fine-grained medium access control to provision channel access. The protocol provides local communication schedules for a multi-hop sensor network and acts only on locally acquired information. Generally, the receive slots in the local schedule of a node and its own original source demand constitute the *demand* at the node, and the transmit slots in the local schedule at a node represents the *supply* at the node. Transmit and receive slots in the local schedule allow the node to know when its neighbors are ready to transmit or to receive from the node.

A node's lifetime is divided into cycles, each of which is further divided into slots. In a two-level tree based architecture, a node can be in one of 6 states in each time slot:

1. Transmit (T) - Transmit a message to parent node, remains in schedule until topology or supply/demand changes.
2. Receive (R) - Receive a message from child node, remains in schedule until topology or supply/demand changes.
3. Advertisement (A) - Broadcast an Advertisement from parent node for an available reservation slot, remains at most one cycle.
4. Transmit Pending (TP) - Send a reservation request to parent node, remains at most one cycle.
5. Receive Pending (RP) - Receive a reservation request from child, remains at most two cycles.
6. Idle (I) - After all current demand at this node has been met, the node can power down during idle slots.

No global initialization is required. The base station initializes by picking a reservation slot at random and listens for a return message during this slot in the following cycle. Each node follows the same procedure to keep the whole network going. Nodes turn the radio off in *Idle* state to conserve energy. FPS is a TDMA-like protocol that trades latency for energy efficiency. The length of a cycle is a key parameter that determines the trade-off between communication latency and energy saving from *Idle* state.

2.2.13 Co-Grid

Co-Grid [66] presents a coverage maintenance protocol for sensor networks that aims to reduce network reconfiguration time upon node failures thus to have better network coverage and connectivity, by improving accuracy of detections of node failure and coverage loss. It defines coverage of a sensor network based on a probabilistic detection model. A point P is covered by a sensor network if the probability that a target, located at P, is detected by the active nodes is above threshold β and the system false alarm rate is below threshold α. A geographic region is deemed as covered if all the points in this region are covered.

[66] investigated three different versions of maintenance protocols with different efficiencies:

Centralized Version In the *Central* protocol, one node is elected among all nodes in the region A to serve as the fusion center. In the coverage configuration phase, the fusion center decides

which nodes should remain active, computing their local false alarm rate to ensure the coverage requirement is met. Initially, all nodes are marked as sleep by the fusion center. In each iteration of the algorithm, a node is marked as active. Given the system false alarm rate threshold and the number of current active nodes, the fusion center derives a local false alarm rate for active nodes. Using active nodes' locations and local false alarm rates, the fusion center finds the location (x_{min}, y_{min}) in region A that has the minimal detection probability P_{Dmin}. If P_{Dmin} is less than β, the fusion center finds the node closest to point (x_{min}, y_{min}) among all sleeping nodes and marks it as active. This process repeats until the minimal detection probability P_{Dmin} in region A is greater than β. Then, the fusion center sends a list of active node IDs and the local false rate it computed to all the nodes in region A. If a node finds its ID in the list, it remains active and sets its decision threshold based on the local false alarm rate; otherwise it goes to sleep and wakes up periodically to check whether it should activate itself by listening to messages from the fusion center.

Se-Grid The centralized version's weakness is obvious: 1) put all the scheduling duty on a single node; 2) long configuration time. In Se-Grid, the whole region is divided into a matrix of identical grids and each grid works separately in a centralized fashion. Se-Grid distributes the scheduling duty across the region and effectively reduces the configuration time, but at the expense of quality of decision-making. This is because Se-Grid restricts decision fusion within each grid, a node can not contribute to the decision fusion of a neighboring group even if it is geographically close to the grid's boundary.

Co-Grid Recognizing the drawbacks of the first two versions, Co-Grid takes a centrist approach: the whole region is divided into overlapping grids, each grid is composed of four identical sub-grids and each sub-grid belongs to up to four grids; the fusion center of each grid is located at the center of the grid. In each iteration of Co-Grid, similar to Se-Grid, a fusion center computes the detection probability of each sample point in a grid and activates the node closest to the sample point with the minimal detection probability, until the minimal detection probability in the grid is above threshold β.

Like most work in large-scale sensor networks, [66] divides the square network field into smaller square grids, and nodes in each grid form a cluster. One thing that differentiates [66] from previous

work is that, Co-Grid is designed based on a distributed detection model that takes into account data fusion among multiple nodes, while others are based on less sophisticated detection models.

2.2.14 Coverage and Connectivity in Sensor Networks with Adjustable Ranges

[78] extends work proposed in [47], which tries putting the center points of the three closest nodes at such positions that th three points form an equilateral triangle with side length $\sqrt{3}r_s$, where r_s is the radius of the disks. [47]'s assumption of identical sensing range of each node inevitably leads to excessive coverage overlapping, a by-product of pursuing full network coverage. [78] relaxes the condition of uniform sensing range to reduce the overlapping to have better energy efficiency.

Coverage with two adjustable sensing ranges (Model II) In this scenario, the nodes are placed in the following manner:

- Cover the area with non-overlapping large disks such that each disk "touches" six disks. The touching point is called "crossing".
- The area enclosed by three adjacent disks is not covered by the large disks and will be covered with a smaller disk. That is, three crossings are on the circumference of the smaller disk.

Coverage with three adjustable sensing ranges (Model III) To further reduce coverage overlapping, [78] proposes using three different sensing ranges:

- Cover the area with non-overlapping large disks such that each disk "touches" six disks.
- The area enclosed by three adjacent disks is uncovered. Embed a small disk in the area so that it touches all three large disks. Three new uncovered areas are generated, which are covered by three medium disks.

Simulation results from [78] shows Model II has better coverage ratio than Model I (original work from [47]) and Model III, especially when node density is low or sensing range is small; while Model III has better energy efficiency.

Chapter 3

Sensor Network Model

Different applications require different types of sensors installed on the sensor node, such as thermal, acoustic, magnetic and motion that are used to monitor environmental conditions like temperature, noise level, electro-magnetic level and movement respectively. One algorithm or protocol could hardly be universally applicable to the wide spectrum of applications in wireless sensor networks. In this chapter, we first look at the category of applications that benefit most from our research; then we present in detail the sensor network model that is used in our study; lastly, we fast forward to the future to look at how our work will still be effective as the hardware/software technology advances and application requirements change.

3.1 Target Sensor Network Applications

As [2] has discussed, wireless sensor networks have been deployed in fields like military, environmental control, health care and intelligent home, etc. As much as the sensor networks in different applications share the characteristics of *wireless communication*, *sensing capability*, *limited computational capacities and memory*, *low-power* and *low cost*, they have their own uniqueness to meet different requirements. For example, military applications require constant monitoring and real-time response from the network, should something happen in the area of interest; environmental applications need the network to work for a significant length of time (as long as more than a year) without intervention from human being.

3.1.1 Application attributes

Our scheme is designed for sensor network applications that have the following attributes:

- High redundancy: the sensor nodes deployed in the network should be at least as many as several

times number of minimal set of nodes that are required to be active to meet the requirement from the application. High-density, which often leads to high-redundancy in wireless sensor networks, is widely considered one of key features that differentiate wireless sensor networks from conventional wireless ad-hoc networks [2]. High-redundancy is a way wireless sensor networks offset the negative impact from premature node failures due to reasons like internal hardware/software outages and harsh working environment.

- Limited resource: the sensor node operates in low-power mode on non-rechargeable battery. When a node's battery is drained out, it's considered a dead node, not being able to communicate or sense. It's not uncommon in a sensor network application that, small-size and low-cost constrain the power a sensor node could carry, while harsh working environment makes it difficult to recharge or replace the dead battery.

- Location-awareness: each sensor node is stationary and knows its own location, which can be obtained either through localization devices, such as GPS, or through certain localization algorithms [7, 8, 79–82]. Further, the sensor node knows the geographical location of the network field in which it's deployed. Based on a node's location and sensing radius, it gets to know what area it covers.

- Full coverage: the full coverage is required for our target sensor network application. As we will discuss in Section 3.2, coverage and connectivity are two separate issues in wireless sensor networks. One doesn't guarantee the other. Full coverage means every point in the network area is in the sensing range of at least one sensor node; while connectivity dictates how well the sensor nodes in the network field are connected. For most sensor network applications, coverage is the most basic and first requirement that needs to be fulfilled in that sensing is mostly about monitoring ambient conditions.

- Random node failure: node failure happens in every type of network, but it's more likely to happen in wireless sensor networks for the following reasons: 1) loss of battery power stops a node from functioning; 2) node reliability is less secured due to low-cost manufacturing and packaging; 3) harsh working environment externally poses more harm to the device: flood, lighting and other natural events could all cause node outage.

- Statistically parameterized recovery time: different applications ask for different levels of fault tolerance and recovery speed. Our work doesn't intend to provide any kind of hard real-time recovery from service loss due to node failures. Our target applications should be tolerant of small delay in recovering from service loss. As long as the average recovery time is below a certain threshold, the application should be considered as undisrupted. Our work allows the application to specify the threshold base on its needs through protocol parameters, as is to be presented in Chapter 5.

3.1.2 Example applications

Many applications in civil, non-critical fields, as described in Section 1.1, could benefit from our research, because these applications normally don't require hard real-time reaction time to ambient events, instead place an emphasis on the sustainability of the service, as long as the quality of the service meets certain criteria. Applications that fall into this category include habitat monitoring, intelligent home, etc.

On the other hand, applications like military surveillance require continuous service, any disruption of the service could have severe consequence like loss of personnel life. Our research doesn't apply to this type of mission-critical applications.

3.2 Generic Sensor Network Model

Like most research, we need to skip non-essential details that are normally found in a real-world working system to have an abstract model, based on which we can formulate and solve the problem. In our study, we adopt a sensor network model that captures the essence of wireless sensor networks and is widely used by fellow researchers. The main characteristics of the model are as follows:

Regularity in Sensing and Communication: the popular disk sensor model [46, 48, 65] is used to simplify the analysis and simulation. Specifically, a node's sensing area is a circular disk with the node as the center and the sensing range as radius; and the neighbor nodes that fall into the co-centric circle with the transmission range as radius are considered as 1-hop neighbors. Additionally, all the communication links are symmetric, i.e. if node A's message could reach node B, node B's

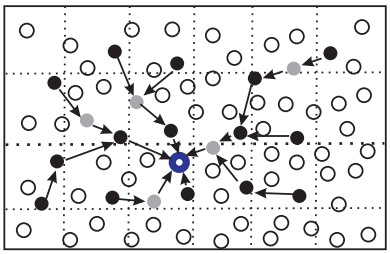


Figure 3.1: Illustration of a wireless sensor network.

message could reach node A too. We believe this abstraction of sensing capabilities and communication links reduces the complexity of the problem considerably while not hurting the legitimacy of our research, for the following reason: i) there is always a disk-shaped area that could be encompassed by the irregularly shaped sensing area; ii) asymmetric communication links normally only constitute a small percentage of all the links in a typical sensor network environment. Even there are a significant percentage of asymmetric links, as long as there are sufficient number of nodes that keep the network connected, two geographic neighbors could always talk, either directly or through multiple-hop link.

Random Sensor Deployment: [83] summarizes three commonly used deployment strategies in sensor network applications: (i) *random deployment*, (ii) *regular deployment*, and (iii) *planned deployment*. In *random deployment*, sensors nodes are evenly distributed across the field following a uniformly distribution; In *regular deployment*, sensors are placed in regular geometric topology, like gird or circle; In *planned deployment*, sensors are placed with higher density in areas where the events are concentrated.

Regular deployment is rare in reality because sensor nodes are often deployed airborne into inaccessible areas, like rural forest, in large batch. And although nodes are deployed non-uniformly in the whole region in *planned deployment*, in small regions, senors are actually distributed randomly. Without the loss of generality, we assume *random deployment* of the sensor networks in our study.

Grid-based Coverage: we adopt the grid-based coverage model [48, 66], illustrated in Figure 3.1. In this popular coverage model, the square network field of interest is imaginarily partitioned into grids, marked by the dotted lines in the figure. The grids should be small enough so that the underlying physical phenomena do not exhibit much variability within a grid. The grid points that fall within a node's sensing area are considered covered by that sensor node. This discretization

approach simplifies the measurement of the coverage, since the network coverage percentage could now be measured by the percentage of the grid points that are covered. The model can be enforced by having nodes exchange the list of grid points it covers with their neighbors, which we call *GridList*. Figure 3.1 shows such a scenario where there are a subset of sensor nodes, represented by darker solid circles, actively monitoring corresponding grid points, while other nodes are not needed for the coverage under the model. With sufficiently large node density, there is a high probability that, out of uniformly randomly deployed nodes, there exists a set of nodes that could collectively cover all the grid points in the network field.

Connectivity: in addition to sensing the physical world, a wireless sensor network is also responsible for delivering the collected data to the applications, usually located at base stations(sink nodes). Network connectivity thus requires that there exists a routing path between every actively sensing node and the sink. In order to achieve this goal, it may be necessary to have more active nodes than just those needed to provide full sensing coverage, i.e., we may need extra nodes to have good network connectivity. In our example, these nodes are represented by the light solid circles in Figure 3.1.

To focus on failure recovery schemes, we choose grid sizes that are small enough to ensure connectivity through coverage: a wireless sensor network that satisfies the coverage requirement is automatically connected. This was also observed in earlier studies [46, 48, 69], which assume the communication range is at least double of the sensing range. With proper selected sensing range and communication range, we can focus on providing sensing coverage in the presence of sensor failures.

Turning Off Redundant Nodes: energy is a scarce resource, for many sensor nodes may not have external power sources other than batteries. A sensor node has three main components: sensor(s), processor, and radio transceiver. Sensors measure physical environment, the processor takes as input the data from the sensors or the network and performs in-network processing, while the radio communicates with the rest of the network. Among the three, the radio is by far the main power consumer [18]. For example, the radio on MICA2 mote has the current draw of 12 mA in transmitting and 8 mA in receiving. It's worth noting that a radio in receiving mode does not necessarily mean the application is receiving any valid packets, it could be merely monitoring the channel. The

energy consumed by the radio on MICA2 mote in transmitting a 30-byte message is roughly equivalent to the energy consumed by an ATMega128 processor executing 1152 instructions [84]. As a result, if we turn on the radio on all the sensor nodes, then no matter how many nodes are deployed, the network can not function longer than 17 months. In practice, to extend network lifetime it is necessary to have redundant nodes turn off their radios and turn them back on only when needed. This is the approach shared by most recent researches in energy conservation in wireless sensor networks [46, 48, 69].

Chapter 4

Persistent-Sentry: A Light-weight Robust Scheme

In majority of shift-based schemes, redundant nodes need to wake up fairly frequently to evaluate the system condition in an attempt to quickly adapt to any changes in the system. However, frequent wake-ups could be wasteful, because the wake-ups and the ensuing message exchanges consume extra energy. Additionally, there are chances that node failures during redundant nodes' sleep time couldn't be detected in time. *Persistent-Sentry* aims to address this dilemma by having a select group of redundant nodes closely watch active nodes to enable longer sleep time for other redundant nodes. Since the select group of redundant nodes surround the active nodes persistently alert, we call this scheme *Persistent-Sentry*, *P-Sentry* for short.

4.1 P-Sentry Scheme

The intuitive way of minimizing damages caused by unexpected death of active nodes is to keep redundant nodes always up so that the failing active node could be replaced immediately. *P-Sentry* shares the same basic idea, except that it only keeps a small number of redundant nodes awake.

4.1.1 Basic idea

We make the simplified assumption that the sentry nodes do not fail before the active nodes do to keep the explanation easy to understand; in later sections, we will discuss how *P-Sentry* handle more realistic situations.

We let the active nodes maintain most of the data structures, while having the redundant nodes only keep track of their own next wake-up time. The most important data structures maintained by the active node are the scheduled wake-up times for its redundant nodes and the IDs of the current persistent sentry (p-sentry) nodes. For a p-sentry node, it only records minimum information about the active node it is guarding. It's worth pointing out that, like active nodes, p-sentry nodes also

have the radio circuitry turned on, but they only participate in activities that have to do with the surveillance of the active node. Hence, the p-sentry nodes have sensor module turned off and don't forward any application traffic.

Next we discuss how the active node chooses p-sentries, establishes wake-up schedules for its redundant nodes, and more importantly, when and how a redundant node turns into a p-sentry.

Initialization: The initialization process in *P-Sentry* is similar to the initialization process in *R-Sentry*, as to be described in Section 5.2.

1. *Neighborhood Discovery*. This service ensures every sensor node identifies its neighbor nodes, and populates its *sensing redundant set*, short for *SRS*, and *GridList*. The *sensing redundant set* (*SRS*) of a node consists of its neighbors whose sensing areas overlap with the node's sensing area, i.e. nodes that can cover the same grid point(s) belong to each other's *SRS*.

   At the beginning, each node in the network sends out a *presence announcement* message including its ID and location. Such messages are flooded within h hops around the source nodes, where h is a small number, usually 1 or 2, as the *presence announcement* only matters to nodes within the vicinity of the source node. After a certain amount of time in the process, every node has received the announcements from all of its *SRS* members, based on which a node can calculate the *GridList* of each node in its *SRS*.

2. *System Bootstrapping*. The initial set of active nodes are selected in a distributed fashion. Our approach is similar to the one employed in CCP [48]. The process takes place right after the *neighborhood discovery* phase, when all the nodes in the network have their radios on and each starts a random back-off timer. Within this back-off period, a node may receive announcements from its *SRS* members about whether they are redundant or not; we call the announcement *redundancy announcement*. As soon as the timer expires, the node determines whether it needs to be active as follows, first it checks the *redundancy announcements* it has received from its *SRS* members. It assumes all the *SRS* neighbors that have not sent the announcement to be active; next, it calculates whether all the grid points in its *GridList* can be covered by the *SRS* neighbors that have decided to be active. If not all the grid points are covered, the node will be active; otherwise, it broadcasts the *redundancy announcement*

message to all the *SRS* neighbors to indicate it would be redundant. Here the random back-off timer serves as tie-break between neighboring nodes. As a result, the nodes with a shorter back-off period are more likely to become active nodes. As soon as the timer on each node is fired, all the sensor nodes would know whether they will be redundant or not.

3. *Initial selection and scheduling*. After the bootstrapping phase, the active nodes follow certain criteria to pick k nodes from their respective *SRS* as its initial p-sentries and calculate wake-up times for the remaining redundant nodes. As soon as the active node has the schedule for every *SRS* member, it broadcasts the schedules to notify *SRS* nodes.

At the end of the initialization phase, the redundant nodes go to sleep with their sleep timers properly set up, the active nodes start collecting and forwarding data, while the p-sentry nodes start guarding the corresponding active nodes.

Detection and Action Other than forwarding the collected data, the active nodes also periodically broadcast heartbeat beacons as part of routing protocol. The p-sentry nodes track these beacons from the active nodes they are guarding. If a p-sentry node does not receive or overhear a beacon from one of its active nodes for a specified period, it assumes the active node has failed and become active to repair the network.

As mentioned earlier, every active node could have more than one p-sentries, whose number k is a parameter in our scheme. If k is more than 2, the failure of an active node would bring more active nodes into the system, as much as $k-1$ more. Following this trend, the number of active nodes could increase rapidly as more active nodes die, which would lead to excessive resource consumption and much shortened network lifetime. Fortunately, a p-sentry node does not necessarily need to become active when the active node dies, because that active node might overlap with its nearby active nodes. To prevent a p-sentry node from unnecessarily turning active, we impose one more condition to determine whether a p-sentry should become active: a p-sentry only becomes active if the number of active nodes in the neighborhood is less than:

$$n_{actNeigh} = \frac{n_{actAvg} \times S_{neigh}}{S_{network}} = \frac{n_{actAvg} \times \pi r^2}{S_{network}}, \tag{4.1}$$

where n_{actAvg} is the number of average active nodes, S_{neigh} is the circular area of the sentry's neighborhood and $S_{network}$ is the area of network field. n_{actAvg} could be an empirical value

learned from previous deployments. The randomness in the node deployment and the selection of active nodes statistically ensures Equation (4.1)'s effectiveness, which is further confirmed by our simulation results.

Similarly, we take advantage of the above statistical property to prevent excessive number of sentries. We make sentries nodes periodically broadcast heartbeats as well, so that they could track the peer sentries in the neighborhood. Every time before a sentry sends out a heartbeat beacon, it checks the counter of the surrounding sentries; if the counter is larger than the threshold $n_{satNeigh}$, we ask the sentry to go to sleep for the minimum lifetime of its neighbor active nodes. We determine $n_{satNeigh}$ as

$$n_{satNeigh} = \frac{n_{satAvg} \times S_{neigh}}{S_{network}} = \frac{n_{satAvg} \times \pi r^2}{S_{network}}, \tag{4.2}$$

where n_{satAvg} is the number of average p-sentry nodes and could be an empirical value too.

