PETRA: TOWARD DEPENDABLE AND AUTONOMIC NETWORKED SENSOR SYSTEMS

by

SAFWAN AL-OMARI

DISSERTATION

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

2008

MAJOR: COMPUTER SCIENCE

Approved by:

_____

Advisor **Date**

_____

_____

_____

_____

ACKNOWLEDGEMENTS

I would like to thank all of the people whose inspiration, support and continuous encouragement help me reach the finish line of this long and tough journey. My advisor, Dr. Weisong Shi, is at the top of this list. He always act as a live example of a consistent, hard-working, and optimistic leader whose support and guidance inspired me to overcome all of the difficult times. I would like also to thank all of the committee members, Dr. Loren Schwiebert, Dr. Hongwei Zhang, and Dr. Nabil Sarhan, for their constructive discussion and criticism. Special thanks to my youngest brother Raja for his helpful occasional research discussions. My sincere thanks also to my MIST Lab fellow members and alumni, Kewei, Sean, Harry, James, Junzhao, Yong, Zhifeng, Brandon, Tung, Hui, Chenjia, Kevin, and all of the other new members, whose discussions and input helped me refine and articulate my thoughts and ideas. I am also thankful to the support and help of our collaborators from the Department of Civil Engineering, Dr. Carol Miller and Tim.

Finally, I would like to convey my endless appreciation of my parents, to whom I will be always in debt for their inspiration, help, and support. I would like also to thank my wife for her continuous support, Amr and Jad (my kids) from whom I will always find inspiration and energy.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Since its conception, Wireless Sensor Networks (WSNs) have been envisioned to operate autonomously for long period of time and without close supervision and human intervention. This key requirement demands that WSNs incorporate features such self-configuration, self-organization, as well as dependability. Self-configuration and self-organization requires the development of adequate system-level support that enables the sensor nodes to automatically and continuously discover its surroundings, including its neighbors and topology information, and to organize into a connected network. On the other hand, node failures, which are the norm rather than the exception in WSNs, should not render the WSN unoperational. In other words, the WSN should be able to withstand node failures and heal on its own and without external intervention. In this thesis, we propose to develop analytical models, system-level support, and network protocols that move WSNs forward toward implementing this vision (i.e., autonomous dependable WSNs).

Our work falls into two major categories. First, we use analytical models from classical reliability theory to formalize and quantify WSN dependability characteristics, these models allow for the systematic integration of WSN dependability as a primary dimension in the design space of the communication stack protocols and algorithms as well as the deployment problem. Perhaps the most prevalent node failures are those caused by harsh environmental conditions. Therefore, we consider non-malicious fail-stop node failures and focus on WSNs reliability and availability aspects of dependability. In order to support some of the assumptions this category, we took the opportunity to analyze failure traces of sensing units. These traces are made available to us from a real-world water system surveillance application. Second, we propose a novel structure of the WSN communication stack based on our Sensor Core (*Score*). In the context of *Score*, we propose and develop fundamental network services that perform automatic low-level discovery services such as

neighbor and topology discovery. *Score* along with discovery services provide a framework that hosts our protocols and provide them with system-level support to collaborate and coordinate with each other in a cross-layer approach.

We utilize our WSN reliability models in attacking two important WSN problems: the deployment problem and the node scheduling problem. The deployment problem is concerned with finding an optimal plan that meets user-defined WSN availability requirement with least total cost. A particular deployment strategy may stipulate an incremental deployment of 10 sensor nodes every month, another deployment strategy, may perform node deployments re-actively whenever a sensor node fails. Each strategy results in different WSN availability and total cost including cost of sensor nodes and cost of conducting the field trips to deploy these nodes. The optimal deployment strategy is driven, in large, by the ratio of the sensor node cost to the field trip cost. In other words, a particular deployment strategy could be the optimal one in a particular application, however, in a different application with different cost ratio, the same strategy may fail to yield the best results (i.e., meets availability requirement with minimum total cost).

Node scheduling protocols have been proven efficient in leveraging high node redundancy to extend the lifetime of the WSN beyond the limited lifetime of a single sensor node. Two nodes are called redundant if both nodes are equivalent in terms of some specific functionality (e.g., connectivity, coverage, etc). A typical node scheduling protocol identifies node redundancy in terms of some functionality and schedule these nodes `ON` and `OFF` to save energy. We argue that in addition to connectivity and coverage, WSN availability/reliability should be considered as a primary factor when scheduling nodes `ON` and `OFF`. After all, turning some nodes `OFF`, decreases the WSN reliability. The first step in introducing WSNs reliability/availability into the design space of the node scheduling protocols is the quantification of WSN reliablity/availability in terms of the reliability of the underlying sensor nodes.

WSNs are believed to be application-specific. Different applications impose different requirements and even different research challenges. Furthermore, different WSN application environments

yield different sensor node failure models, which imply different WSN reliability behavior and mandate different algorithms from the reliability point of view. Therefore, we follow an application-driven approach in our protocol design targeting three different applications; large-scale environmental applications with irregular and random sensor node placement, in-door mission-critical applications, and environmental application with uniform linear sensor node placement. We refer to the first application simply as *environmental*, the second as *SAIL*, which stands for Sensor-Assisted Independent Living Laboratory, and the third as Bridge Scour. A brief introduction to these applications is presented in Chapter 3. For example, in the *environmental* application, connectivity is an important requirement. On the contrary, in the SAIL and Bridge Scour applications, where the sensor nodes are deployed in a much smaller area with controlled node placement, connectivity does not pose a big challenge as the network topology is relatively fixed. Therefore, different node scheduling protocols are required in each application context. In the following two sections, we present a brief introduction to our approach.

1.1 WSN Dependability

Motivated by the fact that sensor nodes are cheap and unreliable devices, we argue that WSN dependability including reliability and availability should play a central role in shaping the WSN protocol stack design and the design and implementation of these protocols. Rather than focusing on failure detection and recovery and proposing arbitrary fault-tolerant protocols, we follow a more systematic approach in addressing WSN dependability issues by formalizing and quantifying WSN reliability in terms of the reliability of the underlying sensor nodes. This approach not only helps in our protocol design, but also allows us to answer more fundamental questions concerning deployment parameters such as deciding the number of nodes required to meet some reliability goals.

In classical reliability theory, a system is viewed as a set of independent components that are connected serially, in parallel, or as compromise (i.e., $\kappa - out - of - m$ system). In a serial system, the system is functioning if and only if all of its components are functioning. In a parallel system, the system is functioning as long as at least one component is functioning. In a $\kappa - out - of - m$ system, at least κ components are required to be functioning for the system to be considered functioning.

A parallel system can be easily mapped into sensor node cluster, the set of clusters can be mapped into a serial system, therefore, the overall WSN can be mapped into a $\kappa - out - of - m$ system. The natural analogy between WSNs and classical systems is the fundamental motivation behind our approach of using classical reliability theory techniques in our work. In addition to this analogy, we have three more important reasons for using classical reliability theory in addressing WSN dependability. First, WSN are deeply embedded in the physical world, which makes them susceptible to the same environmental conditions and suffer similar wear and tear effects to those components of classical systems. Second, using component redundancy is a well-accepted and well-developed approach in boosting overall system reliability in classical systems. Likewise, sensor node redundancy is a well-accepted and promising solution to overcome resource limitations including sensor node unreliability. Third, the failure of one sensor node does not cause the failure of other nodes in WSNs (i.e., independent sensor node failures). Nonetheless, all sensor nodes are expected to follow the same failure model as they suffer the same environmental conditions.

1.2 System Support

Unlike our top-down approach in the first category, in the second category, we follow a bottom-up approach by proposing a novel communication stack structure based on the *Score* framework and implementing low-level system components to support and host protocols developed in the first category.

Contrary to the current Internet protocol stack structure, in which a cross-layer interface is confined to adjacent layers, researchers believe that the communication stack of WSN should be loosely structured in order to allow for an arbitrary interface between the different protocol components. This interface helps in achieving more optimal operation in the resource-limited sensor nodes. Protocol optimizations are usually possible by allowing layers to collaborate more closely when performing their functions, this technique is widely known as *cross-layer design*. We propose *Score* as a framework that facilitates an arbitrary interface while maintaining a modular communication stack. As a core module, *Score* provides other components with the means to maintain and access the *neighbor set* and the *operational state*, which are the fundamental pieces of information

on which all the network components base their actions and optimization. A primary component in the *Score* framework is the *neighbor discovery service*, which probes the nodes in the vicinity to populate *Score* with neighboring nodes. In addition to *active* probing, the neighbor discovery service *passively* intercepts all incoming messages to maintain the neighbor set. *Passive* neighbor discovery mode helps to reduce probe messages and as a result save energy.

1.3 Contribution

In this section we summarize our contributions:

1. *Availability modeling*: we model the availability of WSN in the context of two applications; in-door and large-scale environmental applications. Using these models, we prove that availability/reliability can be introduced successfully into the design space of the node scheduling protocols in a systematic way. This work also serves as a motivation of the need for node scheduling as far as availability/reliability is concerned. we build upon our availability models to tackle two very important WSN problems; WSN deployment problem and node scheduling.

2. *WSN deployment problem*: we define the deployment problem as the problem of deciding the number of sensor nodes and the way these nodes deployed in order to meet user-defined availability requirement with minimum total cost. We formulate WSN deployment problem as an optimization problem, whose solution yields the best deployment strategy including the number of nodes and the number of field trips required over the WSN lifetime.

3. *Node scheduling*: we employ similar availability models to attack the node scheduling problem for in-door WSN applications. Unlike in the deployment problem, we adopt usage-based sensor node failure patterns to capture aging and increasing failure rates. Our analytical and simulation results prove that node scheduling, indeed, improves availability behavior.

4. *Sensor failure traces analysis*: we take an initial step and analyze a one-month sensor failure traces collected from a real-world water system surveillance application. We believe that

this work represents a small step, however, an important one in the right direction toward understanding sensor failure patterns in environmental applications. This experimental-driven approach helps us build more practical protocols and realistic models. In this application, we found that external harsh environmental conditions may be the most important factor on inflicting failures. Furthermore, communication failures, mainly caused by the lack of time synchronization, contribute the largest portion among all failure types.

5. *Redundancy-Aware Topology Control (RAT)*: unlike in the SAIL and Bridge Scour application contexts, in the *environmental with random deployment* application, network connectivity is a major factor that imposes limitations on the node scheduling design space in addition to reliability. Therefore, we address connectivity in the *RAT protocol*, which is discussed in Chapter 8.

6. *Score*: a novel framework that facilitates other network components to collaborate in an arbitrary fashion while maintaining a modular communication stack. A core service in the context of *Score* is the *neighbor discovery* service. This service performs passive as well as active probing and populates *Score* with the neighbor set for other protocols' benefit.

7. Topology discovery service: maintains several topology parameters to support cross-layer design. The topology parameters are accessible by other protocol components through *Score*. As a demonstration of the benefits of this service, we propose *Redundancy-Aware Controlled flooding* (RAC) and *Density-Aware GPSR* (DA-GPSR) protocols.

1.4 Organization

In Chapter 2, we briefly introduce wireless sensor networks with emphasis on node scheduling protocols; Chapter 3 presents three target applications including research challenges, assumptions, and adequate solutions; In Chapter 4, we tackle the deployment problem and discuss optimal solution from the availability and total cost perspectives; Chapter 5 presents our work of modeling the availability of WSNs in the context of the SAIL application; Our real-world sensing failure traces

are presented in Chapter 6; In Chapter 7, we present our *Score* framework including the *negihbor discovery* service, the *topology discovery service*, and finally the *RAC* and *DA-GPSR* protocols as a demonstration of the benefits of the *topology discovery* service. We address connectivity in the context of the environmental application through the *RAT* protocol in Chapter 8. Finally, we conclude our work and discuss future research directions in Chapter 9.

CHAPTER 2

BACKGROUND

We dedicate this chapter to present some background material including overview of Wireless Sensor Networks (WSNs), node scheduling protocols, and basic concepts in reliability theory.

2.1 Overview of Wireless Sensor Networks

The recent development of small wireless sensing devices has opened the door for several new applications. These applications have the potential in changing the way we perceive, monitor, and interact with the physical world. Many application domains have already been mentioned in the literature [3, 18, 73, 76, 79, 98]. For example, sensor nodes can be deployed in the battlefield to monitor and track the enemy's troop movement (i.e., military applications). Sensor nodes can also provide attractive solutions for challenging medical problems; for example, small bio-sensors can function as an artificial retina that replaces a damaged one [73] when deployed inside the human body. Environmental monitoring is another promising application domain for wireless sensor networks. For example, wireless sensor networks can be used to monitor pollution levels in water resources and soil. Also, wireless sensor networks can be used to track and monitor volcanic and earthquake activities, this allows for the early detection of any activities, which helps in avoiding or at least mitigating disasters. Like in out-door applications, WSNs have many promising in-door applications. For example, WSNs can be deployed in the apartments of elderly people providing an innovative non-obtrusive tool to better monitor the activities of elders living in their apartments. In this capacity, WSNs facilitate connecting seniors to their caregivers and the communication of any emergency conditions such as falling of the elder.

Environmental monitoring is one of the earliest envisioned applications of wireless sensor networks. Therefore, research challenges of this particular application caught the attention of many

researchers, which resulted in a large body of work. In this application, researchers envision large-scale deployment, where a large number of sensor nodes are scattered in the sensor field. After deployment, these nodes should organize among themselves and form a multi-hop network and should remain functional for long time periods, several years in some applications [3, 76]. By functional, we mean that the network should be able to perform the original job the network is supposed to do, e.g., sensing the environment and sending the results back to some base station. Since the lifetime of individual sensor nodes is usually on the order of a few months, and that it is infeasible to re-power individual nodes by simply changing their battery, techniques to save the power consumption and so extend the lifetime of wireless sensor networks is vital in real life applications and have been extensively studied in the literature.

Researchers have considered several techniques to decrease power consumption at the different layers. At the application level, aggregation [19, 59, 63], where multiple data messages are diffused and integrated into a single data message, is one of the techniques used to lower the power consumption spent on transmitting redundant data messages from the data sources back to the sink. Several energy-aware routing protocols [33, 72, 74, 75, 95] have been discussed in the literature. In these protocols the current power level of the sensor node is considered by the routing protocol as a criteria in choosing routes between sources and destinations. Trading off network delay to power consumption was leveraged by MAC protocols to introduce low duty cycle operation modes, where nodes turn their radios off if not actively transmitting or receiving [38, 71, 94, 100].

Like other network protocols, topology control was used as yet another attempt to extend the network lifetime and reduce power consumption. The term "topology control" has been used in the literature in two contexts, some people [6, 56, 32, 52, 54, 57, 69] use the term to refer to the problem of adjusting the transmission power of the sensor nodes and so adjusting the network topology, while others [4, 12, 13, 15, 18, 30, 71, 80, 92, 104] use the term to describe the process of turning the sensor nodes radio on and off consequently controlling the network topology and decrease total power consumption. We refer to the latter as the node scheduling technique.

In node scheduling protocols [4, 12, 13, 15, 18, 30, 71, 80, 92, 104], node redundancy was exploited to extend the network lifetime. The key idea in this class of protocols is to identify node redundancy in terms of some functionality (usually communication), cluster redundant nodes together, and finally schedule nodes in the clusters for active or sleep modes in such a way the network is able to perform the original function adequately. Although all topology control protocols that fall in this category share the basic idea, the exact node redundancy definition and criteria for a node to stay active or switch to sleep are still the property of a specific node scheduling protocol.

2.2 Node Scheduling Protocols

Work on node scheduling protocols is widely diverse; we divide its evolution over the last five to six years into three stages. The bulk of the work in the first stage is pure theoretical analysis of the minimum connected dominating set problem. Centralized optimization algorithms are proposed to select the minimum connected dominating set [24, 83, 86], which is known to be an NP-hard problem. This work lacked the practicality which prevented its applicability in real life sensor network deployments. The reported sensor node wireless transceiver module power consumption profile made the node scheduling approach, among others, more appealing in extending the network lifetime, therefore it triggered new interest in the topic. In the second stage, researchers shifted their efforts to address the more practical issues of wireless communication links and investigate localized fully distributed algorithms to select the active node set [4, 12, 13, 15, 18, 30, 71, 80, 92, 104]. The primary goal of the node scheduling protocols in this stage is to select the optimal active node set that most extends the network lifetime, other network properties such as reliability, network capacity, and coverage are ignored. Proposing new node scheduling algorithms was not the primary concern in the third stage. Instead, researchers shifted their efforts again to study and investigate assumptions made in the second stage, such as the assumption that sparse networks improve network capacity. Furthermore, fault tolerance and coverage are considered as important network properties in addition to bare connectivity [6, 8, 21, 25, 66, 85, 91]. In the following subsection, we present several node scheduling protocols in more detail.

2.2.1 Node scheduling algorithm details

In this section, we elaborate on the protocol details of five representative state-of-the-art node scheduling algorithms including GAF, SPAN, ASCENT, ReOrg, and CCANS. Although these protocols share the same basic idea of identifying node redundancy and assigning redundant nodes active and sleep modes, the exact node redundancy definition and scheduling technique is different from one protocol to another.

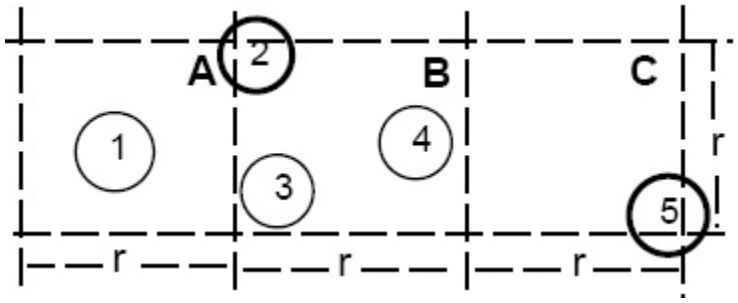


Figure 2.1: Virtual Grid Example: node 2 and node 5 can communicate with each other.

GAF

Based on the physical location and the assumed perfect disc communication range, GAF, in [92], divides nodes into non-overlapping virtual grid cells in such a way any two nodes from adjacent cells can communicate with each other and so only one active node in each cell is enough to maintain a connected network. This definition of virtual grids requires that distance between the furthest two nodes in any adjacent virtual cells be at most R, where R is the communication range. Figure 2.1 depicts three virtual cells: A, B, and C. In cells B and C, we can see that node 2 and node 5 can communicate with each other and so satisfy the definition.

After assigned virtual cells, nodes need to decide which node should remain active and handle routing and which nodes should go to sleep and save energy in each virtual cell. Initially, all nodes start in the `discovery` state, in which the nodes send hello messages to become familiar with nodes in the same virtual cell. To select only one node in each virtual grid, each node sets up a timer

for T_d, once the timer fires, the node switches to active and broadcasts a message telling others that it will take over routing for this virtual cell. If the node receives an active message from another node in the same virtual cell, the node cancels its timer and switches to sleep immediately. Figure 2.2 shows the detailed state transition diagram.

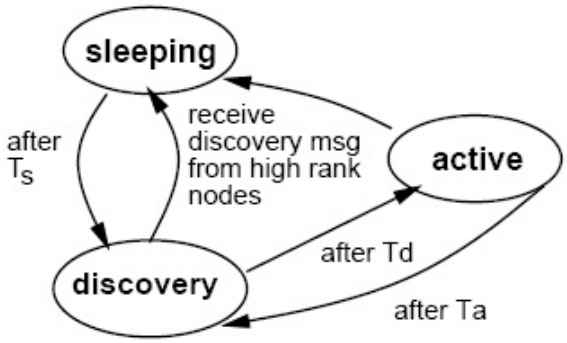


Figure 2.2: State transition diagram in GAF.

From Figure 2.2, we can see that $enat$ (Estimated Node Active Time), T_d (discovery message interval), T_a (node active duration), and T_s (node ranking) can be tuned and adapted to target different requirements of different applications. For example, $enat$ can be set in such a way to balance power consumption among all nodes, or it can be set to make an active node stay active until it drains all of its power. In applications, where nodes can move from virtual cell to another, GAF considers another factor in determining $enat$. GAF uses estimated node grid time, which is the time a node will stay in the same virtual cell (denoted as $engt$), as the $enat$ value. Therefore, before a node moves out of the virtual cell, other nodes will switch from sleep to discovery to take over routing in that virtual cell.

The attractive feature of GAF is that it does not incur high overhead as the virtual grid can be built statically once the communication range is known. The key idea in GAF is that the length of the diagonal crossing any two adjacent cells is less than or equal to the communication range. This accounts for the furthest possible distance between two sensor nodes located at opposite corners of any adjacent cells. Obviously, any two nodes located in the same cluster can communicate with

each other as the dimensions of a single cell has to be less than the communication range. Therefore, having a single active node in each cell is enough to maintain a connected network.

SPAN

SPAN [15] selects a set of coordinator nodes, which stay active all the time and act as a backbone for the network, all other nodes can go to `sleep`. To maintain connectivity, the coordinator nodes have to form a connected backbone, and every non-coordinator node has to be connected (i.e., within the communication range) to at least one coordinator. To select the set of coordinators, all the nodes in the network periodically broadcast "HELLO" messages. The "HELLO" messages consist of the node status (whether a coordinator or not), the node's current set of coordinators, and the node's set of neighbors. In addition to the neighbor set, each node has to maintain two-hop coordinator information (direct coordinators and coordinators of its neighbors). Exchanging "HELLO" messages prepares nodes for the second stage of selecting the set of connected coordinators.

The coordinator eligibility rule at any node i simply states that if node i has any two of its neighbors (nodes k and j), which can not communicate through existing coordinators, then i needs to become a coordinator. Since node i is only aware of coordinators in its two-hop communication range, that means node k and node j can not reach other in at most 2 coordinator hops (i.e., could reach each other through three or more hop coordinators). Before making node i simply become a coordinator if eligible for that, SPAN uses random back off time to delay the decision. During the delay, if the node receives a coordinator announcement, the node re-evaluates its eligibility as a coordinator, otherwise it becomes a coordinator and sends a coordinator announcement.

The random back off time allows SPAN to minimize the total number of coordinators and achieves load balancing among nodes. The back off time minimizes the total number of coordinator nodes for two reasons: first, because it does not allow two nodes to become coordinators at the same time so there are no redundant coordinators in the network. Second, the delay considers the node utility, the more the node's utility is, the less the back off delay. Node i utility simply is the number of nodes that will be connected as a result of node i becoming a coordinator. The

back off time achieves load balancing by considering the remaining power of the node as a factor in the random back off time; nodes with higher remaining power have higher priority to become coordinators. The following formula represents the delay.

$$delay = \left(\left(1 - \frac{E_r}{E_m} \right) + \left(1 - \frac{C_i}{\binom{N_i}{2}} \right) + R \right) \times N_i \times T$$

N_i is the number of nodes neighbors to node i. C_i represents the additional number of node pairs that node i will connect if becomes a coordinator. Therefore, a node with higher C_i becomes coordinator first, and so, fewer coordinator nodes are needed. To avoid collisions, a random value R proportional to N_i and T is used, T is the round trip time of a small packet over the wireless link. E_r and E_m represent the remaining and maximum energy (in Joules) at the node, respectively. Consequently, a node with smaller $1 - \frac{E_r}{E_m}$ will become a coordinator first.

Node i can withdraw from being a coordinator once all of its neighbors can communicate directly or indirectly (through two-hop coordinators) without the help of node i. This gives the opportunity to other nodes to become coordinators. However, to prevent temporary disconnection in the network, before a node switches to sleep if it decided to withdraw as a coordinator, it waits for a while until other nodes assess the situation and suitable node(s) become coordinator(s).

ReOrg

The authors in [18] propose two protocols, ReOrg and ReSync. The former is a node scheduling protocol, while the latter is an energy-saving MAC protocol. These protocols are combined to achieve a higher level of energy saving. As our focus is node scheduling protocols, we present ReOrg in more detail.

ReOrg first builds a relay backbone by choosing primary relay nodes, the criteria for becoming a primary relay node are remaining energy and node degree. These metrics measure the suitability of a node to become a router (i.e., primary relay node). To fill the gaps between relay nodes (if any), non-relay nodes become relay nodes based on their ability to bridge primary relay nodes. Figure 2.3 depicts a set of relay nodes selected by ReOrg.

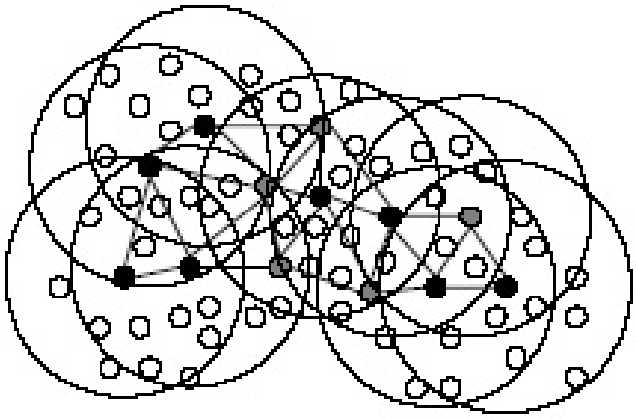


Figure 2.3: Example relay backbone selected by ReOrg in a 70 node network, requiring 19% of nodes to be relays.

The ReOrg protocol proceeds in two phases. In the first phase, nodes exchange "OrgHello" messages in order for each node to identify its immediate neighbors and select the neighbor with the best relay metric (remaining energy and node degree) as its primary relay. A node is considered a neighbor only when the pair-wise link quality is equal to or higher than 80%. Nodes inform their chosen primary relays using a field in the "OrgHello" messages. After primary relays have been chosen, there may be some gaps in the backbone, so non-relay nodes go further with phase II to fill any existing gaps.

In the second phase of ReOrg, each node identifies relays in its two-hop neighborhood and learns whether these relays are connected or not, either directly or through neighboring relays. In order to do that, each non-relay node has to maintain a soft state list of all the primary relays within two hops, and tags each whether it is reachable or not. To minimize the number of selected secondary relay nodes, ReOrg uses the number of primary relays that will be connected by the node, in addition to the normalized relay metric used in the first phase (i.e., remaining energy and node degree). ReOrg resorts to timers to select best secondary relays. Each node sets a timer inversely proportional to the weighted average of the number of disconnected primary relays in its 2 hop neighborhood and the

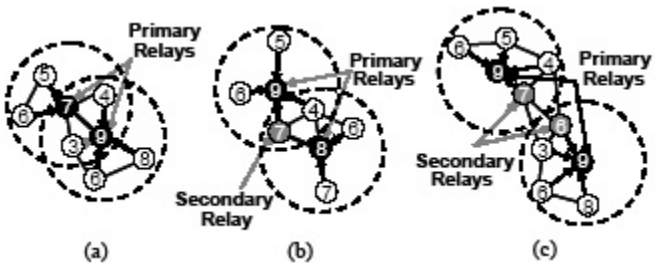


Figure 2.4: Example ReOrg backbone selection. Cases requiring no secondary relays (a), one secondary relay to directly connect primary relays (b), and two secondary relays to connect primary relays (c). (Relay metric indicated in each node)

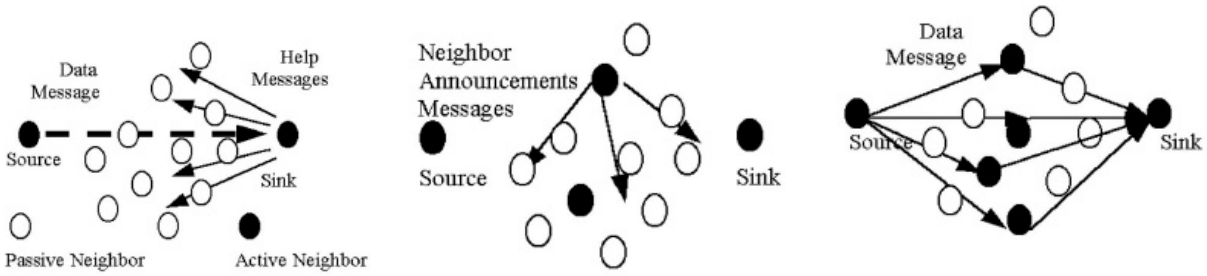
normalized relay metric (remaining energy and node degree). Figure 2.4 shows an example scenario of ReOrg selecting primary and secondary relays.

In Figure 2.4(a), ReOrg selects two nodes as primary relays. As the primary relays are connected, there is no need for secondary relays. In Figure 2.4(b), one secondary relay is enough to connect the two primary relays. Finally, two secondary relays are required in Figure 2.4(c) to bridge the two-hop gap between primary relays.

The rate at which nodes send "ReOrgHello" messages is an important parameter that trades off overhead versus resilience to topology changes. Sending "ReOrgHello" messages at a high rate allows ReOrg to adapt to node failures very quickly, but incurs high protocol overhead, which may limit ReOrg's potential in saving energy. Unlike SPAN, where nodes need to exchange and store the entire neighbor tables in their two-hop neighborhood, ReOrg requires nodes to exchange and store only primary relays. In highly dense topologies, the number of two-hop primary relays tends to be much smaller than the number of immediate neighbors; this limits ReOrg overhead to $o(n)$.

ASCENT

Nodes in ASCENT [12, 13] independently assess the current network performance based on the node density and packet loss rates. If a node experiences high loss rates due to collisions, then it may decide to go to `sleep` to lower the contention in the network. On the other hand, if the node density drops below some threshold, the node joins the network by switching its radio on.



(a) (b) (c)

Figure 2.5: ASCENT active node selection example: (a) communication hole (b) intermediate ASCENT state (c) final ASCENT adaptation.

Figure 2.5 shows a complete ASCENT example scenario, in Figure 2.5(a) a set of nodes are shown where a communication hole exists between the source and destination. As the sink node experiences high loss rate when receiving messages from the source node, it asks nodes in the vicinity to join the set of active nodes and so improve communication quality. Figure 2.5(b) shows an intermediate state where some nodes start to switch to active to compensate for the communication hole. When switching to active, a node broadcasts neighbor announcement messages to inform other nodes that it became active. Finally, Figure 2.5(c) shows the final state after the network stabilizes and a set of active nodes are selected by ASCENT.