Probing and Reinforcing As more and more p-sentry nodes become active, there would be a dearth of p-sentries in the system. However, we need to maintain a certain number of p-sentries to make sure the active nodes are well protected. To reinforce p-sentries, it is just natural to transform sleeping redundant nodes to p-sentries when they wake up from deep sleep. When a redundant node wakes up, it broadcasts a probing message with its node ID included. About the same time, the other redundant nodes following schedules from the same active node will also wake up and probe. All the active nodes that receive a probing message will check its own list of p-sentries, if an active node currently has less than n_s p-sentries, it will take the sender of the probing message as its sentry, update the sentry list, and notify the redundant node of the decision; otherwise, it replies back with the next wake-up time to the redundant node. By doing this, within a small time window, each of the probing redundant nodes would know if it needs to turn into a sentry, or go back to deep sleep. At the end of the time window, the redundant nodes would take actions as instructed. For the purpose of clock synchronization, the active nodes embed their clocks time in the message, so that cumulative clock drift could be avoided locally.

There are two things we would like to point out: first, any active nodes that receive the probing message has the equal chance to take a redundant node as sentry, even though the redundant node was not following the wake-up schedule from them; second, a sentry could serve multiple active nodes. We will elaborate on this in Section 4.2.

4.1.2 Random failure of redundant nodes

So far, we've assumed that the redundant nodes live longer than the active nodes. The reality is, redundant nodes are just as vulnerable as active nodes to external extreme conditions and internal hardware outage. When a non-sentry redundant node fails, not much needs to be done, because neither the current capacity of the network, nor the current reliability of the network are affected. However, when a sentry dies, the current capacity of the network still doesn't change, while reliability of the network changes. The active nodes must be aware of the change in reliability, so that it would be able to regain the reliability whenever it has the chance.

In *P-Sentry*, like the active node, the sentry node also broadcasts the heartbeat signal periodically to inform the active nodes of its existence. When an active node has missed several heartbeats from a sentry node, the active node will remove the sentry node from its sentry list, thinking the sentry node has failed. It will then wait to get a new sentry when one of its redundant nodes comes back from deep sleep. With this mechanism, the failure of a sentry could be captured within a short time and the active nodes would be able to take actions promptly to minimize the adverse impact of the random failures.

4.1.3 How long a redundant node should sleep?

We have redundant nodes sleep for a significantly long to conserve energy, however, they must not oversleep to be able to contribute to the system before extended network service loss occurs. The ideal scenario is, those redundant nodes wake up when a nearby sentry has just become active and the new active node doesn't have enough sentry nodes to have themselves backed up. That said, we would like a redundant node to sleep for the shortest remaining lifetime of its neighboring active nodes, so that it would have a chance to become a sentry. If it turns out none of the active nodes in the neighborhood is dying soon or is in need of more sentries, the redundant node would set its next wake-up time based upon the latest instructions from the neighboring active nodes and go back to sleep. The active node therefore needs to have an estimation of its own remaining lifetime and reply with the estimated remaining lifetime while the redundant nodes are querying.

Our estimation methodology is based on the observation that most physical phenomena are continuous in time, therefore the data collection and packet forwarding are also continuous. As a

result, we can use the average communication rates over the past to predict the future communication rates.

Suppose a node has been active for time T and it has sent out N messages during that period; further, suppose that it has sent n messages during last t time units; then the estimated transmission rate in the next T' period is calculated as

$$s' = (1 - \alpha) \times \frac{N - n}{T - t} + \alpha \times \frac{n}{t} \tag{4.3}$$

, where α is a parameter which dictates the significance of the recent history to the future estimation.

Suppose an active node has remaining energy E', and its remaining lifetime is T'. If that node spends all the remaining time idling, then the total energy consumption is:

$$E_{idle} = p_{idle} \times T'. \tag{4.4}$$

At the same time, Equation 4.3 gives the estimated transmission rate of that node. Since packet size in sensor network applications is usually small and fixed [33], the corresponding energy consumption for sending can be calculated as

$$E_{tx} = p_{tx} \times s' \times T' \times t_{tx}, \tag{4.5}$$

, where t_{tx} is the time spent in transmitting a packet. We should emphasize that primary focus of this study is to design energy conservation protocols when the traffic volume is relatively low. For this reason, Equation 4.5 does not consider energy consumption involved in packet retransmission caused by collision.

When the traffic is low, the incoming traffic rate is equal to the outgoing traffic rate. Consequently, we can calculate the energy consumed in receiving as

$$E_{rx} = p_{rx} \times s' \times T' \times t_{rx}, \tag{4.6}$$

, where t_{rx} is the time to receive a packet.

Since the node will spend a large fraction of its lifetime being idle under the low traffic volume assumption, its total energy expenditure would be:

$$E = E_{idle} + E_{tx} + E_{rx}. \tag{4.7}$$

Therefor, the remaining lifetime of the active node is:

$$T' = \frac{E}{p_{idle} + s' \times t_{tx} \times p_{tx} + s' \times t_{rx} \times p_{rx}} \tag{4.8}$$

Finally, the active node can set the sleep time as $k_1 T'(0 < k_1 < 1)$, where k_1 is a parameter that could be tuned up based on network conditions and application requirement. It's worth noting that the estimation is intended to provide a reference value for the redundant node's sleep time.

As described above, the active node needs to collect statistics such as T, N, t, and n to estimate its remaining lifetime. If it just becomes active, it should inherit these statistics from the previous active node for which it served as a sentry. Hence, the active node and its sentry nodes should synchronize with each other periodically to share these statistics.

4.2 Sharing or Not Sharing?

The active nodes can share redundant nodes, either sentry nodes or non-sentry nodes; in other words, a sentry node could guard multiple active nodes at the same time and a non-sentry redundant node could follow schedules from multiple active nodes. On the other hand, it's also feasible that a sentry node exclusively serves an active node at any moment and a redundant node follows the schedule from just one active node. The other alternative is the mixture of two extreme cases: put a limit on the number of active nodes by which a sentry node could be shared.

The policy of sharing makes the scheme less complicated and more resource efficient while functioning comparably well, as is to be shown in Section 6. This is mainly because:

- In *P-Sentry*, the active node neither needs to maintain schedule for each of its redundant node, nor needs to adjust the schedule if any one of its redundant nodes fails. The only schedule-related information a redundant node needs to get from a neighboring active node is the active node's remaining lifetime. This makes sharing come with little extra cost.

- Under sharing policy, an active node can take as sentry any redundant node from its *SRS*, which saves memory resource and energy consumption involved in bookkeeping and information exchange. For example, when a redundant node wakes up from sleep, any active nodes in the neighborhood can take it as sentry without consulting other active nodes and the redundant node itself.

- As the system continues to run, more and more nodes would die, but we still maintain certain number of active nodes to ensure the service quality. Under this circumstance, eventually there would not be enough redundant nodes to serve as sentry nodes. The sharing policy certainly could defer reaching this point and thus lead to longer network lifetime. The edge of sharing policy here comes just natural in that it takes advantage of the fact that there are service overlapping between nodes.

4.3 Sentry Selection

There are varieties of criteria by which an active node could select n_s sentries out of its *SRS*, depending on the application requirements and network conditions. To name just a few, these criteria include: (1) the residual energy of the redundant node. On most occasions, the more energy a redundant node has left, the better chances it would live longer and be able to guard the active node; (2) the capability the nodes have in replacing the active node's service, like coverage and connectivity; (3) the location of the redundant nodes. Geographically, sentry nodes should not be too close to each other, avoiding being negatively affected at the same time by external events, like malicious attacks or natural physical phenomena.

Sticking with any one of the criteria doesn't come free. Some criterion takes considerable amount of computation and message exchanges to examine, while some criterion even conflicts with other criteria; for instance, criterion (2) and (3) mentioned above contradict each other. The point we are trying to make here is, *P-Sentry* doesn't restrict the system's choice of criteria; instead, it provides with a mechanism by which the selection criteria could be applied to meet the application's needs. In *P-Sentry*, the active node is given the chance in initialization period to sort the redundant nodes in its *SRS* based on the selected criterion and select the first n_s nodes from the sorted list of *SRS*. Later on, when a redundant becomes a sentry the neighboring active nodes of the new sentry would check its sentry list and update it if necessary.

Table 4.1 gives an example of how active nodes adjust their sentry list using aforementioned criterion (1) at some point after the initialization period. For the purpose of simplicity, we use uppercase to label active nodes; each active node's sentry list is decreasingly sorted by the remaining energy at the moment of probing and labeled lowercase in alphabetical order. For instance, node *e*

active node	sentries	sorted SRS
A	{e, l}	{h, e, l, m}
B	{g, i, n}	{o, g, i, u, m, n, t}
C	{i, j, n}	{h, q, k, i, j, n, p, s}
D	{i, j, l}	{l, h, k, i, n, j, m, p}

(a) Before redundant node m's probing

active node	sentries	sorted SRS
A	{e, l, m}	{h, e, l, m}
B	{g, i, m}	{o, g, i, u, m, n, t}
C	{i, j, n}	{h, q, k, i, j, n, p, s}
D	{i, j, l}	{l, h, k, i, n, j, m, p}

(b) After redundant node m's probing

Table 4.1: Illustration of sentry selection based on remaining energy, with k = 3.

has more energy left than node *l*, etc.. Table 4.1 (a) shows the sentry list and *SRS* of active nodes *A*, *B*, *C* and *D*. When redundant node *m* wakes up and probes for active nodes, it detects that node *A* needs one more sentry, so it becomes a sentry for *A* and send out an announcement message. Upon receiving the announcement, active node *B* would know the availability of *m* as a sentry and then check its sentry list; since sentry *m* has more remaining energy than *n*, *B* would replace *n* with *m* as sentry. Since node *m* is not in active node *C*'s *SRS*, *C* doesn't take *m* to replace *n*. The reason why *D* doesn't take *m* is, all of *D*'s existing sentries have more residual energy than *m* does.

One thing we like to point out is, a redundant node becomes a sentry after probing only because one of its neighboring actives nodes has less than k sentries, not because it has more remaining energy than current sentries of the active nodes. Put in the context of the example in Table 4.1, *m* would go back to sleep after probing if active node *A* already has k sentries before the probing. This is to avoid unnecessary overhead caused by frequent sentry replacing, as there is high probability that the existing sentries have less energy left.

4.4 Optimization of P-Sentry

As we've discussed earlier, radio module is the major energy consumer on a sensor node. To further extend the network lifetime, one thing we could do is, turn off the sentry's radio between its heartbeats; in other words, we make sentry nodes sleep as well, but wake them up more frequently so that failing active nodes could still be detected and replaced in time. However, the sleep of sentries changes the way the active nodes and sentries keep track of each other. Some mechanisms described in Section 4.1.1 don't work anymore under the new circumstances, specifically:

- sentry nodes would not be able to monitor and the active nodes by monitoring heartbeats from them. Since the sentries are in sleep mode most of the time, they would miss most of the heartbeat beacons from the active nodes.

- a sentry would miss heartbeat from its neighbor sentry nodes, since they are unlikely to wake up at the same time. As a result, a sentry wouldn't be able to count other sentries in the neighborhood.

To circumvent the first restriction, we make sentry nodes broadcast probing messages when they wake up from short sleep rather than passively collecting heartbeat beacons, and the active nodes must respond to probing message from the sentry nodes. Any active nodes that fail to respond is considered malfunctioning and the sentry nodes that are guarding would become active immediately.

The second restriction poses difficulties in counting peer sentries for the sentry. However, since sentries go to sleep periodically in the optimized version, there is no need to turn off sentries particularly based on the number of neighboring sentries. Therefore, sentries don't need to count its neighboring sentries anymore in the optimized version.

Chapter 5

Rotary-Sentry Scheme for Continuous Sensor Services

In shift-based energy-conservation schemes, like *PEAS* [46] and *ASCENT* [64], a wireless sensor network on duty mainly consists of two types of nodes: active nodes, which perform duties; and redundant nodes, which sleep with their radios off. While active nodes are on duty, redundant nodes shouldn't sleep continuously for an extended period of long time. Rather, they should wake up from time to time to check the conditions of the system. Every node randomly waking up, however, is wasteful and does not yield much benefit. For example, if consecutive wake-ups are far apart from each other, it could harm the chance of timely network recovery.

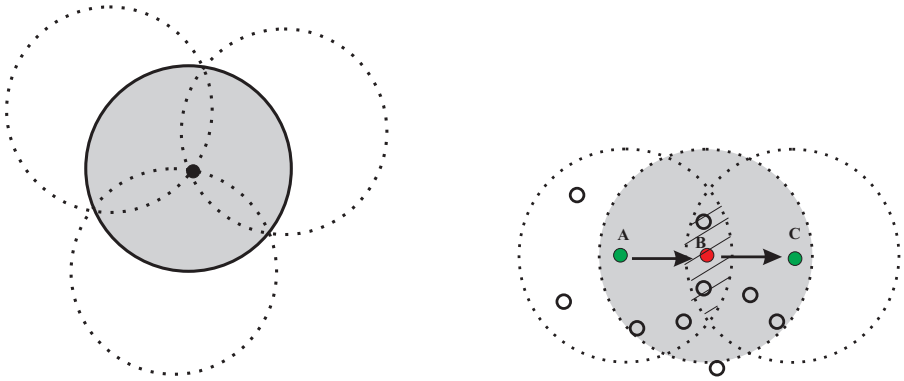
To address this void, we propose a coordinated scheduling protocol, in which the redundant nodes play a role analogous to sentries in real world in the sense that they monitor the health of the active node, and whenever a fault or failure occurs, they jump in to replace the lost node. Since redundant nodes rotate to wake up, we call this scheme *Rotary Sentry*, *R-Sentry* for short. Unlike *P-Sentry*, *R-Sentry* intends to provide a recovery scheme with a statistically configurable average recovery time upon random node failures.

In the following discussions, we use redundant node and sentry interchangeably.

Before going into details of *R-Sentry*, it is beneficial to examine the redundancy among nodes. Every sensor node has a group of neighbor nodes with certain degree of overlapping in communication or sensing capabilities. The *sensing redundant set (SRS)* of a node consists of its neighbors whose sensing areas have overlapping with the node's sensing area. In other words, if two nodes have common grid point(s) covered by them, the two nodes belong to each other's *SRS*.

5.1 Redundant Sets and Gangs

When an active node fails, it can only be replaced by nodes from its *SRS*. However, the replacement of a failed active node can not necessarily be accomplished by simply replacing it with a single



(a) As far as coverage is concerned, a node can usually be replaced by 3-5 nodes with overlapping coverage areas

(b) As far as connectivity is concerned, one node is not always enough to replace the active node even when it is within the communication area

Figure 5.1: One single redundant node is not necessarily sufficient to server in place of an active node.

redundant node, regardless of whether one is trying to repair sensing coverage or network connectivity. This viewpoint is distinctly different from those in earlier studies such as [46, 64], which tried to replace the active node using one redundant node. In fact, it has been shown in [69] that, in a network where nodes are randomly distributed across the field, on average 3-5 neighbor nodes are needed to replace a node in terms of sensing area, as illustrated in Figure 5.1 (a). Connectivity exhibits the similar behavior: after an active node fails, one redundant node, even within that active node's radio range, may not be able to fill the hole as it may not connect to that active node's immediate neighbors on the routing path, which is illustrated in Figure 5.1 (b).

Since an active node can only be replaced by certain combinations of redundant nodes, there is a need for the active node to group all the nodes that belong to its *SRS* into "gangs". Nodes that belong to the same gang can collectively replace the active node. Now, let us look at an example to understand the definition of "gang". In this example, the active node A's *SRS* is $\{B, C, D, E, F\}$, and their respective *GridLists* are shown in Table 5.1. We can see that A's coverage area can be completely replaced by the following combinations of nodes in A's *SRS*: $\{B\}$, $\{C, D\}$, $\{C, E\}$,

node ID	*GridList*	node ID	*GridList*
A	{1, 2, 3, 4}	D	{1, 4, 5, 6, 7}
B	{1, 2, 3, 4, 5}	E	{3, 4, 5, 6, 8}
C	{1, 2, 3, 5}	F	{2, 9, 10}

Table 5.1: An example of *GridList* table.

and $\{D, E, F\}$. As a result, A has four gangs: $\{\{B\}, \{C, D\}, \{C, E\}, \{D, E, F\}\}$, which we call *GangList*.

One thing we would like to point out is, a superset of a gang set technically is also a gang. For example, in Table 5.1, the set {B, C} is also a gang. However, in our work, we only focus on "minimum gangs" – those sets that would no longer be a gang should we remove any member from the set. Grouping nodes from an *SRS* into gangs is essentially a combinatorial problem. To find all the gangs in an online fashion, the basic idea is to enumerate all the subsets of the *SRS*, and mark each subset as a gang candidate whose aggregated *GridList* is a superset of the active node's *GridList*. After all the gang candidates are selected, we next remove those who are supersets of other gangs to obtain the minimum gangs. Table 5.2 shows the pseudo code for a node populating its *GangList* with gangs of sizes no larger than gs. In order to limit the computation of this procedure, we only consider gangs of small sizes.

```
GENE-GANGS(SRS, gs)      // gs : maximum gang size
1   i = 1; clear GL;          // GL : Gang List
2   GCL = SRS;                // GCL : Gang Candidate List
3   while i ≤ gs
4      clear TEMP_GCL;     // TEMP_GCL : temporarily GCL
5      for each set s in GCL
6         if s is a minimum gang
7            then push back s to GL;
8            else push back s to TEMP_GCL
9      if i < gs
10        then GCL = PREPARE-CANDY(TEMP_GCL);
11   return GL

PREPARE-CANDY(TEMP_GCL)
1   clear GCL;
2   for each set t in TEMP_GCL
3      for each node srs in SRS
4         t = srs ∪ t;
5         push back t to GCL;
6   return GCL
```

Table 5.2: Algorithm for generating gang.

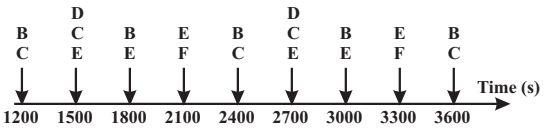
For the purpose of fault tolerance, nodes that belong to the same gang need to wake up simultaneously to completely replace the active node that just failed.

5.2 R-Sentry Algorithm

R-Sentry attempts to bound the duration of service loss. That is, every time an active node fails, *R-Sentry* seeks to make nodes from a gang available within a time interval of Δ, where Δ is the promised upper-bound of service loss time. If an active node has N gangs, then each gang needs to wake up once every $N \times \Delta$. An example schedule is shown in Table 5.3 (b).

Gang	Time to wake up (s)
$\{B, C\}$	1200
$\{D, C, E\}$	1500
$\{B, E\}$	1800
$\{E, F\}$	2100

(a) Gang schedule table



(b) Wake-up schedule

Table 5.3: Illustration of a gang schedule table and the resulting wake-up schedule.

5.2.1 Basic idea

The discussions in this section have simplified assumptions, such as, a redundant node only serves one active node, and the redundant nodes do not fail before the active nodes they protect do. We have these assumptions to keep the explanation easy to understand, and in the following sections, we discuss how to extend *R-Sentry* to handle more realistic scenarios, where a redundant node may belong to the *SRS* of multiple active nodes, and node failures can occur at random time.

We let the active node maintain most of the data structures, while having the redundant node only keep track of its own next wake-up time. The most important data structure maintained by an active node is its gang schedule table, which specifies each gang's next wake-up time. An illustration of such a data structure is provided in Table 5.3 (a). Based on this table, we can infer the wake-up time for each gang. For example, with $\Delta = 300s$, if an active node has 3 gangs, the gang that wakes up at time 1200 also wakes up at time 2100, 3000, 3900, etc., with every two consecutive wake-ups $3 \times \Delta$ apart. A portion of the wake-up events scheduled based on Table 5.3 (a) is shown in Table 5.3 (b).

We next discuss how active nodes establish their gang schedule tables, and more importantly, how they maintain the schedules.

Initialization: During the bootstrapping phase, a wireless sensor network usually runs a host of

initialization services, including neighbor discovery, localization, time synchronization, route discovery, duty-cycle scheduling, etc. Similarly, a wireless sensor network that employs *R-Sentry* needs to run those common services, and additionally the following initialization services:

- *Gang Discovery*. This service ensures every sensor node identifies its neighbor nodes, and populates its *SRS* and *GridList*. At the beginning, each node in the network broadcasts a *presence announcement* message including its ID and location . Such messages are flooded within h hops of the source node, where h is a small number, usually 1 or 2, for the *presence announcement* only matters to nodes within the vicinity of the source node. After a certain amount of time in the process, every node will receive the announcements from all of its *SRS* members, based on which a node can calculate the *GridList* of each node in its *SRS*. After that, a node would be able to form its own *GangList* in the way illustrated in Table 5.2.