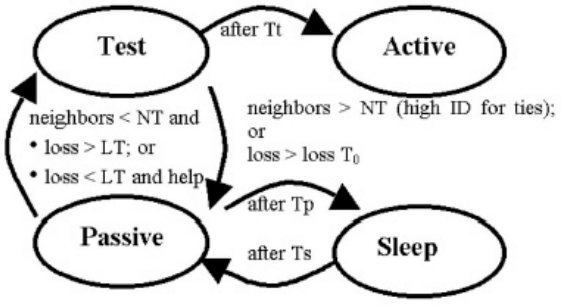


Figure 2.6: ASCENT state transition:T_t, T_p, and T_s denote test, passive, and sleep timers respectively. NT denotes neighbor threshold, LT denotes loss threshold. Loss at T_0 is the loss rate before the node entered the test state.

Nodes in ASCENT could be in one of four states: `sleep`, `passive`, `test`, and `active`. Figure 2.6 shows the detailed state transition diagram. All nodes start in the `test state`, in which a node exchanges data and control messages for a time period of T_t. If the number of nodes reaches some threshold (NT) or the average data loss (DL) is higher than before the node enters the `test` state before T_t expires, the node switch back to the `passive` state. Otherwise, the node enters the `active` state. The intuition behind the `test` state is to give nodes an opportunity to probe the network to see if the addition of new nodes improves connectivity (i.e., decreases loss rate).

While in the `passive` state, the node passively listens on the wireless channel to keep track of the number of active nodes and the loss rate (DL), but the node does not send any data or control messages. A node stays in the `passive` state for T_p. The node switches back to the `test` state if one of two conditions happens, otherwise after T_p expires, the node switches to `sleep`. The two conditions are: the number of active neighbors is less than LT and either the loss rate is greater than DL or the loss rate is less than DL but there is a help announcement.

It is only upon entering the `sleep` state that a node can turn its radio completely off and save energy. A node in the `sleep` state remains asleep for T_s before it wakes up again and switches back to the `passive` state. Making sleeping nodes wake up and switch to `passive` state regularly is important to keep the network responsive to node failures, which results in a disconnected graph. Note that a node in the `active` state stays active until running out of energy.

Timers and thresholds discussed in the previous state transition diagram enrich ASCENT with the ability to tune its behavior and adapt to different application requirements. ASCENT tuning basically allows it to trade off power saving to network agility. For example, the loss threshold (LT) can be set to high values in applications that do not demand high connectivity, while set to low values in applications that are sensitive to packet loss. LT is the maximum loss rate an application can tolerate before a node asks for help from neighboring nodes. Note that high LT increases energy savings, whereas low LT decreases energy saving. Current ASCENT configuration assigns

the following values for the tuning parameters: NT is set to 4, LT is set to 20%, T_p and T_t are set to 2 and 4 minutes respectively.

Nodes in the `test` and `passive` states need to keep track of the number of neighbor nodes and the loss rate. A node is considered a neighbor if we receive a certain percentage of messages over time window. Therefore, ASCENT defines two new concepts, a history window function (CW) and a neighbor loss threshold (NLS). The CW represents the number of lost messages from a certain node over time, while, NLS is the maximum loss rate that a node can reach in order to be considered as a neighbor. The loss rate is maintained as an exponentially weighted moving average (EWMA) in the following form:

$$EWMA_{current} = \rho.\dot{C}W + (1 - \rho) \cdot EMWA_{previous}$$

The value of the constant filter ρ is set to 0.3. The number of neighbor nodes (N) is defined as the number of nodes with loss rate less than NLS. The NLS threshold in calculated as follows:

$$NLS = NLS - \frac{1}{N}$$

N represents the number of neighbors in the previous cycle. The intuition behind this formula is that highly congested channels (i.e., N is big) have high loss rates, and so, the NLS value should be more lenient (i.e., less value) than less congested channels. In other words, the larger the number of neighbors is, the higher NLS is, and so the higher the maximum allowed loss rate, and the more lenient the neighbor eligibility condition.

Maintaining the loss rate and node density in the case of transient node failures and high wireless channel quality variation may incur high communication overhead and adversely affect the integrity of a node's decision of going to `sleep` or `active`. Also, the lack of an interface between ASCENT and the routing layer may incur extra delays and overhead at the routing layer as the routing layer needs to recover broken routes if a node decided to switch to `sleep` in ASCENT.

CCANS

[104] selects a set of active nodes that guarantee sensing coverage and connectivity. The authors considered a two-dimensional sensor field of grid points. The following is some important terminology.

- $G = \{g_1, g_2, ..., g_m\}$ denotes the set of all grid points in the sensor field.
- $S = \{s_1, s_2, ..., s_n\}$ denotes the set of sensor nodes.
- Assume S_i is the set of nodes that can sense grid point g_i, then the probability of detection at point g_i is defined as:

$$p_i(S_i) = 1 - \prod_{s_k \in S_i} \left(1 - p_i^k\right)$$

  where p_i^k denotes the detection probability of point g_i by node s_k.

The CCANS (**C**overage-**C**entric **A**ctive **N**ode **S**election) protocol has two major phases. In the first phase coverage redundancy is evaluated. Based on this redundancy, the set of active nodes that provide sensing coverage is selected. The following formula defines point coverage redundancy of node s_k for point g_i. S_i denotes the set of nodes that provide sensing coverage of point g_i.

$$\xi_i^k = \frac{p_i(S_i \setminus \{s_k\})}{p_i(\{s_k\})}$$

A ξ_i^k value of zero means that no node other than s_k can provide sensing coverage for point g_i, while a value of one or greater, means that node s_k is completely redundant with respect to sensing coverage of point g_i. From the point of view of node s_k, if for all grid points within its sensing coverage can be covered by other nodes (i.e., $\{S_i \setminus \{s_k\}\}$), then node s_k can go to sleep without affecting sensing coverage. This is the basic idea behind the CCANS.

To maintain some fault tolerance in sensing coverage, CCANS keeps some degree of redundancy, which is referred to as the *threshold coverage redundancy*, denoted as ξ_{th}. So a node (s_k) is required to be active based on the following condition:

$$s_k \text{ turns } \begin{cases} \text{off}, & \text{if } \xi_i^k \geq \xi_{th} \\ \text{on}, & \text{otherwise} \end{cases}$$

It is obvious that the higher ξ_{th} is, the more nodes are needed to provide the required sensing coverage redundancy, and so, ξ_{th} can be used to trade off power saving to sensing coverage fault tolerance.

Nodes in CCANS could be in one of three states, `ACTIVE`, `SLEEP`, or `UNSET`. an `ACTIVE` node turns its radio and/or sensing units on, a `SLEEP` node turns both off, while an `UNSET` node is still undecided and may switch to `ACTIVE` or `SLEEP`. CCANS proceeds in two stages, in the first stage, nodes evaluate sensing coverage, while in stage two, nodes assess connectivity. CCANS is a token-based protocol, where the token is assigned to one node at a time (referred to as the token node). Only the token node can turn on or off, therefore, only one node switches `ACTIVE` or `SLEEP` at a time and no conflicts happen. Nodes first forward the token until no `UNSET` node exists in the network, and then nodes pass the token in the backward direction, until the token reaches the first node (sink).

During the first stage, the token node switches to `ACTIVE` or `SLEEP` based on its sensing coverage redundancy using the previous two formulas. However, this decision is not final as the node may have `UNSET` neighbors, which have been assumed to be `ACTIVE`. After that, the token node chooses one of its neighbors to pass the token to, the new token node does the same thing and passes the token until the token reaches all `UNSET` nodes in the network.

In the second stage, only `SLEEP` nodes check their state and connectivity, the state checking is the same as in stage one, except that no `UNSET` nodes exist in the network, which makes the node's decision final. After that, the node checks for connectivity. It is important to note that nodes execute the second stage while the token is being passed in the backward direction.

Figure 2.7 depicts an example scenario of several nodes passing the token back and forth until all the nodes are decided (i.e., no `UNSET` nodes). Figure 2.7(a) starts with all the nodes are `UNSET`, and Figure 2.7(h) ends with all nodes decided.

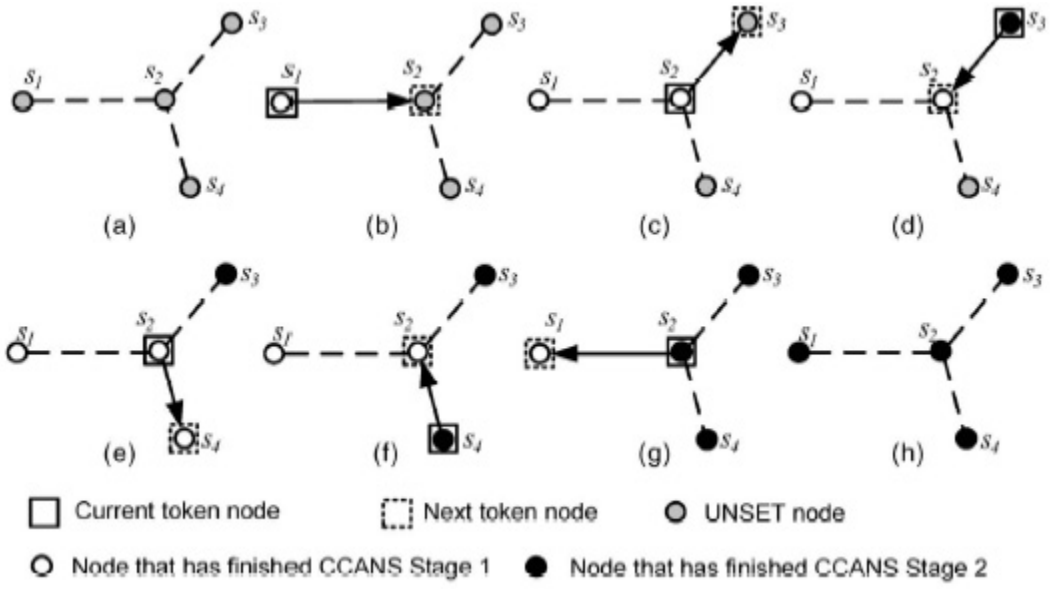


Figure 2.7: Example of token passing in CCANS.

In [30, 80], the authors consider sensing coverage instead of communication connectivity to dispatch nodes to `active` and `sleep` modes. Node i can switch to `sleep` once it can find a set of its neighbors, who's sensing area collectively covers the sensing area of node i. [80] considered symmetric as well as asymmetric sensing radii. However, unless the communication radius (r_c) and the sensing radius (r_s) have some relationship ($r_c \geq 2 \cdot r_s$), using the same off-duty eligibility rule for both sensing and radio module may result in a disconnected network. In the context of query execution, [30] finds an optimal set of nodes to execute the query (least number of active nodes). The selected nodes have to provide coverage as well as connectivity.

We can see that all previous protocols share the same basic idea to turn nodes off and on. First, these protocols identify node redundancy, and then require only one node in each set of redundant nodes to stay active, while allowing others to switch to `sleep`. Table 2.1 summarizes the different node scheduling protocols. Some researchers use the sensing coverage to identify node redundancy, while others use communication coverage to identify redundancy. Also, the exact way the nodes switch to `active` or `sleep` after becoming aware of node redundancy is different. Some papers

Table 2.1: Comparing node scheduling protocols. n is the average number of neighbors in a single-hop communication range.

Protocol	Redundancy Metric	Node Scheduling	Message overhead	Storage overhead	Evaluation
GAF	Physical location and perfect disc model to define communication equivalence.	Nodes within virtual cell exchange hello messages to turn one node on.	constant	constant	Simulation only.
SPAN	Uses two-hop neighborhood lists to explicitly define communication communication redundancy.	Nodes decide to become active or sleep based on the connectivity of the coordinator set + number of nodes that will be connected if the node awake + remaining energy.	All nodes announce state changes.	$o\left(n^2\right)$.	Simulation only.
ASCENT	Uses loss rate and node density to select active node set.	Nodes turn on and off independently based on current loss rate and node density, node scheduling does not guarantee connectivity.	All nodes announce state changes.	$o\left(n\right)$.	Simulation and experimental testbed.
ReOrg	Uses single-hop neighbor lists to explicitly define communication redundancy.	Nodes exchange hello messages to choose active nodes. Nodes with high node degree and most remaining energy are chosen first.	All nodes announce state changes.	$o\left(n\right)$.	Simulation and experimental testbed.
CCANS	Physical location and perfect disc model to define sensing redundancy.	A token passed back and forth in the network to make only one node turn on or off at a time.	All nodes announce state changes.	-	Simulation only.

make nodes decide on and off duty cycles using back-off times to break ties, while others allow only one node to change on or off at a time by means of tokens. Since the radio and the sensing modules are independent (i.e., modules can be turned on and off independently), using either connectivity or sensing coverage to define node redundancy are completely independent and both techniques can be applied concurrently in the same network. Connectivity redundancy results in network topology, and coverage redundancy results in sensing topology. Therefore, trying to find an optimal active node set that achieves connectivity and coverage at the same time is not a big advantage. On the contrary, an optimal active node set for connectivity may conflict with an optimal active node set for coverage.

2.2.2 Other work related to node scheduling

STEM in [71] exploits setup latency to save energy by making nodes aggressively go to `sleep`, while waking up frequently to see if other nodes wish to communicate with them. On its own, STEM is more comparable to energy saving MAC-layer protocols, such as S-MAC [94]. It is the fact that the authors combined STEM with GAF for higher levels of energy saving which makes this work related to the topic of this survey.

In [85] and their later work [91], the authors present theoretical analysis to study the relationship between coverage and connectivity, and to answer the question; does coverage implies connectivity? or vice versa. This work is important for the design of new node scheduling protocols; it allows the designers to focus on one problem (i.e., targeting connectivity or sensing coverage). The authors prove that if sensing radius is at least twice the communication radius, then sensing coverage implies connectivity. The assumption that the communication and coverage is a perfect disc limits the applicability of this kind of results in real life applications.

The authors in [8] presented extensive analysis to study the node scheduling approach in extending the network lifetime and to what degree this approach can leverage node density. This study considered one and two dimensional deployments with nodes uniformly distributed over the sensor deployment area. Three interesting results are drawn in the paper. First, in two dimensional

deployments, the best possible lifetime extension is 80%. Second, node scheduling techniques show a linear lifetime extension relationship to increasing node density. Third, overhead of the node scheduling protocol can severely decrease the protocol's potential in saving energy and so extending the network lifetime. These results provide some guidelines for designing node scheduling protocols. For example, SPAN does not show increasing lifetime extension with increasing node density, this due to the fact that SPAN overhead outweighs its advantage.

In [6], the authors consider a slightly different problem, which is concerned with network connectivity and its relation to node degree. Node degree can be adjusted either by adjusting the transmission power, or by turning nodes on and off. This problem is interesting in the context of node scheduling approach as it gives hints on the minimum node degree that a node scheduling protocol needs to maintain in order to have a connected network. For example, the minimum node degree can provide some insight to the ASCENT number of neighbors threshold value.

The allocation of the **B**ase **S**tation (BS) is considered in [66] in a two-tier topology. In the 1st tier, **S**mall **N**odes (SN) are grouped into clusters. Each cluster has at least one **A**pplication **N**ode (AN) that is reachable by all the SNs in the same cluster. ANs can adjust their transmission power to reach one of the BSs (2nd tier) in a single-hop transmission. An optimal allocation of the BSs minimizes the total power drained at the ANs.

The authors in [21, 11] raise three important aspects of the node scheduling protocol, which conflicts with what previous work intuitively agreed on. First, it is not-necessarily that lower node density decreases interference. Second, lower node densities increase the average path length, which eventually makes the network draw more energy. Third, it discusses the relationship between overhead and node density. Based on their theoretical analysis, if only 10% of the nodes are active, then the average hop length is doubled. If 30% of the nodes are active, then the average path length increases by 30%.

2.3 Reliability Theory

Reliability theory has its origins in probability and statistics. It was originally used as a tool to help life insurance companies decide profitable premiums to charge their customers. The failure of

other systems such as cars, spaceships, and electronic devices resembles life and death events of humans. Therefore, reliability theory was applied to model the lifetime of such devices. Reliability and the lifetime distribution of a particular device is usually drawn using statistical techniques by observing failure times of an identical large population of the device. Furthermore, the reliability of individual devices (components) are used to express the reliability of complex systems consisting of several components connected either in serially or in parallel. In the following two subsections, we discuss basic ideas of component and system reliability.

Component reliability

Let X be a random variable representing the lifetime of a particular component. We define $R(t)$, the reliability of the component, as the probability that the component survives until time t. Thus, $R(t) = Pr\{X > t\} = 1 - F(t)$, where F(t) is the distribution function of the component lifetime (X).

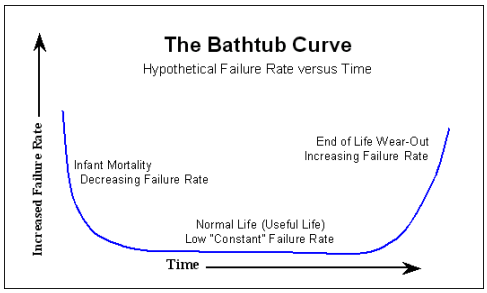


Figure 2.8: Typical bathtub-shaped failure rate function.

A localized version of the reliability function ($R(t)$) is the failure-rate function ($\lambda(t) = \frac{f(t)}{R(t)}$), where $f(t)$ is the probability intensity function of the component lifetime (X). $\lambda(t)$ represents the probability that a component, which survived until time t, fails in the next moment (Δt). In a typical environment, components follow a bathtub-shaped failure rate function. The bathtub-shaped failure rate function consists of three major periods. First, an initial period that exhibits a high and decreasing failure rate in the early stage of the component lifetime. In this period, the failure rate

captures early design and manufacturing defects. Second, a stable period that exhibits a relative constant failure rate. Third, wear and tear period that exhibits an increasing failure rate. Fig. 2.8 depicts a typical failure rate function.

System reliability

The reliability of large systems consisting of several components are usually expressed in terms of the reliability of the underlying components. In a particular system, components are either connected serially, in parallel or combination. Let a system s consist of n components, each with identical reliability r. The system reliability ($R(t)$) is represented in Formula 2.1, Formula 2.2, and Formula 2.3 for serial, parallel, and $\kappa - out - of - n$ composition of the components.

$$R_{series}(t) = r^n \tag{2.1}$$

$$R(t)_{parallel} = 1 - (1 - r)^n \tag{2.2}$$

$$R(t)_{\kappa-out-of-n} = \sum_{i=\kappa}^{n} r^i \cdot (1 - r)^{(n-i)} \tag{2.3}$$

CHAPTER 3

TARGET APPLICATIONS

In this chapter, we present a brief background about our target WSN applications including *environmental*, *SAIL*, and *Bridge Scour* applications. This background helps identify research problems, requirements, and adequate assumptions and solutions.

3.1 Environmental Monitoring

Environmental monitoring is one of the earliest envisioned applications of wireless sensor networks [60, 76]. Many of the research assumptions and challenges tackled in the early research body of work stem from this class of applications. Large-scale deployment (i.e., on the order of several hundred nodes), large environmental area (i.e., the need for multi-hop networks), high node redundancy, and long lifetime requirement (i.e., much longer than the lifetime of a single node) are typical assumptions.

In a typical environmental application, the sensor nodes are scattered randomly in a large environmental area (i.e., sensor field). After deployed, the sensor nodes should self-configure and self-organize into a connected multi-hop network. Furthermore, every point in this field needs to be covered by at least one `active` sensor node (i.e., within the sensing range of the node). The prevalent communication patterns in this application are data dissemination (i.e., one-to-many) and data collection (i.e., many-to-one). The latter is typically used in collecting application-specific data, whereas the former is typically used to re-task the sensor nodes with new queries such as different sampling rates and different sensing parameters. Therefore, coverage and connectivity enjoyed being the primary requirements, in particular, topology management protocols. The requirement of long lifetime and the assumed high node redundancy made the topology management protocols, which leverage node redundancy to schedule nodes `ON` and `OFF`, an attractive solution in extending

the WSN lifetime among other techniques, in particular, given the fact that putting the node's radio into full `sleep` mode is the only feasible way in saving energy.

In light of these requirements and assumptions, most of the proposed topology management protocols (i.e., node scheduling protocols) focused on working around the power limitation and extending the network lifetime while achieving connectivity and coverage. On the other hand, little attention is given to consider node failures. We believe that the unreliability limitation of sensor nodes is as important as that of power limitation. Therefore, reliability-driven topology management protocols are of parallel importance.

3.2 SAIL: Sensor-Assisted Independent Living

It is expected that Adults age 65 years and older will account for more than 20% of the U.S population by the year 2050. As a response to that, in 1991, DHHS created the Healthy People 2000 project, which is the first national effort targeted at reducing disability and promoting physical health in older adults. Despite this initiative, the number of elderly with one or more physical disabilities is increasing. In collaboration with the nursing school at Wayne State University, we propose to apply the technology of wireless sensor networks as a non-obtrusive tool to better monitor the activities of elders in their apartments, providing an innovative approach to connect seniors to their caregivers that facilitates the communication of any emergency conditions such as falling.

Daily physical activity is closely related to the health of senior citizens. Thus daily activity information on a senior citizen, e.g., the distance the person walks in one day, is very difficult to collect in conventional and non-obtrusive ways. This is especially true for long-term activity data, which could be very useful for medical professionals in helping to evaluate health behaviors and design health plans. Emergency conditions detection is also a very important function that wireless sensor networks can provide. Perhaps the most important emergency condition to which elders are susceptible is falling. According to the Centers for Disease Control and Prevention, falls are the leading cause of injury deaths and the most common cause of nonfatal injuries and hospital admissions for trauma. More than one third of adults age 65 and older fall each year, and more than 60% of the people who die from falls are 75 years of age and older. Thus, detecting falls and

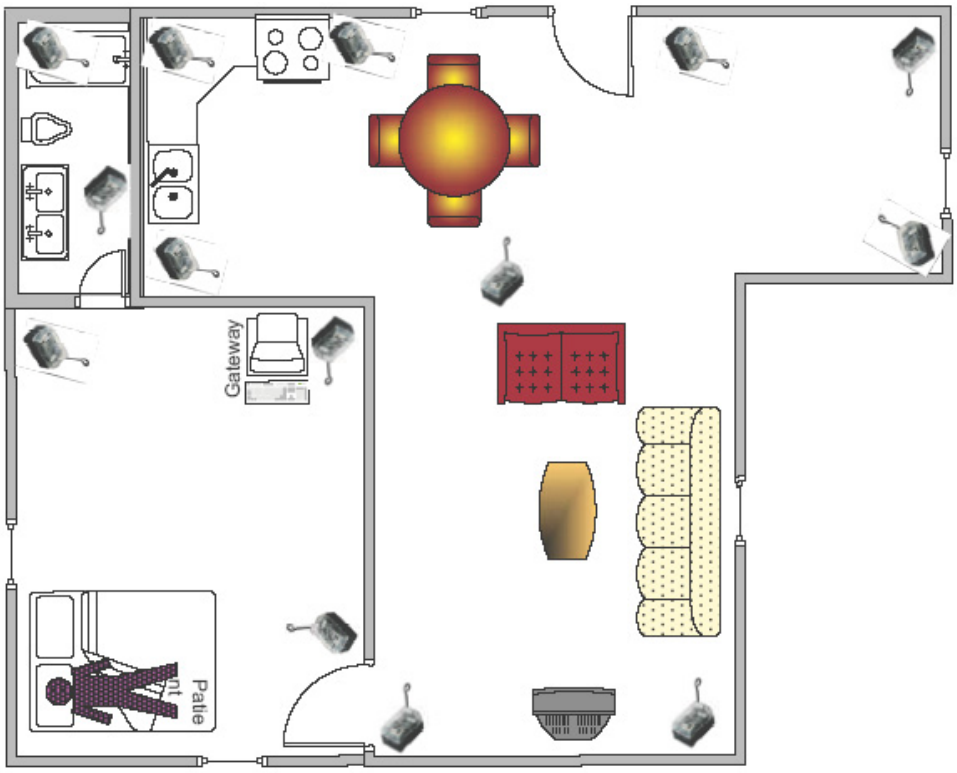


Figure 3.1: Schematic of sensor node deployment in SAIL.

responding quickly is critical. If falls are detected and responded to immediately, more lives will be saved.

In this application, sensor nodes are deployed in each room to provide the required coverage (Fig. 3.1). These sensor nodes are responsible for detecting life-threatening events and communicating them to a base station. The base station alerts caregivers over the Internet. Since the typical communication range of a sensor node is long enough to form a single-hop network between the sensor nodes and the base station, connectivity does not pose a big challenge. However, since the target events are life-threatening, the WSN need to be highly dependable and resilient to node failures.

Target events tend to be short in duration, however, these events are life-threatening and it is vital that the wireless sensor network be available for their detection. Therefore, the *reliability* of the wireless sensor network takes a secondary priority to *availability*. *Reliability* means the probability

of having long continuous and uninterrupted operation of the wireless sensor network. *Availability* means the probability that the wireless sensor network will be available (i.e., functioning) at an arbitrary point of time over its lifetime.

3.3 Bridge Scour

The significance of this project is driven by the financial and human losses attributed to scour related bridge collapse, nationally and internationally. In the United States, approximately 60% of all U.S. highway bridge failures are due to bridge scour [46]. In 1993 alone, more than 2500 bridges were destroyed or severely damaged due to scour caused by severe flooding [62]. However, scour due to severe flooding is not the only concern. The high profile catastrophic collapse of the Schoharie Creek Bridge in New York in 1997 in which 10 people died was caused more by the cumulative effect of pier scour of glacial till than the severe flood which ultimately caused its collapse [64, 46]. In another high profile case that caused seven deaths, I-5 over Los Gatos Creek in California collapsed due to local pier scour during a flood event, but the underlying cause was channel degradation from the previous 28 years of service [46]. In addition to the unacceptable human loss of life and the direct costs associated with bridge repair, the Federal Highway Administration (FHWA) estimates that indirect costs suffered by the public and local business because of long detours and lost production are five times greater than the direct costs of bridge repair [46]. As such, a reliable and effective means for monitoring and predicting scour is necessary.

The project seeks a reliable method for the determination of local scour at bridge piers to improve the safety and reliability of critical infrastructure components. The proposed technique utilizes a networked sensor system relying on thermal, optic, and pressure measurements. This sensor system will be used to assess cumulative scour progress, alert decision makers when scour protection measures are necessary for infrastructure protection, provide a warning system for scour related failure to improve safety on scour susceptible bridges, and provide necessary information for continuously updated predictive scour models.

As indicated in the schematic of Fig. 3.2, bridge scour is characterized by the loss of soil in the vicinity of the bridge support (pier, abutment, embankment). The soil loss results in reduced support

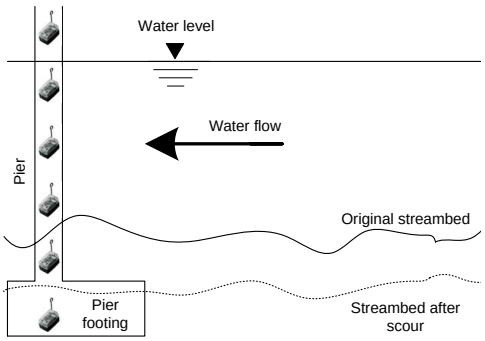


Figure 3.2: Schematic of the bridge scour project sensor node deployment.

for the bridge structure and loads; ultimately, bridge scour can lead to bridge collapse. In response to these issues, the FHWA established a national scour evaluation program as a component of the National Bridge Inspection Program, resulting in the development of the National Bridge Inspection Standards (NBIS). The NBIS requires more than 588,000 U.S. bridges to be inspected every two years for scour and structural stability, and with divers every five years if underwater members are not visible [65, 46]. In addition, the FHWA has published (and updated in 2001) three reports that define bridge scour technology and provide guidance to state DOTs -"Hydraulic Engineering Circular No. 18 (HEC-18) Evaluating Scour at Bridges" [67],"HEC-20 Stream Stability at Highway Structures" [47], and "HEC-23 Bridge Scour and Stream Instability Countermeasures" [48].

As depicted in Fig. 3.2, the sensor nodes will be placed in a linear topology over the the pier of the bridge with very small displacement (i.e., on the order of few centimeters). Based on the scour conditions, some sensors will be submerged in soil, in water, or in the open air. It is the responsibility of each sensor node to decide whether the node is submerged in soil or the scour process caused the sensor to be in water based on observed temperature and light sensor readings. Using this information, the streambed can be recovered with high accuracy.

3.4 Summary

WSNs in all of the application classes are expected to operate autonomously without close administration or maintenance. Therefore, low-level services that provide system-level support and

enable self-* behavior are of particular importance in moving WSNs forward toward real deployment. Services such as *neighbor discovery* and *topology discovery* services are typical examples. Similarly, working around power as well as unreliability limitations of wireless sensor nodes is unavoidable. Leveraging node redundancy to develop node scheduling protocols is an attractive solution. However, the requirements and assumptions that the node scheduling protocol need to consider are different in each application. For example, coverage and connectivity plays a major factor in turning nodes `ON` and `OFF` in the environmental application, whereas, in the *SAIL* project it does not.

In the following chapter, we introduce our availability modeling of WSN for environmental applications, we use these models to formulate and solve the deployment problem seeking an optimal deployment strategy that meets user-defined availability requirement with least total cost.