- *Schedule Bootstrapping*. The initial set of active nodes are determined by the underlying coverage and initialization protocols. Our approach is similar to the one employed in CCP [48], which can be broken down into three steps: 1) after *presence announcement* exchange phase, every node stays active and starts a random back-off timer, collecting *redundancy announcement* from its *SRS* members; 2) when the timer expires, a node checks its *SRS* members' redundancy status and determines if all grid points in *GridList* are covered by the non-redundant *SRS* members. If yes, it considers itself redundant and broadcasts *redundancy announcement*, otherwise it considers itself non-redundant and doesn't take any actions; 3) at the end of bootstrapping phase, the non-redundant nodes calculate their gangs' schedules and flood them within a small number of hops, staying active; while the redundant nodes, upon receiving the schedules, record their own wake-up time or the earliest one if it receives multiple schedules.

At the end of the initialization phase, the redundant nodes go to sleep with their sleep timers properly set up, while the active nodes start collecting and forwarding collected data.

Probing: A sentry (i.e. redundant) node periodically wakes up, as scheduled, to probe the active node. If the active node has failed, sentry node will become active to resume network service; otherwise, they go back to sleep.

When a sentry node wakes up, it broadcasts a probing message with its node ID enclosed. Around the same time, the other sentries of the same gang will also wake up and probe. If the target

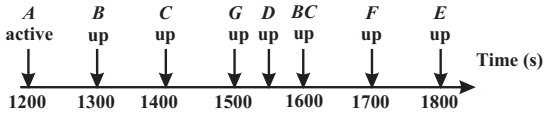
active node is still alive, it will match the node IDs contained in the probing messages with the gang whose scheduled wake-up time is closest to the current time. If the match is successful, the active node updates the gang schedule table by incrementing the wake-up time by the round cycle $N \times \Delta$. Finally, the active node sends the sentries a reply message which has two fields: *NextWakeTime*, and *CurrentTime*. The *CurrentTime* field is used to synchronize the clocks between the sentry nodes and the active node.

The above discussion covers the scenario where the active node is still alive while sentries are probing. If the active node has failed before sentries probe, the sentries will not receive the reply message, and they conclude the active node has failed. If this is the case, the sentries should become active to resume interrupted services. The design of *R-Sentry* thus ensures that, whenever one of the active node fails, its duty will be fully taken over by other nodes roughly within Δ, under the assumptions that the sentries outlive the target active node and corresponding communication time is negligible compared to Δ. Therefore, *R-Sentry* can limit the service loss period within a configurable threshold.

5.2.2 Dynamically establishing schedules for new active nodes

After sentry nodes become active, these new active nodes face several challenges. The main challenge stems from the fact that the association between the new active node and the redundant nodes that belong to its *SRS* has not been established. On one hand, the redundant nodes are still following the schedules from the previous active node, not aware of the fact that the active node has changed; on the other hand, the new active node does not have a schedule for its gangs, and therefore, it will not be guarded properly. Further complicating the problem is that it is impossible for the new active node to communicate with the redundant nodes that are in sleep for their radios are turned off to conserve energy. As a result, we can only attempt to establish the schedule gradually as more and more redundant nodes wake up in group down the stretch.

The algorithm for a new active node to establish its schedule is rather straightforward, yet effective. The idea is, if the redundant node that is probing is not associated with a wake-up time in the gang schedule table, the new active node will assign the next available wake-up slot to the gang that contains this redundant node. If the node is part of multiple gangs, the gang with smallest number of nodes will be picked.



(a) Events observed by new active node A.

{B, C}	-
{D, E}	-
{E, F}	-

(b) $t = 1200$

{B, C}	1600
{D, E}	-
{E, F}	-

(c) $t = 1300$

{B, C}	1600
{D, E}	-
{E, F}	-

(d) $t = 1400$

{B, C}	1600
{D, E}	-
{E, F}	-

(e) $t = 1500$

{B, C}	1600
{D, E}	1900
{E, F}	-

(f) $t = 1550$

{B, C}	2200
{D, E}	1900
{E, F}	-

(g) $t = 1600$

{B, C}	2200
{D, E}	1900
{E, F}	2500

(h) $t = 1700$

{B, C}	2200
{D, E}	1900
{E, F}	2500

(i) $t = 1800$

Figure 5.2: Illustration of how a new active node A establishes its gang schedule table.

To better understand the algorithm, let us walk through an example shown in Figure 5.2. Suppose node A is the new active node, and its *GangList* is {B, C}, {D, E}, and {E, F}. The Δ is 300 seconds. Figure 5.2 (a) illustrates the wake-up events A observes in the establishing phase, and for each event, it shows the corresponding schedule table in the subsequent figures (Figures 5.2 (b-i)). In particular, the establishing process consists of the following steps:

1. Node A becomes active at time 1200, when A's schedule is empty, shown in Figure 5.2 (b).
2. At time 1300, node B wakes up. Node A then assigns next available wake-up slot, i.e. the current time incremented by Δ, 1600, to gang $\{B, C\}$, and updates the gang schedule table accordingly, shown in Figure 5.2 (c).
3. At time 1400, node C wakes up. Node A finds C is already scheduled, so it does not update the gang schedule table. It just simply sends a reply message to C to instruct C to wake up at 1600, shown in Figure 5.2 (d).
4. At time 1500, node G wakes up. A notices G does not belong to its *SRS*, so it does not update the gang schedule table. A sends a reply message to G with a large sleep interval, shown in Figure 5.2 (e). If G serves other active nodes, it will receive a much shorter sleep time from them.

5. At time 1550, node D wakes up. A assigns next available wake-up slot, 1900, to gang $\{D, E\}$, and updates the gang schedule table accordingly, shown in Figure 5.2 (f).
6. At time 1600, node B and C wake up according to the schedule. Since node A's table is not fully occupied yet, A assigns the next available wake-up slot, 2200, to them, shown in Figure 5.2 (g).
7. At time 1700, node F wakes up. node A assigns the next available wake-up slot, 2500, to node E and F, shown in Figure 5.2 (h).
8. At time 1800, node E wakes up. Node A does not update the schedule table because E is already scheduled. A sends a reply message to E, requesting E to wake up at time 1900, shown in Figure 5.2 (i).
9. After that, A has filled its gang schedule table, and it will be able to handle the subsequent waking sentries by incrementing their wake up time by $3 \times \Delta = 900$.

We should note that a new active node normally could establish its gang schedule within a reasonable amount of time, because in *R-Sentry*, every redundant node has to wake up periodically, as scheduled by its active node(s) before sleep.

5.3 Scheduling Redundant Nodes that Serve Multiple Active Nodes

Earlier discussion assumes that one redundant node serves only one active node. In this section, we look at how *R-Sentry* handles situations where a redundant node may serve multiple active nodes.

The main challenge here lies in that when a redundant node probes, how it handles schedules from multiple active nodes. In *R-Sentry*, when a sentry node probes, it only includes its own ID in the probing message. Each of the active nodes that receive the probing messages will examine the difference between the wake-up time it scheduled earlier for the redundant node and the current time. If the difference is smaller than a pre-defined threshold, the active node determines it needs to update the wake-up time for the redundant node, it then calculates the next wake-up time, and sends a reply message back, which contains the following information: the next wake-up time T_{next}, the current time T_{curr}, and the active node's ID; if the difference is above the threshold, the active node determines the wake-up time it generated earlier for the redundant node is still valid, so it will simply enclose the previously scheduled wake-up time in the reply message. After receiving

reply messages from all of its active nodes, the redundant node calculates the sleep interval based on schedules from each of its active nodes, chooses the shortest one as the next sleep interval, and also synchronizes its clock accordingly.

5.4 Dynamically Adjusting Schedules due to Failed Redundant Nodes

In most sensor network applications, failures occur not only to active nodes, but also to redundant nodes, even when they are in sleep mode. For instance, a catastrophic event, such as lightning, affects sensor nodes equally, regardless of the operation mode. When a redundant node fails, the active node can not rely on the gangs that contain the failed node, and should remove these gangs from its schedule.

{B, C}	1200
{D, E}	1500
{F, G}	1800

(a) $t = 1000$

{B, C}	2100
{D, E}	1500
{F, G}	1800

(b) $t = 1500$

{B, C}	3000
{D, E}	2400
{F, G}	2700

(c) $t = 2400$, A still has not heard from D

{B, C}	3000
-	-
{F, G}	2700

(d) $t = 2400$, A cleans up its gang schedule table

{B, C}	3000
{F, G}	3300

(e) $t = 2700$

{B, C}	3600
{F, G}	3300

(f) $t = 3000$

Figure 5.3: An active node detects failures in redundant nodes and adjusts its schedule accordingly.

In *R-Sentry*, dynamically adapting the active node's schedule for its sentry nodes is rather straightforward. If the active node does not hear from a redundant node in k consecutive rounds (k usually is a small number such as 2), it simply removes the gangs that contain the missing node from the *GangList*. To better present the details, let us look at an example illustrated in Figure 5.3, where the active node A has the following gangs, {B, C}, {D, E}, and {F, G}; and the Δ is 300 seconds. Node D fails at time 1000. What happens to A is:

1. At time 1000, node D fails. Node A's gang schedule table is shown in Figure 5.3 (a).
2. At time 1500 (Figure 5.3 (b)), node A only receives probes from E, not D. A decides to wait for one more round (900 seconds in this case) before taking actions.
3. After one round, at time 2400, node A still has not heard from D (Figure 5.3 (c)). A then concludes that D has failed, and removes the gang $\{$D, E$\}$ from the schedule (shown in Figure 5.3 (d)). At this time, A will still send a reply message to E that includes a reasonably long sleep time, like 2Δ.
4. A will schedule the remaining two gangs as usual, with the only exception that the round duration now becomes 600 seconds. As a result, gang $\{$B, C$\}$ will wake up at time 3000, 3600, 4200, etc., while gang $\{$F, G$\}$ will wake up at time 2700, 3300, 3900, etc. Therefore, A will still receives probes every 300 seconds.

Chapter 6

Evaluation of P-Sentry and R-Sentry

In this chapter, we first define a simplified sensor failure model, then we examine our sentry-based protocols' performance against PEAS [46] in the presence of random sensor node failures.

6.1 Sensor Failure Model

We introduce sensor failures into our simulation to model the fact that random node failures are norms instead of exceptions in wireless sensor networks. The *catastrophic failure model* used in our simulation is a coarse-grained one, in which a percentage of sensor nodes that are alive, but not necessarily active, could "die" due to external catastrophic events, like natural disasters. By "die", we mean the node stops functioning completely as an electronic device, though in reality, it's highly likely that not all components in the node are affected by external events. The scenarios of partial failure would be discussed in Chapter 7.

The *catastrophic failure model* is mainly defined by two parameters: (i) failure period f_p: the mean time between consecutive external catastrophic events. The actual interval between two consecutive events is of random length that follows an exponential distribution; and (ii) failure percentage $f_{\%}$: the mean percentage of the living nodes that are affected in a particular event. In a catastrophic event, the actual percentage of affected nodes is a random number between 0 and $2\times f_{\%}$.

6.2 Home-grown Event-driven Simulator

We implemented *P-Sentry*, *R-Sentry* and PEAS with our home-grown simulator *USenSim*. *USenSim* is a discrete event-driven simulator that is intended to model large-scale sensor networks. Our sentry-based scheme shines when there is high node redundancy; and there has to be sufficient

number of active nodes to have valid simulations that reflect the real-world scenarios. In order to achieve the network scale of thousands of nodes, which is difficult to simulate with more detailed network simulators such as NS-2 [85], *USenSim* assumes a constant transmission delay and serialized transmissions among nodes that compete the channel, similar as the one adopted in [48]. *USenSim* doesn't simulator wireless network stack and any aspects that is not directly related to the scheme being evaluated. This design choice makes the simulator light-weight and extremely resource-efficient compared to NS-2.

We believe the simplified network model does not prevent us from proving the validity and effectiveness of our scheme, because our scheme is orthogonal to the protocols in other network layers, and we are evaluating schemes on the same simulation framework, instead of against any real network testbed. We actually used NS-2 to evaluate *DADA* - predecessor of *R-Sentry* with less nodes, and saw similar performance gain as seen in *USenSim*.

6.3 Simulation Model and Settings

Unless explicitly specified, the simulation environment has 600 sensor nodes, uniformly randomly deployed in a $50 \times 50m^2$ square field. Each grid is a $1 \times 1m^2$ square. The initial energy of a sensor node is uniformly distributed between 50 and 60 Joules, which allows the node to function around $2250 \sim 2700$ seconds if the radio is in idle (listen only) mode. Every 50 seconds, each active node transmits a packet containing collected data to the sink, which is located at the center of the field and has persistent power supply. The neighbor nodes that are located within the distance of $15m$ constitute *Sensing Redundancy Set (SRS)*. Due to the high redundancy, gang size of 1 is used in the simulations, unless stated otherwise.

For those parameters that are specific to PEAS, we have tried various parameter values, and choose the setting that produces the best results: probing range $R_p = 3m$, desired probing rate

Routing protocol	Shortest path	**Sensing range**	10 m
Communication range	20 m	**Transmission power**	0.0801 w
Receiving power	0.0222 w	**Idle power**	0.0222 w
Sleeping power	0.0000006 w	**Sensing power**	0.001 w
Bandwidth	19.2 Kbps	**Packet size**	44 Bytes

Table 6.1: Simulation Platform Parameters

$\lambda_d = 0.02$, initial probing rate $\lambda_s = 0.02$, and $k = 32$. We would like to emphasize that we chose a rather small probing range for PEAS to guarantee all the grid points are covered since PEAS does not explicitly require every grid point to be covered. The network/energy parameters used in our simulations are mostly taken from PEAS for fair comparison and are summarized in Table 6.1.

6.4 Performance Metrics

Coverage ratio refers to the percentage of the grid points that are covered. Our sentry-based schemes and PEAS all attempt to achieve a good coverage ratio throughout the lifetime.

θ **network lifetime** is the time the network lasts until the coverage ratio drops below θ and never comes back. We use θ lifetime to measure the scheme's capabilities of preserving coverage and recovering coverage loss from node failures. In actual simulations, we stops the simulation when the coverage ratio drops below a certain percentage that is less than θ. We think the coverage ratio shouldn't drop below a certain percentage for a sensor network to be usable.

Coverage loss time is the duration from the point when a grid point loses coverage to the point when its coverage recovers. The average coverage loss time indicates how quickly a failed active node can be replaced by the redundant nodes.

Packet delivery ratio is the ratio of number of packets received by the sink to the number of packets sent out by the active nodes in a specific length of time window. The packet delivery ratio indicates the connectivity of the network.

6.5 Simulation Results

6.5.1 Service Availability

The motivation behind the study is to provide uninterrupted services as long as possible throughout a sensor network's lifetime. Figures 6.1 (a) and (b) show how the coverage ratio evolves over the time for *R-Sentry* and PEAS. In this experiment, we have $\Delta = 50$ seconds in both schemes. We observe that *R-Sentry* can offer a 90% coverage ratio until 2.5×10^3 second, while PEAS's coverage ratio drops below 90% from 2.0×10^3 second. This is because in *R-Sentry*, once an active node

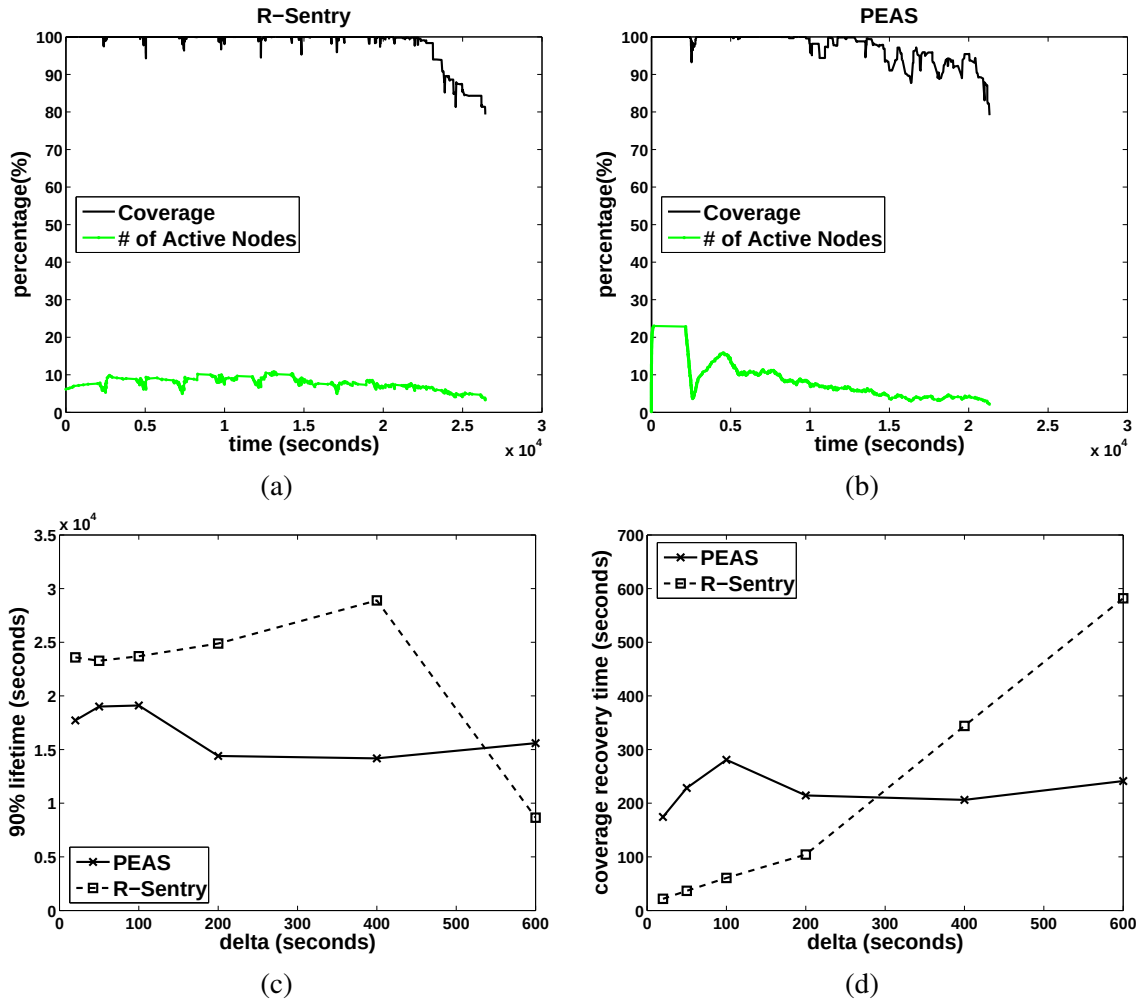


Figure 6.1: Network coverage throughout the lifetime of a wireless sensor network: (a) R-Sentry, and (b) PEAS. The 90% network lifetime and the average coverage recovery time with varying Δ are shown in (c) and (d).

fails, it can be quickly replaced by a gang, while in PEAS, there is no guarantee that the redundant node can fully replace the active node.

We can also confirm our hypothesis by looking at the time series of the percentage of active nodes in both cases. *R-Sentry* can maintain a steady number of active nodes, which as a result guarantees a high coverage ratio; while in PEAS, the number of active nodes decreases over the time due to oversleeping. We note that the drop of the number of active nodes in PEAS at around 2500 second is due to the fact that most of the initial active nodes failed at that time. We did not observe a drop in *R-Sentry* because it can quickly recover from failures.

6.5.2 Algorithm Configurability

Being configurable is a valuable feature for an algorithm because the performance of an algorithm with configurability can be easily tuned up to meet application requirements. In this set of experiments, we have f_p and $f_{\%}$ fixed at 5000 seconds and 5% respectively, and collected the corresponding average coverage loss time with varying Δ. The result is shown in Figure 6.1 (d). We observe that *R-Sentry* demonstrates good configurability: for any given Δ value, the resulting average coverage loss time is always below Δ, and often slightly higher than $\Delta/2$. PEAS, however, fails to do so: no correlation is observed between the actual coverage loss time and the desirable coverage loss time.

Thanks to the capability of maintaining higher coverage ratio and lower coverage loss time, *R-Sentry* can make the sensor network function for a much longer time, as shown in Figure 6.1 (c). The interesting thing we observe from Figure 6.1 (c) is, the 90% network lifetime increases slightly as Δ goes up and then drops significantly when Δ reaches 600 seconds. This can be explained by the fact that, a larger Δ can save more energy by making redundant nodes wake up less frequently and sleep longer. The significant drop of 90% network lifetime, however, is because the likelihood of the overall coverage ratio falling below 90% and never coming back is higher due to oversleep. Another factor that plays a role is, we stop simulation when the overall coverage drops below 80%. With relatively large Δ, the probability is also higher when the coverage drops below 80% before it could possibly come back above 90%.