CHAPTER 4

HIGHLY AVAILABLE AND LOW COST WSNS

In this chapter, we attack the sensor network deployment problem. We believe that the deployment problem shapes the capabilities and limitations of all WSN protocols and algorithms including localization, MAC, and so on. We define the deployment problem as the problem of deciding how many sensor nodes should be deployed in the sensor field over how many phases over it's lifetime. In this chapter, we seek the optimal deployment strategy that meets user-defined availability requirement with minimum total cost taking into consideration node failures resulted from harsh environmental conditions and changing field trip to sensor node cost ratio. We, first, model WSN and total cost as functions of the deployment plan, then, we formalize the deployment problem as a 2-dimensional optimization problem and provide iterative numerical solutions that target minimum and average availability. Our modeling and optimization enable us to explore cost-benefit tradeoffs, we believe, this is a solid step toward bringing the cost as an explicit dimension in the design space of WSN protocols. We compare the performance of the optimized solution (denoted as **pro-active**) to two other solutions: on-demand deployment solution (denoted as **on-demand**) and a single-visit solution (denoted as **at-front**). The former strategy, represents a solution that does not plan for node failures in advance, on the contrary, it schedules for future deployments on an on-demand basis only. The latter strategy, accounts for node failures, however, it deploys all the nodes at front with no later field trips. Unlike the **pro-active**, **on-demand** and **at-front** can not adapt to a changing field trip to sensor node cost ratio due to their inherit limitations, **pro-active** adapts its plan and maintains lower total cost and achieve comparable WSN availability. We show that **pro-active** outperforms **at-front** and **on-demand** in terms of the total cost per availability unit in all application scenarios.

For example, using **pro-active** costs $7 compared to $40 and $280 per total uptime in case of **on-demand** and **at-front** respectively. We further found that **at-front** is by far the worst among the three strategies.

4.1 Introduction

WSN application deployment and lifetime can be divided into two major stages: *planning stage and operational stage*. Operational stage is concerned with the development of network and application protocols. On the other hand, the planning stage, which is usually referred to as the deployment problem, pertains to deciding network parameters. Network parameters include *how many* nodes should be deployed, *how* and *where* to place them, and how to structure them (e.g., flat, two-tier, etc), and others [27, 82]. Hereafter, we refer to these three questions as the key aspects of the deployment problem. Deciding these parameters is typically driven by coverage and connectivity [6, 9, 45, 68]. We believe that the deployment problem occupies a very unique and inherit central role in WSN as careful estimation of the deployment parameters is essential to the proper operation of later protocols in the operational phase.

There have been several papers addressing the deployment problem [35, 37, 102], in particular, the sensor placement aspect [37, 49]. In these efforts, the authors studied the problem of deciding the number of nodes and where to place them in order to guarantee coverage and connectivity. They considered various application contexts that constraint the way node deployment is conducted. Our work complements existing work by looking at the deployment problem from availability and total cost point of views and addressing the *how many* and *how* (i.e., single or multiple deployment phases) aspects of the problem. In this chapter, we use the term *deployment problem to refer to the problem of deciding how many sensor nodes should be deployed in the sensor field over how many incremental deployments*. We aim at meeting user-defined WSN availability requirement with minimum total cost taking into consideration node failures resulting from external harsh environmental conditions and changing field trip to sensor node cost ratio. In spite of the vision that sensor nodes will be very cheap in the near future, we believe that field trip to sensor node cost ratio will not always be in favor of the field trip. For example, in developing countries, technology is more

expensive than labor. Therefore, it is expected that sensor cost will be considerable compared to field trip cost. On the contrary, consider a home application scenario, where sensors are deployed in the home and within the reach of the end-user, deployment cost will be minimal as the end-user himself may be in a position to perform incremental deployments. WSN availability is defined as the probability of having a functioning WSN at any point of time. Total cost, on the other hand, accounts for the cost of all the sensor nodes to be deployed and the cost of conducting the field trips (i.e., incremental deployment). A simple deployment strategy may stipulates the deployment of all sensor nodes at once, with no further deployments, another strategy may perform extra deployments on an on-demand basis only (i.e., whenever a node fails). These deployment strategies result in different WSN availability and total cost. We assume that node placement (i.e., the *where* aspect of the deployment problem) is decided by other means [14, 37, 49], furthermore, we adopt a two-tier WSN structure, in which the sensor nodes are grouped into clusters. A two-tier WSN structure has been supported by existing work such as in GAF [92] and the latter Tenet architecture [27]. WSN availability is defined by the availability of having at least κ nodes in each cluster.

We propose three deployment strategies: **at-front**, **on-demand**, and **pro-active** strategies. **at-front** represents a deployment strategy that considers the hostility of the environment and the required WSN lifetime to estimate the number of nodes required to be deployed at front to achieve the target availability without any further deployment visits. Having only a single deployment visit prevents **at-front** from adapting to a variable field trip to sensor node cost ratio. The **on-demand** strategy, represents an optimistic strategy in which no planning is conducted at all to account for future node failures, on the contrary, further deployment visits are scheduled on an on-demand basis only whenever the number of available sensor nodes in a particular cluster drops below some user-defined threshold.

The **pro-active** strategy, on the other hand, plans for for future node failures and considers different cost ratios. In the **pro-active** strategy, we formalize the deployment problem as a two-dimensional optimization problem and provide iterative numerical solutions that satisfy user-defined minimum and average availability requirements with minimum total cost. We perform extensive

simulations and compare the performance of the different deployment strategies in terms of total cost per availability unit in changing application settings. In general, we observe that **pro-active** outperforms the other two strategies, **at-front** and **on-demand** in all settings, we also find that the **at-front** deployment strategy is by far the worst. We find that **pro-active** adapts very well to a changing cost ratio and maintains lower cost per availability unit. For example, adopting **pro-active** costs \$10 compared to \$70 and \$300 per total uptime in case of **on-demand** and **at-front** respectively.

Our contribution in this work is four-fold. First, we provide a general definition of the deployment problem as time-of-deployment vector and number-of-nodes-to-deploy matrix, this definition accommodates the expression of any deployment strategy. Second, model WSN availability and total cost as functions of the deployment strategy, which facilitates the investigation of different deployment strategies in the availability-cost design space. Third, we use these models to formulate a confined and practical version of the deployment problem into a two-dimensional optimization problem, solving which, results in a deployment plan that achieves user-defined availability with least cost. Fourth, we move one step, albeit an initial step, closer toward understanding and exploring the cost-benefit tradeoffs of the deployment problem in the context of a particular application and business model.

The rest of the chapter is organized as follows. Section 4.2 discusses the deployment problem and its components. In Section 4.3, we illustrate the big picture of our modeling and present our definitions and assumptions. We discuss the **pro-active** strategy in Section 4.4, and the **at-front**, and **on-demand** deployment strategies in Section 4.5. Our simulation and evaluation results are discussed in Section 4.6. Finally, we compare to existing literature, conclude our chapter in Section 6.3 and Section 6.4 respectively.

4.2 Deployment Problem

The *deployment problem* is concerned with deciding *how many* sensor nodes we need for a particular WSN application, as well as, *how* and *where* these sensor nodes should be deployed in order to achieve connectivity, coverage, reliability, availability, and many other characteristics [44].

The *how many* component of the deployment problem is concerned with deciding the number of sensor nodes that must be deployed to meet user and systems requirement [6]. The *where* component pertains to deciding the location of the sensor nodes relative to each other and in the sensor field [22, 37, 61, 103]. Finally, the *how* component is concerned with the question whether these nodes should be deployed at once, or incrementally over multiple field trips.

The *deployment problem* shapes the capabilities and restrictions of all operational WSNs protocol and algorithms including localization, MAC, topology control, routing, and data aggregation protocols. For example, the *where* component of the deployment problem may stipulates the placement of a few beacon sensor nodes in a pre-determined locations to assist the localization protocol in its mission [10]. Therefore, we believe that the *deployment problem* enjoys an inherit fundamental role in WSNs research.

Deciding *how many* nodes we need to overcome node failures resulting from hostile environment, perhaps, is trivial to justify. This may not be the case when it comes to justifying why do we need to decide *how* these nodes should be deployed (i.e., incremental deployment). There are many scenarios where incremental deployment brings advantages and alleviates challenges. For example, when we face uncertainty about the application environment that domain experts cannot predict, early deployment phases of the incremental deployment can play an exploratory role to unveil these uncertain elements such that latter deployment phases can be conducted with more certainty. These exploratory phases, not only help improving WSN efficiency and performance, but also alleviates the pressure on the WSN designer to get the deployment right in one step.

Like other WSN design options, the deployment problem window of feasible solutions is confined by the specific application. For example, in military applications, accurate placement of the sensor nodes is not possible, therefore, the *where* component of the deployment problem must take into consideration the uncertainty of the ultimate sensor node locations in the battle field [103]. Furthermore, incremental deployment in such applications may not be an option due to the short term mission of the WSN and severe fatality of the application environment.

In this work, we attack the *how many* and *how* components of the deployment problem and assume sensor placement is decided by other means. We use the term *deployment problem to refer to the problem of deciding how many sensor nodes should be deployed in the sensor field over how many incremental deployments*. A simple deployment strategy may stipulates the deployment of all sensor nodes at once, with no further deployments. We aim at finding the optimal deployment plan that meets user-defined availability requirement with minimum total cost taking into consideration node failures resulted from harsh environment conditions and changing field trip to sensor node cost ratio. As a first step in formalizing this optimization problem, we define both WSN availability and total cost as functions of the deployment strategy. WSN availability is defined as the probability of having a functioning WSN at any point of time. Total cost, on the other hand, accounts for the cost of all the sensor nodes to be deployed and the cost of conducting the field trips during the operational stage.

4.3 Model

Given a user-defined availability requirement, our modeling and optimization finds the optimal deployment plan that minimizes total cost. WSN availability at time t is defined by the number of nodes deployed in the field by that time, the way these nodes are clustered together, and the sensor node failure model. The total cost, on the other hand, is defined by the total number of nodes deployed, the total number of field trips, cost of the sensor node, and the cost of the field trip. Before we can proceed with formalizing availability and total cost in terms of the deployment strategy, it is vital to discuss a few preliminaries.

4.3.1 WSN structure

The sensor nodes of a WSN are arranged into N clusters, each cluster j consists of a set of $\eta_j(t)$ sensor nodes, $\eta_j(t)$ represents the number of nodes deployed in cluster j by the time t, note that $\eta_j(t)$ is a function of the deployment strategy. This structure is similar to GAF [92] and Tenet [27]. In this well-known and accepted WSN structure strategy, it is assumed that all the sensor nodes that belong to the same cluster are redundant in the sense that any κ set of nodes out of the deployed

$\eta(t)$ nodes are enough to maintain a functioning cluster and WSN. Therefore, the availability of each cluster, denoted as $A_c(t)$ can be modeled as a parallel structure, whereas, the availability of the WSN, denoted as $A_{wsn}(t)$ can be modeled as a serial structure. Since the number of deployed nodes in each cluster $\eta_j(t)$ is a function of the deployment plan, the availability of the WSN becomes a function of the deployment plan. Figure 4.1 depicts the reliability block diagram of the WSN. Each dotted-box represents a single cluster, which consists of several nodes connected in parallel. The set of clusters collectively are connected serially and form the WSN.

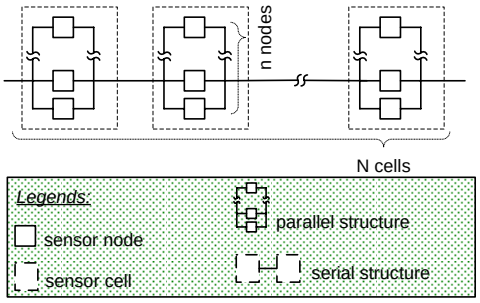


Figure 4.1: Reliability block diagram of a WSN.

4.3.2 Node failure model

Accounting for node failures dictates the assumption of sensor node failure model. In this work, we target deployment-based failures. Deployment-based failures account for failures resulted from harsh environmental conditions. For example, severe wind storms that will destroy the sensor node(s), falling objects such as trees, and wild animals that could step over the sensor node or even could eat it. We believe that such harsh environmental conditions will, in most cases, render the sensor node totaly dead, therefore, we adopt a fail-stop failure pattern. Deployment-based failures are independent of the time a sensor node spend active. In contrast, usage-based failures depends on the time a sensor node spends in the active mode, the more a sensor node is in the active mode, the more the probability that the node fails.

To model such sensor node failures, we adopt the exponential distribution to model the lifetime of sensor nodes. The exponential distribution exhibits a constant failure rate, which implies that a sensor node that has just been deployed has the same probability of failing as a node deployed long time ago. We believe this memoryless feature of the exponential distribution allows us to capture the type of node failures we are targeting and model the environment hostility. In other words, it does not matter whether the node has been deployed for several months or has just been deployed to be eaten by an animal for example, and therefore, fail. Since all the nodes are deployed in the same environmental region, we further assume that all sensor nodes independently follow the same failure model.

In fail-stop failure pattern, the probability that a sensor node is available at any time t, equals to the probability that the lifetime (i.e., τ) of the sensor node is greater than t. This cumulative probability distribution function is known as the survival function and defined as $S_i(t) = e^{-\lambda t}$ in the case of exponential lifetime distribution [28]. The survival function, $S_i(t)$, represents the unconditional probability that the sensor node will survive beyond time t, whereas, failure rate, λ, represents the conditional probability intensity that node i will fail in the next moment, given that it has survived until time t (i.e., $\lambda = Pr\{\tau_i \in [t + \mathrm{d}t] | \tau_i > t\}$), where τ_i is the lifetime of node i. Note also that the mean node lifetime (i.e., $E[\tau]$) equals to $\frac{1}{\lambda}$. Given the fail-stop failure pattern, sensor node availability at time t (i.e., the probability of having a functioning sensor node at time t) has a cumulative probability distribution of $S(t)$.

4.3.3 Incremental deployment and cost ratio

As we have discussed in Section 4.1, any solution to the deployment problem is confined by the limitations imposed by the particular application. A key assumption in our work is that the application provides the flexibility to conduct multiple deployment visits (i.e., incremental deployment). We argue that this is the case in a large set of envisioned WSN applications. For example, environmental applications [60], in-home deployments [18], and many others. From node failure point of view, incremental deployment offers the flexibility in postponing node deployment until we need

them, which may help in avoiding node failures. Ideally, we would like to postpone further node deployments immediately after nodes in the sensor field fail. Furthermore, we assume that the cost of a sensor node (denoted as C_{sensor}) and the cost of conducting the field trip (denoted as C_{trip}) may have different ratios. The cost of a visit accounts for any administrative costs associated with the field trip, such as personnel salaries and labor costs. The sensor cost represents the cost of actual sensing hardware.

Changing cost ratio is leveraged by the **pro-active** deployment strategy to maintain a lower total cost. Intuitively, having a very costly field trip expenses and very cheap sensor node may lead us in reducing the number of field trip deployments while increasing the number of nodes deployed in each trip to maintain the required availability level.

A changing field trip to sensor node cost ratio (i.e., $C_{trip} : C_{sensor}$) could be the result of several factors. First, proximity of the sensor field, for instance, far-away sensor field may cause the cost ratio to tip in favor of field trip cost. Second, hostility of the environment, for example, deploying sensor nodes in a volcano may require very expensive safety measures which drives the field trip cost significantly high. Third, expensive sensing hardware that may cause the scale to tip in favor of sensor node cost, which may require us to increase the number of field trips and decrease the number of sensor nodes to deploy in each trip. Fourth, geographical factors may result in different cost ratios in different countries even for identical application. Contrary to developed countries, in developing countries labor is typically cheaper than technology, this results in lower field trip costs and higher sensor node costs.

4.3.4 Availability and cost model

Now, we are in a position to present WSN availability (i.e., A_{wsn}) and the total cost (i.e., C_{total}) as functions of the deployment strategy, which can be seen as a plan in which sensor nodes are deployed incrementally over several phases at specific times, in each deployment phase, a specific number of nodes are added in each cluster. This can be formalized as a tuple: $D\langle \vec{\tau}, \vec{\eta} \rangle$. $\vec{\tau} = (\tau_1, \tau_2, \tau_i, \ldots, \tau_m)$ is a vector representing the time of each deployment phase, where $1 \leq i \leq m$

and m represents the total number of deployment phases such that $\tau_m < T_{max}$ (i.e., the last node deployment should be conducted before the end of the WSN lifetime required by the end-user). $\vec{\eta} = (\eta_{11}, \eta_{12}, \ldots, \eta_{ij}, \ldots, \eta_{mN})$ is a two dimensional matrix with each element representing the number of nodes to be deployed in phase i at cluster j, where $i \in [1, m]$ and $j \in [1, N]$. Note that both the availability of the WSN (i.e., A_{wsn}) and the total cost (i.e., C_{total}) become functions of the deployment strategy, and therefore, our goal can be summarized as *finding the optimal deployment plan ($D\langle \vec{\tau}, \vec{\eta} \rangle$) that meets a particular availability requirement and minimizes the total cost.*

Availability of each cluster can be modeled as a κ-out-of-n system, and the availability of the WSN as a serial system of the clusters. The availability of the WSN at a particular time $t \in [0, T_{max}]$ depends on the number of nodes deployed by the time t in each cluster (denoted as $\eta_j(t) = \sum_{i=1}^{\lfloor t/m \rfloor} \eta_{ij}$ for cluster j). On the other hand, the total cost (denoted as C_{total}) of a particular deployment plan is defined as the cost of the sensor nodes plus the cost of all visits.

$$A_{wsn}(t) = \prod_{j=1}^{N} \left(\sum_{i=\kappa}^{\eta_j(t)} S_i(t)^i \cdot (1 - S_i(t))^{\eta_j(t) - i} \right) \tag{4.1}$$

$$C_{total} = C_{trip} \cdot m + C_{sensor} \cdot \sum_{i=1}^{m} \sum_{j=1}^{N} \eta_{ij} \tag{4.2}$$

Depending on the cost ratio $C_{trip} : C_{sensor}$, different deployment plans result in different costs. For example, if the cost of a sensor node is negligible compared to the cost of a field trip, then it is probably best to deploy all the sensor nodes at front, in other words $m = 1$. Whereas, in case of expensive sensing hardware, we may want to perform extra deployments only when necessary, in other words, when deployed nodes fail.

The **at-front** and **on-demand** strategies lie at the extreme ends of the solution spectrum, whereas, **pro-active** traverses the solution spectrum for optimal solution. **At-front** confines its deployment plan to one visit and tries to satisfy the availability requirement by keeping on increasing the number

nodes to deploy at-front. On the other end, **on-demand** puts no limits on the number of deployment visits, while minimizing the number of nodes to deploy in each visit. The **pro-active** strategy, on the other hand, leverages the flexibility provided by incremental deployment and adapts to the changing cost ratio and adjusts the number of deployment visits and the number of nodes to deploy in each visit to satisfy availability requirement with minimal cost.

4.3.5 User requirement

We provide two ways to allow the end-user in expressing availability requirement: minimum availability (denoted as min_A) and average availability (denoted as avg_A). Minimum availability represents a lower bound on the probability of having an available WSN at any time over the WSN lifetime, it is defined formally as $min_A = \min_{t=0}^{T_{max}} A_{wsn}(t)$. It allows the end-user in expressing a WSN availability lower-bound, which could be useful in mission-critical applications. Average availability represents the probability of having an available WSN on average at any point of time during the WSN lifetime and defined formally as $avg_A = \frac{\sum_{t=0}^{T_{max}} A_{wsn}(t)}{T_{max+1}}$. avg_A is proportional to total uptime of the WSN, for example, an avg_A value of 0.5 results in $\frac{T_{max}}{2}$ of total uptime.

4.4 Pro-active Strategy

The definition of the deployment strategy in Equation 4.1 has many variables, which makes the optimization problem very complicated and the deployment plan impractical to carry out in the real world. For example, it is easier for the WSN administrators and application scientists to carry out a deployment strategy that mandates regular field trips every six months, rather than a strategy that requires field trips at irregular times. Therefore, we impose several constraints on this definition. First, the deployment phases are equally splitted in time, therefore, $\vec{\tau}$ reduces to a vector of m values with T_{max}/m increments. Second, the number of nodes deployed in each field trip are equal, in other words, for a fixed j, n_{ij} are equal $\forall i \in [1, m]$. Third, the number of nodes deployed in each cluster is the same, in other words, for a fixed i, n_{ij} are equal $\forall j \in [1, N]$. These constraints reduce the deployment plan into a tuple of two scalar values $D\langle\, m, n \,\rangle$, where m is the number of

field trips, and n is the number of nodes to be deployed in each cluster at each field trip. We use the new deployment plan definition in the **pro-active** strategy.

Under the new deployment definition, the WSN lifetime is divided into m equal deployment phases of length $\Delta = T_{max}/m$ each. During the first deployment phase (i.e., $t \in [0, \Delta)$), we have exactly n nodes deployed in each cluster, in the second deployment phase (i.e., $t \in [\Delta, 2 \cdot \Delta)$), we have $2 \cdot n$ nodes deployed in each cluster. Let $\Omega = \lceil t/\Delta \rceil \in [1, m]$ represents the deployment phase, then, $\eta_j(t) = \Omega \cdot n \ \forall j \in [1, N]$. To simplify our analysis, we confine κ in our mode to 1, this allows us to reduce the optimization problem in case of min_A into a one-dimensional one. This is still reasonable from the practical point of view, for example, in GAF [92], the sensor field is divided into grids, in each grid at least one sensor node needs to be available to achieve connectivity or coverage, and consequently, have an available WSN.

Note that all the nodes in the same patch (i.e., nodes that have been deployed in the same deployment phase) have the same survival function, and that for patch $\omega \in [0, \Omega - 1]$, the survival function is, simply, a time-shift of $S(t)$, formally, $S_\omega(t) = e^{-\lambda(t-\omega\Delta)}$, note that the total number of sensor patches equals to the number of deployment phases. The availability of cluster can be derived as follows:

$$
\begin{aligned}
A_c(t) &= 1 - Pr\{\text{all nodes have failed by time } t\} \\
&= 1 - Pr\{\text{all nodes of the } 1^{st} \text{ patch have failed}\} \\
&\cdot Pr\{\text{all nodes of the } 2^{nd} \text{ patch have failed}\} \ldots \\
&\cdot Pr\{\text{all nodes of the } \omega + 1^{th} \text{ patch have failed}\} \\
&= 1 - \prod_{\omega=0}^{\Omega-1} \left(1 - e^{-\lambda(t-\omega\Delta)}\right)^n \\
&\text{where } \Omega = \lceil t/\Delta \rceil \text{ is the number of patches deployed} \\
&\text{by time } t
\end{aligned}
$$

In order to have an available WSN at time t, all clusters must be available at that time, hence, the availability of the WSN is defined as shown in Equation 4.3. The total cost for **pro-active** deployment strategy reduces as in Equation 4.4.

$$\begin{aligned} A_{wsn}(t) &= Pr\{\text{all clusters are available at time } t\} \\ &= \prod_{i=1}^{N}\left[1-\prod_{\omega=0}^{\Omega-1}\left(1-e^{-\lambda(t-\omega\Delta)}\right)^{n}\right] \\ &= \left[1-\prod_{\omega=0}^{\Omega-1}\left(1-e^{-\lambda(t-\omega\Delta)}\right)^{n}\right]^{N} \end{aligned} \tag{4.3}$$

$$C_{total} = C_{trip}\cdot m + N\cdot n\cdot m\cdot C_{sensor} \tag{4.4}$$

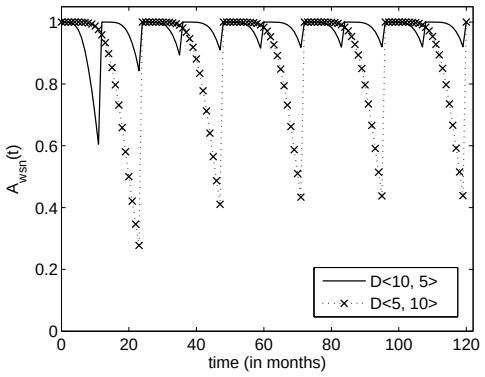


Figure 4.2: Availability of WSN. The x-axis represents time in months, the y-axis represents the probability of having a functioning WSN at that time.

Figure 4.2 represents the availability of a WSN with 10 clusters (i.e., $N = 10$), a sensor node mean lifetime of 13.8 months (i.e., $\lambda = 0.072$), and a deployment lifetime of 10 years (i.e., $T_{max} = 120$ months). The x-axis represents the WSN lifetime in months, whereas, the y-axis, represents the WSN availability. The lines corresponds to two deployment plans, the un-marked line represents availability for a deployment plan of 10 phases and a 5-nodes deployment in each phase in each

cluster (i.e., $D\langle\, 10, 5\,\rangle$), whereas the marked line represents a deployment plan of 5 phases and a 10-nodes deployment in each phase in each cluster (i.e., $D\langle\, 5, 10\,\rangle$). Note that both deployment plans consume the same number of nodes (i.e., 50 nodes in each cluster over the 120 months), however, their availability is totaly different. Intuitively, deploying more nodes at front results in better availability at the beginning, however, more nodes will be susceptible to failures which results in worse availability as the time goes by.

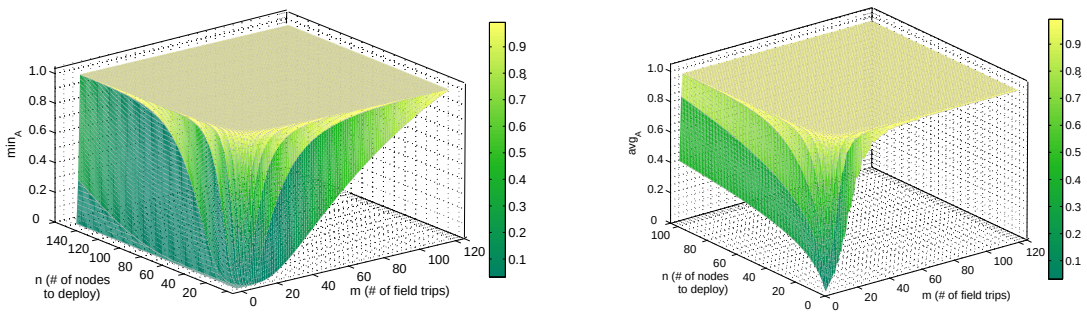
$$min_A = \left[1 - \left(1 - e^{-\lambda \frac{T_{max}}{m} - 1}\right)^n\right]^N \tag{4.5}$$

$$avg_A = \frac{\sum_{t=0}^{T_{max}} A_{wsn}(t)}{T_{max} + 1} \tag{4.6}$$

From Figure 4.2, it is clear that min_A happens at the end of the first deployment phase (i.e., $t = \Delta - \epsilon$), where ϵ is a number close to zero. Note that each cluster will have only n nodes deployed by then. Therefore, min_A can be reduced as in Equation 4.5. avg_A is defined as in Equation 4.6.

Figure 4.3 shows how availability changes as a function of the deployment plan $D\langle\, m, n\,\rangle$, min_A in Figure 4.3(a) and avg_A in Figure 4.3(b). In both figure, the x-axis represents the total number of deployment phases (i.e., m), the y-axis represents the number of nodes added in each phase at each cluster (i.e., n), each point on the xy-plane, hence, represents one deployment plan in the **pro-active** strategy, the corresponding min_A and avg_A for each plan are shown on the z-axis in Figure 4.3(a) and Figure 4.3(b) respectively. The other WSN parameters are chosen as follows: $N = 10$, $\lambda = 0.072$ failures/month, and $T_{max} = 120$ months.

We can see that min_A and avg_A are consistently increasing in m and n. We further observe from Equation 4.4 that the total cost is also increasing in m and n. We leverage these properties to devise a termination condition (refer to Figure 4.6 line 6) and locate a global optimal solution and avoid exhaustive search of the open-ended solution space (i.e., m, n pairs) in our optimization.



(a) (b)

Figure 4.3: Clarifying WSN availability as function of deployment plan, min_A in (a), avg_A in (b).

4.4.1 Optimization

Now, we are in a position to formalize and solve the optimization problem, in other words, finding $D\langle m, n \rangle$ (denoted as Optimal$_D$) that meets availability requirement (i.e., min_A or avg_A) and minimizes the total cost (i.e., C_{total}). Finding Optimal$_D$ is equivalent to searching for the lowest point on a surface representing the total cost. The domain of this surface is a plane consisting of all the pairs, $D\langle m, n \rangle$, that satisfy availability requirement (either min_A or avg_A). Note that the cost surface is consistently increasing in m and n. Therefore, in our search for the global minimum, which represents Optimal$_D$, we can confine ourselves to the points $D\langle m, n \rangle$ located on the lower boundary of this solution domain. In the case of optimization for minimum availability, we can reduce the optimization problem into a single dimension domain. In the following two subsection, we discuss the optimization problem for min_A and avg_A consecutively.

Optimizing for minimum availability

In this subsection, we are concerned with finding the deployment plan that meets user-defined minimum availability requirement (min_A) and minimizes the total cost (C_{total}). Solving Equation 4.5 for n as shown in Equation 4.7 and substituting in Equation 4.4, allows us to express C_{total} in terms of m alone. Thus, our optimization problem reduces to searching for m that minimizes total cost. We choose m as the independent variable, rather than n, since the number of possible

values for m can be bounded by practical limits. In other words, the time between deployment visits can be bounded with a minimum value (denoted as Δ_{min}) by considering some practical issues. After finding m that minimizes total cost, we can find corresponding n simply by plugging-in m and min_A in Equation 4.7.

$$n = \left\lceil \frac{\ln\left(1 - \sqrt[N]{min_A}\right)}{\ln\left(1 - e^{-\lambda \frac{T_{max}}{m}}\right)} \right\rceil \tag{4.7}$$

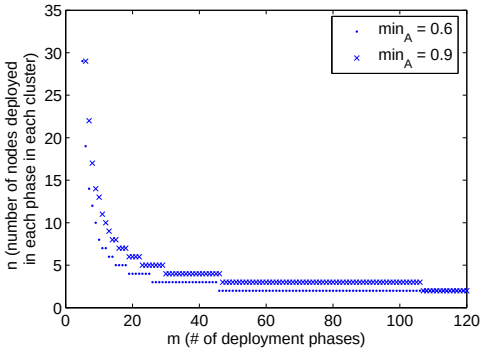


Figure 4.4: Solution space, each point represents one deployment plan that achieves min_A.

Figure 4.4 illustrates the solution plane for min_A. The x-axis represents the number of visits m, the y-axis represents the number of nodes to deploy in each visit, n. Each point in this plane represents one deployment plan in the **pro-active** strategy. The line in the figure represents a lower bound on the deployment plans that satisfy a specific minimum availability requirement. Observe that as the number of visits increases (i.e., m), the number of nodes that should be deployed to satisfy a particular min_A decreases, and vice versa. Depending on the $C_{trip} : C_{sensor}$ cost ratio assignment, each point on this lower-bound line results in a unique total cost value. Recall that the total cost is increasing in m and n, therefore, the optimal deployment plan that minimizes C_{total} is located on this line, in essence, our optimization solution traverses this line to search for the minimum total cost.

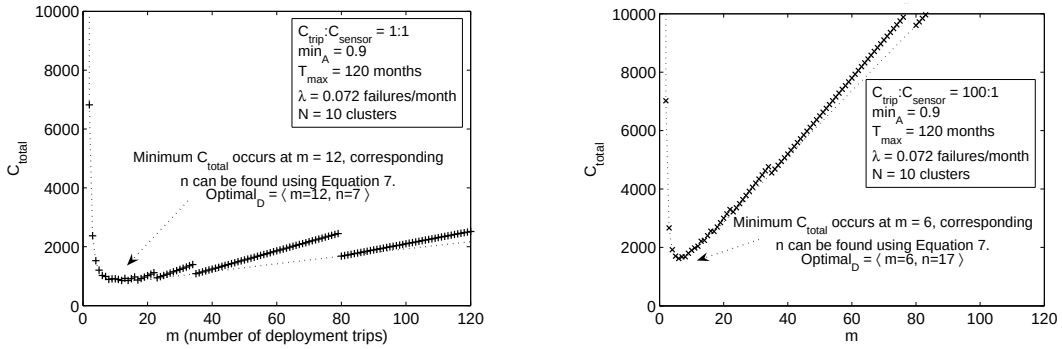


Figure 4.5: Illustrating the pro-active strategy adaption to different $C_{trip} : C_{sensor}$ cost ratio assignments. A cost ratio of 1:1 is used in (a) and a cost ratio of 100:1 is used in (b).

Figure 5.7 demonstrates how the **pro-active** strategy adapts to a changing cost ratio in choosing the optimal deployment plan. The x-axis represents the number of deployment trips, the y-axis represents the total cost. Both Figure 5.7(a) and Figure 5.7(b) have exactly the same parameters except the cost ratio $C_{trip} : C_{sensor}$. In Figure 5.7(a), we use a cost ratio of $1 : 1$, and in Figure 5.7(b), we use a cost ratio of $100 : 1$. It can be easily seen that as the cost of the field trip (i.e., C_{trip}) increased with respect to the cost of the sensor node (i.e., C_{sensor}), the **pro-active** strategy, chooses to decrease the number of field trips (i.e., m) and increase the number of nodes to deploy in each trip (i.e., n) to lower the total cost and maintain the same level of minimum availability.

Optimizing for average availability

Unlike in optimizing for min_A, we use a two-dimensional iterative numerical algorithm to solve the optimization problem for avg_A.

Figure 4.6 shows the pseudo code of our approach. We iterate over all the solution points m and n on the lower boundary of the deployment points that meet avg_A, calculate the corresponding total cost (C_{total}) using Equation 4.4, and find m and n that achieve the minimum total cost. As we did in Section 4.4.1, m takes on the values 1 up to $\frac{T_{max}}{\Delta_{min}}$, with one step increments (line 2 in

```
Input: C_trip, C_sensor, T_max, Δ_min, avg_A
Output: Optimal_D ⟨m, n⟩
 1. Optimal_Cost = INF
 2. for m = 1 to (T_max / Δ_min)
 3. begin
 4.     n = 1
 5.     calculate tAvg_A for D<m, n> // use Eqn. 6 and Eqn. 3
 6.     while ( tAvg_A< avg_A) // terminate once avg_A is satisfied
 7.     begin
 8.         n = n + 1
 9.         calculate tAvg_A for D ⟨m, n⟩ // use Eqn. 6 and Eqn. 3
10.     end while
        // D⟨m, n⟩ satisfies avg_A

11.     C_total = m . C_trip + N . m . n . C_sensor
12.     if (C_total < Optimal_Cost)
13.     begin
15.         Optimal_D = ⟨m, n⟩
16.     end if
17. end for
```

Figure 4.6: The pseudo code for finding the optimal deployment plan that meets avg_A with minimum total cost.

Figure 4.6). For each each value of m, we keep on incrementing n until avg_A is satisfied (lines 6 to 10 in Figure 4.6).

4.4.2 Cost-benefit analysis

From the business point of view, a very important issue that deserves a careful study is *return on investment*. Managers are usually concerned with monetary profit they can achieve for each dollar they spend. Our model allows to address this issue in a comprehensive way.

Figure 4.7 demonstrates that the total cost per availability unit (e.g., min_A, avg_A) is not homogeneous for all target availability levels. In each of the figures, we use the **pro-active** strategy approach to obtain the optimal deployment plan for different min_A and avg_A user requirements (shown on the x-axis), find the corresponding C_{total} for each plan, and finally get the total cost per availability unit, min_A in Figure 4.7(a), avg_A in Figure 4.7(b), and total uptime in Figure 4.7(c). In Figure 4.7(a) for example, if we wish to achieve a min_A of 0.4, each min_A unit costs twice as much as the min_A unit cost if we target min_A value of 0.6. In Figure 4.7(c), we can observe that one month of total uptime costs around $7 if we are seeking a deployment plan that guarantees 100 months of total uptime, whereas, we may have to pay more than $9 per month if we target a

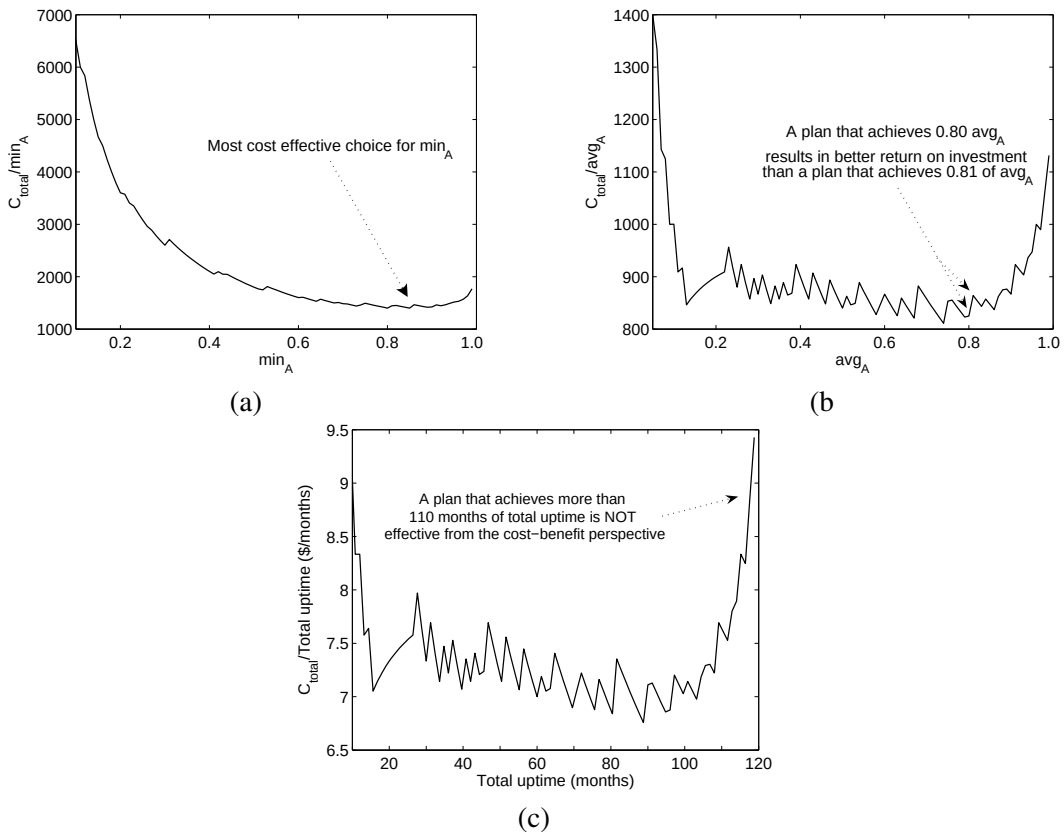


Figure 4.7: Understanding cost-benefit for pro-active deployment strategy, x-axis can be transformed into benefit for a particular business model, y-axis represents cost. User requirement is expressed as min_A in (a), avg_A in (b), and total uptime in (c).

deployment plan that achieves 118 months of total uptime. Figure 4.7(b) suggests that we may want to target a deployment plan that achieves 0.75 avg_A in order to get the best deal on a unit of avg_A we obtain.

This non-homogeneity has implications on investment options. For simplicity, let us study a scenario where the profit is \$P constant-multiplication of the total uptime, in other words we make a profit of \$P for every month of total uptime of the WSN (i.e., profit = \$P $\cdot$ the x-axis value in Figure 4.7(c)). Thus, profit to cost ratio becomes \$P multiplication of the line in Figure 4.7(c). For example, let P=\$10, according to Figure 4.7(c), if we use a deployment plan that achieves 85 months of total uptime, our profit is obviously \$850, using the curve in the figure, we find that our

cost would be $570 = 6.7 \cdot 85$ resulting in \$1.5 return-on-investment, however, we achieve only \$1.05 for every dollar we pay if we adopt a deployment plan that targets 119 months of total uptime. Likewise, the curves in Figure 4.7(a) and in Figure 4.7(b) represent profit to cost ratios in case of min_A and avg_A respectively.

In real-life applications and business models, a simple linear relationship between total uptime and profit is unlikely, if the availability of the WSN is very low, chances the application loses reputation and customers, and eventually profit, is very high. However, our analysis can be a powerful tool in studying cost benefit tradeoffs. It is worth mentioning that targeting minimum availability (Figure 4.7(a)) could be of particular importance in cost-benefit analysis, in some mission critical applications, such as assisted-living. Having unavailable WSN, which may result in the undetection of an emergency condition, could be very damaging to the application reputation, which may lead to severe business losses.

4.5 Ad-hoc deployment strategies

In this section, we present two more simplistic deployment strategies: an ad-hoc **on-demand** strategy and an **at-front** strategy. **At-front** represents a deployment strategy that considers the hostility of the environment and the required WSN lifetime (i.e., λ and T_{max}) to estimate the number of the nodes required to be deployed at front to achieve the target availability without any further deployment visits. Whereas, the **on-demand** strategy, represents an optimistic strategy in which no planning is conducted at all to account for future node failures, on the contrary, further deployment visits are scheduled on an on-demand basis only when a fraction of the sensor nodes in a particular cluster fail.

4.5.1 At-front deployment strategy

The **at-front** deployment accounts and plans for node failures by estimating the required number of nodes needed to meet user-defined availability, however, only one deployment visit, at the beginning of the WSN lifetime, is conducted. Limiting the number of deployment visits to one rules out any possibility to consider the $C_{trip} : C_{sensor}$ cost ratio.

To estimate the number of nodes required to meet user-defined availability, WSN availability under the **at-front** deployment strategy can be modeled as a simple k-out-of-n system. By specializing the general formula in Equation 4.1, we can model the availability of the WSN as follows:

$$A_{\text{at-front}}(t) = \left(1 - \left(1 - e^{-\lambda t}\right)^{n}\right)^{N} \tag{4.8}$$

Again, n is the number of nodes to be deployed at $t = 0$ in each one the N clusters of the WSN. Like in the **pro-active** deployment strategy, user-defined availability is expressed as min_A or avg_A. For a given WSN of size N and a deployment lifetime of T_{max}, finding n that achieves min_A or avg_A can be formulated as a simple one-dimensional optimization problem using the following formulas of min_A and avg_A:

$$min_{A_{\text{at-front}}} = \left(1 - \left(1 - e^{-\lambda T_{max}}\right)^{n}\right)^{N} \tag{4.9}$$

$$avg_{A_{\text{at-front}}} = \sum_{t=0}^{T_{max}} \left(1 - \left(1 - e^{-\lambda t}\right)^{n}\right)^{N} \tag{4.10}$$

The **at-front** deployment strategy exhibits a very high availability at the beginning of the WSN lifetime, however, the availability degrades significantly toward the end lifetime of the WSN. Therefore, it shows a very weak scalability to increasing maximum WSN lifetime (i.e., T_{max}) and increasing failure rate (λ) as we present in Section 4.6.

4.5.2 On-demand deployment strategy

As we have mentioned, the **on-demand** strategy is optimistic in the sense that it does not plan for node failures in advance. However, it performs incremental deployment(s) whenever some sensor nodes fail and the fraction of the remaining nodes drops below some user-defined threshold. Planning and conducting deployment visits is not instantaneous, in other words, there is a minimum time period between time of the detection of the need for node deployment and conducting the actual

deployment (T_{prep}). This constrain captures some real-life physical limitations. In our simulations, we make T_{prep} and the minimum time between two consecutive deployment visits in the **pro-active** strategy (Δ_{min}) the same.

We used a fraction of 0.7 and initial nodes of 2, every time the number of remaining nodes drops below 0.7 of the original, a deployment visit is scheduled at $t + T_{prep}$ and the number of nodes in the cluster is brought back to 2.

4.6 Simulation and Evaluation

In this section, we simulate and evaluate the performance of the **pro-active** node deployment strategy and compare it to more simplistic deployment strategies: **on-demand** and **at-front** and observe how well these deployment strategies adapt to changing application deployment settings.

To capture the primary performance dimensions in our study: *total cost* and *WSN availability*, we coin our performance metrics as cost per availability unit and observe the performance of the different deployment strategies in changing application deployment settings.

4.6.1 Performance metrics

We define three performance metrics, cost per min_A, avg_A as low level metrics, and cost per total uptime as a high level metric. The former metrics are direct measure of the goodness of the optimization process, whereas, the latter is a more meaningful measure from the user point of view. Due to the extremely large cost per availability unit of the **at-front** deployment strategy, we had to draw the y-axis on a log scale, the x-axis is drawn on a linear scale.

4.6.2 Simulation setup

We perform four sets of experiments to study the scalability of the different deployment strategies to different parameters. First, we study the effect of increasing failure rate (i.e., λ), this helps us understand how well each of the deployment strategies scales to more and more hostile environments. Second, observe how the different deployment strategies adapt to a changing cost ratio (i.e.,

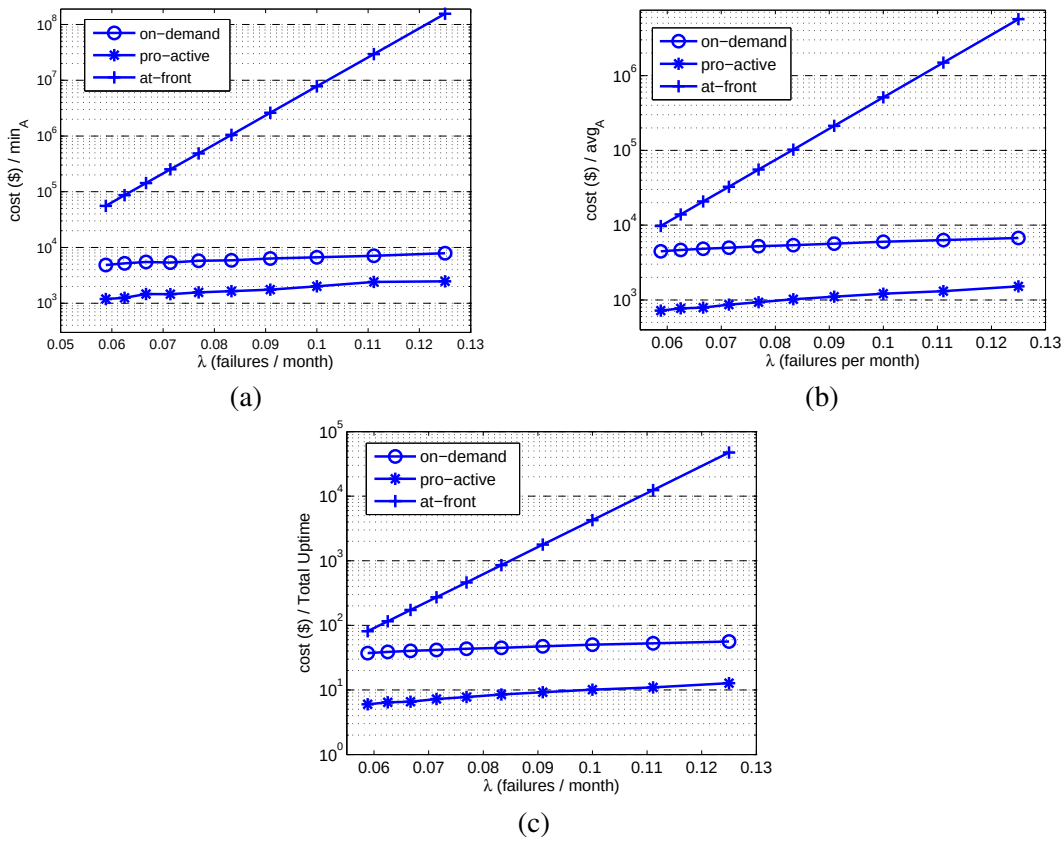


Figure 4.8: Comparing **pro-active**, **on-demand**, and **at-front** deployment strategies as environment hostility increases (i.e., increasing λ). (a) total cost per min_A, (b) total cost per avg_A, and (c) total cost per total uptime.

C_{trip} : C_{sensor}), this is important to see how these strategies behave in different settings due to application specifics or geographical variations. Third, we study the different deployment strategies scalability with respect to an increasing WSN size, by size we mean number of clusters in the WSN (i.e., N). Fourth, we observe the scalability of these strategies to an increasing deployment time (i.e., T_{max}). Table 4.1 lists the parameter value ranges of the four experiment sets.

The WSN is arranged into N clusters, the WSN is available if and only if at least one sensor node is available in each cluster. The sensor nodes fails independently following exponential distribution with parameter λ as defined in the particular deployment scenario being simulated. Finally, the simulation is allowed to run until the maximum deployment time (i.e., T_{max}) is reached or all the

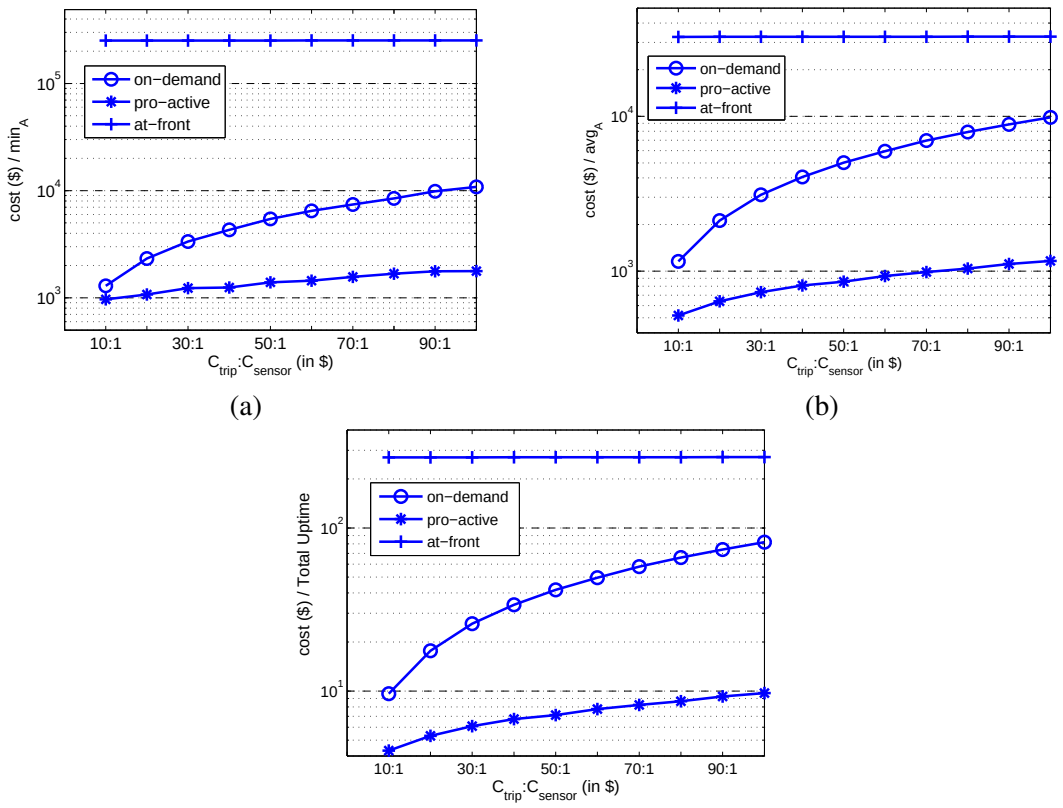


Figure 4.9: Comparing **pro-active**, **on-demand**, and **at-front** deployment strategies as the cost ratio changes (i.e., $C_{trip} : C_{sensor}$ gets larger). (a) total cost per min_A, (b) total cost per avg_A, and (c) total cost per total uptime.

sensor nodes have failed and no future deployments are scheduled. The total cost (i.e., C_{total}) of each run is calculated according to the cost ratio ($C_{trip} : C_{sensor}$) and the number of deployments and number of nodes deployed (i.e., m and n). For each deployment scenario (i.e., deployment settings), we perform simulations of 200 deployments and record the average min_A, avg_A, total uptime, and the total cost. In all of the figures, we present the total cost per availability on the y-axis, the x-axis shows the value of the changing parameter according to the application scenario.

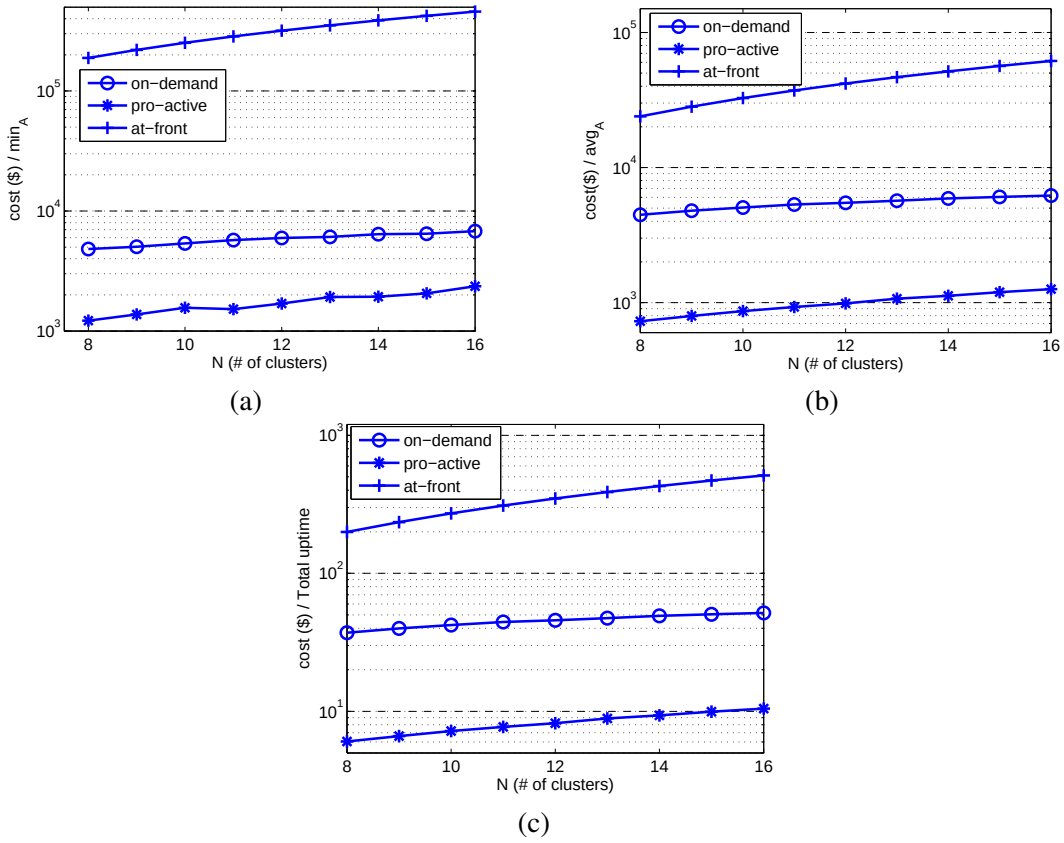


Figure 4.10: Comparing **pro-active**, **on-demand**, and **at-front** deployment strategies as the WSN number of clusters increases (i.e., N increases). (a) total cost per min_A, (b) total cost per avg_A, and (c) total cost per total uptime.

4.6.3 Evaluation results

Environment Hostility. In the first set of experiments in Figure 4.8, we study the performance of the deployment strategies as the environment hostility increases from 0.059 up to 0.125 failures per month drawn linearly on the x-axis, this results in an average sensor node lifetime of 17 months down to 8 months with 1 month decrements. T_{max}, N, C_{trip} : C_{sensor} are set to 120 months, 10, and $50:1$ respectively. The y-axis depicts total cost per min_A in Figure 4.8(a), total cost per avg_A in Figure 4.8(b), and total cost per total uptime in Figure 4.8(c). The y-axis is drawn on a log scale. In general, we observe increasing cost per availability unit as the environment gets harsher for all deployment strategies. Harsher environment results in more node failures, which requires more

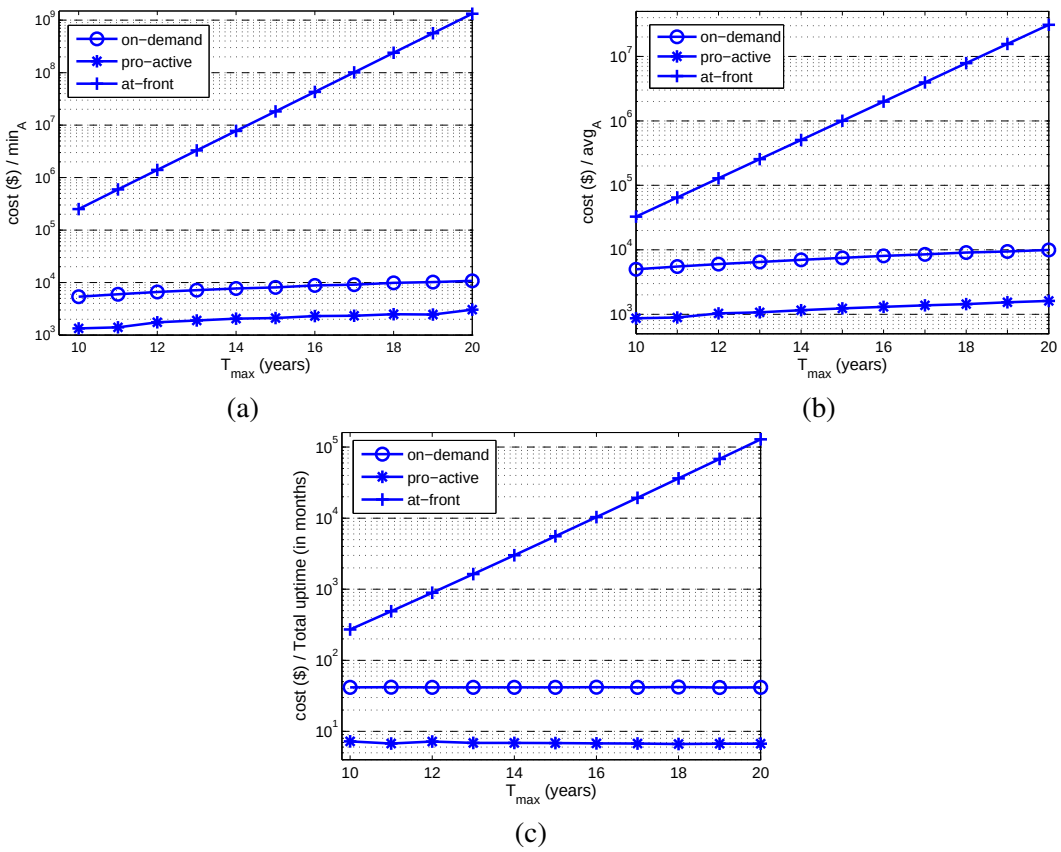


Figure 4.11: Comparing **pro-active**, **on-demand**, and **at-front** deployment strategies as deployment time increases (i.e., T_{max} increases). (a) total cost per min_A, (b) total cost per avg_A, and (c) total cost per total uptime.

nodes to be deployed over more frequent deployment visits, this drives the total cost per availability unit up.

As a general observation in Figure 4.8, we found that the **pro-active** deployment strategy exhibits lower total cost per availability unit compared to the **at-front** and **on-demand** strategies. Furthermore, it scales better to increasing failure rate. We further found that the non-incremental **at-front** deployment strategy is by far the worst strategy on the basis of point-to-point comparison and its scalability to increasing failure rate. In Figure 4.8(a), employing the **pro-active** strategy results in the lowest cost per minimum availability unit. For example, for a λ of 0.125 (i.e., average node lifetime is 17 months), **on-demand** requires more than three times the total cost compared

Scenario	Parameter	Value range
Environment Hostility	λ	0.059 - 0.125
Cost Ratio	$C_{trip} : C_{sensor}$	10:1 - 100:1
WSN size	N	10 - 17 clusters
WSN lifetime	T_{max}	120 - 240 months

Table 4.1: The tuning parameters of different evaluation scenarios.

to that of **pro-active** deployment strategy, and **at-front** requires more than five *orders of magnitudes* higher cost per min_A unit. We also observe that the **pro-active** strategy scales better than **on-demand** and **at-front** strategies with respect to increasing failure rate. Total cost in **on-demand** grows at least twice as fast as that growth of the **pro-active** deployment strategy, whereas, **at-front** grows exponentially, recall that the y-axis is a log scale.

We observe similar results in Figure 4.8(b), the **pro-active** deployment strategy outperforms both the **on-demand** and **at-front** strategies and decreases the total cost per a avg_A unit. For example, employing the **on-demand** strategy costs more than 6 times per avg_A when employing the **pro-active** strategy and more than 13 times when **at-front** is used. As far as scalability is concerned, **on-demand** grows at a rate 2.5 times faster than that of **pro-active**, **at-front** grows exponentially.