Parameter Δ is not applicable to *P-Sentry* since it doesn't intend to provide with any bounded recovery time.

6.5.3 Fault Tolerance

It's not unusual that sensor nodes die before energy drains out [46,64,74]. In this set of experiments, we evaluate the robustness of the schemes against random node failures under the *catastrophic failure model*.

In Figures 6.2 (a) and (b), we fix $f_{\%}$ as 5%, and vary the value of f_p. Since *R-Sentry* has sentry nodes guarding active nodes and can dynamically adapt the schedule table of an active node to accommodate the failures of its redundant nodes, it is rather robust against random node failures.

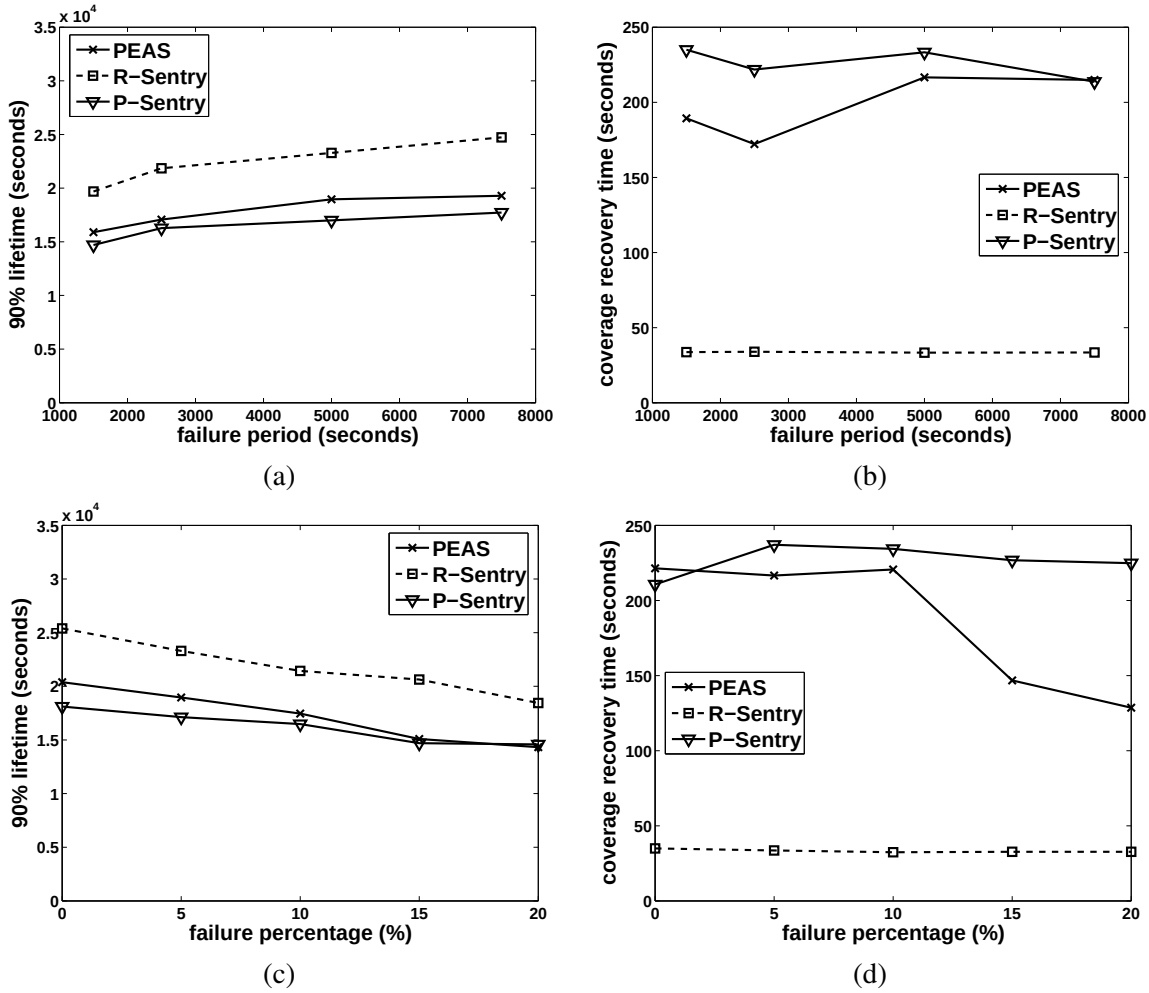


Figure 6.2: The impact of f_p on 90% network lifetime and average coverage loss time is shown in (a) and (b) (with $f_{\%}$ = 5%, and $\Delta = 50$ seconds). The impact of $f_{\%}$ on 90% network lifetime and average coverage loss time is shown in (c) and (d) (with $f_p = 5000$ seconds, and $\Delta = 50$ seconds).

Figure 6.2 (b) tells that, regardless of the failure rate, *R-Sentry* is able to replace a failed active node within around 50 seconds, which is the value of Δ. On the other hand, in PEAS, the average service loss period is much longer. As a result, *R-Sentry* provides a much better 90% network lifetime than PEAS across all the failure rates, shown in Figure 6.2 (a). We also observe the similar trend from the results when we fix f_p and vary $f_{\%}$, as shown in Figures 6.2 (d) and (c).

In this set of simulations, *P-Sentry* and PEAS demonstrate comparable performance. Even though oversleep is unlikely to happen on *P-Sentry*, the fact that a persistent sentry has similar operation power as an active node increases the overall system operation power and thus leads to shorter network lifetime compares to *R-Sentry*

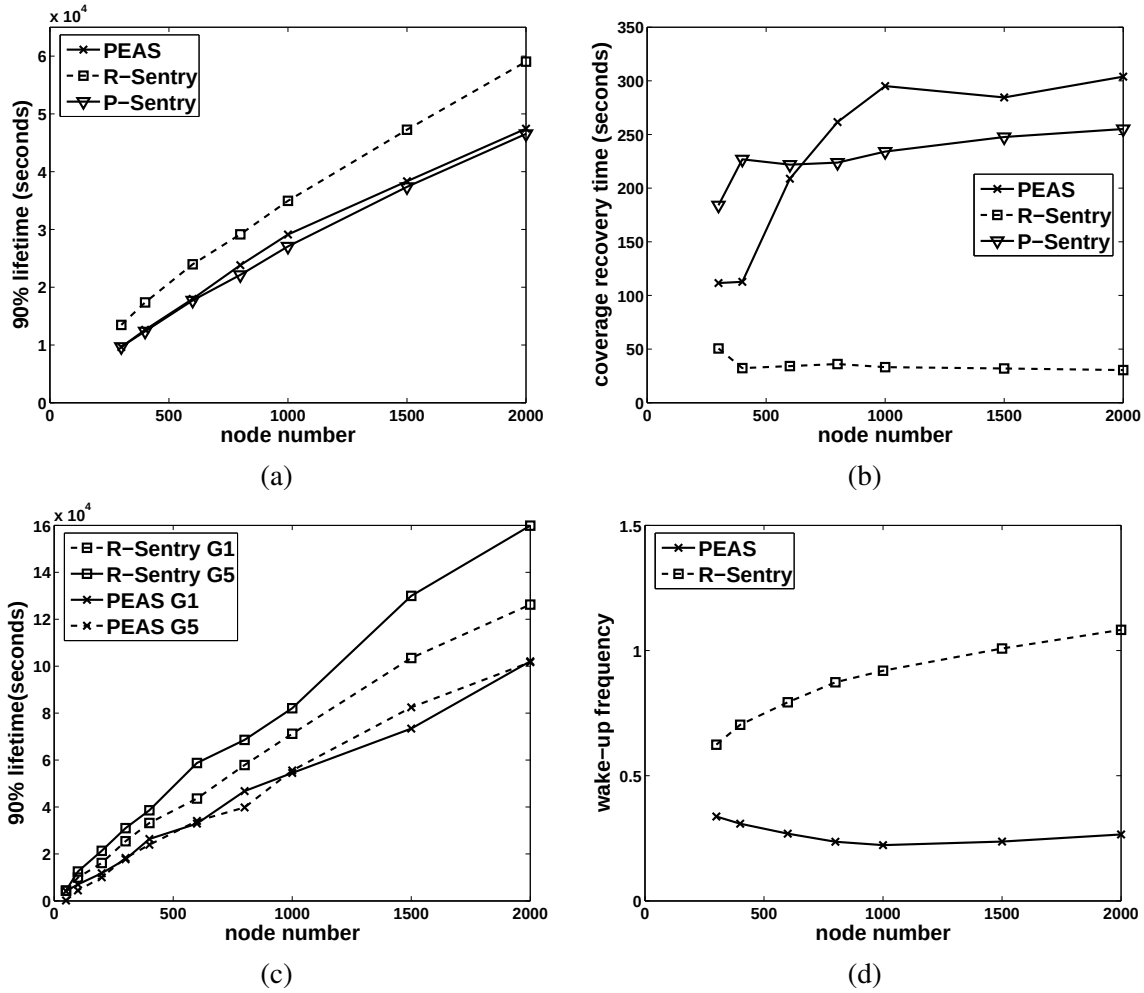


Figure 6.3: Scalability

6.5.4 Energy Efficiency and Scalability

Many sensor network applications seek to achieve longer network lifetime by deploying more nodes. Therefore, the ability to translate larger number of sensor nodes into longer network lifetime is critically important to node scheduling algorithms like *R-Sentry* and PEAS. In this set of experiments, we vary the number of sensor nodes, and measure the 90% network lifetime. The result, reported in Figure 6.3 (a), shows that *R-Sentry* always has longer lifetime than PEAS by roughly 30% as node number increases.

This is because *R-Sentry* efficiently maintains a good coverage ratio for much longer time than PEAS through careful scheduling. In fact, we see that the 90% lifetime in *R-Sentry* almost scales linearly with the number of nodes, and that the difference between the two schemes increases as

the number of nodes increases. At the same time, *R-Sentry* is able to keep the average coverage recovery time around Δ regardless of the node number, as shown in Figure 6.3 (b).

Figure 6.3 (d) shows the wake-up frequency, which refers to number of node wake-ups per second, for both *R-Sentry* and PEAS. PEAS maintains the wake-up frequency roughly at the same level even as node density increases, mainly because its system parameter λ_d, which is $1/\Delta$ in our simulation setup, determines the system aggregate probing rate, i.e. overall wake-up rate. While in *R-Sentry*, even though it tries to achieve that every active node is probed every Δ seconds regardless of the node density, higher node density still leads to modest increase in wake-up rate due to the fact that there are slightly more active nodes at any point in the course of the whole network lifetime as node density goes up.

6.5.5 Connectivity

Our simulations also show that, *P-Sentry*, *R-Sentry* and PEAS all achieve more than 95% packet delivery ratio during 90% lifetime, which confirms our claim in Chapter 3 that a wireless sensor network that satisfies the coverage requirement is automatically connected as long as the communication range is at least double of the sensing range.

6.6 Discussion

6.6.1 Energy Overhead in R-Sentry

The ability to wake up appropriate redundant nodes in time ensures *R-Sentry*'s fault-tolerance. However, on the other hand, it also entails additional energy overhead for requiring extra message exchanges between nodes. Assuming no packet loss, when a redundant node wakes up, it sends out a probing message, and will receive n reply messages, where n is normally smaller than the number of active nodes at the moment in the system. To better understand the overhead, let us next look at one example scenario: assuming there are 2000 nodes and $1 \times 1m^2$ grids, and further assuming that each node on average wakes up 75 times during the 90% lifetime and receives 16 replies during each wake up. This adds up to a total of 1200 reply messages. Compared to our simulation trace, these numbers are rather aggressive. Given the power specifications in Table 6.1, the amount of

energy consumed in transmitting the aforementioned control packets would be:

$$(75 + 1200) * (25 * 8/20000) * 0.06 = 0.765 \quad Joules \tag{6.1}$$

, which is less than 2% of the initial energy level. Therefore, we take the viewpoint that the energy overhead in *R-Sentry* is rather reasonable.

6.6.2 The Implication of Grid Size

The grid is virtual, however its size plays a role in *R-Sentry*: larger size usually leads to longer lifetime because fewer active nodes are needed to cover all the grid points, which we confirmed in our experiments. Specifically, we run simulation with two grid sizes: $1 \times 1m^2$, referred to as $G1$ and $5 \times 5m^2$, referred to as $G5$. The results are shown in Figure 6.3 (c). We can see that the 90% network lifetime of *R-Sentry* can be further improved by the system with a larger grid size. In fact, a grid size of $5 \times 5m^2$ extends the lifetime by 30%. This is mainly because there are less number of active nodes with larger grid size in *R-Sentry*. However, excessively large grid would compromise connectivity since the network would be disconnected due to low density of active nodes. On the other hand, since PEAS does not rely on the concept of grids, its performance is not influenced by the grid size – the inconsistency between $G1$ and $G5$ in the case of PEAS is caused by artifacts of the simulations.

6.6.3 Impact of Gang/Sentry Size

The reason why sentry-based scheme could outperforms other schemes is that, when an active node stops working, its sentries would take over in time to ensure continuous service. A ensuing question would arise naturally: how many sentry nodes are sufficient? Section 2.2.10 has discussed this issue and gives an answer, in the context of one active node and multiple redundant nodes. The analysis could serve as guidance in selecting the right size of gang in *R-Sentry* and right number of sentries in *P-Sentry*, however, it fails to give either the optimal size of gang and the optimal number of sentries. This is largely due to the overlapping in both coverage and redundant node set between neighboring active nodes, which we call *Neighborhood Effect*. *Neighborhood Effect* makes it difficult to quantify the contribution of any single redundant node. [69] has performed an in-depth analysis on the relationship between the probability of network full coverage, network redundancy percentage and

the number of each active node's active neighbors, as shown in the Table 2.3. The result shows how expensive and difficult, if not impossible, it is to have full redundancy in a deterministic sense.

In the presence of *Neighborhood Effect*, it's intuitive to imagine that an active node in *R-Sentry* needs to schedule a gang of smaller size than suggested by [69], to have itself fully covered in a statistical sense. The same thinking should also be applicable to *P-Sentry*. Given the setup that is used in the simulation: $15m$ neighbor range, $50 \times 50m^2$ network field, and the initial number of active nodes: somewhere between 30 and 40 out of total number of nodes 600, as confirmed by our simulation trace, we could roughly know by calculation that, each active node initially has about $8 - 11$ active neighbors [1].

If each active node schedules its own gang, it is inevitable that the gang member could also be shared by other active nodes, thus creating redundancy in terms of redundant nodes if the gang size used is decided solely based on scenarios without considering *Neighborhood Effect*. The redundancy in redundant nodes certainly provides an extra level of insurance against random node failures, it could also bring down the network lifetime if the energy overhead of this extra-level redundancy is significant.

We study the impact of gang size by running the same set of experiments on *R-Sentry*, with gang size as 1 and 2 (referred to as *A1* and *A2* respectively). The result is presented in Figure 6.4.

During the course of the 80% network lifetime, Figure 6.4 (a) and (b) show that both *A1* and *A2* maintain less than 10% of of nodes active, hence steady network coverage and longer network lifetime. The downward spikes on the coverage curve indicate the events of coverage loss, caused by the failure of active nodes, and coverage recovery, thanks to redundant nodes taking over dead active nodes. *A2* has slightly more active nodes and slightly longer 90% lifetime, which doesn't come as surprise because larger gang size often leads to more redundant nodes available when an active node fails.

As Δ goes up, both *A1* and *A2* have longer coverage recovery time, however the recovery time goes up more in *A1* than in *A2*, as shown in Figure 6.4 (d). In the scenario where Δ is 600, network lifetime in *A1* drops significantly, as shown in Figure 6.4 (c). This is all because in *A1*, there are less redundant nodes in probing mode when a active node dies. When the Δ is as long as 600, recovery

[1] $8 = 30 \times \frac{\pi \times 15^2}{50 \times 50}, 11 = 40 \times \frac{\pi \times 15^2}{50 \times 50}$

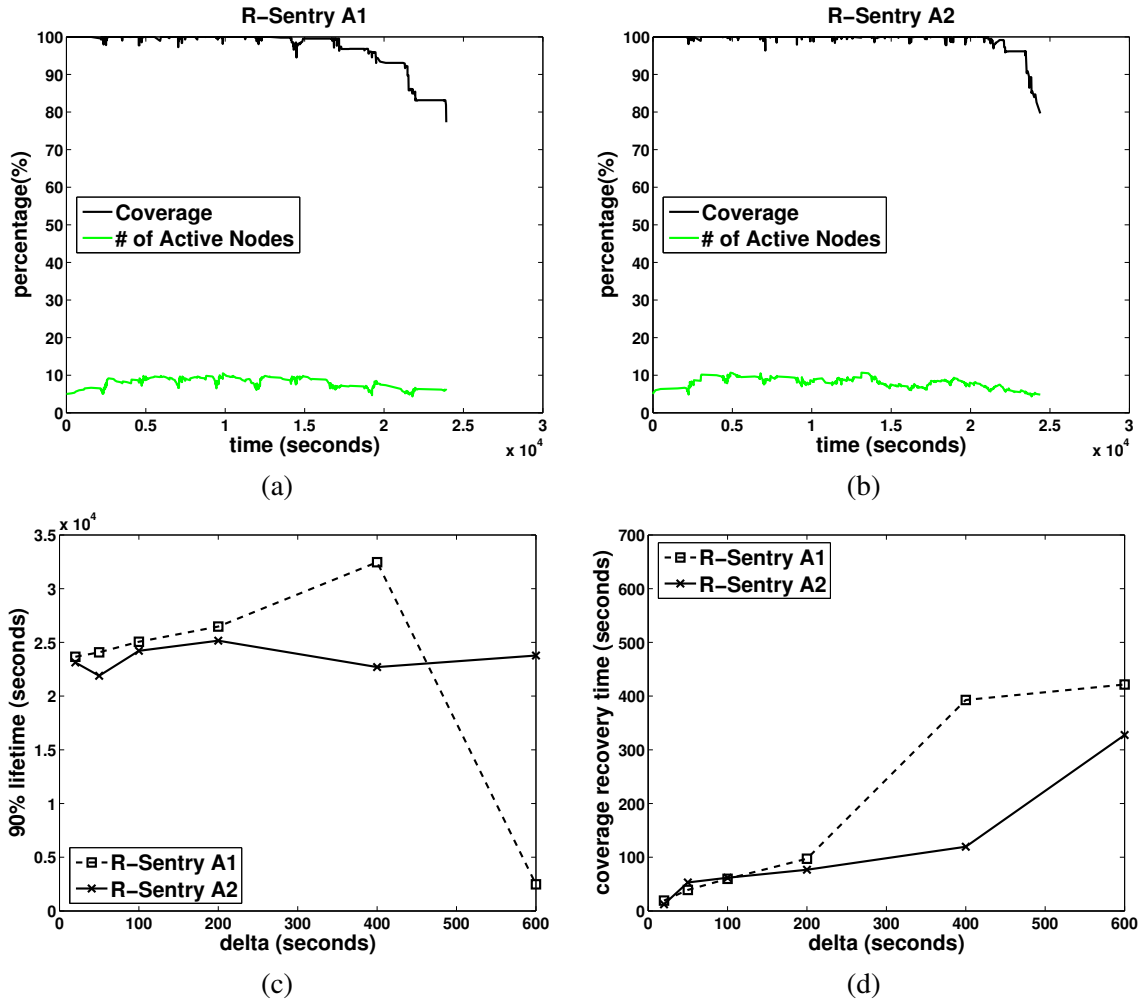


Figure 6.4: The impact of gang size on R-Sentry (I): network coverage throughout the lifetime: (a) R-Sentry A1, and (b) R-Sentry A2, with $\Delta = 50$; the 90% network lifetime and the average coverage recovery time with varying Δ are shown in (c) and (d), where $f_{\%} = 5\%$ and $f_p = 5000$

takes so long that the whole system couldn't maintain the network coverage above 80% before it could bounce back to above 90% level, if it would ever.