Compared to Figure 4.8(a), the total cost per avg_A tends to be cheaper for each of the corresponding strategies, this is simply because guaranteeing min_A requires more nodes and deployment visits than targeting avg_A. While the absolute number on the y-axis in Figure 4.8(a) and Figure 4.8(b), perhaps, does not represent meaningful quantity from the end user point of view, the relative quantities indeed do. In Figure 4.8(c), on the other hand, the absolute numbers can be interpreted in a more useful way as the total cost (e.g., dollars) per a unit of total uptime of the WSN (e.g., months). For example, in a deployment environment with a failure rate of 0.1 failures per month, the **on-demand** strategy requires almost fifty dollars per one month of total uptime, whereas, **pro-active** can achieve the same total uptime for only ten dollars. **At-front** requires around four thousand dollars, which is significantly more.

Cost Ratio. In the second set of experiments, as shown in Figure 4.9, we study the effect of changing the field trip to sensor node cost ratio (i.e., $C_{trip} : C_{sensor}$) on the performance of the different deployment strategies, and how well these strategies adapt the number of deployment visits (i.e., m) and number of nodes in each visit (i.e., n) to maintain good cost to availability investment. As we mentioned earlier, this ratio may change as a result of geographical characteristics, sensor technology costs, and so on. We change $C_{trip} : C_{sensor}$ from $10 : 1$ up to $100 : 1$ incrementing C_{trip} by 10 in each step. λ, T_{max}, and N, are set to 0.071 failures per month, 120 months, and 10 clusters respectively.

We observe that **pro-active** can adapt m and n and achieve lower total cost per availability unit for all $C_{trip} : C_{sensor}$ ratios. For example, in Figure 4.9(a), adopting the **on-demand** strategy costs at least 5 times per min_A unit more compared to adopting **pro-active** when the cost of a deployment visit is 90 times more expensive than the cost of a single sensor node (i.e., $90 : 1$). Observe that when the cost ratio is close to one, the performance of **pro-active** and **on-demand** is close, this is because that **pro-active** loses its key adaptation advantage, where changing the number of deployment visits (i.e., m) and the number of nodes to deploy in each visit (i.e., n) can no longer help in driving the cost down when the cost of deployment visit and cost of sensor node is comparable. However, as the cost ratio grows, **pro-active** starts to increasingly outperform **on-demand**. We, further, observe that the performance of the **at-front** strategy performance is far worse than **on-demand** and **pro-active** and that it is almost not affected by the increasing cost ratio. This is due to the fact that the total number of nodes deployed is decided only based on the availability and is not affected by the changing ratio, furthermore, there is only one deployment visit, therefore, the total cost in the case of **at-front** increases by a very small amount (un-noticeable on a log scale) equal to the increase in C_{trip} in Figure 4.9.

WSN Size. To study the scalability of the deployment strategies to increasing WSN size defined as the number of clusters (i.e., N), we contrast the cost per availability unit under the different deployment strategies in Figure 4.10. N changes from 8 clusters up to 17 clusters. λ, T_{max}, and $C_{trip} : C_{sensor}$ are set to 0.071 failures per month, 120 months, and $50 : 1$ deployment visit to

sensor node cost ratio respectively. N effect on cost per availability unit is two-fold, first, increasing N increases the number of total nodes to be deployed, which drives the cost up, second, increasing N decreases WSN availability, which also increases cost per availability. Thus, increasing N has a cumulative effect of increasing the deployment cost per availability for all deployment strategies, which is what we can observe in Figure 4.10.

In general, we observe that **pro-active** outperforms both **on-demand** and **at-front**. In Figure 4.10(a) for example, **on-demand** costs almost five times more than **pro-active** per minimum availability unit for a WSN of size 13 (i.e., $N = 13$), **at-front**, on the other hand, costs more than 2 orders of magnitudes than that of **pro-active** cost. In Figure 4.10(c) shows that for a WSN size of 16, **pro-active** costs \$10 per one month of total uptime, compared to \$40 and \$500 for **on-demand** and **at-front** respectively. Furthermore, we observe that **pro-active** scalability for increasing WSN size is better than **on-demand** and **at-front**, for example in Figure 4.10(b), **on-demand** cost per average availability grows almost 2.5 times faster than that of **pro-active** and more than five times in case of **at-front**.

WSN Lifetime. Finally, we investigate the performance of the different deployment strategies as WSN required lifetime (i.e., T_{max}) increases from 10 months up to 240 months in terms of the total cost per minimum availability in Figure 4.11(a), cost per average availability in Figure 4.11(b), and cost per total uptime in Figure 4.11(c). λ, $C_{trip} : C_{sensor}$, and N are set to 0.071 failures per month, $50 : 1$ cost ratio, and 10 clusters respectively. Increasing WSN lifetime allows for more node failures, as a consequence, more deployment visits and sensor nodes are needed , which increases the total cost per min_A and avg_A availability units as we observe in Figure 4.11(a) and Figure 4.11(b) respectively. Figure 4.11(c) represents a normalized total cost. Each point in Figure 4.11(c) represents the ratio of the total cost to the corresponding T_{max}. Ideally, we expect that we pay the same cost per a unit of total uptime whether the WSN runs for 10 years or 20 years. Both **on-demand** and **pro-active** can achieve a stable cost per unit of total uptime no matter how long is T_{max}. **at-front**,

on the other hand, does not scale well for increasing T_{max} as deploying all the nodes at the beginning results in a very low availability toward the end of the WSN lifetime, especially, when T_{max} gets longer and longer.

4.7 Related Work

Our work complements previous efforts and attacks the deployment problem from a new angle. We target high availability and low total cost taking into consideration node failures and changing field trip to sensor node cost ratio. Among others, the node placement aspect of the deployment problem enjoyed the attention of many researchers [37, 49]. In these efforts, the authors investigated several node placement techniques that guarantee coverage and connectivity. They considered controlled [14, 23, 81, 84] as well as un-controlled environments [49, 81, 103]. The latter stipulates random node locations, whereas, the former allows hand-placement of nodes. Our work uses and complements existing node placement techniques by considering node failures and total cost.

Incremental deployment has been leveraged by several previous efforts to overcome application and environment uncertainty [17, 35, 36, 102, 103]. For example, in [35], the authors employed the the idea if incremental deployment, one at a time, to provide better coverage for mobile WSN in unknown environments. Early deployed nodes, discover the environment, and provide informed experience to later nodes to move to appropriate locations and provide better coverage. Our work uses incremental deployment to adapt to changing field trip to sensor node cost ratio and lower the total deployment cost.

The authors in [17] propose to use incremental deployment to achieve user-requirement path-exposure taking into consideration node and field trip costs as well as a maximum budget of sensor nodes. Path-exposure is a measure of the likelihood of detecting a target traversing the region using a given path. The higher the path exposure, the better target detection is. Our work is different than their work in several ways. First, we take into consideration node failures resulted from harsh environments. Second, our model allows the user to express availability requirement, a maximum required WSN lifetime, and network parameters (i.e., number of clusters), given these requirements, our optimization finds the optimal deployment plan including how many nodes to deploy over how

many visits. Third, we don't impose any limitations on the total number of nodes that could be deployed in our problem formulation. Fourth, we contrast our optimized deployment plan to other ad-hoc strategies and show that it outperforms these solutions in different application scenarios.

4.8 Summary

In this chapter, we attack the deployment problem from availability and total cost perspectives. We define the deployment problem as finding how many nodes to deploy in each cluster and when to deploy them. We formulate the deployment problem as a two-dimensional optimization problem whose solution yields the optimal deployment plan that satisfies availability requirement with a minimum total cost. In our evaluation, we show that the **pro-active** strategy (i.e., optimized solution) outperforms **on-demand** and **at-front** in terms of cost per availability unit.

In the next chapter, we present WSN availability modeling in the *SAIL* application. Unlike the *environmental* applications discussed in this chapter, the sensor nodes are deployed in-door and chances of catastrophic events are less probable, therefore, we employ the usage-based failure model and attack the node scheduling problem with emphasize on maintaining high WSN availability.

CHAPTER 5

MODELING THE AVAILABILITY OF AUTONOMOUS IN-DOOR WSN

Availability analysis and modeling in autonomous and remotely administered systems that are composed of cheap and failure-prone components is vital to redundancy management, which includes the prediction of the required number of components and the way these components are scheduled `ON` and `OFF`. Targeting the application of wireless sensor networks for the monitoring of elderly people living in their apartments, we use techniques from reliability theory to model the WSN as a κ-out-of-m system with independent components. In addition to predicting the required redundancy to meet the desired availability behavior early in the planning phase, we use our modeling to show that scheduling these nodes `ON` and `OFF` later on in the operational phase does indeed improve the availability behavior over the entire system lifetime. To validate our model, we design and implement a simulator using nesC/TOSSIM. Our analytical as well as experimental results show that using node scheduling almost doubles the expected WSN total uptime.

5.1 Introduction

Wireless Sensor Networks (WSNs) are envisioned to operate autonomously for long periods of time without close maintenance and supervision. This makes modeling and predicting WSNs availability behavior a very important and challenging problem. This problem includes two aspects. *One is about planning for the required number of nodes to meet desired availability behavior. The other is in controlling the way these nodes are turned* `ON` *and* `OFF` *(i.e., node scheduling) to improve the system's availability*. By availability we mean the probability that the system (i.e., WSN) will be available (i.e., functioning) at an arbitrary point of time over its lifetime. In contrast to availability, reliability means the probability of having longer continuous and uninterrupted operation of the system.

Developing such models requires a clear understanding of sensor node failure models. From the perspective of failure models, we can identify two application classes: in-door [18] and out-door [79, 98] WSN applications. The latter tends to have a hostile deployment environment, which results in more frequent and unexpected node failures due to harsh environmental conditions [93, 96]. Whereas, the former tends to provide a more controlled deployment environment, which results in fewer and more expected sensor node failures. Therefore, adopting the usage-based failure model, in which the probability of failure depends on the time a node spends in the `ON` mode, is more reasonable. Despite the controlled deployment environment in in-door WSNs, sensor nodes are still failure-prone due to their resource limitations and cheap cost. Therefore, WSN deployments are envisioned to involve a high degree of node redundancy [13, 42, 93] to overcome these limitations. Our work in this chapter is inspired by the application of WSNs for the monitoring of elderly people living in their apartments(Chapter 3).

Node redundancy and scheduling, which has been referred to as topology management, has been studied extensively in the literature [1, 13, 15, 18, 92]. In previous efforts, node redundancy and scheduling is used basically to extend the WSN lifetime beyond the lifetime of a single sensor node. Nodes are redundantly deployed so that multiple sensors are able to perform the same function. Some nodes are turned `OFF` and save energy, while others stay `ON` and perform the function. Two functions were considered, connectivity and coverage, which relate to connectivity and coverage topologies respectively. Our work complements their work by emphasizing availability as a new requirement on the node redundancy and scheduling algorithms. As topology (either connectivity or coverage) is not a concern in this work, we prefer to use redundancy management instead of topology management in this chapter.

5.1.1 Motivation of Node Scheduling

Note that turning a node `OFF` in the usage-based failure model prevents node failures. Hence, node scheduling does indeed affect WSN availability behavior. In Fig. 5.1, we show an interesting result on how node scheduling can affect WSN availability. The x-axis represents time and the y-axis

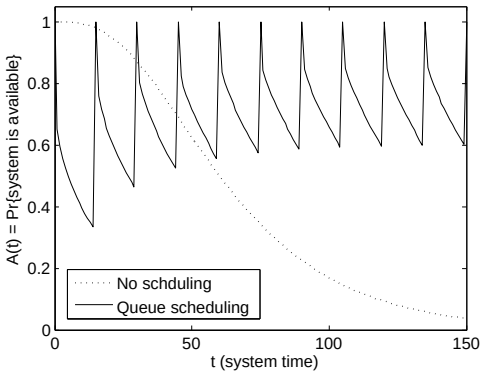


Figure 5.1: Effect of node scheduling on the availability of WSN.

represents system availability. In the *queue scheduling* scheme, the m sensor nodes are divided in groups of κ' nodes each, these groups are turned `ON` sequentially in a queue-like schedule one group at a time. As shown in the figure, *queue scheduling* exhibits more desirable availability behavior over the system lifetime than turning all the nodes `ON` since the beginning (i.e., the *no scheduling* scheme). We differentiate four aspects to describe the more desirable features. First, *queue scheduling* exhibits a higher average availability, which means that at an arbitrary time, *queue scheduling* has a higher probability that the system will be available on average. Second, *queue scheduling* exhibits a higher minimum availability, which is useful in planning for worst-case scenarios. Third, *queue scheduling* exhibits less variation in the system's availability at different stages of the system lifetime, in other words, the system is more stable. In mission-critical systems it is undesirable to have a highly available system in an early stage, while having poor availability in a later stage of the system lifetime as the *no scheduling* scheme does. Fourth, *queue scheduling* results in a longer expected total uptime.

5.1.2 Our Contribution

Our contribution in this work is four-fold. First, we model the WSN availability using sound techniques from the reliability theory [28]. Second, we use the model to solve the redundancy management problems: *deciding the number of redundant nodes needed to meet desired availability*

behavior, and the way these nodes should be scheduled to improve the availability behavior. Third, we define and formalize four performance availability metrics to compare the performance of different scheduling schemes. Fourth, we show that scheduling nodes in the usage-based failure model does indeed improve availability behavior. We show analytically and experimentally that the expected total uptime under the *queue scheduling* scheme is almost double that of the *no scheduling* scheme.

The rest of the chapter is organized as follows. Section 5.2 provides more details on the application background and requirements. Section 5.3 states formal definitions of our performance metrics and assumptions. In Section 5.4, we move on to present our node scheduling schemes and their availability modeling, and finally formalize and solve the redundancy management problems. Section 5.5 presents the evaluation results. Finally, we present related work and conclude our work in Section 5.6 Section 5.7, respectively.

5.2 Application Context and Background

Despite several initiatives, the numbers of elders with one or more physical disabilities due to lack of tools to monitor the elders' physical activities, who are living in their apartments, is still on the rise. In the *SAIL* application, sensor nodes are proposed to be deployed in each room to allow for remote unobtrusive monitoring and detection of any life-threatening accidents such as falling of elderly people.

There are several unique requirements of the application and unprecedented limitations and features of WSNs that mandate novel approaches to network management. First, *remote administration and unattended WSN operations*, which bring in the need for autonomous, self-managing, and self-healing capabilities. Second, *short-duration events*. Events that need to be detected tend to have a short time duration (e.g., falling), which makes availability take precedence over reliability. In this application, high availability of the WSN is very important to detect potential events, however, high reliability is not. Third, *relatively fixed topology*. Nodes that are deployed in the same room can be considered to form one cluster, in which any node is able to do what any other node can do. Fourth,

mild deployment environment, which makes the usage-based sensor node failure model a very reasonable assumption. Fifth, *large volume*. Wireless sensor nodes are envisioned as power-limited, resource-limited, and failure-prone devices [93], however, these sensor nodes are expected to be very cheap, which makes it feasible **to** deploy them in large volume. These characteristics make the use of node redundancy to overcome these limitations an attractive solution [13, 42, 93].

Designing systems that are highly available on one hand, and autonomous and self-healing on the other hand, puts more demand on developing tools and system models that capture node failure behavior and facilitate the prediction of the required number of nodes that are needed to empower the system to withstand failures and maintain availability and autonomy. In other words, an autonomous system should be designed to sustain failures on its own without external intervention. Therefore, we focus in this work on developing analytical models to help answer two fundamental questions: *how many nodes are required and the way these nodes should be scheduled to obtain best availability behavior in terms of either minimum availability, average availability, expected total uptime, or stability*.

Unlike usage-based failure models, deployment-based failure models capture node failures resulted from harsh environmental conditions and affect all the sensor nodes deployed in the field regardless whether these nodes are turned `ON` or `OFF`. Therefore, deployment-based failure is more suitable to out-door WSN applications and the use of a usage-based failure model is more reasonable in our application. We assume that all the sensor nodes have similar initial power and perform a similar workload, which enable them to function for an identical maximum period of time T_{max}. We further assume that all the nodes independently follow the same failure model and once a node fails, it never becomes available again (i.e., fail-stop).

5.3 Problem Statement and Metrics

The purpose of our work is to model the availability of the sensor cluster (denoted as $A(t)$) in terms of the availability of the underlying components (denoted as $S_i(t)$) under two scheduling schemes; *no scheduling* and *queue scheduling*, formalize the redundancy management problem in

each scheduling scheme, and finally to compare their performance in terms of minimum availability, average availability, stability, and expected total uptime. We use techniques from reliability theory [28] to develop these analytical models and to formalize and solve the redundancy management problems. We consider the sensor nodes of each room to form one cluster, out of which at least κ sensor nodes have to be available in order to have an available cluster. κ sensor nodes are needed instead of only one node to rule out sensor reading errors due to faulty sensors and noisy environment [58]. Deciding the value of κ is outside the scope of this work and is considered as input to our model. We use m to represent the number of nodes in each cluster. Basically, we model the availability of the system as a κ-out-of-m system. In the rest of the chapter, we use the terms system and cluster interchangeably.

We formally define our performance metrics as follows. The minimum availability (denoted as $\min_{A(t)}$) is defined as $\min_{t=0}^{T_{max}} A(t)$. The average availability (denoted as $\text{avg}_{A(t)}$) is defined as $\frac{\sum_{t=0}^{T_{max}} A(t)}{T_{max}+1}$. Stability (denoted as $\sigma_{A(t)}$) is defined as $\sqrt{\frac{1}{T_{max}+1} \sum_{t=0}^{t=T_{max}} (A(t) - avg_{A(t)})^2}$. The total uptime time and expected total uptime are denoted as U and E$[U]$, respectively and defined as the total time and expected total time, in which the system is available. Formal definitions of U and E$[U]$ for the *no scheduling* and *queue scheduling* schemes are presented and explained in Subsection 5.4.1 and Subsection 5.4.2, respectively.

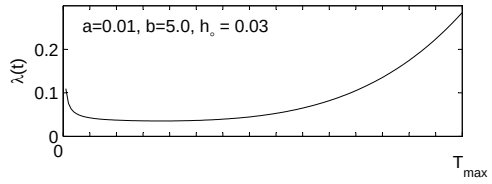


Figure 5.2: Sensor node failure-rate function.

The redundancy management problem has slightly different settings in each scheduling scheme. In the *no scheduling* scheme, all the nodes are made `ON` since the beginning and so, we are only left with finding the required number of nodes (i.e., m) to meet some availability requirements (i.e., either $\min_{A(t)}$, $\text{avg}_{A(t)}$, $\sigma_{A(t)}$, or E$[U]$). Whereas in the *queue scheduling* scheme, the m nodes are

divided into groups of κ' nodes each, and made ON in a queue-like manner, therefore, we have two problems to solve. First, find m and the corresponding κ' to meet some availability requirements. Second, given m, find κ' that optimizes availability in terms of either $\min_{A(t)}$, $\text{avg}_{A(t)}$, $\sigma_{A(t)}$, or E$[U]$. Note that the *no scheduling* scheme is indeed a special case of the *queue scheduling* scheme (i.e., make $\kappa' = 1$), however, we prefer to model the *no scheduling* scheme separately for two reasons. First, modeling the *no scheduling* scheme is easy as a classical κ-out-of-m system. Second, this scheme needs *no scheduling* management at all, which makes the implementation different from the *queue scheduling*.

The lifetime of usage-based components is typically divided into three periods, each with a different failure rate. First, an early period with decreasing failure rate; these failures are due to design and manufacturing faults. Second, a stable period with a very low and stable failure rate. Third, a wear-out period at the end of the component lifetime with increasing failure rate, these failures are due to normal wear and tear. A widely known and accepted approach to model this behavior is to use a bathtub-shaped *failure rate* function, denoted as $\lambda_i(t)$. $\lambda_i(t)$ represents the conditional probability intensity that node i will fail in the next moment, given that it has survived until time t (i.e., $\lambda_i(t) = Pr\{X_i \in [t + \mathrm{d}t] | X_i > t\} = \frac{-S_i(t)\prime}{S_i(t)}$), where $X \in [\,0, T_{max}\,]$ represents node i lifetime and $S_i(t)$ is known as the survival function and represents the unconditional probability that node i has no failures by time t and so the node is available at time t. It is known that $S_i(t) = exp\{-\int_0^t \lambda(\tau)\mathrm{d}\tau\}$ [28]. In this work, we use a failure rate function proposed recently in [5]. $\lambda_i(t)$ and the corresponding $S_i(t)$ are defined as follows:

$$\lambda_i(t) = a\,b(a\,t)^{b-1} \;+\; (\frac{a}{b})(a\,t)^{\frac{1}{b}-1} \;+\; h_\circ \tag{5.1}$$

$$S_i(t) = \exp\{-(a\,t)^b \;-\; (a\,t)^{\frac{1}{b}} \;-\; h_\circ t\} \tag{5.2}$$

Fig. 5.2 depicts $\lambda(t)$ with assumed values for a, b, and $h_\circ$. Based on our assumption that once a node dies it never becomes available again (i.e., fail-stop), we may think of $S_i(t)$ as the availability of node i at time t, which equals to $Pr\{$*node i is available at time t*$\}$. Also, note that since all the nodes follow the same failure model, we can simply use $\lambda(t)$ and $S(t)$ in the rest of the chapter.

5.4 Availability Modeling

In the following two subsections, we present availability modeling of two scheduling schemes, *no scheduling* and *queue scheduling*, including formalizing and solving the redundancy management problems in the context of these scheduling schemes.

5.4.1 The No Scheduling Scheme

We simply model the availability of the cluster as a classical κ-out-of-m system. We say that the sensor cluster is available at time t with probability $A_{no}(t)$ if and only if there exists at least κ nodes available at time t, put formally as follows:

$$A_{no}(t) = \sum_{i=\kappa}^{m} \binom{m}{i} S(t)^i \cdot (1 - S(t))^{(m-i)} \tag{5.3}$$

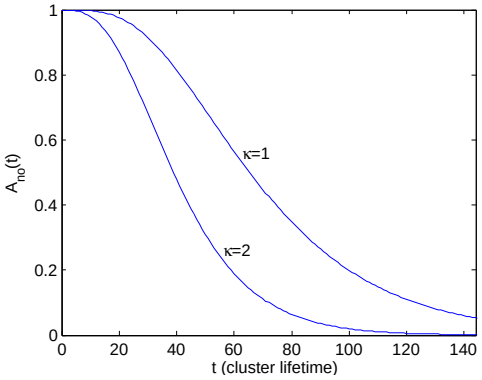


Figure 5.3: No scheduling availability.

Fig. 5.3 shows an $A_{no}(t)$ with $m = 12$ and $\kappa = 1$ and 2, and a $T_{max} = 144$. To find E$[U]$, note that the system as a whole exhibits a fail-stop behavior following its fail-stop components (i.e., sensor nodes). In other words, once there are less than κ sensor nodes available, the system fails and never becomes available again. Therefore, U is equivalent to a random variable representing the time until first failure (denoted as τ) with $Pr\{\tau > t\} = A_{no}(t)$. Note that the random variable

$\tau \geq 0$, and so the expected time until the first failure and hence E$[U]$ can be calculated as follows:

$$\text{E}[U] = \sum_{t=0}^{T_{max}} A_{no}(t) \tag{5.4}$$

As there is *no scheduling* in this scheme (i.e., all nodes are ON all the time, we are left with one question for the model to answer:

PROBLEM 1: GIVEN A VALUE OF κ, FIND THE LOWEST m NEEDED TO MEET EITHER $\min_{A_{no}(t)}$, $\text{avg}_{A_{no}(t)}$, $\sigma_{A_{no}(t)}$, or E$[U]$.

PROBLEM 1 is a simple optimization problem that can be solved iteratively over m starting with $m = \kappa$, incrementing m by one each time, and checking whether the current value of m meets the requirement (i.e., $\min_{A(t)}$, $\text{avg}_{A(t)}$, $\sigma_{A(t)}$, or E$[U]$). To find $\min_{A_{no}(t)}$ for a given m, simply substitute ($t = T_{max}$) in Equation (5.3). Note that $A_{no}(t)$ is consistently decreasing with time as we can observe from Fig. 5.3, hence, $min_{A_{no}(t)} = A_{no}(T_{max})$. For $\text{avg}_{A(t)}$ and $\sigma_{A(t)}$, we simply follow the definition to calculate them for a given m value. From the definition of $\text{avg}_{A(t)}$ in Section 5.3, we get $(\text{T}_{max} + 1) \cdot \text{avg}_{A_{no}(t)} = \sum_{t=0}^{\text{T}_{max}} A_{no}(t)$. By substituting in Equation (5.4), we get:

$$\text{E}[U] = \text{avg}_{A_{no}(t)} \cdot (\text{T}_{max} + 1) \tag{5.5}$$

Thus, the value of m needed to meet the $\text{avg}_{A(t)}$ requirement is the same value of m that is needed to meet E$[U]$. In other words, the solution of PROBLEM 1 for $\text{avg}_{A(t)}$ and E$[U]$ is the same.

5.4.2 Queue Scheduling Scheme

Unlike the *no scheduling* scheme, *queue scheduling* divides the m nodes into $\eta = \frac{m}{\kappa'}$ groups (denoted as g_i, where $i = 1, \ldots, \eta$). Each group, g_i, consists of κ' nodes, where ($\kappa \leq \kappa' \leq m$). Given these η groups, the time is divided into η epoches with equal periods denoted as $\Delta = \frac{T_{max}}{\eta}$. Each group g_i is turned ON, in a queue-like scheduling, at the beginning of its corresponding epoch (denoted as ϵ_i).

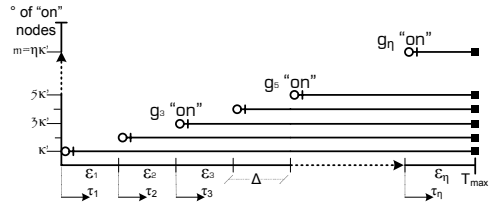


Figure 5.4: Timeline of queue scheduling.

Fig. 5.4 depicts the *queue scheduling* timeline. The x-axis represents time, while the y-axis represents the total number of ON nodes shown as discrete value multiples of κ'. Unlike in the *no scheduling* scheme, the total uptime (i.e., U) in *queue scheduling* is different from the time until the first failure since the system may fail and becomes available again when a new group is turned ON. However, in a single epoch, the system exhibits a fail stop behavior. Therefore, we can define the total uptime of the system as the summation of random variables representing the time until the first system failure in each time epoch (shown as τ_i in Fig. 5.4). Formally, $U = \sum_{i=1}^{\eta} \tau_i$, where, $0 \leq \tau_i \leq \Delta$. Hence:

$$\mathrm{E}[U] = \sum_{i=1}^{\eta} E[\tau_i] \tag{5.6}$$

Now, we turn our attention to find the system availability ($A_Q(t)$), which also represents the probability distributions of the random variable τ_i. Perhaps the first thing that comes to mind when trying to model the availability of *queue scheduling* is the renewal process model. Unfortunately, the fact that renewals (i.e., bringing node groups (g_i) ON) are asynchronous to failures causes two major inconsistencies with the classical renewal process model assumptions. First, *lack of instantaneous repairs*. In other words, should the system fail during epoch i (i.e., ϵ_i), it will not be available until the beginning of the next time epoch (i.e., ϵ_{i+1}). Recall that in our usage-based failure model assumption Section 5.3, nodes have to be in the OFF mode to avoid failures, which makes them un-responsive to external events and therefore can not be asked to become active on the spot in

case a failure is detected. On the other hand, in unattended and remotely administered WSNs, human intervention is infeasible and violates the key non-obtrusive application requirement. Second, *non-homogeneity of the availability probability distribution during different time epoches.* In other words, τ_i in Fig. 5.4 are not identically distributed.

```
1. function result=Q(t, e, i, p)
2. t' = t – (e - 1) * Δ
3. if (e == 1){ // base case
4.   if (i > k'){
5.      return 0 // no enough nodes to choose from
6.   }
7.   return (k' i) * S(t')^i * (1 - S(t'))^(k' - i) * p
8. }else{
9.     minJ = max(0, i - (e - 1) * k')
10.    if (min > k'){
11.       return 0 // no enough nodes to choose from
12.    }
13.    maxJ = min(i, k')
14.    tmpP = 0
15.    for (j = minJ; j <= maxJ; j++){
16.      current_p = p * (k' j) * S(t')^j * (1 - S(t'))^(k' - j)
17.      tmpP = tmpP + Q(t, e - 1, i - j, current_p)
18.    }
19.    return tmpP
20. }}
```

Figure 5.5: Q: finds recursively the probability of having exactly i nodes available.

In light of the above *queue scheduling* algorithm complications, we use a recursive numerical function to model the system availability at an arbitrary time instance (i.e., $A_Q(t)$). Let $Q(t, e, i, p)$ be the probability that there are exactly i nodes available at time t, then:

$$A_Q(t) = \sum_{i=\kappa}^{\kappa' \cdot e} Q(t, e, i, 1.0), \text{ where} \tag{5.7}$$
$$e = \left\lfloor \frac{t}{\Delta} \right\rfloor + 1$$

The function **Q** finds the probability of having exactly i nodes available out of $e \cdot \kappa'$ nodes that are already turned ON by time t. Q considers all the possible i node combinations by looping recursively over e. Fig. 5.5 lists the pseudocode of function Q. In line 2 of Fig. 5.5, Q finds the total ON time (t') of the current group (represented by the variable e) as a shift of the global time t. Hence, the availability of the current node group becomes $S(t')$. Lines 3 through 6 in Fig. 5.5

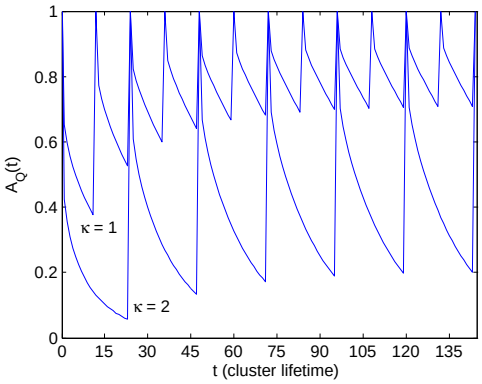


Figure 5.6: Queue scheduling availability.

represent the base case scenario (i.e., $e = 1$), in which only one group of nodes exists. Therefore, Q returns the probability of having i nodes available out of κ' ON nodes. Lines 15 through 17 loops recursively over all the possible combinations. Fig. 5.6 shows $A_Q(t)$ for $m = 12$, $\kappa = 1$, and $\kappa = 2$, and $T_{max} = 144$. κ' is set equal to κ for simplicity.