A1 and *A2* exhibit similar performance in scenarios with varying failure percentage and fixed failure period, and varying failure period and fixed failure percentage, as shown in Figure 6.5. Both *A1* and *A2* performs well in terms of network lifetime and coverage recovery time. *A2* doesn't seem to benefit from "larger" gang size. This could be explained as follows:

1. Gang size decides how many redundant nodes should wake up to probe. At each catastrophic event, $f_{\%}$ of healthy nodes, either in active, probing or sleeping mode, are affected and die. Both *A1* and *A2* do well in keeping only a minimal number of nodes active

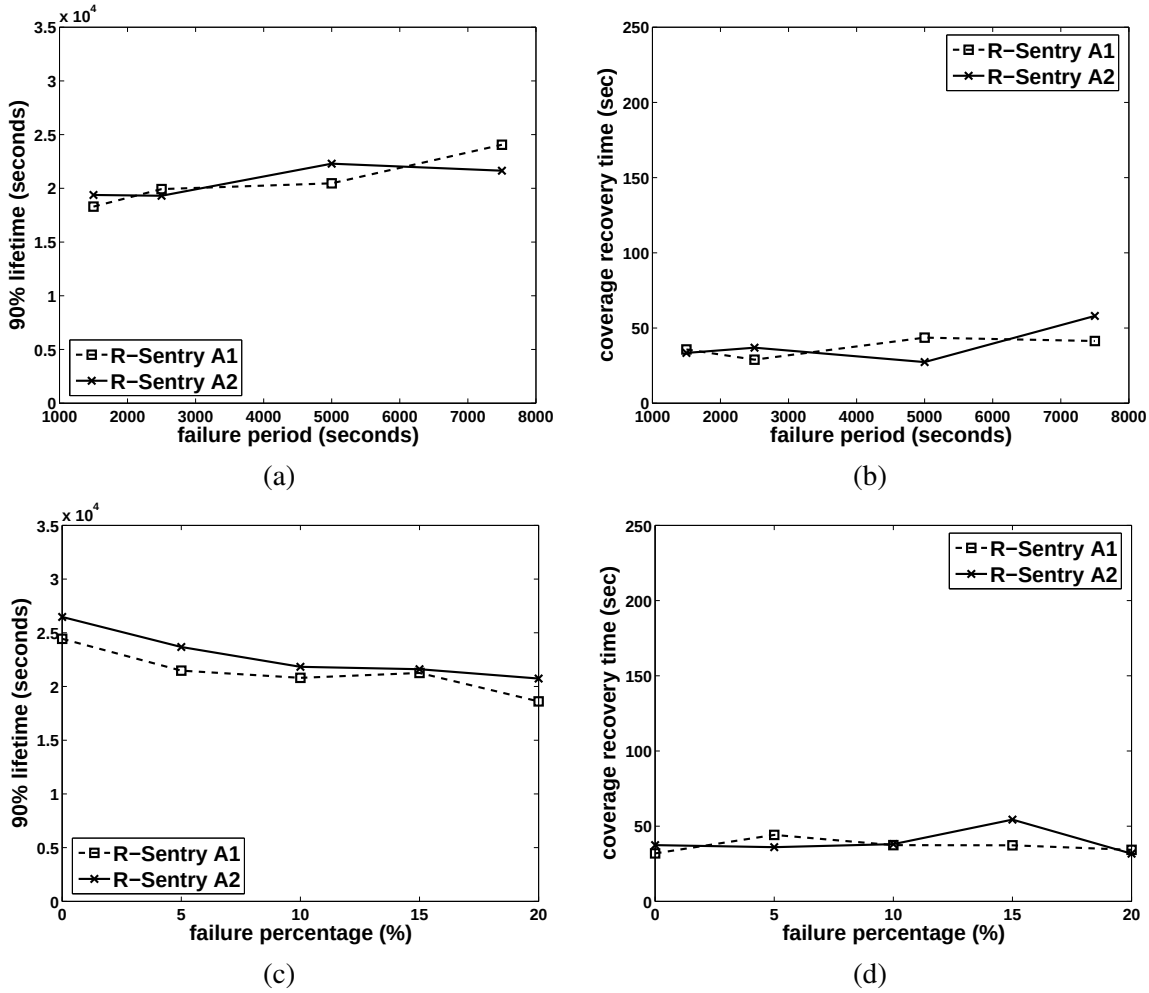


Figure 6.5: The impact of gang size on R-Sentry (II): 90% network lifetime and average coverage loss time shown in (a) and (b), with $f_{\%}$ = 5%, and $\Delta = 50$ seconds; 90% network lifetime and average coverage loss time shown in (c) and (d), with $f_p = 5000$ seconds, and $\Delta = 50$ seconds.

to have 90% plus network coverage, as already shown in Figure 6.4 (a) and (b). This means a catastrophic event has almost the same impact on the current network state in *A1* and *A2*.

2. The fact that each active node adjusts schedules for its redundant nodes based on the current network state greatly reduces the impact from catastrophic event on the future network state, because rescheduling of wake-up time on the remaining redundant nodes contains the impact of the event no further than the next round of probing from redundant nodes, hence the advantage of larger gang size isn't as noticeable as otherwise.

The simulation result seems to indicate that *R-Sentry* doesn't benefit from larger gang size as much as suggested by analysis in [69]. This however is not a reflection of the illegitimacy of gang

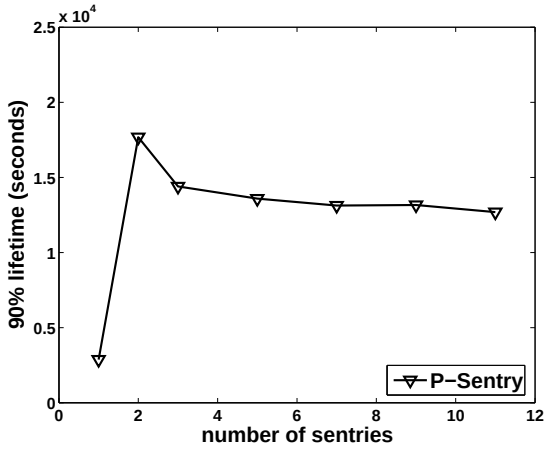


Figure 6.6: Impact of sentry size on P-Sentry

concept, it's mainly because that *Neighborhood Effect* dilutes the effectiveness of gang with larger size. To verify this point, we conducted a separate experiment where redundant nodes are only allowed to serve one active node, the results show that the system benefits from larger gang size significantly in this setup.

Neighborhood Effect also plays a role in *P-Sentry*, as shown in Figure 6.6. *P-Sentry*'s performance peaks when the sentry size is 2, and any sentry size larger than 2 doesn't make a big difference.

Chapter 7

Attacks on R-Sentry and the Defenses

We have shown that *R-Sentry* exhibit a great degree of tolerance of random node failures under the *catastrophic failure model*, which assumes that a sensor node is either intact or halts completely when hit by an external catastrophic event. An affected node in this model appears as one with a shortened lifetime and the impact from a node's premature death is nothing more than a coverage hole that would be recovered later with the help from its sentry nodes. The situation, however, could be far more challenging when some of the sensor nodes behave with malicious intention. To get the network operate properly and effectively is a tougher undertaking when there are a percentage of network nodes trying to undermine the network. Having only a minimum set of nodes active, as is in *R-Sentry*, leaves the whole network more vulnerable than when all the nodes are active. The attack from any single active node could severely disrupt the system.

In this chapter, we explore how *R-Sentry* performs when it's under attack from within the network, and how we improve *R-Sentry* to withstand the attack. We first give an overview of general security issues in the domain of sensor networks, then we present two attack models under the context of *R-Sentry*: passive attack and proactive attack, and our defense schemes against the attacks.

7.1 Security Requirements from Wireless Sensor Networks

Wireless sensor network is a special type of wireless ad-hoc network. It has its own unique set of requirements in security, in addition to the common security features found in traditional networks [86, 87].

7.1.1 Data confidentiality

In wireless sensor networks, data confidentiality means:

- Sensor readings should not be leaked to unintended or unauthorized entities.
- Communication channel is secured against attacks like eavesdropping.
- Node identity information, like encryption key and identifier, should be kept safe to prevent attacks like traffic analysis.

Symmetric key encryption [36, 88] is widely considered the main cryptography to ensure data confidentiality in wireless sensor networks, for the fact that public-key encryption is too expensive in a resource constrained environment like wireless sensor networks.

7.1.2 Data integrity

Data confidentiality ensures that the legitimate data is unavailable to adversary nodes, but it doesn't guarantee the data is intact in transmission. An adversary node could still alter packet to prevent valid data reaching intended receiving nodes, even if the adversary node couldn't decipher the content in the packet. Data corruption or loss could also happen due to harsh ambient conditions. We call the characteristic data integrity that a receiving node should always receive what the sending node originally sends. In most sensor network applications, data integrity could be achieved through data authentication [36].

7.1.3 Data freshness

Unharmed confidential data isn't necessarily valuable to sensor network applications. A lot of time-sensitive sensor network applications, like fire detection, require the freshness of the data, i.e. the data has to be recent enough to reflect the current conditions of the environment being monitored. More importantly, if shared-key strategy is employed in the system, the data must be fresh to avoid replay attack. Typically, shared keys need to change over time, but it could take hours even days for new shared keys to propagate to the entire network, which makes it vulnerable to replay attack.

SNEP, proposed by SPINS [36], tries to address the data freshness issue by identifying two types of freshness: *weak freshness*, which partially ensures message order, and *strong freshness*, which provides total order on request-response pair.

7.1.4 Availability

Adapted traditional encryption algorithms have been deployed in wireless sensor networks to provide a security infrastructure, however the extra energy consumption in encryption/decryption and additional message exchanges take a toll on the lifetime of a sensor node. From perspective of network QoS, the efforts in achieving the data confidentiality and data integrity undermine data availability, because shortened node lifetime always leads to less network service time to upstream applications. Further, the fact that wireless sensor nodes normally don't have the same level of physical security as enjoyed by traditional network could be another source of discounted data availability in wireless sensor networks. Data availability is just another security feature that requires more efforts in wireless sensor networks than in traditional networks [46, 89–91].

7.1.5 Privacy

Privacy in wireless sensor networks are complicated by the fact that sensor nodes operate in low-power mode and are severely constrained by limited resources. PARIS project [92] categorizes the privacy in wireless sensor networks into three groups:

Source location privacy : the physical or virtual location of communication participants may be sensitive information that is undesirable for an adversary to know. Most wireless sensor networks won't be attended after the deployment. The adversaries could easily tamper or destroy a sensor node if it knows its location. Further, tracking and monitoring applications often require the location information not to be disclosed to unauthorized parties to preserve the privacy of the target object. [93] takes on the issue of source location privacy from the perspective of routing by utilizing probabilistic flooding.

Temporal privacy : paired with the location of the data source is the time at which the data was collected. Together, the availability of spatial and temporal information to an adversary constitute a serious privacy breach as this information allows an adversary to track the information origin. [94] tries to enhance the temporal privacy by randomizing the delay of packages in delay-tolerant sensor networks.

Data size/traffic privacy : the size of a message can allow an adversary to infer many things.

Although an adversary might not be able to decrypt sensor messages, it might be able to deduce information about target environment of interest by observing the size of the packets and the amount of traffic crossing the sensor network, .

Privacy in wireless sensor network is often achieved as part of data confidentiality since location information is an integral part of sensor readings [36, 38, 39, 95, 96], though it's not identical to the data confidentiality issue.

7.1.6 Authentication

Preventing unauthorized access to legitimate data and tampering from adversaries is not sufficient to have the network secured. The network also needs to detect and reject false or fake data fabricated by adversaries. Non-legitimate data could harm the network's ability to make quick and sound decision; even worse, maliciously forged data could lead to an attack on the system. A receiving node should only accept data from a legitimate neighbor instead of a random node. At the same time, certain sensor nodes might send command to peers to exercise administrative function, where authentication is a must for the whole network to operate as designed. SPINS [36] proposed μTESLA to achieve authenticated broadcast in resource-constrained environment. LEAP [97] uses a globally shared symmetric key for broadcast messages to the whole group.

7.2 Common Attacks in Wireless Sensor Networks

Wireless sensor networks are particularly vulnerable to several attacks that are commonly seen in traditional networks. Attacks could be launched in a variety of ways, including Denial of Service (DoS) attacks, traffic analysis, privacy violation, physical attacks, and so on [86, 87].
In this section, we walk through a list of attacks that are commonly seen in wireless sensor networks.

7.2.1 Denial of Service attacks

DoS attacks on wireless sensor networks range from simply jamming the sensor's communication channel to more sophisticated attacks, such as altering 802.11 MAC protocol or any other layer of the network stack to deny authorized access to the communication channel.

Due to the constraint in power capacity and computational capability, guarding against a well orchestrated DoS attack on a wireless sensor network can be nearly impossible. A more powerful node can easily jam a sensor node and effectively prevent the sensor network from performing its intended duty.

[98] identifies a few DoS attacks on routing protocol in wireless sensor networks, including "Block Hole", "Wormholes" and a few others. [37] gives an overview of DoS attacks on different communication layers in wireless sensor networks.

7.2.2 The Sybil attack

The Sybil attack is defined as a malicious device illegitimately taking on multiple identities [99]. It was originally described as an attack that is able to defeat the redundancy mechanisms of distributed data storage systems in peer-to-peer networks [100]. The Sybil attack is also effective against routing algorithms, data aggregation, voting, fair resource allocation and foiling misbehavior detection. All the techniques of this category involve a node utilizing multiple identities to receive disproportionally large share of resources . For instance, in a sensor network voting scheme, the Sybil attack might utilize multiple identities to generate additional votes. Similarly, to attack the routing protocol, the Sybil attack would rely on a malicious node taking on the identity of multiple nodes, and thus routing multiple paths through a single malicious node.

7.2.3 Node replication attack

Node replication attack is an attack similar to Sybil attack that compromises the network by manipulating node identities. In node replication attack, an attacker node seeks to add a node to an existing sensor network by replicating the node identifier of an existing sensor node [101]. A node replicated in this fashion can severely disrupt a sensor network's performance: packets can be corrupted or even misrouted. This can result in a disconnected network, false sensor readings, etc. If the replicated node is inserted into a strategically crucial point in the network, the attacker could possibly get the control of a segment of the network, if not the whole network.

7.2.4 Physical attack

Physical attack is not a major concern in traditional networks, but it requires major attention in wireless sensor works, because they are normally deployed in hostile outdoor environments. In such environments, the small form factor of the sensors, coupled with the unattended and distributed nature of their deployment makes them highly susceptible to physical attacks, for example, threats of physical node destruction [102]. Unlike the previously mentioned attacks, most physical attacks, like flooding, and animal stomping, don't intend to interfere with any layer of network protocol and might not even have malicious intention behind them, however, they could permanently destroy sensors either partially or entirely; while some physical attacks are far more intrusive: attackers can extract cryptographic secrets, tamper with the associated circuitry, modify programming in the sensors, or replace them with malicious sensors under the control of the attacker, as described in [103]. [104] has shown that standard sensor nodes, such as the MICA2 motes, can be compromised in less than one minute.

7.3 Security Issues under the Context of R-Sentry

Theoretically, a wireless sensor network running *R-Sentry* could suffer all the attacks described in Section 7.2, and should have most of, if not all, the security features presented in Section 7.1. The fact that *R-Sentry* is a scheduling algorithm in nature and is not tightly coupled with other system modules makes it straightforward to achieve generic network security by applying existing security solutions. Hence, we would like to focus on the security issues that arise because of *R-Sentry*'s unique features, one of which is the validity of schedules. We want to see how adversaries could attack *R-Sentry* with attempt to invalidate schedules, how *R-Sentry* performs under the attacks, and how *R-Sentry* thwarts the attacks by installing new defensive mechanisms.

7.3.1 Assumptions on security infrastructure

Our focus is the attack scheme that is specifically coined against *R-Sentry* and the corresponding defenses; in other words, we like to find out how a node in abnormal condition could undermine the performance of *R-Sentry*, and how we would defend within the framework of *R-Sentry*. Given the fact that the existing security protocols and algorithms are orthogonal to the *R-Sentry* scheme,

it's reasonable to assume the availability of the following security infrastructure in *R-Sentry*:

Key management is essential to any network that uses encryption to enhance security. Random key pre-distribution schemes [105–108] could be used to distribute shared keys. LEAP [97] proposed a protocol that a multiple keying mechanisms, while PIKE [109] describes a mechanism for establishing a key between two sensor nodes that is based on the common trust of a third node.

Cryptography in wireless sensor networks is far more challenging than in traditional networks given the constraints in resources. [88, 110, 111] are representatives of symmetric cryptography in wireless sensor networks, while [112] describes a public-key infrastructure for wireless sensor networks.

With security infrastructure properly in place, we further assume *R-Sentry* has the following security characteristics:

Secure broadcasting requires that only members of broadcast group could receive and decrypt the broadcast message. [113] describes a secure broadcast protocol that uses greedy routing-aware key distribution algorithm.

Node authentication could be achieved through works like μTESLA and its variants [36]. This prevents a node sending messages with forged identities, be it replicated IDs or fake IDs.

Data freshness is achieved to a certain extent, i.e. data with delay of longer than a threshold length of time could be detected and discarded.

Having discussed security features and common attacks in general in wireless sensor networks, we turn our attention to the attack and defense in the context of *R-Sentry*. We will present two attack model that are coined specifically against *R-Sentry* and our defense scheme against the two types of attacks respectively. The attacker in our attack model has to take control of nodes first, or find a way to make nodes behave against *R-Sentry*. The way an attacker comprises a node could take the form of physical attack, DoS attack, node impersonation or any other possible way in real world, but it's the attacking behavior against *R-Sentry* after the take-over that is of interest in our study. The goal of adversaries is always the same: prevent redundant nodes from waking up in time and thus compromise the service availability from the network.

In the rest of the chapter, we first present the passive attack model and our defense - *Pre-emptive R-Sentry*; then we discuss two proactive attack schemes, and the corresponding defense schemes.

7.4 Passive Attack on R-Sentry and the Defense

Any of *R-Sentry*-specific attacks has to be made possible by exploiting system loopholes or attacking weak links in the system. To better understand the origin of the passive attack model, we examine the sensor node from system perspective by breaking down the node hardware.

7.4.1 The anatomy of a typical sensor node

A wireless sensor node typically is comprised of four modules in hardware, as described in Chapter 2: micro-processor, radio, logger and sensor board. Any one of these module could fail due to reasons like, physical tampering, external extreme conditions or internal hardware error. Failure of different modules has different impact on a sensor network system that is deployed with energy-saving scheduling algorithms like *R-Sentry*. Here, we particularly look at how *R-Sentry* is affected by the malfunction of each module.

Micro-processor is the brain of a sensor node, in which software (normally an embedded operating system, like TinyOS [13]) manages the operation of the node. The failing of the micro-processor could lead to erratic behavior from any components on the sensor node, and eventually halts the sensor node's operation. It's fair to say that malfunction of micro-processor stops any activities on the sensor node, as if it is removed from the network.

Logger is an optional component typically located on an external flash memory and is mainly used to hold collected data for further in-network processing. Though a sensor network system with *R-Sentry* could conduct in-network data processing, *R-Sentry* itself doesn't rely on any form of in-network processing. Our analysis in Chapter 2 has shown that, given the system parameters in table 2.1, each node only need a few Kilobytes to hold the data structures required by *R-Sentry*. Hence we take the view that the failing of logger doesn't have significant impact on *R-Sentry*.

Radio is just as important as micro-processor to a wireless sensor node. In wireless sensor

networks, the only way to transmit the data from individual sensor nodes to base station is through the wireless links. When the radio module fails, the node is isolated from the rest of the network and there is no way the node could contribute to the application. Based on this reasoning, when the radio fails, the node is believed to stop working completely, even though all the other modules could still function properly.

Sensor Board is where sensor module is located and is usually attached to the node through expansion slot [1]. If the sensor module stops working, the sensor node wouldn't be able to sense the surrounding environment, and would leave a service hole within the network, either in coverage, connection, or both. Put in the context of *R-Sentry*, the effect of a malfunctioning sensor board could be worse. Suppose the other modules still function properly and are not aware of the failure of the sensor module, the active node in *R-Sentry* could still respond to the probings from its redundant nodes, and generate schedules for them, misleading redundant nodes to believing that the active nodes still functions as it appears: covering its neighborhood.

Based on the above analysis, we have the following arguments regarding the impact of a faulty component on *R-Sentry*:

1. The outage of *logger* alone virtually has no impact on *R-Sentry*.

2. Regardless of the healthiness of other modules, failure of *micro-processor* or *radio* causes the node to appear as dead to the application, because the node couldn't contribute to the network operation.

3. Malfunction of *sensor board* could give a false impression of good network coverage to the surrounding redundant nodes and prevent *R-Sentry* from waking up redundant nodes in time.

It's not difficult to see that the scenario described by the second argument is perfectly covered by the catastrophic failure model in Section 6.1, because the affected node just exhibit the same behavior as dead node in *R-Sentry*; while the third argument could be the foundation for the passive attack scheme that we will present next.

7.4.2 Attacking scenario

R-Sentry relies on the redundant nodes to wake up at scheduled time to ensure the demise of an active node could be caught within Δ time. As a result, any coverage hole left due to the malfunction of an active node would be recovered within Δ statistically. This feature requires active nodes, as long as they are healthy, actively sense and transmit readings back to the base station to ensure their surrounding area is covered.

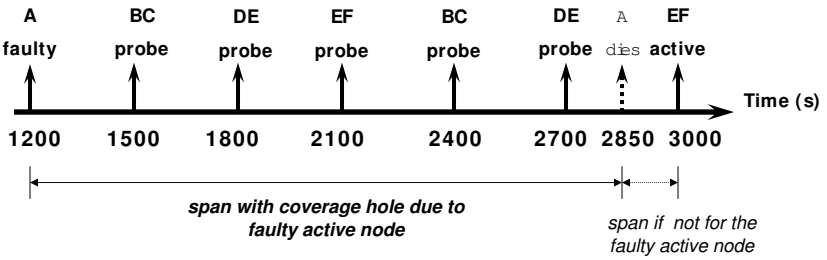
What would happen then if the node is attacked physically, and as a result, sensor board stops working properly; or a node is injected with a piece of malicious code that derails node's sensing activities? It's very likely that random readings would be collected by sensor board, sent to micro-processor for processing if there is any, and eventually forwarded to the base station. This essentially causes a coverage hole because there is no correct sensor reading for the area covered by nodes with malfunctioning sensor board. This attacking scenario certainly could also happen when there is no adversary at all but there are outages in sensor boards due to internal errors or hash working conditions. Regardless, we call such node passive attacking node in the context of *R-Sentry*.

7.4.3 Attacking model

A passive attacking active node appears just like a regular active node, except that it doesn't report sensor reading back to the base stations, with the intention of creating a coverage hole in the network in a stealthy manner. What makes things worse is, attacking node still as usual generates schedules for the redundant nodes and responds to any communication requests from peer nodes. By doing this, the malicious active node could prevent its redundant nodes from becoming active timely and thus leave the coverage hole uncovered, potentially until the malicious node itself dies. What could be worse is, if malicious nodes have external power source and operate for extended longer time, the network would suffer more severely, because the percentage of malicious nodes in the network gets higher and higher as more and more non-malicious nodes run out of power as time goes.

One noteworthy characteristic of the above attacking scenarios is, the malicious node doesn't seek to tamper or manipulate schedules: it still follows *R-Sentry* in generating schedules without any

change or adjustment for the malfunctioning sensor board. This is why we call it a passive attacking model.



(a) Wake-up sequence

B,C	1500
D,E	1800
E,F	2100

(b) t = 1200

B,C	2400
D,E	2700
E,F	3000

(c) t = 2100

Figure 7.1: Passive attack leaves the system uncovered

Figure 7.1 illustrates how a passive attacking node causes the coverage hole uncovered for longer time than when the active node is not in attacking mode. In this example, active node A has gangs of $\{B, C\}, \{D, E\}$ and $\{E, F\}$; Figure 7.1 (b) and 7.1 (c) shows A's gang schedule table at time 1200 and 2100 respectively. If active node A starts attacking by not reporting actual sensor readings at time 1200, though its gangs wake up and probe periodically based on schedules generated by node A (shown in Figure 7.1(a)), the coverage hole caused by node A's malicious action could not be recovered until at time 3000 when gang $\{E, F\}$ detects A's death, leaving certain grid points uncovered for 1800 seconds. Since the active node A dies at time 2850, the coverage loss could have lasted for just 150 seconds if node A operates normally until it dies. We present an enhanced version of *R-Sentry* to counter the passive attack.

7.4.4 Pre-emptive R-Sentry

As we have shown, malicious sensor nodes pose great challenge to *R-Sentry*. To neutralize the impact of the malicious nodes, one solution is to detect the malicious nodes at the early stage. One way we could do is to have the base station perform analysis on the sensor readings collected by active nodes. Since one active node's coverage more or less overlaps with its neighboring active

nodes, if one node's reading is significantly different from its neighbors', there is a good chance this active node is compromised. This approach however requires certain degree of coverage overlapping to have a satisfactory detection rate.

Instead of taking an reactive approach of detecting, we would like to be proactive by limiting the active time of possible malicious sensor nodes.

Basic idea

In *R-Sentry*, once a node becomes active, it stays active until it dies. In other words, when a malicious node becomes active, it could be harming the system until it dies. In recognition of this, it is just natural to limit the active time of malicious sensor nodes to limit the effect from the attack. Since no prior knowledge is available regarding each node's intention, and there is no detection mechanism is in place, we will simply impose a quota β on the time a node could be continuously active. When an active node uses up its quota, it goes to sleep voluntarily. When its redundant nodes wake up, some of them would become active as they learn that the original active node is no longer active. Because of the fact that each active node has to go to sleep voluntarily after it uses up its quota regardless of its energy level, we call this scheme *Pre-emptive R-Sentry*, inspired by the pre-emptive process scheduling algorithm in operating system.

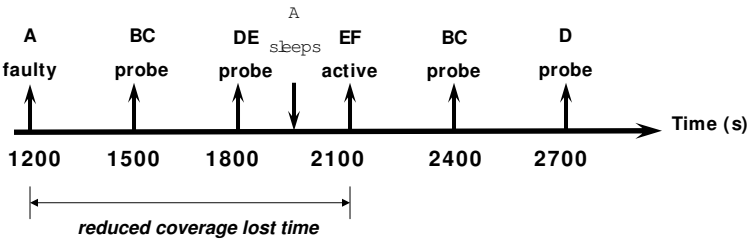


Figure 7.2: Illustration of Pre-emptive R-Sentry against faulty nodes

Figure 7.2 shows how *Pre-emptive R-Sentry* reduces the coverage loss time caused by the attack from passive malicious node, with the same setting as scenario in Figure 7.1. Malicious node A goes to sleep around time 1950 due to the quota constraint, which leads the gang $\{E, F\}$ to take over when it wakes up at time 2100. The benefit is obvious, the coverage loss time is reduced from 1800 seconds to 900 seconds.

We have to be careful about the length of β, which we refer to as *shift period*. If it's too long, the

effectiveness of the scheme would be limited, because the malicious active node may stay active for too long; if it's too short, the whole system would experience state transitions too frequently, which would leave the system in an unstable state and incur excessive overhead related to state transition.

Challenges for Pre-emptive R-Sentry

Pre-emptive R-Sentry introduces more frequent node state transitions, which creates problems for the scheduling scheme itself, in addition to the overhead imposed on each individual sensor node. In regular *R-Sentry*, a redundant node follows the schedules from neighbor active nodes until the redundant node turns active. The same could still be said in *Pre-emptive R-Sentry*, but at the same time, we need to decide how long an active node should sleep after it continuously serves for duration of β. We suggest an active node sleep for a short interval, for the following reasons:

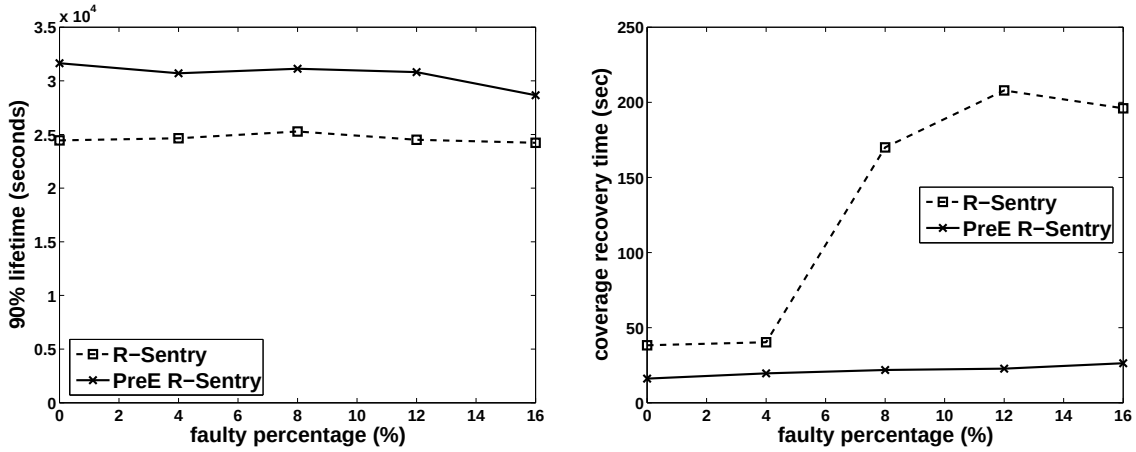
1. The active node should be given the chance of being scheduled by the neighboring active nodes as early as possible once it becomes redundant, so that it could start guarding the new active nodes soon.

2. The once active node should inform the newly activated nodes of its existence as early as possible, because the newly activated nodes didn't have updated information about the neighborhood while they were in sleep and they need to update their knowledge about the neighborhood to build gang schedule tables that reflect the situation as accurately as possible.

7.4.5 Experiment Evaluation

Simulation setup

We use the same simulation platform as discussed in Chapter 6. To better understand the impact of the passive malicious sensors, we modify the catastrophic failure model by injecting malicious nodes into the networks. Specifically, a certain percentage of the total 600 nodes operate with the intention to undermine the network. These malicious nodes are randomly uniformly distributed across the network field. They operates until running out of energy, just like other normal nodes. The only difference is, passive malicious nodes don't report sensor readings to the base station.



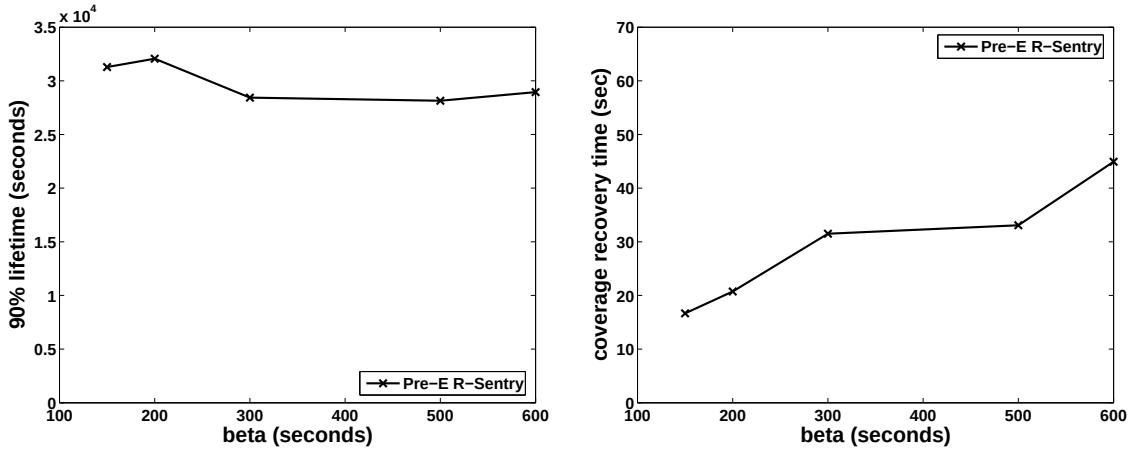
(a) (b)

Figure 7.3: Impact of passive malicious sensor nodes

Simulation results

Figure 7.3 shows how the system performs as the percentage of malicious nodes increases with β as 200 seconds. From Figure 7.3 (a), we can see both regular *R-Sentry* and *Pre-emptive R-Sentry* maintains a certain level of 90% lifetime as the percentage of malicious nodes increases. This is because, with the faulty percentage relatively small, the overlapping of the active nodes' coverage alleviate the coverage hole caused by the malicious nodes. The benefits of *Pre-emptive R-Sentry* however is highlighted in Figure 7.3 (b): the average coverage recovery time stays nearly constant in *Pre-emptive R-Sentry*, while in regular *R-Sentry*, the average coverage recovery time increases as the percentage of the malicious nodes goes up. our explanation is, *Pre-emptive R-Sentry* limits the active time of the malicious nodes, thus allows the redundant nodes to become active to recover the coverage earlier than regular *R-Sentry* does. This behavior agrees with our analysis in Section 7.4.3.

Shift period β governs how long a malicious active node could harm the system before it dies and how dynamic the system composition could be. Figure 7.4 observes the impact of β with percentage of the malicious nodes as 12%. Figure 7.4 (b) depicts the trend of the average coverage recovery time over the shift period. As expected, overall, the coverage recovery takes longer as the β increases. In Figure 7.4 (a), we surprisingly notice that the lifetime is not significantly affected by the shift period. Our explanations are: first, the frequency of node wake-up is not directly tied with the length of the shift period, which is confirmed by the simulation trace. The redundant



(a) (b)

Figure 7.4: Impact of shift period

nodes wake up as scheduled, regardless of whether they would become active or not. This means the overhead from the node wake-up doesn't change much over different shift periods; second, given the definition of 90% lifetime in 6.4, the longer the network coverage is maintained at certain level, the longer the lifetime is, regardless of the total energy that has been consumed, which gives shorter shift period the edge in terms of lifetime. But excessively short shift period does incur significant overhead, including energy consumption in radio state switch, and time delay. This effect however is not apparent in Figure 7.4 (a), noting that the network lifetime when β is 200 seconds is only slightly better than the network lifetime when β is 150 seconds. We believe this is because our event-driven simulator couldn't simulate these circuit-level behaviors. We expect to confirm this phenomenon in a real sensor platform in the future work.

7.5 Proactive Attack on R-Sentry and the Defense

A malicious sensor node in passive attack mode doesn't seek to manipulate its neighboring nodes in any way other than implicitly appearing to be active and functional. The malicious node simply responds to communication requests, implicitly giving false information about its sensing capability. Given the nature of *R-Sentry* as a scheduling algorithm, a distorted schedule could potentially be more harmful and disruptive to the system.

An active node in "proactive attack" mode attempts to send manipulated schedules to its redundant nodes, with the intention to disrupt *R-Sentry* thus leading to shorter network lifetime and longer

recovery time. Two types of proactive attack on *R-Sentry* and the respective defense are discussed in this section.

7.5.1 Secretive mode

An proactive attacking node in secretive mode looks no different from an node in passive attack mode: it still pretends to be functional, responding to communication request from redundant nodes while not reporting sensor reading. However, it responds with manipulated schedules with the intention to discount *R-Sentry*'s effectiveness.

A proactive attacking node could potentially make any schedule that doesn't follow *R-Sentry*. To conceal its identity, the attacking node in secretive mode makes schedules that lead its redundant nodes to sleep reasonably longer than normal *R-Sentry* schedules.

The setup of this type of attack resembles unsolicited longer Δ on top of passive attack model. Thanks to the secretiveness nature of this type of mis-scheduling, it's difficult for the redundant nodes to detect the attacking nodes. Again, our defense doesn't intend to detect these secretive attacking nodes. Section 7.4 has shown that pre-emptor could successfully thwart the passive attack. The impact from unsolicited longer Δ from a percentage of active nodes should be similar from longer pre-configured Δ. In Section 6.5, it has been shown that *R-Sentry* performs well with reasonably long Δ. All this points to *Pre-emptive R-Sentry* as a working defense scheme against proactive attack in secretive mode, as is confirmed by our experiments in Section 7.5.3.

7.5.2 Brute-force mode

An active node in secretive mode tampers the schedule to a limited extent in order to keep its stealthy state; an active node could also aggressively make such schedules that its redundant nodes sleep forever. We refer to the attacking mode in the latter case as brute-force mode.

An attacking node in brute-force mode doesn't have any intention to conceal its identity. It simply launches a trivial yet effective attack: schedule the redundant nodes to sleep forever, leaving the network as short-lived as the current set of active nodes. Brute-force attack from active nodes virtually neutralizes the benefits that a scheduling algorithm like *R-Sentry* brings to the table.

Brute-force attacking scheme

To better study how *R-Sentry* reacts to and defenses against brute-force attack, attacking nodes in the brute-force attack model is made report sensor readings back to base station. As described in Chapter 5, during initialization period, a minimal set of nodes are selected to be active, and the rest of the nodes, deemed redundant for the time being, switch to sleep mode to preserve energy, hoping to contribute when the current set of active nodes stop functioning later. *R-Sentry* scheme essentially comes down to getting redundant nodes back to work when needed.

We will see in Section 7.5.3 that the attack from a limited percentage of malicious active nodes, which try to get their redundant nodes sleep longer than scheduled by *R-Sentry*, has limited impact on the network, thanks to the *Neighborhood Effect* described in Section 6.6.3. To launch an effective attack, our scheme employed by the brute-force attacking nodes has the following features:

- An attacking node always schedule its redundant nodes to sleep longer than the active node's estimated lifetime with the intention of neutralizing *R-Sentry*.
- Manipulating schedule is the only way a malicious node attacks. To isolate the impact from passive attack, attacking nodes in brute-force mode report sensor reading to base station.
- A potential attacking node doesn't attack while in sleep mode.

Defense against brute-force attacking

When a redundant node receives consecutive schedules from the same active node that asks it to sleep unreasonably long, it has legitimate reason to believe that the schedules from that same active node are polluted and should not be followed. Then comes the question: what schedule should the affected redundant node follow? Certainly the node could still follow other active nodes that it considers are trustworthy, but what if majority of the active nodes in its neighborhood are compromised, or a redundant node just couldn't receive valid schedules from active nodes for some reason?

If we go back to Chapter 5 and revisit how an active node makes schedules for its redundant nodes, we'll see that there are mainly three parameters that dictate schedule making:

1. Δ
2. gang size
3. number of redundant nodes near the active node

Both item 1 and item 2 are pre-configured and known to redundant nodes, while item 3 is something only active node itself knows. It seems the defense against attacks of invalid schedules from active nodes essentially comes down to figuring out the number of redundant nodes near the malicious active nodes. However, it would not be straightforward for a node to detect the existence of another node if there is no direct communication link between them. Since redundant nodes don't necessarily wake up at the same time, it's not feasible to resort to the help from intermediate nodes to establish the indirect connection between two redundant nodes; and even if the percentage of redundant nodes that wake up at the same time reaches a point that makes the indirect communication meaningful, the ensuing overhead and complexity in communication and node tracking, and the so-so quality of information derived through non-neighbor redundant nodes make not a viable answer collecting information on relevant redundant nodes through indirect communication.

Given the aforementioned observations, we step back from the thinking that we have to defend by restoring the untampered schedules. Instead, we adopt random schedules that is not bound by any information from active nodes. If the random function follows a context-aware distribution, it's highly likely to achieve relatively sub-optimal performance as complex schemes do, while not being penalized with overhead in communication and computation as much [114, 115]. Here we employ an uniform distribution that has the redundant nodes sleep for a time that is randomly distributed between Δ and an upper bound. The choice of Δ as lower bound is obvious, because Δ is the minimal time for a redundant node to sleep in *R-Sentry*, as described in Chapter 5. Upper bound is not as straightforward, but we can use the information that is available to a redundant node to deduce one.

There is certainly little valuable information available in those manipulated schedules, but there is still certain piece of valid information that is available to or could be derived by redundant nodes. These information includes:

- **Current network lifetime** $t_{curr\_net\_time}$

A redundant node could easily get this from its internal timer.

- **Remaining functioning nodes** n_{remain_nodes}

  The number of remaining functioning nodes (either in active mode or sleep mode) n_{remain_nodes} could be roughly derived by the following equation:

$$n_{remain\_nodes} = N - \frac{t_{curr\_net\_time}}{t_{active\_life}} \times s_{active\_set} \times r_{adj}, \tag{7.1}$$

  where N is the total number of nodes, $t_{curr\_net\_time}$ is the current network lifetime, t_{active_life} is the lifetime an node could sustain continuously in active mode, s_{active_set} is the average number of nodes that need to be active to maintain the network coverage above the pre-configured threshold, and r_{adj} is an adjustment factor that is normally larger than 1.

  The actual values of N and $t_{curr\_net\_time}$ are available to any nodes; an estimation of s_{active_set} could be provided by network administrator based on historical data or be calculated based on grid size, sensing range, neighbor range; r_{adj} could be set based on environment conditions, sensor node reliability, or any other conditions that could cause a node stop working prematurely, like the catastrophic events that we model in Chapter 5.

- **Average number of redundant neighbor nodes of each active node** n_{avg_redud}

  We could derive n_{avg_redud} by the following equation:

$$n_{avg\_redud} = \frac{n_{remain\_nodes} - s_{active\_set}}{s_{active\_set}} \times r_{overlap} \tag{7.2}$$

  where $r_{overlap}$ is a factor that reflect the fact that redundant nodes could be shared by multiple active nodes. $r_{overlap}$ normally takes a value that is larger than 1.