Now, we shift our gear to formalize and solve the redundancy management problems for the *queue scheduling* scheme. As in the *no scheduling* scheme, first the redundancy management problem is concerned with finding the needed number of nodes (i.e., m) to meet the desired availability requirement put formally as follows:

PROBLEM 2: GIVEN κ, FIND THE LOWEST m AND CORRESPONDING κ' NEEDED TO MEET EITHER $\min_{A_Q(t)}$, $\text{avg}_{A_Q(t)}$, $\sigma_{A_Q(t)}$, OR $\text{E}[U]$.

Again, we may solve this simple optimization problem iteratively over m and κ' starting from $m = \kappa$, incrementing m by one, and checking against the requirement. For each value of m, κ' is changed from κ up to m. Fig. 5.6 illustrates the cluster availability for the queue scheduling schemes with two different κ values. We can observe from Fig. 5.6 that $min_{A_Q(t)}$ happens at $t = \Delta - 1$, hence, $min_{A_Q(t)} = A_Q(\Delta - 1)$. To find $\text{E}[U]$, we may rewrite $avg_{A_Q(t)}$ as a piece-wise summation with Δ time intervals as follows:

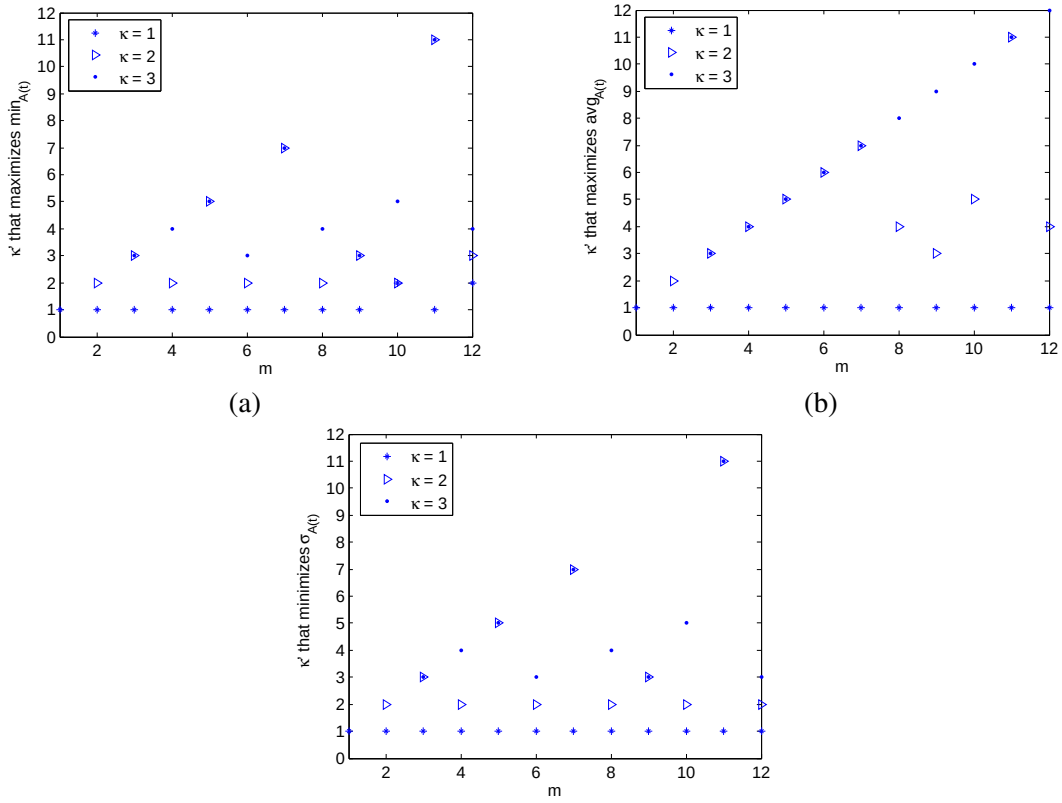


Figure 5.7: The solutions of the PROBLEM 3: (a) for $\min_{A(t)}$, (b) for $\text{avg}_{A(t)}$, and (c) for $\sigma_{A(t)}$.

$$\text{avg}_{A_Q(t)} = \frac{\sum_{t=0}^{\Delta-1} A_Q(t) + \cdots + \sum_{t=(\eta-1)\cdot\Delta}^{\eta\cdot\Delta-1} A_Q(t)}{T_{max}+1} \tag{5.8}$$

Note that the i^{th} summation term in Equation (5.8) equals $E[\tau_i]$, hence, from Equation (5.6), $E[U]$ equals $avg_{A_Q(t)} \cdot T_{max} + 1$. Therefore, optimizing for $avg_{A_Q(t)}$ is the same as optimizing for E$[U]$.

We can re-formulate PROBLEM 2 to find the optimal κ', given a budget of m nodes from which at least κ nodes should be available. Putting the problem this way is useful in two scenarios. First, if there is a limited budget on the allowed number of nodes m in the planning phase. Second, if κ'

needs to be adapted during the operational phase by considering the actual new m nodes available in the WSN after some node failures. The optimization problem is put formally as follows:

PROBLEM 3: GIVEN m AND κ, FIND κ' THAT MAXIMIZES $\min_{A_Q(t)}$, $\text{avg}_{A_Q(t)}$, OR MINIMIZES $\sigma_{A_Q(t)}$.

As PROBLEM 1 and PROBLEM 2, PROBLEM 3 can be solved iteratively over κ' starting from $\kappa' = \kappa$ and incrementing κ' one by one until $\kappa' = m$, and finding the optimal κ'. Fig. 5.7 shows solutions of PROBLEM 3 for $\min_{A_Q(t)}$ (Fig. 5.7(a)), $\text{avg}_{A_Q(t)}$ (Fig. 5.7(b)) and $\sigma_{A_Q(t)}$ (Fig. 5.7(c)). From these three figures, we can easily observe that the optimizing values of κ' for $\min_{A_Q(t)}$, $\text{avg}_{A_Q(t)}$, and $\sigma_{A_Q(t)}$ are not the same. For example, for $m = 8$ and $\kappa = 3$, κ' that optimizes $min_{A_Q(t)}$ is 4, whereas, κ' that optimizes $avg_{A_Q(t)}$ is 8.

5.5 Simulation and Evaluation

We design and implement a simulator in the nesC/TOSSIM environment to verify our modeling results and to show that *queue scheduling* outperforms *no scheduling* in terms of $\min_{A(t)}$, $\text{avg}_{A(t)}$, $\sigma_{A(t)}$, and E$[U]$. Therefore, node scheduling is a useful technique for improving the system availability for in-door WSNs. nesC [26] is a well-known programming language for WSNs and TOSSIM [51] is a simulator that can simulate nesC applications.

5.5.1 Simulation Setup

We arrange the m sensor nodes into one cluster with a pre-determined cluster head, the cluster head is responsible for performing optimization and finding κ', dividing the m sensor nodes into η groups, and assigning each group the time it is supposed to become `ON` (denoted as α_i in Fig. 5.8). The cluster head is made aware of the total number of nodes in the cluster (i.e., m) and the required number of `ON` sensor nodes (i.e., κ). As we discussed earlier, m is determined in the planning phase using our model, whereas κ is an input to the simulator and made available at the cluster head as a booting parameter. We argue that this centralized and fixed clustering structure is reasonable and supported by the Tenet architecture recently proposed in [27]. In the simulator environment, we maintain a global simulation time component that is responsible to maintain and advance the global

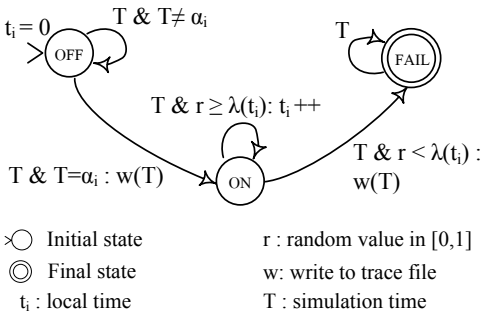


Figure 5.8: The state-transition diagram.

simulation time, denoted as T, and to notify all the sensor nodes in the cluster at each time unit by firing an event at the nodes. The global simulation time starts at zero and stops at T_{max}, which marks the end of the simulation. Each sensor node is responsible to maintain its local simulation time (denoted as t_i), which represents the node's time since it became `ON` and basically is a shift of the global simulation time T. t_i is used as input to the failure rate function (i.e., $\lambda(t)$) to decide whether the node should fail or not in the next moment. The sensor nodes are responsible to write two events, *start time* and *fail time*, on the simulation trace.

Fig. 5.8 shows a three-state diagram explaining the sensor node state transitions. All the sensor nodes start in the `OFF` state and stay in that state until their start time assigned by the cluster head does not equal the announced global time T. Otherwise, the sensor nodes switch to the `ON` state and write their start time to the global simulation trace. While in the `OFF` state, the node's local time (i.e., t_i) is not incremented. Once the node enters the `ON` state, it starts incrementing its local time (i.e., t_i) and checks whether the node should fail (i.e., switch to `FAIL` state) or not whenever a new T is announced. Recall that we use $\lambda(t_i)$ as the failure probability, therefore, we use a random variable uniformly distributed over the interval [0,1], (denoted as r in Fig. 5.8), and compare it to $\lambda(t_i)$ to achieve the right failure probability. Should the node fail, it writes its failure time to the global simulation trace and switches to the `FAIL` state and stays in that state until the end of the simulation. Once all the nodes enter their `FAIL` state, the simulation ends.

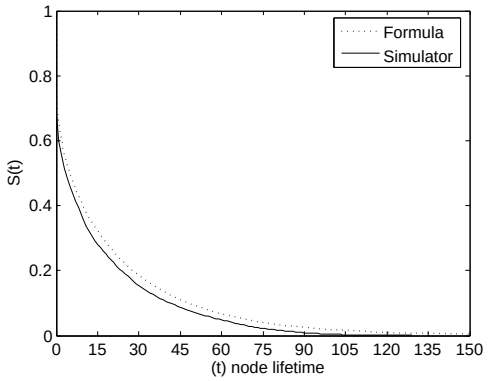


Figure 5.9: Generating S using the simulator and validating it against the formula.

5.5.2 Model Verification

Before presenting our evaluation results, we validate our simulator by re-generating the same single node survival function (i.e., $S(t)$ in Equation (5.2)). Later on, we use the simulator to validate our scheduling schemes modeling.

Fig. 5.9 compares the survival function drawn directly from Equation (5.2) to which we get from the simulator. The x-axis represents time (t), the y-axis represents failure probability. We perform a thousand single-node simulation runs and record failure times. Each point represents the ratio of the number of runs in which the node was available to the total number of runs at time t. We can observe an excellent match, where the average difference between experimental S values and analytical S values over all values of t equals to 0.025. This demonstrates the accuracy of our simulator.

In Fig. 5.10 we move on to validate our scheduling schemes availability modeling in Equation (5.3) and Equation (5.7) against the system availability obtained experimentally from the simulator. The x-axis represents time, the y-axis represents system availability. For each scheduling scheme we perform one thousand runs, record single-node failure times, and calculate system availability at time t as the ratio of runs in which at least one node is available (i.e., $\kappa = 1$) at time t, to the total number of runs. We observe an excellent match between analytical and experimental data with an average difference of 0.093 and 0.065 in the *no scheduling* and *queue scheduling* schemes, respectively. There are two more important observations that need further explanation. First, the

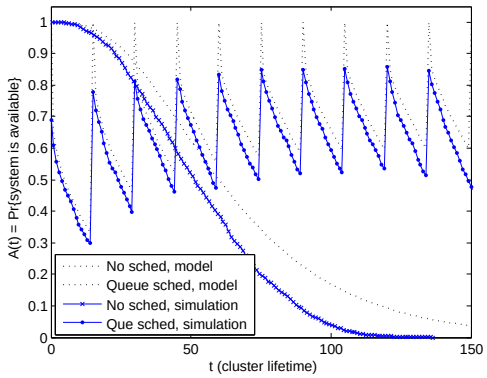


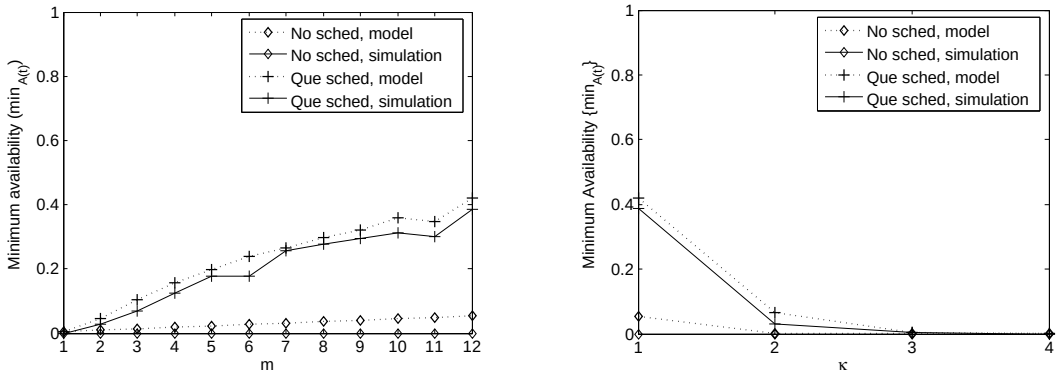
Figure 5.10: Matching model and simulation.

mismatch between the model and the simulator is bigger than that of the single node scenario in Fig. 5.9, whose average difference is 0.025. Second, the mismatch in the *no scheduling* scheme is bigger than that of the *queue scheduling* scheme. The observed bigger mismatch in both cases is due to the mismatch accumulation resulting from the several nodes in the system. This is easy to understand in the first case. In the second scenario, note that the number of nodes that are turned ON in the *no scheduling* scheme is bigger than that of the *queue scheduling* scheme, which results in large-mismatch accumulation. This explanation is also supported by the observation that the mismatch grows as more and more nodes are turned ON as time passes in *queue scheduling*.

5.5.3 Evaluation Results

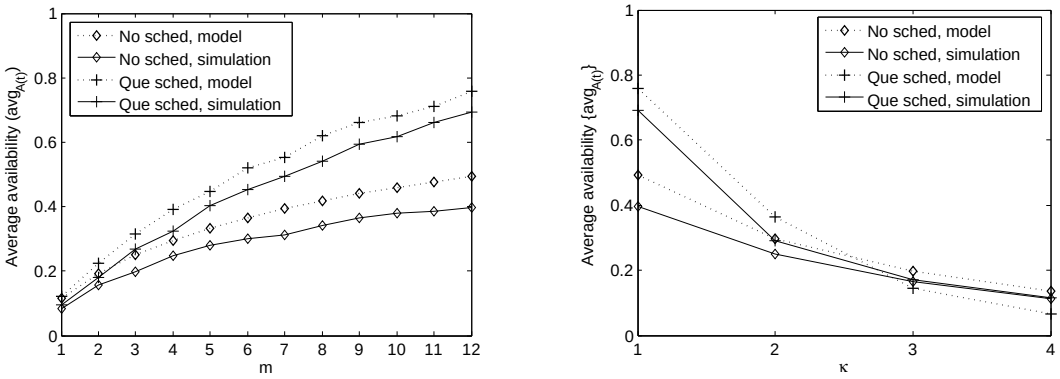
In this subsection, we present our evaluation results to show that node scheduling has the ability to improve the system availability. We compare the performance of the *no scheduling* and *queue scheduling* schemes in terms of $\min_{A(t)}$, $avg_{A(t)}$, $\sigma_{A(t)}$, and $\mathrm{E}[U]$ as node redundancy increases. We show that *queue scheduling* indeed exhibits better results than *no scheduling* in terms of all metrics. We control the node redundancy by changing both m and κ. In *queue scheduling*, κ' is chosen to optimize the corresponding performance metric under evaluation.

Fig. 5.11 shows the analytical as well as experimental minimum availability comparison results as m increases in Fig. 5.11(a), and as κ increases in Fig. 5.11(b). Experimentally, we perform one



(a) effects of increasing m. (b) effects of increasing κ.

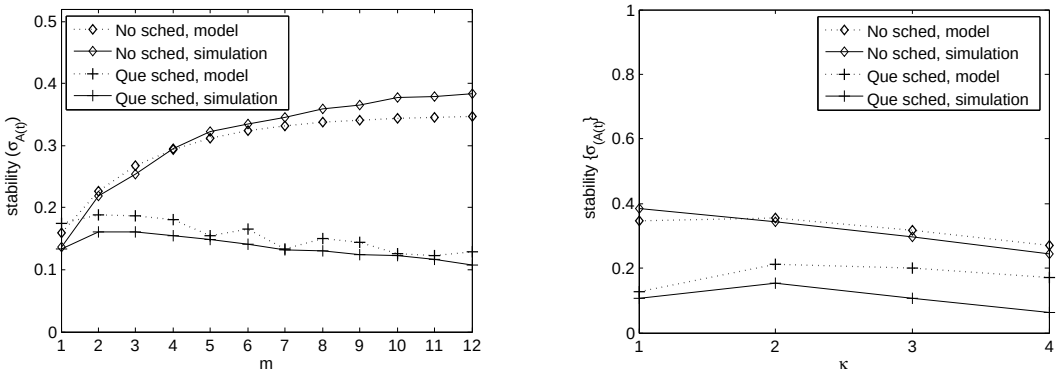
Figure 5.11: Comparing $min_{A(t)}$ under no scheduling and queue scheduling.



(a) effects of increasing m. (b) effects of increasing κ.

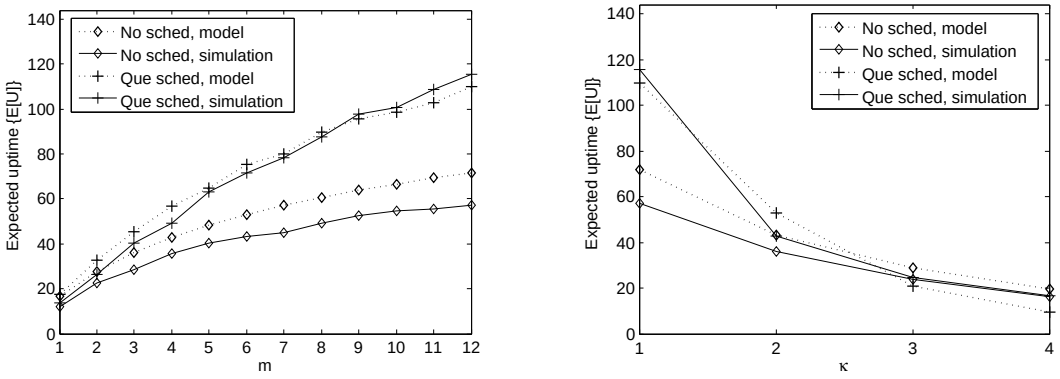
Figure 5.12: Comparing $avg_{A(t)}$ under no scheduling and queue scheduling.

thousand runs for each m and κ value, calculate the system availability as we did in Fig. 5.10 for all $t \in [0, T_{max}]$, and finally find the corresponding minimum availability. We can see that *queue scheduling* exhibits a consistent and larger increase in the minimum availability compared to *no scheduling*. For $m = 12$ in Fig. 5.11(a), the minimum availability goes from almost zero in the *no scheduling* scheme up to 4.0 in the *queue scheduling* scheme. Also, note that as m and κ get closer (i.e., less redundancy), the performance of *queue scheduling* and *no scheduling* becomes closer, which is simply because *queue scheduling* converges to *no scheduling*. In other words, if m and



(a) effects of increasing m. (b) effects of increasing κ.

Figure 5.13: Comparing $\sigma_{A(t)}$ under no scheduling and queue scheduling.



(a) effects of increasing m. (b) effects of increasing κ.

Figure 5.14: Comparing E$[U]$ under no scheduling and queue scheduling.

κ are the same, the only way to schedule the nodes is to make all of them ON from the beginning, which is the same as *no scheduling*.

Like in Fig. 5.11, in Fig. 5.12 we perform one thousand runs for each m and κ value, calculate the system availability, and finally find corresponding average availability over $t \in [0, T_{max}]$. Again, we observe that *queue scheduling* outperforms *no scheduling*, in particular for high node redundancy. For example, for $m = 12$ and $\kappa = 1$ in Fig. 5.12(a), *queue scheduling* almost doubles the average system availability compared to *no scheduling*. For $m = 12$ in Fig. 5.12(a), the average availability

goes from 0.4 up to 0.7 based on the experimental results, and from 0.5 up to 0.75 based on the model.

In Fig. 5.13, we move on to compare *no scheduling* and *queue scheduling* in terms of stability. As we mentioned earlier, we use standard deviation as a measure of stability (i.e., the y-axis). Note that unlike Fig 5.12 and Fig. 5.11, a lower value on the y-axis in Fig. 5.13 means better stability. Again, we observe better stability in case of *queue scheduling*. Furthermore, we observe that *queue scheduling* achieves better and better stability as redundancy increases, whereas, *no scheduling* keeps on getting worse as redundancy increases.

Finally, in Fig. 5.14 we compare *no scheduling* and *queue scheduling* in terms of the expected total uptime as m increases in Fig. 5.14(a) and as κ increases in Fig. 5.14(b). The y-axis represents the total expected uptime. Again, we observe that *queue scheduling* outperforms *no scheduling* by increasing the total time in which the system is available. For example, for $m = 12$ in Fig 5.14(a), the system total uptime is almost 105 hours when *queue scheduling* is used compared to less than 60 hours when *no scheduling* is used. The improvement is almost double.

In summary, we conclude that *queue scheduling* outperforms *no scheduling* significantly in terms of all metrics.

5.6 Related Work

Sensor node redundancy and node scheduling have been extensively studied in the wireless sensor network research community [1, 13, 15, 18, 92]. The basic idea in their work is to use node redundancy and scheduling to work around the battery lifetime limitation of sensor nodes and extend the network lifetime while maintaining coverage and connectivity. Our work complements their work by adding a new dimension (i.e., availability) in the node redundancy and scheduling protocol design space and exploring more sensor node failure models, namely usage-based, in addition to the trivial running-out-of-battery sensor node failure model. Furthermore, coverage and connectivity are minor objectives when performing the node scheduling process in our application, as all the nodes in the same cluster can communicate directly to the cluster head and can provide the same

coverage. This setting has been also supported in the latest wireless sensor network architecture (i.e., Tenet) proposed in [27].

In [96], the authors propose an adaptive sleeping schedule of redundant nodes based on application demands and network conditions. They associate a backup set with each active node. The backup sets are made active regularly to take over in case an active node failure is detected. They considered two node failure models; aging failure model and catastrophic event failure model, which correspond to our usage-based and deployment-based failure models. Our work differs from theirs in two aspects. First, they adopt a reactive failure detection and recovery approach in contrast to our proactive approach, in which we optimize for the best possible scheduling scheme in advance given a specific failure model. Second, we address a more fundamental problem, which is predicting the required node redundancy to reach a desired fault tolerance behavior. In a sense, their adaptive scheduling protocol can be integrated in our solution framework, which may result in more efficient scheduling schemes in different application contexts. PEAS, proposed in [93], leverages node redundancy and scheduling to overcome harsh deployment environments, which cause frequent node failures. In other words, they target deployment-based failure model. Like DADA [96], their approach is reactive in contrast to our proactive approach.

5.7 Summary

In this chapter, we develop analytical models of WSN availability based on reliability theory. The model allows for the prediction of the required number of nodes to meet desired availability behavior as well as the way these nodes should be scheduled. Furthermore, we show that in the usage-based failure model, node scheduling improves availability significantly. Due to the small number of large-scale real WSN deployments, the wireless sensor network research community still lacks real traces of node failures that allow the development of strong sensor node lifetime models. Therefore, we adopt the well-known bathtub failure model to capture usage-based sensor node failures. However, our availability modeling can be easily adapted to employ any sensor node

failure model once it becomes available. In the next chapter, we take the opportunity to study a one-month sensor failure traces made available to us through the United States Army Corps of Engineers (USACE).

CHAPTER 6

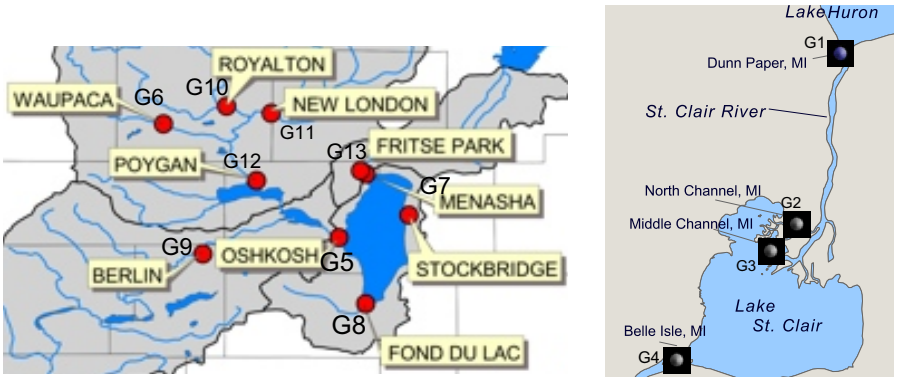
FAILURE ANALYSIS

In this chapter, we take an initial step and analyze a one-month worth of collected sensor failure traces. These traces are collected from a real-world water system surveillance application. Our findings in this chapter supports some of the assumption we adopted in the previous two chapters. Next, we provide some details on the water surveillance application and how these sensing units deployed and operate in this application.

6.1 Sensing System Setup

The United States Army Corps of Engineers (USACE) in Detroit District has 22 data collection platforms. These sensor nodes or gauges are deployed around the St. Clair and Detroit rivers in southeast Michigan as well as the Lake Winnebago watershed southwest of Green Bay, Wisconsin. One month data in January 2008 from 13 of the 22 gauges were made available for this study. Each sensor node collects battery voltage, water level and precipitation except the "Dunn Paper" gauge (G1) which collects battery voltage, air temperature and water temperature. However, precipitation data for the St. Clair/Detroit river system is not used in this work, because that "data" in the raw files is simply an artifact of the gauge programming. For convenience, we name each sensor node as G1, where 'G' stands for "gauge.". G1, G2 and G3 are located on the St. Clair River; G4 is located on the Detroit River; and G5 through G13 are scattered around the Lake Winnebago watershed. Gauges G5 through G13 are shown in Figure 6.1(a), the remaining gauges are shown in Figure 6.1(b).

The gauges are equipped with satellite transmitter units. The data collection system works as follows: sensing units of each gauge continuously record measurements locally, these measurements are transmitted over the GOES satellite channel every hour (for gauges with low baud rate transmitters) and every four hours (for high baud rate transmitter equipped gauges). Data is then sent from



(a) (b)

Figure 6.1: Map of gauge location. (a) Lake Winnebago Watershed. (b) St. Clair River and Detroit River

the satellite to a central location in Wallops Island in Virginia, where the data samples are collected and arranged in files for later download through a regular ftp service. We conducted our analysis directly using the un-decoded files. This raw data set has not been subject to any quality control procedure, and thus provide a good opportunity to study failures happening in sensor network. Water level and precipitation are sampled once every hour, whereas voltage is sampled once every hour or every two hours. Water level is measured against the IGLD Datum 1985, which works as the base to measure current water level. So negative water level means it is below the local IGLD Datum 1985. Precipitation data is supposed to be constantly increasing (except when the gauge resets as part of its normal operation). To figure out how much precipitation fell over a one-hour period, the difference between two consecutive samples are calculated and reported. The measurements for voltage and precipitation are in volts and inches respectively. Water level is reported in meters, centimeters and feet. For convenience, we converted the readings for water level in meters.

6.2 Failure Analysis

In this section we study the failure patterns of the sensor system including communication related failures and sensing hardware related failures. First, we present a few important definitions.

TTF denotes Time To Failure and represents the time between two consecutive failures. Mean TTF (MTTF) is a measure of the system reliability.

TTR denotes Time to Repair that is the time it takes the system to recover from a failure. A system that exhibits a small Mean TTR (MTTR) typically maintains high availability.

Total time: represents the total system lifetime including functioning as well as failing periods.

Uptime: the total time a system is in the functioning mode, in contrast, **Downtime** is the total time, in which the system is un-available. The following two equations illustrate the relationship between these values:

$$MTTF = \frac{Total\ time}{number\ of\ failures} \tag{6.1}$$

$$Downtime = MTTR \cdot number\ of\ failures \tag{6.2}$$

6.2.1 Methodology

For each sensing parameter (i.e., Water level or Precipitation), we organize the readings as a discrete time series and locate missing or corrupted readings. Each missing or corrupted reading is considered a failure. For each time series, we record the **Number of Failures**, **TTFs**, and **TTRs** for each individual failure type independently.

In our investigation of the raw data traces, we discovered several failure types. Some of these failures are related to communication failures, while others are pertinent to the sensing hardware itself. Table 6.1 lists all the failure types we encountered in the raw data along with a simple description for each one of them. The first four failures in the figure (i.e., Comm-T1 to Comm-T4) are communication failures between the gauge station and the satellite unit. The last two failures (i.e., H/W-T1 and H/W-T2) are sensing hardware malfunctioning. H/W-T1 represents a fail-stop failure, where the sensor simply fails to report a reading on time, whereas, in H/W-T2, the sensor reports a corrupted reading (i.e., unreadable values).

Table 6.1: Failure types and their description.