- **Remaining network lifetime** $t_{remain\_net\_life}$

  With the number of remaining functioning nodes available, the remaining network lifetime could be derived using the following equation:

$$t_{remain\_net\_life} = \frac{n_{remain\_nodes}}{s_{active\_set}} \times t_{active\_life} \tag{7.3}$$

  The above equation assumes that all the remaining nodes would contribute to extend the network lifetime, which is not always the case, but the Equation 7.3 does provide an

estimation with a good accuracy and the historical data could always be used to calibrate the value derived though the equation.

- **Remaining lifetime of the redundant node** $t_{remain\_node\_life}$
  Remaining energy is readily available to the node, hence remaining life time of a node is also available, as long as the operation power of a node doesn't change.

With the above information available, we choose the minimal value among $t_{remain\_net\_life}$, $t_{remain\_node\_life}$ and $n_{avg\_redud} \times \Delta$ as the upper bound for the following reasons:

1. A redundant node won't be able to contribute if it sleeps past the estimated network lifetime.
2. A redundant node won't be able to contribute if it dies during the sleep.
3. A redundant shouldn't sleep longer than the $n_{avg\_redud} \times \Delta$ according to *R-Sentry* algorithm.

We refer to this minimal value as $SleepInterval_{upperbound}$.

Detection of invalid schedules

Unlike *Pre-emptive R-Sentry* against passive attack, our defense of random scheduling against brute-force attack requires detection of invalid schedules. Given the brute-force nature of the attack, invalid schedules in brute-force attack are normally not as deceptive as in secretive mode, because all a malicious node wants to do is to send excessively long sleep time, hoping redundant nodes would follow whatever schedules are received.

Our analysis has shown that a redundant node shouldn't sleep longer than $SleepInterval_{upperbound}$, thus $SleepInterval_{upperbound}$ could be part of the equation that decides if a schedule is valid. To be in line with our original motive of being relatively simple and effective, a feature enjoyed by random scheduling, we use a fraction of $SleepInterval_{upperbound}$ as the criterion: $SleepInterval_{upperbound} \times F$, where F is a positive floating number that is less than 1. Larger F would increase the threshold for random scheduling to kick in while smaller F would lower the threshold. F is just one of the many parameters that is adjustable in *R-Sentry* to adapt to different needs in real world.

7.5.3 Experiment Evaluation

The simulation platform and basic setup for proactive attack is identical to the one used in simulations for passive attack, as describe in Section 7.4.5. Changes specific to new attacking modes will be described later in corresponding sections.

Secretive mode

A new parameter, Δ_{scale}, is introduced in secretive mode of proactive attack to measure how the system responds to manipulated schedules. Specifically, an active in secretive attacking mode scales up Δ by a factor of Δ_{scale} in making schedules for its redundant nodes. In other words, $\Delta_{scale} \times \Delta$ replaces Δ in original equations that generate schedules.

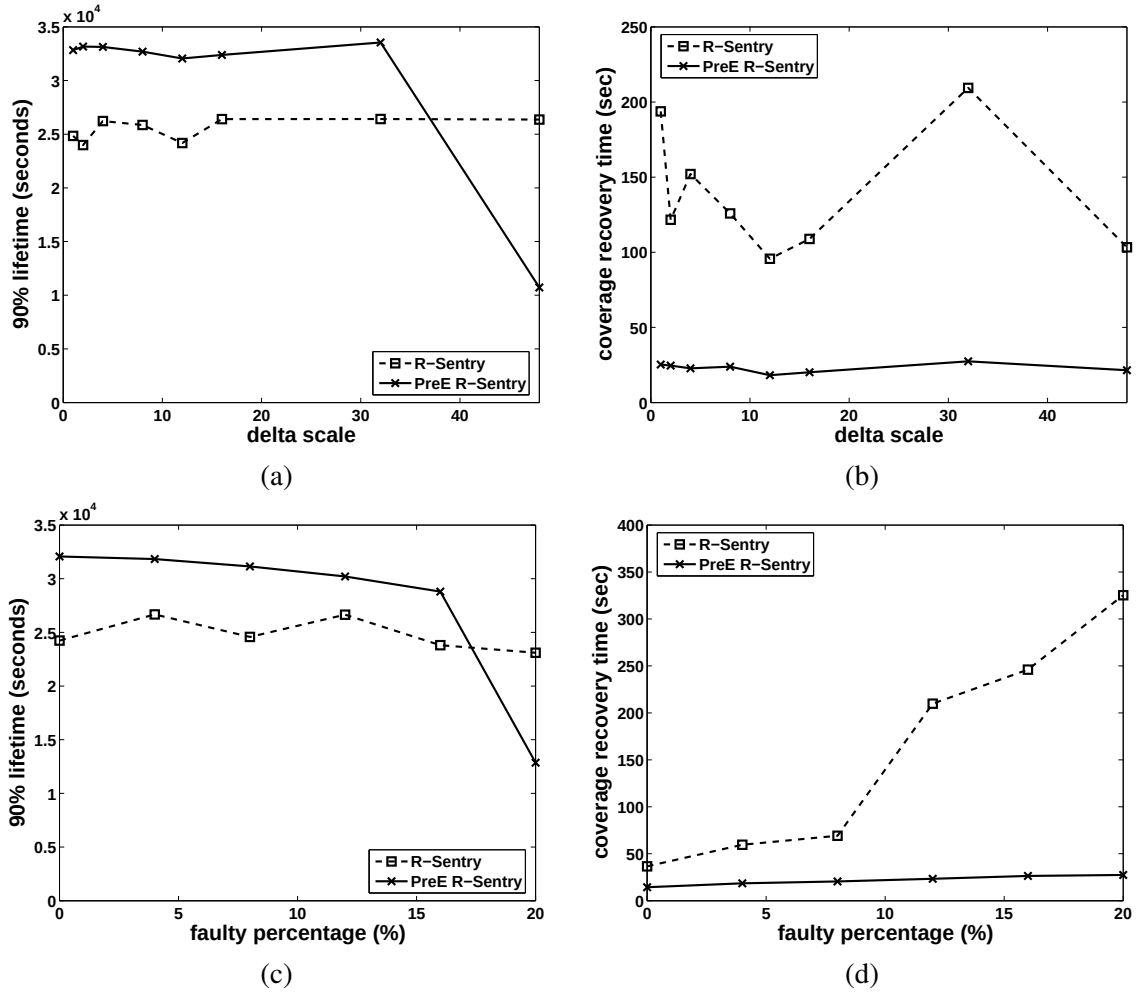


Figure 7.5: Proactive attacking in secretive mode

We first increase the Δ_{scale} while keeping fixed the percentage of attacking nodes as 12% and Δ as 50. Figure 7.5 (a) and (b) show that *Pre-emptive R-Sentry* achieves consistent coverage recovery time, and performs better in network lifetime until when Δ_{scale} is over certain value beyond 30. Before Δ_{scale} goes beyond 30, the performance comparison between *R-Sentry* and *Pre-emptive R-Sentry* here is very similar to the one in passive attack(see Figure 7.3) . This is because, secretive mode of proactive attack suffers from the same passive attack that malicious active nodes don't report sensor reading, which is neutralized by imposing pre-emptor on all active nodes, and the malign schedule from malicious active nodes, which is mitigated thanks to *Neighborhood Effect* described in Section 6.6.3. What happens when Δ_{scale} passes over a threshold value is, at certain point during a transition period from one group of active nodes to the next group of active nodes, there are not enough active nodes in the network that keeps the network coverage above a certain threshold, which is 80% in our simulation setup. *Neighborhood Effect* alleviates the negative impact from tampered schedules, but couldn't completely shield the network from being affected. In Figure 7.5 (c) and (d), we keep Δ_{scale} fixed at 8 and increase the percentage of attacking nodes. Similar patterns are observed:*Pre-emptive R-Sentry* nicely maintains coverage recovery time consistently low, and has longer network lifetime until the percentage of malicious nodes goes beyond certain threshold. Again, this is because of insufficient number of active nodes during transition period.

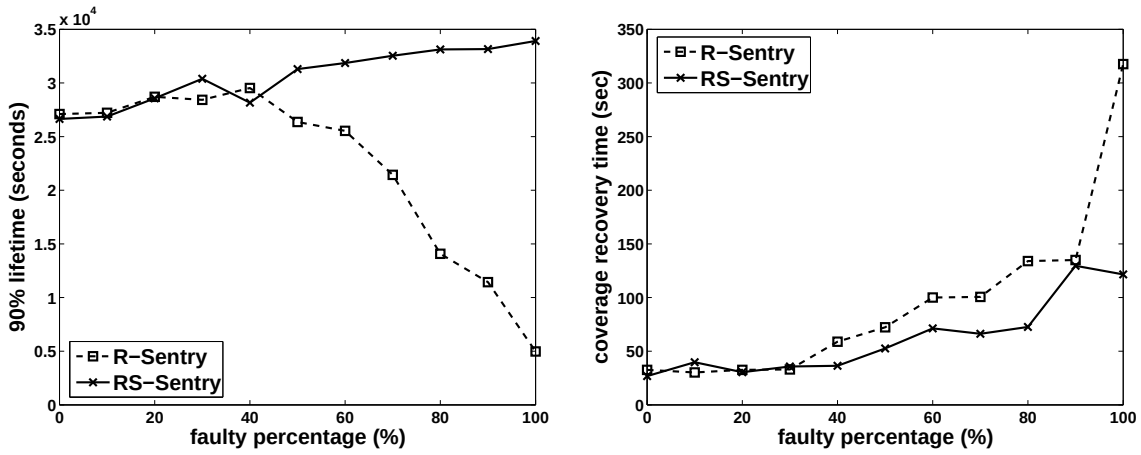
At the same time, we noticed that *R-Sentry* actually reacts to the proactive attack in secretive mode pretty well. This is largely due to the *Neighborhood Effect* and the limited impact from malicious nodes. After all, malicious active nodes have to maintain its secretive state by not distorting the schedule too much.

Brute-force mode

To evaluate the effectiveness of our defense scheme of random scheduling against brute-force attack, we increase the malicious node percentage from 0 until 100 by the increment of 10 in both *R-Sentry* and *Random Scheduling R-Sentry*, *RS-Sentry* for short. In our simulation setup, malicious nodes send excessively long sleep time hoping to get redundant nodes sleep for as long as $\frac{InitialEnergy}{p_{idle}}$, and we choose F as 0.5.

Figure 7.6 shows how the 90% network lifetime and average coverage recovery time change in

R-Sentry and *RS-Sentry*.



(a) (b)

Figure 7.6: Proactive attacking in brute-force mode

R-Sentry shows its resilience and robustness in the presence of invalid schedules. It performs well until the faulty percentage hits 40%. This is mainly because of *Neighborhood Effect* discussed in Section 6.6.3. A redundant node receives schedules from multiple active nodes located in its neighborhood. According to *R-Sentry*'s design, a redundant node always follows the one that schedules it to sleep for the shortest time, which means, a redundant node won't sleep for excessively long period as long as there is still healthy active node in its neighborhood, and there is still a good chance that a redundant node could contribute to extending the network's lifetime. However, as the percentage of faulty nodes goes up further, the percentage of valid schedules received by a redundant node goes down, and the likelihood of a redundant node not receiving any valid schedule goes up significantly. This explains the significant drop of network lifetime in *R-Sentry* when the faulty percentage goes beyond 40%.

With random scheduling, the impact from manipulated schedules that lead to excessively long shortest sleep time is contained. At the end of probing period, if a redundant node concludes the shortest sleep time it receives is invalid based on criterion described in Section 7.5.2, it simply ignores the schedules from active nodes and adopts a schedule generated by random scheduling. Though this random scheduling doesn't give an answer as well-calculated as by *R-Sentry* with no malicious nodes, it certainly produces a schedule that reflects the system state to a certain extent and keeps the redundant node in the loop by avoiding sleeping unreasonably long. The result is an

improved network lifetime and reasonably good coverage recovery time.

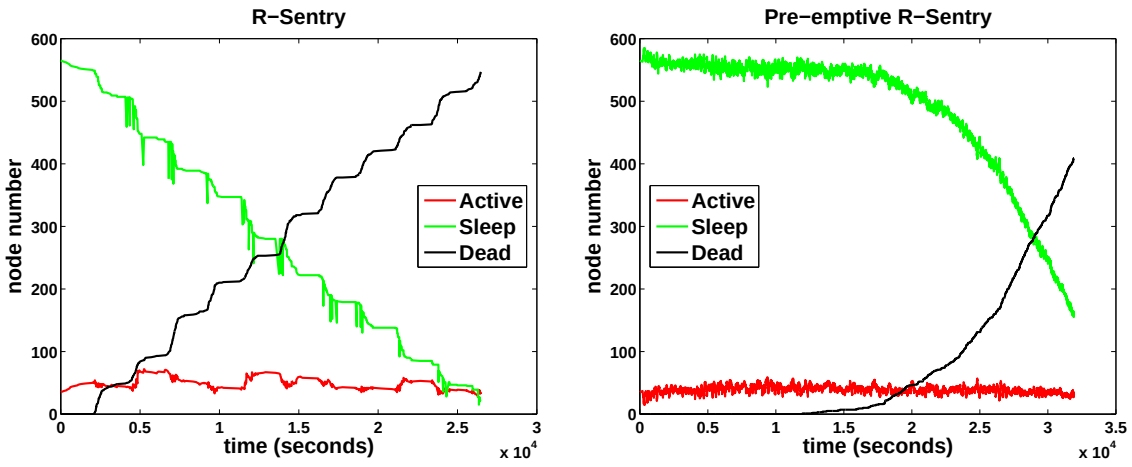
One interesting observation from Figure 7.6 is, network lifetime actually goes up slightly after the random scheduling kicks in when the percentage of malicious nodes goes beyond 40%. This, however, comes at the expense of average recovery time, as shown in Figure 7.6 (b).

7.5.4 Discussion

The study presented in this chapter on the passive and proactive attack in *R-Sentry* reveals interesting facts that are not exposed in previous chapters and also raises questions concerning the applicability of the attack and defense schemes. We will briefly discuss these issues in this section.

Bonus from being pre-emptive

The introduction of pre-emptor into *R-Sentry* is intended to to counter the attack from passive malicious sensor nodes. Surprisingly, our simulation result reveals that pre-emptor gives us some bonus performance boost. When we look at the data point whose malicious node percentage is 0 in both plots of Figure 7.3, we find it interesting that even when there is no passive malicious nodes, *Pre-emptive R-Sentry* still outperforms regular *R-Sentry*. Our explanation is, in *Pre-emptive R-Sentry*, the system experiences changes in a more gradual manner in the quantity of the active nodes, as well as the quantity of the sleep nodes. This largely lowers the probability of not having enough nodes at proper positions to maintain the network coverage at certain level.



(a)R-Sentry (b)Pre-emptive R-Sentry

Figure 7.7: Change of system state

Figure 7.7 illustrates how the numbers of nodes in different states change during the course of the 90% network lifetime. *Pre-emptive R-Sentry* has a more smooth curve than regular *R-Sentry*.

Uncooperative passive malicious nodes

Pre-emptive R-Sentry imposes a quota on continuous time active nodes can operate for and performs well against passive attacking. This quota-based approach, however, requires cooperation from malicious nodes. If a malicious node ignores the quote restriction, continuing to operate after it uses up its quota, pre-emptor just couldn't be enforced.

It's certainly not reasonable to ask malicious nodes to cooperate, even though it's possible that the quote restriction could be still enforced if software module that controls the node mode is not compromised. One ultimate solution is to have a hardware timer embedded in the power controller on the sensor node. Reprogramming(compromising) hardware module is far more difficult than reprogramming software stack on any electronic devices. With the help from a hardware timer, *Pre-emptive R-Sentry* could be implemented regardless of malicious nodes' willingness to cooperate.

Possibility of brute-force attack

As pointed out in Section 7.4.1, passive attack could be launched by malicious parties intentionally annihilating the sensor module, or could be a byproduct of malfunctioning sensor module in sensor nodes.

In proactive attack, both attacking modes could happen when a percentage of nodes are injected with malicious code that changes logic in generating schedules. Another possible scenario could be that redundant nodes receive the signal from active nodes but couldn't decode the signal to recover the schedules, for example on occasions when there is electromagnetic interference like traffic jam from malicious parties, or electromagnetic wave from nearby power lines etc. Whichever the case may be, it's not unusual that a large percentage of nodes are affected.

Chapter 8

Summary

Providing long-lived, fault-tolerant sensor services requires the network to be able to quickly recover from coverage loss due to frequent node failures. That is, if an active node fails, its coverage should be quickly resumed by the redundant nodes, which were in sleep mode to conserve energy. Earlier node scheduling solutions, such as PEAS [46], adopt completely random schedules among redundant nodes. A random schedule cannot guarantee a redundant node will wake up timely when the active node fails, nor can it guarantee the redundant node that happens to wake up can fully recover the coverage hole.

R-Sentry addresses these issues by grouping redundant nodes into "gangs" and careful scheduling of the "gangs". *P-Sentry* overcomes the drawback of excessive number of wake-ups the *R-Sentry* suffers and fits in scenarios where *R-Sentry* might not work best due to costly wake-up.

Defense against passive and proactive attack on *R-Sentry* have also been addressed in the thesis. In the case of passive attack, malicious party doesn't attack the essence of *R-Sentry* – scheduling, instead active nodes stop reporting sensor readings while still making valid schedules for their redundant nodes. We introduced pre-emptor into *R-Sentry* to limit the continuous active time of each active node without pro-actively identifying malicious nodes.

In the case of proactive attack, malicious party goes straight to manipulating schedules of redundant nodes. It turns out that *R-Sentry* has an inherent robustness against attacks from a limited percentage of malicious nodes. However, when the attack turns into a brute-force manner by blatantly generating invalid schedules, *R-Sentry* loses its basis of working properly. Random scheduling comes to rescue by applying random schedules that are bound between Δ and an upper bound that is closely tied to the current system state.

The study in the thesis doesn't intend to be universally applicable to any popular wireless sensor networks. It's a research work that is largely based on experiments on our home-grown simulation

platform. However it's the author's intention to delve into the scheduling scheme in wireless sensor networks and provide guidances and insights from some interesting observation presented in the thesis. Moving forward, however advanced the technology may be, scheduling-based schemes like *R-Sentry* could always boost the system performance that is either bare-bone or less planned.

References

[1] CrossBow Technology, http://www.xbow.com/Products/productdetails.aspx?sid=174. *MICA2 Information Center*.

[2] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Journal of Computer Networks*, pages 393–422, 2002.

[3] Vipul Singhvi, Vipul Singhvi, Carlos Guestrin, James H. Garrett Jr., and H. Scott Matthews. Intelligent Light Control using Sensor Networks. In *Proceedings of the ACM SenSys 2005*, November 2005.

[4] UPC. http://www.upccode.net/.

[5] L. M. Ni, Y. Liu, Y. C. Lau, and A. P. Patil. Landmarc: Indoor location sensing using active rfid. In *the 1st IEEE International Conference on Pervasive Computing and Communications (PerCom'03)*, March 2003.

[6] Mario Chiesa, Ryan Genz, Franziska Heubler, Kim Mingo, Chris Noessel, Natasha Sopieva, Dave Slocombe, and Jason Tester. RFID: a week long survey on the technology and its potential. *Harnessing Technology Project*, March 2002.

[7] Radu Stoleru, Tian He, John A. Stankovic, and David Luebke. A high-accuracy, low-cost localization system for wireless sensor networks . In *Proceedings of the ACM SenSys 2005*, November 2005.

[8] Radu Stoleru, Pascal Vicaire, Tian He, and John A. Stankovic. StarDust: a flexible architecture for passive localization in wireless sensor networks. In *Proceedings of the ACM SenSys 2006*, October 2006.

[9] David Madigan, Eiman Elnahrawy, Richard P. Martin, Wen-Hua Ju, P. Krishnan, and A. S. Krishnakumar. Bayesian Indoor Positioning Systems. In *Proceedings of IEEE INFOCOM*, March 2005.

[10] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. Peh, and D. Rubenstein. Energy-Efficient Comiputing for Wildlife Tracking: Design and Tradeoffs and Early Experiences with Zebranet. In *asplos02*, pages 96–107, 2002.

[11] CrossBow Technology, http://www.xbow.com/Products/productdetails.aspx?sid=253. *Imote2 Information Center*.

[12] TI CC2420 Transceiver. http://focus.ti.com/docs/prod/folders/print/cc2420.html.

[13] TinyOS. http://www.tinyos.net/.

[14] Sun SPOT. http://www.sunspotworld.com/.

[15] Mark Stemm and Randy H Katz. Measuring and reducing energy consumption of network interfaces in hand-held devices. *IEICE Transactions on Communications*, E80-B(8):1125–1131, August 1997.

[16] Oliver Kasten. Energy Consumption. http://www.inf.ethz.ch/personal/kasten/research/bathtub/energy_consumption.html.

[17] TI CC1000 Transceiver. http://focus.ti.com/docs/prod/folders/print/cc1000.html.

[18] W. Ye, J. Heidemann, and D. Estrin. An Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *Proceedings of IEEE INFOCOM'02*, June 2002.

[19] Joseph Polastre, Jason Hill, and David Culler. Versatile Low Power Media Access for Wireless Sensor Networks. In *Proceedings of the ACM SenSys 2004*, November 2004.

[20] Tijs van Dam and Koen Langendoen. An adaptive energy-efficient MAC protocol for wireless sensor networks. In *Proceedings of the ACM SenSys 2003*, November 2003.

[21] Matthew J. Miller and Nitin H. Vaidya. A MAC Protocol to Reduce Sensor Network Energy Consumption Using a Wakeup Radio. *IEEE Transaction on Mobile Computing*, 4(3), May-June 2005.

[22] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In *Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networks (MobiCOM)*, August 2000.

[23] M. Gerla, T. J. Kwon, and G. Pei. On Demand Routing in Large Ad Hoc Wireless Networks with Passive Clustering. In *Proceedings of WCNC*, 2000.

[24] M. Younis, M. Youssef, and K. Arisha. Energy-aware routing in cluster-based sensor networks. In *Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, October 2002.

[25] P. Krishna, N. Vaidya, M. Chatterjee, and D. Pradhan. A Cluster-based Approach for Routing in Dynamic Networks. In *ACM SIGCOMM Computer Communication Review*, April 1997.

[26] Y. Xu, J. Heidemann, and D. Estrin. Geography-informed Energy Conservation for Ad Hoc Routing. In *Proceedings of the Seventh Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 2001)*, July 2001.

[27] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: An Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks. In *Proceedings of the Seventh Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 2001)*, July 2001.

[28] Young-Bae Ko and Nitin H. Vaidya. Location-aided routing (LAR) in mobile ad hoc networks. In *Mobile Computing and Networking*, pages 66–75, 1998.

[29] Y. Yu, D. Estrin, and R. Govindan. Geographical and energy-aware routing: A recursive data dissemination protocol for wireless sensor networks. Technical Report UCLA-CSD TR-01-002, UCLA Computer Science Department, May 2001.

[30] Jae-Hwan Chang and Leandros Tassiulas. Energy conserving routing in wireless ad-hoc networks. In *Proceedings of IEEE INFOCOM'00*, pages 22–31, 2000.

[31] R. Ramanathan and R. Rosales-Hain. Topology control of multihop wireless networks using transmit power adjustment. In *Proceedings of IEEE INFOCOM'00*, pages 404–413, 2000.

[32] Yong Yao and J. E. Gehrke. The Cougar Approach to In-Network Query Processing in Sensor Networks. *Sigmod Record*, 31(3), September 2002.

[33] Y. Sankarasubramaniam, O. Akan, and I. Akyildiz. ESRT: Event-to-Sink Reliable Transport in Wireless Sensor Networks. In *proceedings of the ACM MobiHoc Conference*, 2003.

[34] C. Wan, S. Eisenman, and A. Campbell. CODA: Congestion Detection and Avoidance in Sensor Networks. In *proceedings of the ACM SenSys 2003*, 2003.

[35] Jaewon Kang, Yanyong Zhang, and Badri Nath. TARA: Topology-Aware Resource Adaptation to Alleviate Congestion in Sensor Networks. *IEEE Transactions on Parallel and Distributed Systems*, 18, July 2007.

[36] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. SPINS: Security Protocols for Sensor Networks. In *Proceedings of the Seventh Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 2001)*, July 2001.

[37] A. Wood and J. A. Stankovic. Denial of Service Attacks in Sensor Networks. *IEEE Computer*, 35(10):54–62, October 2002.

[38] Adrian Perrig, David Wagner, and Jack Stankovic. Security in Wireless Sensor Networks. *Communications of the ACM*, 47(6), June 2004.

[39] Elaine Shi and Adrian Perrig. Designing Secure Sensor Networks. *IEEE Wireless Communications*, 11(6), December 2004.

[40] A. Mainwaring, J. Polastre, R. Szewczyk, and D. Culler. Wireless Sensor Networks for Habitat Monitoring. In *ACM International Workshop on Wireless Sensor Networks and Applications*, 2002.

[41] G. Simon, A. Ledeczi, and M. Maroti. Sensor Network-Based Countersniper System. In *Proceedings of the ACM SenSys 2004*, November 2004.

[42] Ning Xu, Sumit Rangwala, Krishna Kant Chintalapudi, Deepak Ganesan, Alan Broad, Ramesh Govindan, and Deborah Estrin. A Wireless Sensor Network For Structural Monitoring. In *Proceedings of the ACM SenSys 2004*, November 2004.

[43] Chao Gui and Prasant Mohapatra. Power Conservation and Quality of Surveillance in Target Tracking Sensor Networks. In *Mobicom'04*, September 2004.

[44] T. He, S. Krishnamurthy, J. A. Stankovic, T. Abdelzaher, L. Luo, R. Stoleru, T. Yan, L. Gu, J. Hui, and B. Krogh. An Energy-Efficient Surveillance System Using Wireless Sensor Networks. In *Proceedings of the ACM MobiSys 2004*, June 2004.

[45] J.F. Zeigler. Terrestrial Cosmic Rays. *IBM Journal of Research and Development*, 40(1):19–39, January 1996.

[46] F. Ye, G. Zhong, S. Lu, and L. Zhang. PEAS: A Robust Energy Conserving Protocol for Long-lived Sensor Networks. In *3rd International Conference on Distributed Computing Systems (ICDCS '03), Rhode Island*, May 2003.

[47] Honghai Zhang and Jennifer C. Hou. Maintaining sensing coverage and connectivity in large sensor networks. *Wireless Ad Hoc and Sensor Networks: An International Journal*, 1(1-2), January 2005.

[48] X. Wang, G. Xing, Y. Zhang, C. Lu, R. Pless, and C. Gill. Integrated coverage and connectivity configuration in wireless sensor networks. In *Proceedings of the ACM SenSys 2003*, pages 28–39, November 2003.

[49] T. Yan, T. He, and J. Stankovic. Differentiated Surveillance for Sensor Networks. In *Proceedings of the ACM SenSys'03*, November 2003.

[50] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin. Highly-resilient, energy-efficient multipath routing in wireless sensor networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(4):11–25, October 2001.

[51] A. Banerjea. Simulation study of the capacity effects of dispersity routing for fault tolerant realtime channels. In *Conference proceedings on Applications, technologies, architectures, and protocols for computer communications*, pages 194–205, 1996.

[52] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: a Tiny Aggregation Service for Ad-Hoc Sensor Networks. In *OSDI'02*, 2002.

[53] F. Ye, H. Luo, S. Lu, and L. Zhang. Dissemination protocols for large sensor networks. *Wireless sensor networks*, pages 109–128, 2004.

[54] B. Deb, S. Bhatnagar, and B. Nath. Reinform: Reliable information forwarding using multiple paths in sensor networks. In *28th Annual IEEE International Conference on Local Computer Networks*, 2003.

[55] Tian He, John A Stankovic, Chenyang Lu, and Tarek Abdelzaher. SPEED: A Stateless Protocol for Real-Time Communication in Sensor Networks. In *The 23th International Conference on Distributed Computing Systems*, 2003.

[56] L. Li and P. Sinha. Throughput and energy efficiency in topology-controlled multi-hop wireless sensor networks. In *The 2nd ACM international conference on Wireless sensor networks and applications*, 2003.

[57] N. Li and J. C. Hou. FLSS: A Fault-Tolerant Topology Control Algorithm for Wireless Networks. In *The 10th annual international conference on Mobile computing and networking*, 2004.

[58] Xiang-Yang Li, Peng-Jun Wan, Yu Wang, and Chih-Wei Yi. Fault tolerant deployment and topology control in wireless networks. In *The 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 117–128, 2003.

[59] D. M. Blough, M. Leoncini, G. Resta, and P. Santi. On the symmetric range assignment problem in wireless ad hoc networks. In *IFIP 17th World Computer Congress - TC1 Stream / 2nd IFIP International Conference on Theoretical Computer Science*, pages 71–82, 2002.

[60] L. Li, J. Y. Halpern, P. Bahl, Y. Wang, and R. Wattenhofer. Analysis of a cone-based distributed topology control algorithm for wireless multi-hop networks. In *the twentieth annual ACM symposium on Principles of distributed computing*, pages 264–273, 2001.

[61] R. Rajaraman. Topology control and routing in ad hoc networks: a survey. *SIGACT News*, 33(2):60–73, 2002.

[62] C. Bettstetter. On the minimum node degree and connectivity of a wireless multihop network. In *the 3rd ACM international symposium on Mobile ad hoc networking & computing*, pages 80–91, 2002.

[63] Y. Xu, J. Heidemann, and D. Estrin. Adaptive energy-conserving routing for multihop ad hoc networks. Research Report 527, USC/Information Sciences Institute, 2000.

[64] A. Cerpa and D. Estrin. ASCENT: Adaptive Self-Configuring sEnsor Networks Topologies. In *Proceedings of IEEE INFOCOM*, June 2002.

[65] Di Tian and Nicolas D. Georganas. A coverage-preserving node scheduling scheme for large wireless sensor networks. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 32–41, 2002.

[66] Guoliang Xing, Chenyang Lu, Robert Pless, and Joseph A. O'Sullivan. Co-Grid: An Efficient Coverage Maintenance Protocol for Distributed Sensor Networks. In *Information Processing in Sensor Networks (IPSN 2004)*, April 2004.

[67] J. Wu and H. Li. On calculating connected dominating set for efficient routing in ad hoc wireless networks. In *Proceedings of the 3rd international workshop on Discrete algorithms and methods for mobile computing and communications*, page 714, August 1999.

[68] M. Kochhal, L. Schwiebert, and S. Gupta. Role-based hierarchical self organization for wireless ad hoc sensor networks. In *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, page 714, September 2003.

[69] Y. Gao, K. Wu, and F. Li. Analysis on the redundancy of wireless sensor networks. In *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, pages 108–114, September 2003.

[70] A. Salhieh and L. Schwiebert. Power-aware Metrics for Wireless Sensor Networks. *International Journal of Computers and Applications*, 26(4), 2003.

[71] X. Y. Li, P. J.Wan, and O. Frieder. Coverage inWireless Ad-Hoc Sensor Networks. *IEEE Transactions on Computers*, 52(6), June 2003.

[72] S. Shakkottai, R. Srikant, and N. B. Shroff. Unreliable Sensor Grids: Coverage, Connectivity and Diameter. In *Proceedings of IEEE INFOCOM*, April 2003.

[73] Santosh Kumar, Ten H. Lai, and Jozsef Balogh. On k-Coverage in a Mostly Sleeping Sensor Network. In *Mobicom'04*, September 2004.

[74] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-Efficient Communication Protocols for Wireless Microsensor Networks. In *Proceedings of the Hawaian International Conference on Systems Science*, January 2000.

[75] Douglas M. Blough and Paolo Santi. Investigating Upper Bounds on Network Lifetime Extension for Cell-Based Energy Conservation Techniques in Stationary Ad Hoc Networks. In *Proceedings of MobiCom'02*, September 2002.

[76] Manish Bhardwaj and Anantha P. Chandrakasan. Bounding the Lifetime of Sensor Networks Via Optimal Role Assignments. In *Proceedings of IEEE INFOCOM*, pages 1587–1596, June 2002.

[77] Barbara Hohlt, Lance Doherty, and Eric Brewer. Flexible Power Scheduling for Sensor Networks. In *Information Processing in Sensor Networks (IPSN 2004)*, April 2004.

[78] J. Wu and S. Yang. Coverage and Connectivity in Sensor Networks with Adjustable Ranges. In *2004 International Workshop on Mobile and Wireless Networking (MWN)*, August 2004.

[79] K. Langendoen and N. Reijers. Distributed localization in wireless sensor networks: a quantitative comparison. *Comput. Networks*, 43(4):499–518, 2003.

[80] T. He, C. Huang, B. Blum, J. Stankovic, and T. Abdelzaher. Range-free localization schemes for large scale sensor networks. In *Proceedings of the 9th annual international conference on Mobile computing and networking (Mobicom 2003)*, pages 81 – 95, 2003.

[81] Z. Li, W. Trappe, Y. Zhang, and B. Nath. Robust Statistical Methods for Securing Wireless Localization in Sensor Networks. In *Proceedings of the IEEE/ACM IPSN'05*, 2005.

[82] N.Bulusu, J. Heidemann, D. Estrin, and Tommy Tran. Self-configuring Localization Systems: Design and Experimental Evaluation, August 2002.

[83] S. Tilak, N. B. Abu-Ghazaleh, and W. Heinzelman. Infrastructure tradeoffs for sensor networks. *ACM Workshop on Wireless Sensor Networks and Applications (WSNA)*, September 2002.

[84] S. Mohan, F. Mueller, D. Whalley, and C. Healy. Timing analysis for sensor network nodes of the atmega processor family. In *Proceedings of the 11th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2005.

[85] NS-2. http://www.isi.edu/nsnam/ns/.

[86] John Paul Walters, Zhengqiang Liang, Weisong Shi, and Vipin Chaudhary. *Security in Distributed, Grid, and Pervasive Computing*, chapter 17. Auerbach Publications, CRC Press, 2006.

[87] Mayank Saraogi. Security in wireless sensor networks. In *ACM SenSys*, 2004.

[88] C. Karlof, N. Sastry, and D. Wagner. Tinysec: A link layer security architecture for wireless sensor networks. In *Proceedings of Second ACM Conference on Embedded Networked Sensor Systems*, pages 162–175, November 2004.

[89] M. Cardei and J. Wu. Energy-Efficient Coverage Problems in Wireless Ad Hoc Sensor Networks. *Journal of Computer Communications on Sensor Networks*, 2004.

[90] Shengchao Yu, A. Yang, and Y. Zhang. DADA: A 2-Dimensional Adaptive Node Schedule to Provide Smooth Sensor Network Services against Random Failures. In *Proceedings of the Workshop on Information Fusion and Dissemination in Wireless Sensor Networks*, 2005.

[91] Shengchao Yu and Yanyong Zhang. R-Sentry:Providing Continuous Sensor Services Against Random Node Failures. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2007.

[92] Wade Trappe and Yanyong Zhang. NeTS-NOSS: PARIS: A Framework for Privacy Augmented Relaying of Information from Sensors. *National Science Foundation*, (0435043), 2004-2007.

[93] P. Kamat, Y. Zhang, W. Trappe, and C. Ozturk. Enhancing source-location privacy in sensor network routing. In *the 25th IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2005.

[94] Pandurang Kamat, Wenyuan Xu, Yanyong Zhang, and Wade Trappe. Temporal privacy in wireless sensor networks. In *27th International Conference on Distributed Computing Systems (ICDCS)*, 2007.

[95] C. Ozturk, Yanyong Zhang, and Wade Trappe. Source-location privacy in energy-constrained sensor network routing. In *2nd ACM workshop on Security of ad hoc and sensor networks*, 2004.

[96] J. Deng, R. Han, and S. Mishra. traffic analysis attacks in wireless sensor networks. In *First IEEE/CreateNet Conference on Security and Privacy for Emerging Areas in Communication Networks*, 2005.

[97] S. Zhu, S. Setia, and S. Jajodia. Leap: efficient security mechanisms for largescale distributed sensor networks. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 62–72. ACM Press, 2003.

[98] C. Karlof and D. Wagner. Secure routing in sensor networks: Attacks and countermeasures. In *First IEEE International Workshop on Sensor Network Protocols and Applications*, May 2003.

[99] J. Newsome, E. Shi, D. Song, and A. Perrig. The sybil attack in sensor networks: analysis & defenses. In *Proceedings of the third international symposium on Information processing in sensor networks*, pages 259–268. ACM Press, 2004.

[100] J. Douceur. The sybil attack. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS?02)*, February 2002.

[101] B. Parno, A. Perrig, and V. Gligor. Distributed detection of node replication attacks in sensor networks. In *Proceedings of IEEE Symposium on Security and Privacy*, May 2005.

[102] X. Wang, W. Gu, K. Schosek, S. Chellappan, and D. Xuan. Sensor network configuration under physical attacks. *Technical Report (OSU-CISRC-7/ 04-TR45), Dept. of Computer Science and Engineering, The Ohio-State University*, July 2004.

[103] X. Wang, W. Gu, S. Chellappan, Dong Xuan, and Ten H. Laii. Search-based physical attacks in sensor networks: Modeling and defense. *Technical report, Dept. of Computer Science and Engineering, The Ohio-State University*, February 2004.

[104] C. Hartung, J. Balasalle, and R. Han. Node compromise in sensor networks: The need for secure systems. *Technical Report CU-CS-988- 04, Department of Computer Science, University of Colorado at Boulder*, 2004.

[105] H. Chan, A. Perrig, and D. Song. Random key predistribution schemes for sensor networks. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, page 197. IEEE Computer Society, 2003.

[106] L. Eschenauer and V. D. Gligor. A key-management scheme for distributed sensor networks. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 41–47. ACM Press, 2002.

[107] J. Hwang and Y. Kim. Revisiting random key pre-distribution schemes for wireless sensor networks. In *Proceedings of the 2nd ACM workshop on Security of Ad hoc and Sensor Networks (SASN'04)*, pages 43–52. ACM Press, 2004.

[108] D. Liu, P. Ning, and R. Li. Establishing pairwise keys in distributed sensor networks. *ACM Trans. Inf. Syst. Secur*, 8(1):41–77, 2005.

[109] H. Chan and A. Perrig. Pike: Peer intermediaries for key establishment in sensor networks. In *Proceedings of Infocom 2005*, 2005.

[110] N. Gura, A. Patel, A. Wander, H. Eberle, and S. Shantz. Comparing elliptic curve cryptography and rsa on 8-bit cpus. In *Workshop on Cryptographic Hardware and Embedded Systems*, August 2004.

[111] R. Watro, D. Kong, S. Cuti, C. Gardiner, C. Lynn, and P. Kruus. Tinypk: securing sensor networks with public key technology. In *Proceedings of the 2nd ACM workshop on Security of Ad hoc and Sensor Networks*, pages 59–64. ACM Press, 2004.

[112] D. J. Malan, M. Welsh, and M. D. Smith. A public-key infrastructure for key distribution in tinyos based on elliptic curve cryptography. In *Proceedings of First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, 2004.

[113] L. Lazos and R. Poovendran. Secure broadcast in energy-aware wireless sensor networks. In *IEEE International Symposium on Advances in Wireless Communications*, 2002.

[114] L.A. Belady. A study of replacement algorithms for virtual storage computers. *IBM Systems Journal*, 5:78–101, 1966.

[115] M.W. Rhodehamel. The bus interface and paging units of the i860 microprocessor. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 380 – 384, October 1989.

Curriculum Vita

Shengchao Yu

2012 Ph.D. in Electrical and Computer Engineering, Rutgers, The State University of New Jersey, New Jersey

2008 M.S.. in Electrical and Computer Engineering, Rutgers, The State University of New Jersey, New Jersey

2002 M.S.. in Information System, Peking University, Beijing, China

1999 B.E. in Electrical Engineering, Tsinghua University, Beijing, China

2003-2008 Graduate Research Assistant, WINLAB, ECE Department, Rutgers University, New Jersey

2002-2003 Graduate Research Assistant, ECE Department, Iowa State University, Iowa

2008 Y. Guo, S. Yu, H. Liu, S. Mathur, and K. Ramaswamy, "Supporting User Interactivity in a Mesh-based P2P VoD System", Fifth IEEE Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, January, 2008

2007 S. Yu and Y. Zhang, "R-Sentry: Providing Continuous Sensor Services against Random Node Failures", the IEEE International Conference on Dependable System and Network (DSN), 2007

2007 Y. Guo, S. Mathur, K. Ramaswamy, S. Yu, and B. Patel, "PONDER: Performance Aware P2P Video-on-Demand Service", in proceedings of IEEE GLOBECOM, November, 2007

2006 Y. Guo, S. Mathur, K. Ramaswamy, S. Yu, and B. Patel, "Ponder: Providing commercial-quality video-on-demand service using peer-to-peer network", in Technical report, Corporate Research, Thomson Inc., 2006

2005 S. Yu, A. Yang, and Y. Zhang, "DADA: A 2-Dimensional Adaptive Node Schedule to Provide Smooth Sensor Network Services against Random Failures", WICON Workshop on Information Fusion and Dissemination in Wireless Sensor Networks, 2005

2004 J. Kang, B. Nath, Y. Zhang, and S. Yu, ”Adaptive Resource Control Scheme to Alleviate Congestion in Sensor Networks”, 1st Workshop on Broadband Advanced Sensor Networks, 2004