Failure Type	Message in raw trace files	Description
Comm-T1	"ADDRESS ERROR CORRECTED"	Unknown reasons. We could not reach technical people who could provide explanation.
Comm-T2	"MISSING SCHEDULED DCP MESSAGE"	Communication failure due to lack of time synchronization between the gauge station and the satellite unit.
Comm-T3	"MESSAGE RECEIVED ON WRONG CHANNEL"	The message was not received on the channel that has been assigned to that particular gauge station.
Comm-T4	"MESSAGE OVERLAPPING ASSIGNED TIME WINDOW"	Communication failure due to lack of time synchronization between the gauge station and the satellite unit.
H/W-T1	Blank	Sensor failure, no reading was reported by the sensor on time. This represents a fail-stop sensor failure.
H/W-T2	Corrupted reading	Sensor failure, the data format is corrupted, unreadable reading.

6.2.2 Failure Analysis by Type

To understand the relative importance of these failure types, we draw their relative occurrence in the raw data traces for all locations combined in Figure 6.2(a) and the total downtime due to the particular failure type in Figure 6.2(b). Figure 6.2(a) gives an idea of how frequent a particular failure type is, whereas, Figure 6.2(b) clarifies how severe that failure is, in other words, how long it takes to recover from the failure.

In Figure 6.2(a), we observe that 56% of the total number of failures are of type Comm-T2 communication failure, all other communication related failures (i.e., Comm-T1, Comm-T3, and Comm-T4) collectively account for only 6% of the total number of failures. Comm-T2 as well as Comm-T4 are directly related to the lack of time synchronization between the gauge station and the satellite unit, thus, the lack of time synchronization constitutes 58% of the total number of failures. Failure to report measurements by the sensor hardware on time (i.e., H/W-T1 failure) accounts for 34% of the total number of failures, in contrast, reporting corrupted data (i.e., H/W-T2 failure)

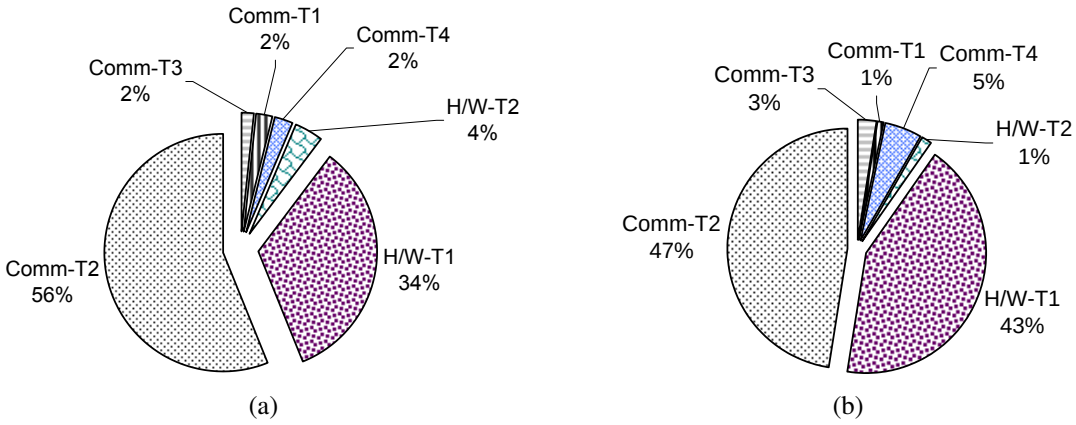


Figure 6.2: Understanding relative importance of different failure types. (a) shows their relative frequency (b) shows their contribution to system total downtime.

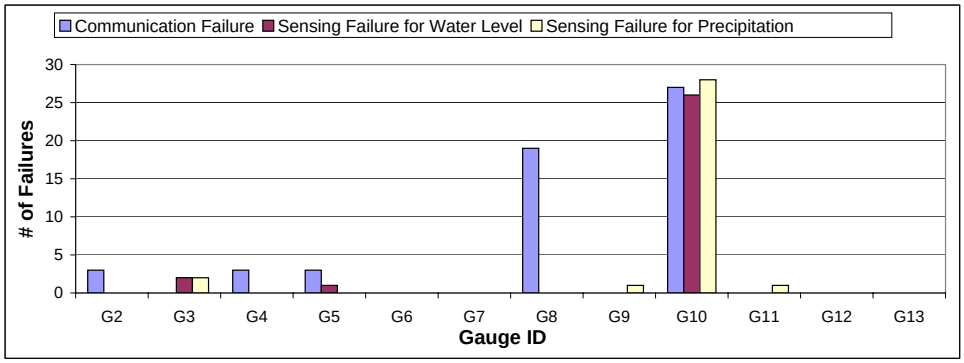
accounts for only 4%. This observation suggests that fail-stop failures are more common in the real world environmental applications.

Figure 6.2(b) allows us to observe the importance of the different failure types from a different perspective, in particular, how long it takes to recover from a particular failure type. For example, Although Comm-T4 and Comm-T1 failure types account for the same percentage of the total number of failures (i.e., 2% as shown in Figure 6.2(a)), it seems that recovering from a Comm-T4 failure takes more time compared to recovering from a Comm-T1 failure, which allows us to conclude that Comm-T4 failures are more important than Comm-T1 failures, in other words, Comm-T4 failures contribute 5% to the system total downtime, whereas, Comm-T1 contributes only 1% as shown in Figure 6.2(b). We also, observe that sensing hardware failures (i.e., H/W-T1 and H/W-T2 failures) account for 44% of the total system downtime.

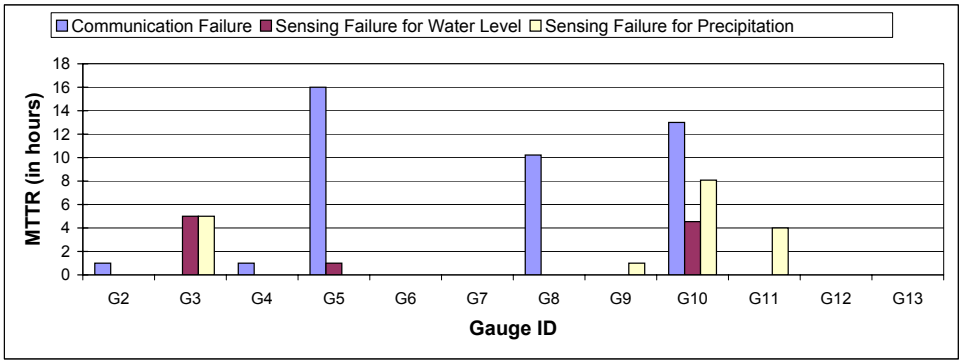
We observe in Figure 6.2 that we have two equally important major failure types, communication related failures and sensing hardware failures. Based on these findings and after consultation with field experts, we decide to merge these failure categories and abstract them into two failure classes: communication failures and sensing hardware failures in the rest of this section.

6.2.3 Failure Analysis by Location

In this subsection, we study failure characteristics at different locations, which allows us to understand the effect of the environment on inflicting failures on the system. At each location, we record the Number of failures and MTTR for each failure type and draw them in Figure 6.3(a) and Figure 6.3(b) respectively. Note that MTTF is directly proportional to the Number of failures (refer to Equation 6.1), therefore, including MTTF offers no insight in our analysis.



(a) Effect of environment on failure frequency.



(b) Effect of environment on MTTR.

Figure 6.3: Understanding effect of external environment on inflicting failures.

Figure 6.3 allows us to observe that gauge G10 experience much more failures compared to the other gauges, because of the hostile environment surrounded G10. This suggests that the external environment plays a significant role in the failure frequency and pattern of the system. We also observe in Figure 6.3(a) that the environmental impact is uniform in inflicting different failure types.

For example, Figure 6.3(a) shows that gauge G10 suffered around 26 communication failures, 26 water level sensor failure, and 28 precipitation sensor failures, whereas, Gauge G3 experienced 1 water level sensor failure, 1 precipitation sensor failure, and 0 communication failures.

In Figure 6.3(b), we observe that different failure types need different recovery time. For example, at gauge G10 in Figure 6.3(b), precipitation sensor failure takes more time on average to recover from a failure compared to water level sensor failure. Surprisingly, communication failures exhibit much longer repair time.

6.2.4 Summary and Implications

Based on our findings, we believe that the lack of time synchronization is a major source of communication failures, this makes time synchronization algorithms of particular importance in remotely deployed environmental monitoring sensor applications. Sensor hardware failures are also a major source of failures in these applications, we found that fail-stop failure is the most common failure pattern in this category. We further observed that different sensor hardware exhibit different failure characteristics, in particular, different repair times. Finally, we found that external environmental conditions, perhaps, are the most important factor on inflicting failures in environmental applications. This makes deployment-based failures particularly important, in which failures are independent of aging.

6.3 Related Work

There have been a few efforts that focus on understanding failure patterns of computer hardware components, in particular hard disks [39, 70]. Our work employs similar techniques and investigates a different type of system that is deployed in an open environment, therefore, we believe that its failure behavior is totaly different from classical computer systems. To the best of our knowledge, there is no prior work that specifically studies and analyzes real sensor failure traces so that our work is a leading exploration step in this direction. Most existing work on WSN reliability assumes exponential lifetime distribution of sensor nodes [2, 96], and here we take a second look of such assumption by investigating real sensor system failure traces. Although the current data set

spans a short period of time and does not permit us to draw strong conclusions regarding lifetime distributions, we believe that our work brings new insights in understanding sensor device failure patterns.

6.4 Summary

In this work, we use real sensor failure traces collected by 13 sensor nodes deployed by the United States Army Corps of Engineers (USACE) around St. Clair and Detroit rivers in southeast Michigan as well as the Lake Winnebago watershed southwest of Green Bay, Wisconsin. We found that the lack of time synchronization is a major source of communication failures in the system, which suggests that we should pay more attention to time synchronization protocols in remotely deployed WSN environmental applications. We also found that external harsh environmental conditions may be the most important factor on inflicting failures on sensor nodes in outdoor applications.

CHAPTER 7

SYSTEMS SUPPORT

In this chapter we move our attention to present and discuss *Score*, which defines the structure of the communication protocol stack and provides protocol components with the adequate means to collaborate in a cross-layer approach. In addition to *Score*, we present two important protocol components that run in the context of *Score* and provide system support for our other protocol components: the *neighbor discovery* and *topology discovery* services. To demonstrate the benefits of the *Score* framework and its system support advantages, we present and discuss two higher-level communication protocols, a routing protocol, Density-Aware-GPSR (*DA-GPSR*), and a dissemination protocol, Redundancy-Aware Controlled flooding protocol (*RAC*).

7.1 Score

The Internet protocol stack is widely known for its modular layered design, in which the cross-layer interface is confined to adjacent layers, where a higher layer uses services provided by the layer immediately beneath it in the stack. Researchers think that this will not be the case in the future wireless sensor network (WSN) communication stack due to several reasons including limited energy supply, limited computational power, and unreliable wireless communication [16, 43]. These limitations make the need for more optimal solutions another primary requirement besides modularity. Protocol optimizations are usually possible by allowing layers to collaborate more closely when performing their functions; this technique is widely known as cross-layer design. It is believed that cross-layer design is the key to the self-optimization of WSN communication stack protocol layers, and so overcome the energy and computing limitations of wireless sensor nodes [7, 16, 29, 43].

Typically, in cross-layer design, some pieces of information at one layer are used to improve the performance of another layer in the communication stack. For example, a routing protocol can

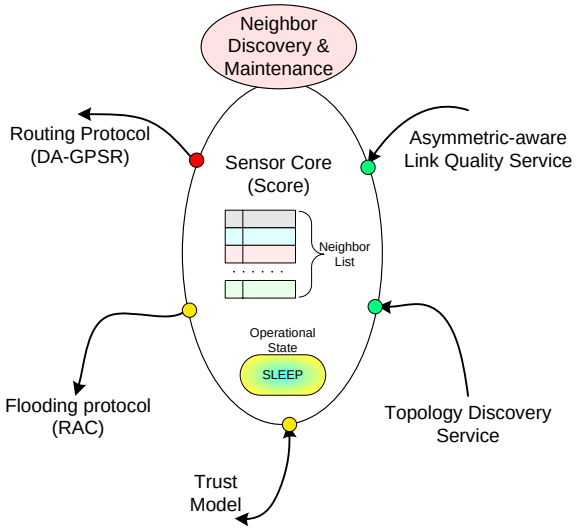


Figure 7.1: *Score* vision as a baby frog. *Score* is depicted as the body, neighbor discovery as the head, terminals as different network components.

consider link quality provided by a link quality service when selecting a path from a source to a destination to improve the end-to-end delivery rate. Likewise, a topology management protocol can take advantage of the node's duty schedule maintained by the application to put the node into a full sleep mode –when idle– and save energy. The former example represents a traditional top-down interface, while the latter represents an unusual bottom-up interface. An effective cross-layer design framework should allow for an arbitrary interface between any two protocol components, yet maintain enough modularity among the different communication stack components. In this section, we present *Score*, a framework to facilitate cross-layer design and maintain network component modularity.

7.1.1 Score *Framework Vision*

We envision *Score* as a framework that facilitates other network components to collaborate in an arbitrary fashion while maintaining a modular communication stack. As a core module, *Score* provides other components with the means to maintain and access the *neighbor set* and the *operational state*, which are the fundamental pieces of information upon which all the network components base

their actions and optimization. Fig. 7.1 depicts this vision as a baby frog. The body represents the *Score* module including the *neighbor list* and the *operational state*. The head represents a neighbor discovery component that maintains the neighbor list using interfaces provided by *Score*. Each one of the baby frog terminals represents a network component in the communication stack that has access to the neighbor list and monitors the operational state. Note that any interface between any of the network components (i.e., terminals) has to go through *Score* (i.e., the body). Arrows differentiate provider from consumer services. For example, the topology discovery service inserts parameters into *Score*, whereas, the dependable routing reads them out to perform dependable routing.

```
interface SCore{
    // Sequential Access Iterator commands
    command result_t first();
    command result_t next();
    event result_t nextDone(uint16_t neighborID);

    // Ramdom Access Iterator command
    command result_t seek(uint16_t n_id);
    event result_t seekDone(result_t success);

    // SCore Reader
    command result_t read(uint8_t *neighbor);
    event result_t readDone(uint8_t *neighbor);

    // SCore Writer
    command result_t write(uint8_t *neighbor);
    event result_t writeDone(result_t result);
}
```

(a)

```
interface State{

   // To change the node's current state
   command result_t change(uint8_t newState)

   // Fired whenever the node's state changed
   event result_t changed(uint8_t newState);

}
```

(b)

Figure 7.2: *Score* APIs, (a) neighbor set abstraction API and (b) state interface.

7.1.2 Score *Framework Features*

Score provides three basic mechanisms and interfaces to facilitate network components collaboration. First, *a unified neighbor set abstraction*. Second, *a modular cross-layer interface*. Third, *a cross-layer coordination* mechanism.

Neighbor set abstraction API

Using the `Score` access interface (Fig. 7.2(a)), a network component can read or write any neighbor record simply by pointing at the required record and performing a read or a write. Moving the pointer can be done in two ways, sequentially using the *first* and *next* commands, or randomly using the *seek* command (Fig. 7.2(a)). Following the nesC/TinyOS philosophy, *Score* provides split-phase operations to keep the sensor node responsive to external events [26]. *Score* does not impose any limitations and is not involved in deciding which nodes are included in the neighbor set. In other words, *Score* only provides the mechanism and not the policy. We present an example neighbor discovery service in Section 7.2.

Cross-layer Interface

Score plays a significant role in decoupling network components by providing a mechanism for them to interface and communicate without the need for pair-wise interfaces. *Score* defines a global neighbor record structure. In this structure, each network component is allocated a number of bytes. The network components can use these bytes to annotate the neighbors with useful information that other components wish to access. For example, a trust network component can rank the neighbors based on some trust criteria, and annotate the neighbors with this value. Another component, the routing protocol for example, can access these trust values through *Score* and exclude untrusted neighbors while building a routing tree.

To keep the `Score` access interface simple and general, *Score* does not provide individual read and write commands to read and write specific fields in the neighbor record, it only supports reading and writing entire records. By doing so, *Score* is not severely involved and dependent on a particular

neighbor record structure, which we think can change in different WSN applications. Reading and writing entire records raises the need for *Score* to prevent network components unintentionally or intentionally (malicious component implementations) from overwriting each other's information in the neighbor record. Therefore, each network component is assigned a *writing mask*, This mask (for short) is statically defined in *Score* according to the current neighbor record structure. Each time a network service writes a neighbor record, *Score* will first apply the mask, on the new record, which sets all the unrelated bits to zeros, and then perform a bit-wise *OR* operation with the old neighbor record. The masking process does not only provide inter-protocol overwrite protection, it also allows for multiple writers at the same time with no need for inter-component synchronization (each component writes its own bytes only in the shared neighbor record).

Cross-layer coordination

Score supports cross-layer coordination by maintaining a sensor node *operational state*. This *state* (e.g., `DISCOVERY`, `BOOTED`, `SLEEP`, and `ACTIVE`) describes the current sensor node operational status. Each network protocol can react in its own way when a new *state* is announced by *Score*. For example, a *neighbor discovery* service will send neighbor probing messages if the node state changes to `DISCOVERY` (a `DISCOVERY` state means there are not enough neighbors in *Score*), while a routing service will hold its protocol messages as there are not enough neighbors to maintain a routing tree, and so save the node's precious energy from being wasted for nothing. *Score* provides a *state* interface (Shown in Fig. 7.2(b)), which provides a command to change the node's current state and uses an event to announce state changes. Any network component wishing to react to state changes must provide an implementation of the *changed* event, in which the component can take the appropriate action.

7.2 Neighbor Discovery Service

The *neighbor discovery* service works in *passive* as well as *active* modes. In the *active* mode, the *neighbor discovery* service actively probes nodes in the vicinity to populate *Score* with new neighbors, while in the *passive* mode, the *neighbor discovery* service passively intercepts incoming

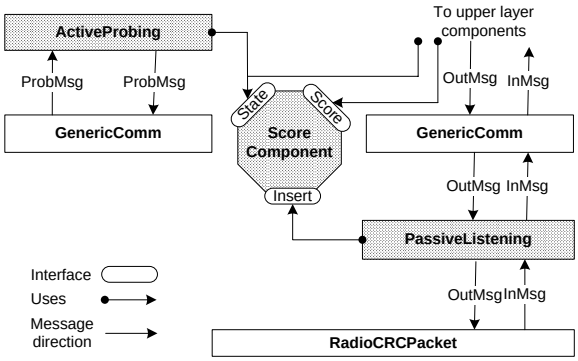


Figure 7.3: *Score* and the *neighbor discovery* service in the TinyOS communication stack. Our components are shown as shaded boxes.

messages, extracts the source address, and inserts a new record into *Score*. Other network services and protocols access the set of neighbors transparently using the *Score access API* with no need for direct interface with the *neighbor discovery* service. Fig. 7.3 depicts *Score*, the *neighbor discovery* service (represented by the *activeProbing* and *passiveListening* modules), and how they fit in the TinyOS communication stack. The *activeProbing* module is responsible for active probing whenever *Score* announces a `DISCOVERY` node operational state, while the *passiveListening* module passively intercepts incoming messages and inserts neighbor records into *Score*. Note that the *passiveListening* module is located below the GenericComm module so that it can benefit from all incoming messages and not only from those targeted for the neighbor discovery service (i.e., *probReplyMsg* messages). Doing so helps in reducing the need for costly active mode neighbor discovery.

7.2.1 Active Probing

To perform active *neighbor discovery*, the *activeProbing* module simply broadcasts probe messages (i.e., ProbMsg(i)), which consist of the source node address (sender). Nodes in the vicinity (receivers) reply by sending messages consisting of the the receiver's address and the original sender address. As we present in Section 8.3.1, including the original sender address in the probe reply messages is important to build the shared neighbor sets in RAT. The *active probing* module registers

with *Score* for node operational state. Whenever the operational state changes to `DISCOVERY`, the *activeProbing* module starts probing for neighbors so that *passiveListening* can insert new neighbor records into *Score*. Once the operational state changes back to `BOOTED`, *activeProbing* holds back its probing messages.

7.2.2 Passive Listening

The *passiveListening* module is placed under the active messaging layer (i.e., GenericComm in Fig. 7.3) so that it can intercept all incoming messaging and not only those targeted to the *neighbor discovery* service, and so lowers the need for costly active *neighbor discovery*. To make this possible, the source node address has to be included as the first two bytes of the payload of all the packets, so that the *passiveListening* module can simply extract the first two bytes of any incoming message and insert a new neighbor record into *Score* without paying attention to the message type.

A ProbReply(i, j) message, on its own, tells any third party receiving node that both i and j are neighbors. As we discuss in Chapter 8, RAT leverages this by overhearing these message to build and maintain the shared neighbor sets, which is important for the RAT protocol.

7.3 Topology Discovery Service

Wireless Sensor Networks (WSNs) have been known as a self-configuring, self-organizing, self-optimizing, and self-healing type of networks [40, 78, 97]. The rule of thumb in these self-* networks is the localization of communication, computation, and finally decision making. Protocol layers running at sensor nodes located in a specific neighborhood collaborate and coordinate their efforts to achieve a general goal without the aid of entities external to the network. As a first step toward this collaboration, each sensor node should acquire the adequate topology information that describes the node's relation to other nodes in its vicinity. Our focus in this section is to present a topology discovery service that works in the context of *Score* and supports cross-layer design. We demonstrate the advantages of this service through two case studies: routing in irregular topologies (DA-GPSR) and controlled flooding (RAC) in Section 7.4 and Section 7.5, respectively.

In this section, we present a topology discovery service that maintains several topology parameters to support cross-layer design. To the best of our knowledge, we are the first to propose such a service that explicitly aims to support cross-layer design. The topology parameters are accessible by other protocol components through *Score*. Sensor field dimensions, the total number of nodes, node degree (i.e., the number of nodes within the sensing/communication range), and node density in a specific sensor field area are typical examples of topology parameters. A routing layer, for example, may choose to route around areas with high node density to avoid high levels of collisions.

7.3.1 The Topology Discovery Service in the Score Framework

On its own, the topology discovery service does not map into any of the OSI reference model layers and it represents a typical network service that solely supports cross-layer design. Other traditional network layers, such as MAC and routing layers, uses the topology parameters maintained by the topology discovery service in order to improve their performance. In order to build and maintain these parameters, the topology discovery service actively sends and receives protocol messages. For example, neighboring nodes exchange their neighbor lists to find *communication redundancy* and *freshness*. At node x with neighbor list (NS_x), *communication redundancy* and *freshness* are defined for each node ($y \in NS_x$) as the cardinality of ($NS_x \cap NS_y$), and ($NS_y \setminus NS_x$) respectively. After calculating the topology parameters, the topology discovery service publishes them into *Score* so that other network components can access them at will. It is vital to note that the introduction of the topology discovery service does not impose severe restructuring of the communication stack. *Score* allows the network components to access the topology parameters without the need for direct interfaces with the topology discovery service. Fig 7.4 shows the skeleton of the topology discovery service in *Score*.

It is important to differentiate topology discovery from neighbor discovery. Neighbor discovery mainly focuses on discovering a node's neighbor list, usually by simple probing messages. Topology discovery, on the other hand, builds on the neighbor list information to provide higher-level

```
module TopologyDiscoveryM{
 uses interface Score;
 ...
}
implementation{
 // pointer to the current neighbor record
 NeighborRecordPtr p;
 int16_t redundancy, freshness;
 ...
 event TOS_MsgPtr Receive.receive(TOS_MsgPtr m){
   // Receive message from y and
   // save neighbor list into NS(y)
   ...
   // Start calculating redundancy and freshness
   Score.first()
 }
 event result_t nextDone(uint16_t neigborID){
   // Got the next neighbor in my neighbor list
   ...
   for all e in NS(y){
     if (e == neighborID)
       redundancy++;
   }
   // move to next neighbor in my neighbor list
   if (!call Score.next()){
     // end of neighbor list, then find freshness
     freshness = |NS(y)| - redundancy;
     //put the pointer at y's record
     call Score.seek(y);
   }
 }
 event result_t seekDone(result_t suc){
   // update values of y's record
   p->redundancy = redundancy;
   p->freshness = freshness;
   call Score.write(p);
   ...
 }
}
```

Figure 7.4: Topology discovery skeleton implementation in *Score*. Upon receiving a neighbor list from node y, current node loops over the neighbor set using *Score* and calculates communication redundancy and freshness. Finally, current node updates relevant bytes in y's neighbor record.

information with semantics. For example, what is a node's *communication redundancy* with each node in the node neighbor list.

7.3.2 Topology Parameters

Topology parameters describe in general the sensor field including its physical dimensions (i.e., size) and the number of nodes deployed in that field. This deployment implies several other topology parameters including connectivity and coverage topologies, and average node density in general as well as node density in different locations in particular. This deployment also dictates how many

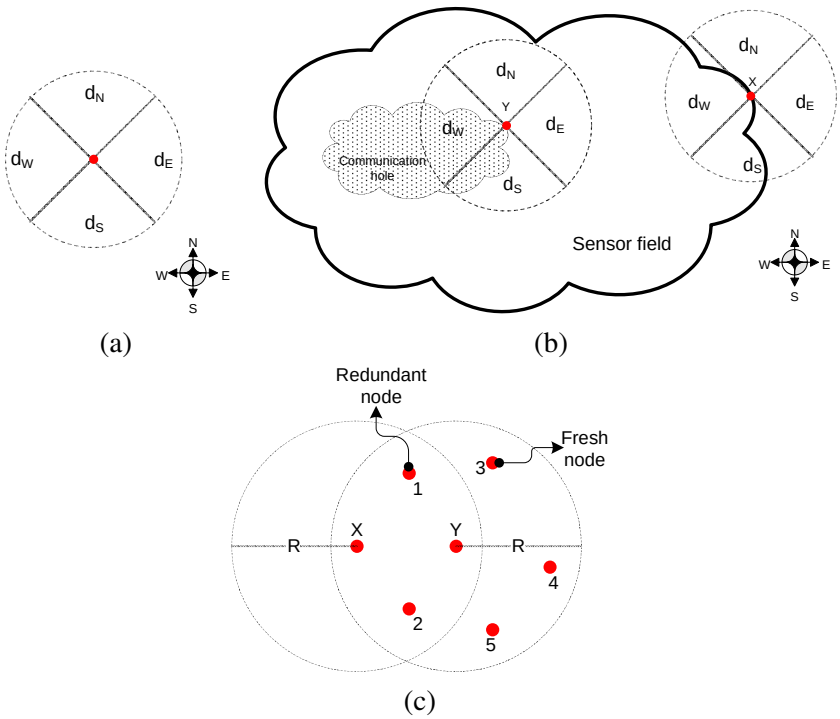


Figure 7.5: Illustrating topology parameters: (a) shows a sensor node at the center of its nominal communication area, which is divided into four directional areas. (b) depicts a sensor field with a communication hole (i.e., shaded area), node X located on the boundary exhibits zero d_N and d_E. Node Y also exhibits very low d_W which suggests a communication hole in that direction. (c) depicts several nodes as red dots with their respective communication range R. X has communication redundancy and freshness values of 2 and 3 with node Y, respectively. Nodes 1 and 2 are redundant, while nodes 3 , 4, and 5 are fresh.

neighbors each node has in its neighbor list (i.e., node degree) on average and identifies nodes with above and below average node degree. The link quality value for a node with each one of its neighbors can also be considered as a topology parameter. Collectively, the topology parameters describe a node's topological (connectivity and coverage) relationship with its immediate neighbors as well as its topological position in the sensor field (i.e., on the boundary of the connectivity graph or on a boundary of a communication hole or high density area). The following is a list of these parameters:

- *Physical dimension* of the sensor field: describes the shape and dimensions of the sensor field. This information along with an estimation of the nominal communication range, can be used, for example, by a routing protocol to get an estimate of the longest (hop count) possible path between any two nodes in the sensor field. Such an estimate could be useful for a routing layer to recover from routing loops (i.e., when the path exceeds the maximum hop count).

- *Total number of nodes*: specifies the total number of nodes the network has in total. It can be used, along with sensor field dimensions, to calculate deployment node density. A topology management protocol can use this as an indication of how aggressive the protocol should be when putting nodes to sleep.

- *Node density*: could be either communication or sensing density, which is simply the number of neighbors in a node's communication/sensing range divided by communication/sensing area. A node density can be used by a topology management protocol to maintain a specific number of active nodes at any time.

- *Directional density*: represents a node density in a particular direction (i.e., NORTH, SOUTH, EAST, WEST). Figure 7.5(a) explains the directional density. A large difference in a node's directional densities may suggest that the node is located on the topological boundary of the network, some communication/sensing hole, or a high density area.

- *Boundary*: a flag that indicates whether a node is located on the boundary of a sensor field area with irregular topological properties. An area is considered topologically irregular if the area has above average node density (high node density), or below average node density (communication hole). The boundary flag may also indicate that the node is located on the topological boundary of the network. Boundary nodes can collaborate to seize irregular areas. A routing protocol becomes aware of such areas and avoids routing through them, for example, routing around communication holes. Figure 7.5(b) depicts a sensor field with a communication hole

area (i.e., shaded area). We can notice that a node on the boundary of the communication hole and a node on the sensor field boundary exhibit a variation in their directional densities.

- *Communication/sensing redundancy*: a node's redundancy with each one of its neighbors describes the number of nodes that both of them share in their neighbor lists.

- *Communication/sensing freshness*: a node's freshness with each neighbor is the number of nodes that exist in the neighbor's neighbor list and not in the node's neighbor list. Fig. 7.5(c) illustrates communication redundancy and freshness.

  Both *communication redundancy* and *freshness* can be used, for example, in a controlled flooding protocol (Section 7.4), where the total number of transmissions required to disseminate a message is minimized by selecting particular wireless links when forwarding a message.

7.4 Redundancy-Aware Controlled Flooding

In this section, we demonstrate the utility of the topology discovery service using a data dissemination and flooding protocol. Data dissemination is one of the two popular communication patterns in WSNs in addition to data collection. A data message (usually a query) has to be sent from a single node (i.e., the sink) to reach every other node in the network. A basic approach to do flooding is to make the sink start the process by broadcasting the message. Every receiving node, in its turn, re-broadcasts the message until all the nodes get the message. It is vital that each node forwards the same message only once, otherwise, loops will happen. We refer to this approach as the blind flooding protocol (denoted as **Blind**). In this case study, we exploit topology redundancy information provided by the topology discovery service in order to reduce the total number of transmissions required to disseminate the data message. We refer to our approach as **R**edundancy-**A**ware **C**ontrolled flooding (denoted as **RAC**). In **RAC**, among the receiving nodes, only the node that has the least communication redundancy with the sender re-broadcasts the message. This increases the chance that the data message will reach more nodes that have never seen the message before,

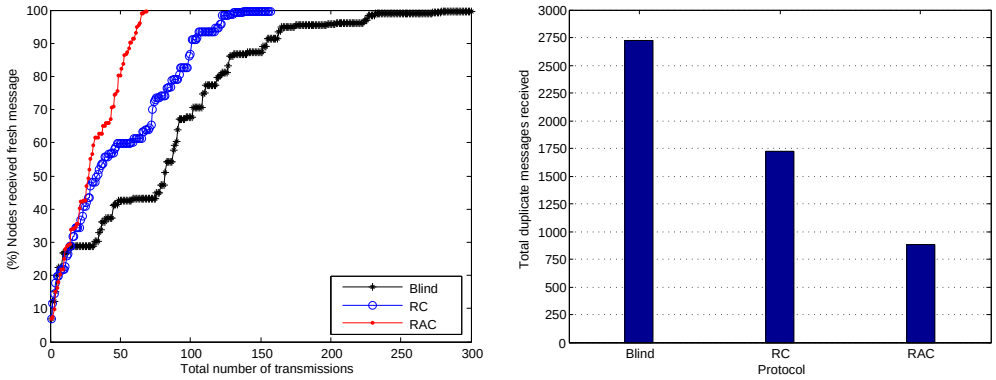
and so, reduces the required number of transmissions overall. We also compare **RAC** to **R**andom **C**ontrolled flooding protocol (denoted as **RC**), in which the forwarder node is chosen randomly. We refer to **RAC** and **RC** together as controlled flooding protocols.

7.4.1 Controlled Flooding Protocols

Based on who (i.e., sender versus receivers) decide(s) the node that should re-broadcast a message, we can differentiate two approaches to controlled flooding (i.e., sender-based and receiver-based). In sender-based flooding, the sender node chooses the node that should pick up the flooding process, while in the receiver-based flooding, receiver nodes decide among themselves a single node to pick up the flooding.

The sender-based controlled flooding approach maintains a single thread of flooding in the network at any time, which potentially keeps the total number of transmissions as low as possible. However, if the selected node to re-broadcast has already seen a copy of the message, it will not be interested to re-broadcast, and therefore, the only thread of flooding will vanish, leaving some nodes in the network unaware of the message. Therefore, the sender-based approach does not provide enough guarantee that all nodes get the message.

After receiving a message in the receiver-based controlled flooding approach, each one of the receiver nodes backs off for a specific amount of time before re-broadcasting. During the back off period, a node suppresses its transmission once the node gets another copy of the message from another node in the vicinity. It is clear that several flooding threads may exist in the network at the same time; however, this approach makes sure that at least one node eventually picks up the flooding process and so, all nodes in the network get at least one copy of the message. The remaining question is how the receiver nodes calculate their back off time. This is where we differentiate **RAC** from **RC**. In **RAC**, each receiver node sets the back off time proportionally to its communication redundancy with the sender node, so that receiver nodes with the least communication redundancy with the sender re-broadcast first. As a result, more new nodes get the message. In **RC**, back off times are chosen randomly.



(a) (b)

Figure 7.6: Comparing the performance of Blind, RAC, and RC protocols.

7.4.2 Simulation Setup and Evaluation

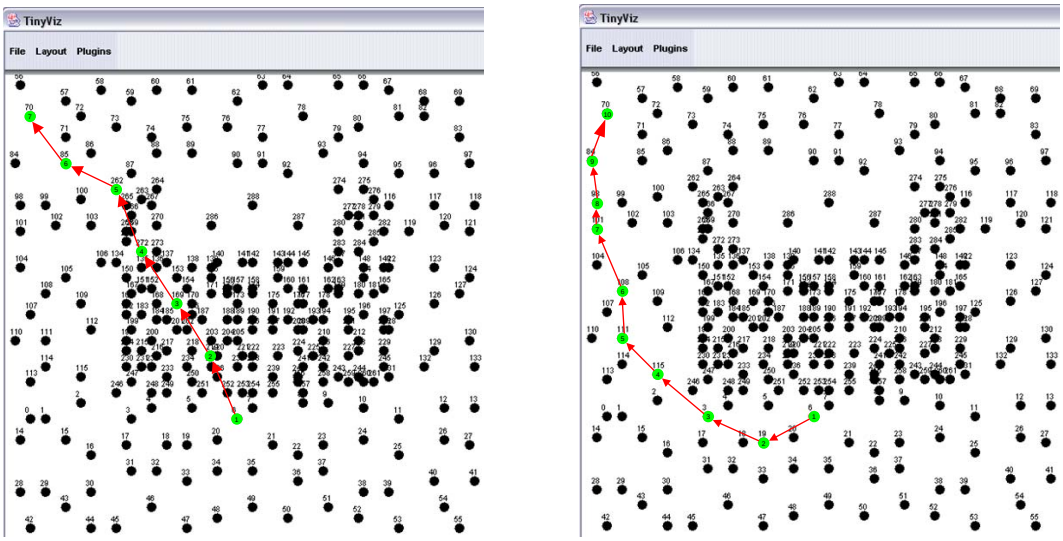
We implement **Blind**, **RC**, and **RAC** on top of the topology discovery service in the nesC/TinyOS platform. We compare the performance of the protocols in terms of two performance metrics. First, the number of nodes that receive a fresh message copy for each message transmission per flooded message. Second, the total number of duplicates received by all nodes per flooded data message. Using each flooding protocol, we disseminate 100 messages from the sink in a wireless network of 300 nodes distributed randomly over a (100x100) units squared sensor field. The nodes' nominal communication range is 15 units. We take the average over the 100 message floodings and present the data in Fig. 7.6.

Fig. 7.6(a) shows the cumulative number of fresh messages received, shown as the y-axis, for each message transmission in the network, shown as the x-axis. We observe two important results. First, **RAC** outperforms **RC** and **Blind** as for each transmission, there are always some nodes that receive the message for the first time (i.e., the line consistently increasing), while, in **RC** and **Blind**, many transmissions are useless, shown when the line moves horizontally. Second, **RAC** requires around 70 transmissions in total to disseminate a message compared to over 150 and 300 for **RC** and **Blind**, respectively. Fig. 7.6(b) shows the total number of duplicates received as the y-axis for

each one of the protocols as shown on the x-axis. We observe that **RAC** reduces the total number of duplicates to almost half those of **RC** and around one third those of **Blind**.

7.5 Routing In Irregular Topologies

In this section, we use the topology discovery service to develop a modified version of the GPSR routing protocol to improve end-to-end (E2E) performance in irregular topologies. By irregular topologies we mean topologies where wireless links are not uniformly distributed in the sensor field (i.e., some sensor field areas have high connectivity and so high interference levels, and some areas with low connectivity and so lower interference levels). We argue that irregular topologies are more probable to happen in real life scenarios than regular topologies as there are many reasons for irregular topologies to happen such as non-uniform node deployment, environmental conditions that make the nominal communication range variable at different locations in the sensor field, and node failures.



(a) GPSR Route (b) DA-GPSR Route

Figure 7.7: TOSSIM snapshot of routes chosen by GPSR compared to routes chosen by DA-GPSR. In both (a) and (b), data messages goes from node 6 to node 70.

While routing from a source to a destination, DA-GPSR avoids areas with high node density in contrast to GPSR, which strictly routes directly toward the destination. The rationale behind avoiding areas with high node density is that these areas are more susceptible to collisions and have higher levels of interference, which results in degradation in one-hop success rate and ultimately degradation in E2E performance. As a result of avoiding areas with high node density, routes chosen by DA-GPSR tend to have a higher hop count than routes chosen by GPSR. Nonetheless, in irregular topologies, using shortest hop count routes obviously will not lead to the best E2E delivery rate possible as these routes may have to pass through areas with high interference levels. However, it may be not intuitive whether routing around those areas will result in better E2E delay.

The basic idea of DA-GPSR is to augment the link quality of the next hop sensor node on the route to the destination in addition to the distance to the destination used by GPSR. Two factors affect the link quality of a wireless link, SNR and interference level. The former is controlled by environmental factors and we don't have much power to improve it, while the latter is mainly controlled by the number of potential transmitters on a given wireless link. As presented in [31], interference has the larger effect on link quality and total network capacity. We employ the interference definition of a wireless link introduced in [11], and further, define route interference level as the maximum and average interference level over wireless links on the route.

7.5.1 Network Model

We assume a 2-dimensional sensor field with N sensor nodes distributed over the sensor field. All sensor nodes have the same transmission range r. If a node transmits a packet, all the nodes within a distance r of the transmitting node, which form the node's neighbor set, will receive the message. Since in this case study, we are interested in interference only, we assume a perfect pairwise wireless channel (i.e., no packet losses due to signal fading). Finally, we assume that the relative physical location of the nodes is available to them by means of some localization protocol.

7.5.2 MAC Protocol

In this subsection we describe the underlying MAC protocol used with GPSR and DA-GPSR. To avoid excessively long running time of the TOSSIM bit-level simulation, we use packet-level simulation with simulated collisions. Our MAC protocol employs a TDMA channel access with packet acknowledgement and retransmissions. As discussed in [13], in a network with S transmission slots and d potential forwarding nodes, the probability that node i forwards a message successfully is defined as follows:

$$Pr(suc.) = \left(\frac{S-1}{S}\right)^{d} \tag{7.1}$$

The MAC protocol employs a simple acknowledgement mechanism to control packet retransmissions. Upon receipt of a data message, the receiver node sends a short acknowledgement message back to the sender. If the sender node does not receive the acknowledgement within a round trip time, it assumes that the data message has been lost, and so retransmits the data message one more time. The sender node continues to retransmit a data message until the message is successfully received or it reaches the maximum number retransmissions (**MAX_RET**), and so, give up. To avoid the congestion control problem and keep the MAC protocol simple, each data message is allowed enough time to propagate from the source node to the destination node in our simulation. The single-hop success rate with retransmissions, therefore, is defined as follows:

$$Pr(suc.\ with\ ret.) = 1 - \Big((1 - Pr(suc.))^{MAX\_RET}\Big) \tag{7.2}$$

From the above equations, we can see that S, d, and **MAX_RET** are the parameters that control the single-hop success rate, and therefore, E2E performance. S and **MAX_RET** are MAC protocol configuration parameters, while d is determined by the route that the routing protocol (i.e., GPSR and DA-GPSR) chooses from the source node to the destination.

7.5.3 DA-GPSR

Let S_i be the neighbor set of node i, $|x, j|$ the Euclidean distance between node x and node j, and $|S_x|$ be the node degree of node x. At node i, we introduce two ranking values for each node $x \in S_i$ given a final destination node j. First, $dRank_{x,j}$ is the distance rank of neighbor x. Second, $degRank_x$ is the degree rank of neighbor x. $dRank_{x,j}$ ranks a node (i.e., x) based on its physical proximity to the destination (i.e., j), while $degRank_x$ ranks a node (i.e., x) based on its link quality with the current node (i.e., i). Interference of wireless link (i, x) equals to the summation of $|S_i|$ and $|S_x|$. Thus, for node i, the link quality with any of its neighbors boils down to the node degree of that neighbor (i.e., node degree of node i is the same for all neighbors). To bring distance rank and the degree rank into the same domain values, we normalize both values as follows:

$$degRank_x = \frac{\max_{m \in S_i}(|S_m|) - |S_x|}{\max_{m \in S_i}(|S_m|)} \tag{7.3}$$

$$dRank_{x,j} = \frac{\max_{m \in S_i}(|m, j|) - |x, j|}{\max_{m \in S_i}(|m, j|)} \tag{7.4}$$

To keep DA-GPSR a localized algorithm, we use local maximums over the neighbor set of the node to normalize $dRank_{x,j}$ and $degRank_x$.

$$Rank_x = \alpha \cdot dRank_{x,j} + (1 - \alpha) \cdot degRank_x \tag{7.5}$$

Under DA-GPSR forwarding rules, node i ranks the nodes in its neighbor set (i.e., S_i) according to formula 7.5 when forwarding to destination node j. Then, node i forwards the data message to the neighbor with the highest $Rank$ value. We use an α value of 0.7 in our simulation. Using an α value of 1.0 brings DA-GPSR to the original the GPSR algorithm.

Using the interference level of a wireless link introduced in [11], we further define *Avg_route_interference_level* and *Max_route_interference_level* to describe route quality. Let $route_{a,b}$

pass through nodes $\{x1, x2, ..., xn\}$. Then,

$$Avg\_route\_interference\_level = \frac{\sum_{i=1}^{n} |S_{xi}|}{n-1} \tag{7.6}$$

$$Max\_route\_interference\_level = \max_{i=1}^{n} |S_{xi}| \tag{7.7}$$

Figure 7.7 compares routes chosen by GPSR to routes chosen by DA-GPSR, GPSR tends to choose routes with less hop count, however, DA-GPSR chooses routes with lower *Avg_route_interference_level* and *Max_route_interference_level*. In Figure 7.7(a) and Figure 7.7(b), the GPSR route has a hop count value of 6, an $Avg\_route\_interference\_level$ value of 29.8, and a $Max\_route\_interference\_level$ value of 46, while the DA-GPSR route has a hop count value of 9, an $Avg\_route\_interference\_level$ value of 10.9, and a $Max\_route\_interference\_level$ value of 19. As we show in Subsection 7.5.4, the impact of the interference level of a route outweighs the impact of hop count on the ultimate E2E performance.

7.5.4 Evaluation

Performance metrics

We use the following four E2E performance metrics to compare GPSR to DA-GPSR:

- E2E delivery rate: the ratio of the number of sent messages at the data source to the number of data messages received successfully at the destination node.

- modified Hop Count (mHC): is the number of hops required to route a data message from the source to the destination. Each retransmission is counted as an extra hop.

  Retransmissions may result from a lost data or ACK message. In a single hop transmission, if the data message itself is lost, then the retransmission will count toward the mHC, but if the data message is successfully received and the ACK is lost, the retransmissions will not count

toward the mHC, as the data message did go through. In order to capture lost ACK messages, we define Transmissions Per Data message (TPD).

- Transmissions Per Data message (TPD): the number of transmissions –including data and ACK messages– performed by the nodes on the route between the source and destination. TPD accounts for both retransmissions resulting from lost data and ACK messages.

- E2E delay: the time in milliseconds required to deliver a data message from source node to destination node on the route.

Simulation results

We have two sets of experiments. The first set of experiments studies the effect of the MAC level parameters (i.e., **S** and **MAX_RET**) configuration on the E2E performance of GPSR and DA-GPSR. This set of experiments gives us an insight into how DA-GPSR performs in contrast to GPSR given the same underlying MAC protocol settings. From the application point of view, the specific MAC level settings are not the primary concern. Instead, E2E delivery rate is what actually matters. Therefore, in the second set of experiments, we compare the performance of GPSR and DA-GPSR in terms of E2E delay, mHC, and TPD given the same achievable E2E delivery rate.

In Fig. 7.8, we fix **S** to 24 and vary the **MAX_RET** value from 0 to 15. Then, for the same source/destination nodes, we send 100 data messages using GPSR and DA-GPSR and calculate E2E delivery rate, mHC, E2E delay, and TPD in Fig. 7.8(a), Fig. 7.8(b), Fig. 7.8(c), and Fig. 7.8(d) respectively. From Fig. 7.8(a), we notice that the E2E delivery rate is zero when **MAX_RET** equals zero. This means that not a single data message arrived successfully at the destination. Hence, mHC, E2E delay, and TPD for a **MAX_RET** value of zero are not defined. Therefore, **MAX_RET** value of zero is removed from the domain in Fig. 7.8(b), Fig. 7.8(c), and Fig. 7.8(d).

We can observe from Fig. 7.8(a) that E2E delivery rate improves very fast under DA-GPSR compared to GPSR. Fig. 7.8(a) shows that for a **MAX_RET** value of 4, DA-GPSR already achieved over 90% E2E delivery rate, while GPSR is lagging behind with an E2E delivery rate value of only

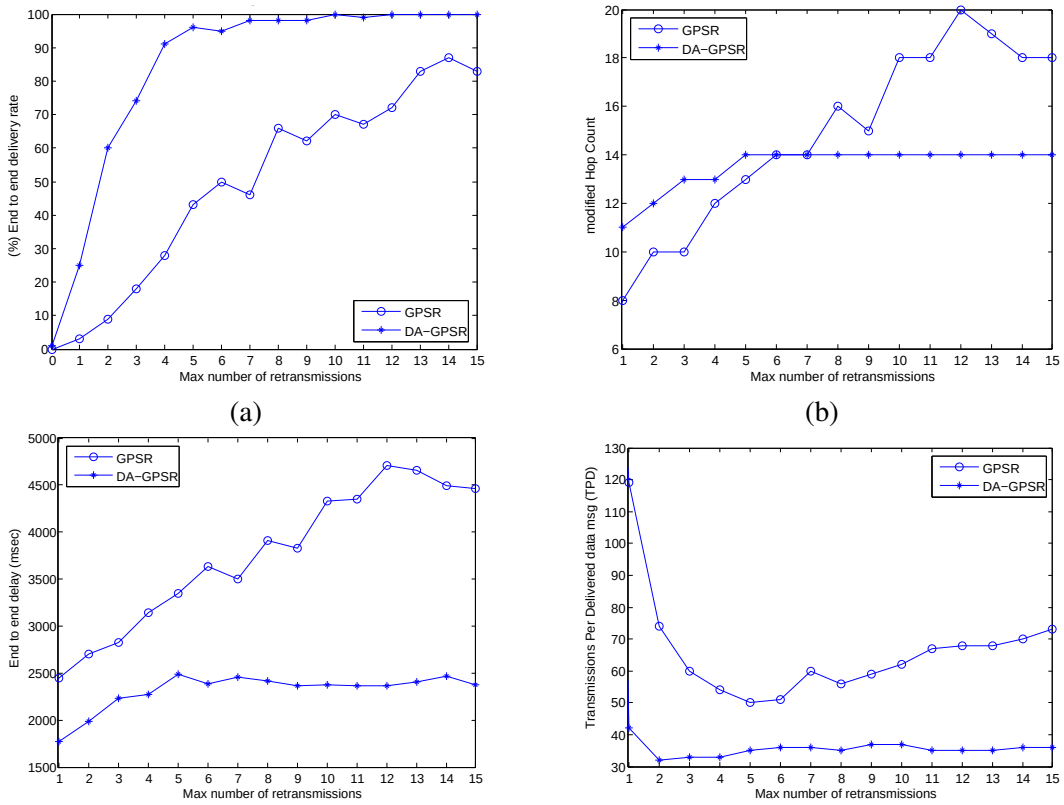


Figure 7.8: Comparing the effect of MAX_RET on the performance of GPSR and DA-GPSR. S is fixed to 24.

40%. Also, DA-GPSR shows more reliable E2E delivery rate performance compared to GPSR, in which, E2E delivery rate keeps fluctuating even for high **MAX_RET** values.

In Fig. 7.8(b), the y-axis represents the mHC. Therefore, the lower the y-axis value the better. We observe that GPSR starts with lower mHC values when **MAX_RET** is small. This is due to the fact that routes under GPSR tend to have a lower hop count. However, as the figure shows, DA-GPSR catches up very fast with GPSR to provide a stable and reliable mHC, which is less than that of GPSR, for **MAX_RET** values over 6. Besides, for those **MAX_RET** values where GPSR outperforms DA-GPSR, the E2E delivery rate is very low as we can infer from Fig. 7.8(a). This point is made more straightforward to observe in the second set of experiments.

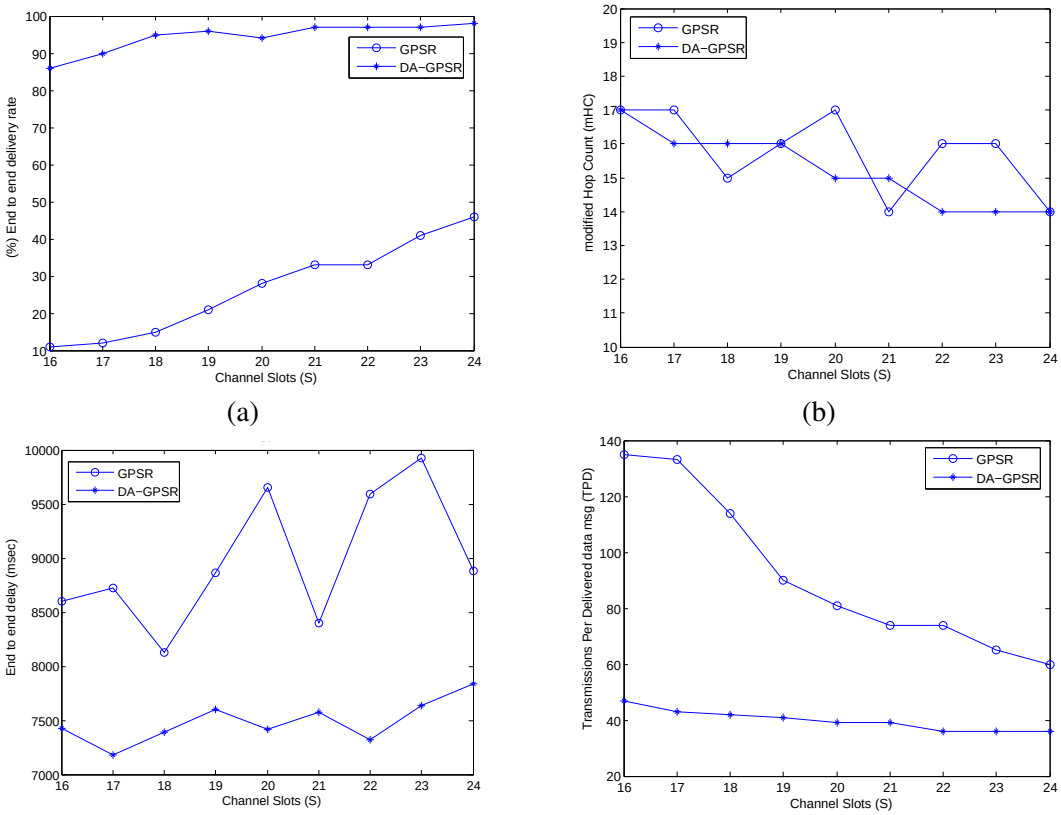


Figure 7.9: Comparing the effect of number of transmission slots (S) on the performance of GPSR and DA-GPSR. MAX_RET is fixed to 7.

Fig. 7.8(c) studies the effect of **MAX_RET** on E2E delay in milliseconds. Intuitively, E2E is driven by two factors, mHC and **S**. mHC accounts for the actual number of hops including retransmissions a data message travels from the source to the destination, and **S** accounts for the time delay at each hop (back off time a node needs to wait before forwarding a message). Since **S** is the same for both GPSR and DA-GPSR in this figure, one expects E2E delay to follow mHC in Fig. 7.8(b) exactly. However, the figure suggests this is not the case. When we examined the trace files, we observed that processing time (i.e., single hop processing delay) spent by GPSR is higher than that spent by DA-GPSR. The reason is that in GPSR, nodes on the route have to process a higher number of nodes in their neighbor sets when forwarding a data message (i.e., nodes in areas with high node

density have larger neighbor sets). In addition, we observe that under DA-GPSR, E2E delay tends to stay constant once the **MAX_RET** exceeds a specific value (5 in the figure), while, E2E delay under GPSR tends to keep increasing.

In Fig. 7.8(d), we study the relationship between **MAX_RET** and TPD. We observe that TPD starts at very large values, then it drops to reach a minimum value, and then its value increases a bit and stabilizes. This is due to the fact that the relationship between TPD and **MAX_RET** has an explicit and implicit side. **MAX_RET** explicitly controls TPD as it defines an upper bound on the allowed number of possible transmissions for each data message (i.e., the higher the **MAX_RET** is, the higher the TPD). On the other hand, TPD is affected implicitly by **MAX_RET**, since the higher the **MAX_RET** is, the better the single-hop and E2E delivery, and so the less the need for retransmissions (i.e., the higher the **MAX_RET** the lower the TPD). Fig. 7.8(d) shows that DA-GPSR outperforms GPSR as it needs fewer transmissions to deliver a data message from the source to the destination. After TPD stabilizes, DA-GPSR takes less than 30 transmissions, including acknowledgements, to deliver a single message to the destination, while GPSR takes more than 60 transmissions to deliver the same data message.

In Fig. 7.9, we fix the **MAX_RET** value to 7 and change **S** from 16 to 24 and run the same experiments as in Fig. 7.8 in order to study the relationship between **S** and E2E delivery rate, mHC, E2E delay, and TPD in Fig. 7.9(a), Fig. 7.9(b), Fig. 7.9(c), and Fig. 7.9(d), respectively. As evident in the figures, the effect of **S** on the performance of GPSR and DA-GPSR is not as big and consistent as that of **MAX_RET**. This is due to the fact that the effect of **S** on the algorithms performance, except E2E delay, stems only from **S's** effect on single hop success rate. Furthermore, from formula 7.1, we notice that increasing **S** by one causes a very small increase in single-hop success rate.

In Fig. 7.9(a), we observe that the E2E delivery rate improves slightly as S increases. Also, it is very clear that DA-GPSR achieves much better E2E delivery rate for the same value of **S**. Fig. 7.9(b) demonstrates how mHC decreases as **S** increases, which is simply because the number of retransmissions required to deliver a data message is smaller at each hop. GPSR and DA-GPSR exhibit a comparable mHC.

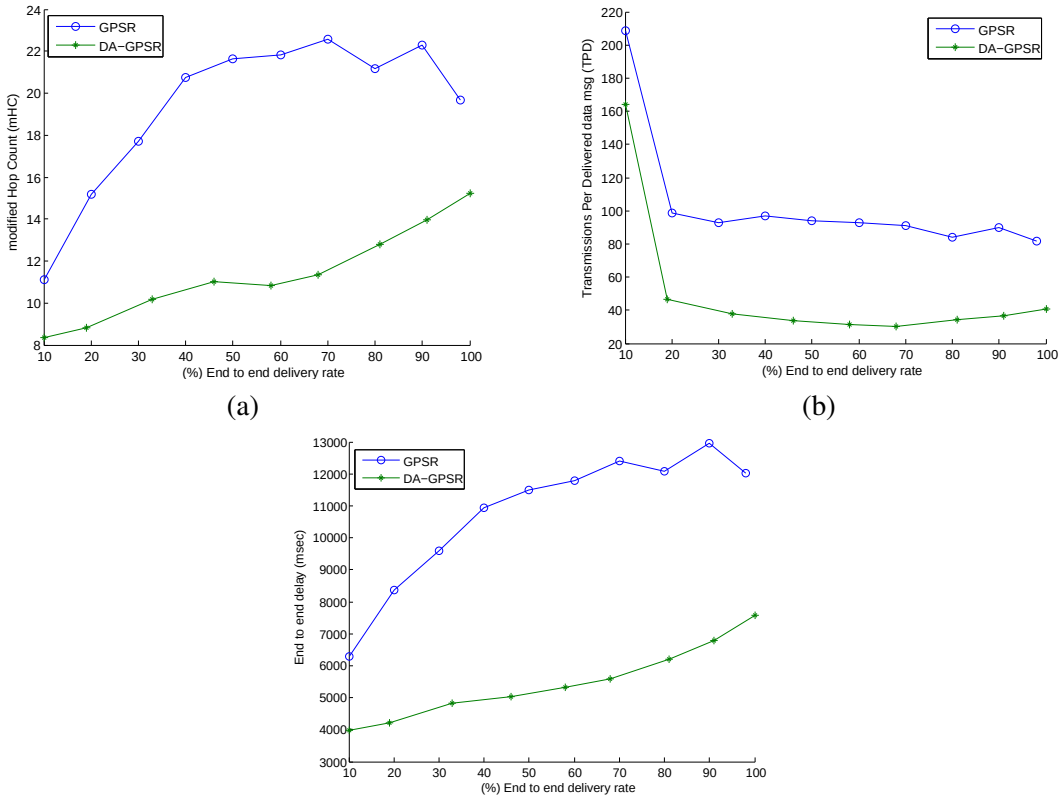


Figure 7.10: Combining MAX_RET and S into E2E delivery rate and comparing GPSR and DA-GPSR.

Like Fig. 7.8(d), the relationship of **S** and E2E delay is double-sided. Increasing **S** improves single-hop success rate, which decreases the need for retransmissions, and so decreases single-hop delay and hence, E2E delay. On the other hand, increasing **S**, increases single-hop delay as nodes need to back off for a longer period of time on average before forwarding a message. However, since the effect of **S** on single-hop success rate is not that big, the second factor dominates. Therefore, increasing **S** has a net effect of increasing E2E delay. Finally, in Fig 7.9(d), we notice that increasing **S** improves single-hop success rate, and so, lowers the need for retransmissions and ultimately TPD. DA-GPSR takes advantage of the improved single-hop success rate and achieves much lower TPD.

In Fig. 7.10, we abstract **S** and **MAX_RET** into E2E delivery rate and compare the performance of GPSR and DA-GPSR in terms of mHC, E2E delay, and TPD. As we mentioned, it is important

to contrast the performance of GPSR to that of DA-GPSR given the same achievable E2E delivery rate. As we have no control to vary E2E delivery rate and observe the other performance metrics, we have to make several runs, in which we vary **S** from 16 to 24 and vary **MAX_RET** from 0 to 16 in order to get the full E2E delivery rate domain values (i.e., from 0% to 100%). For each run, we record mHC, E2E delay, and TPD. Then, we sort and group the runs based on E2E delivery rate. In each group, we calculate the average value of mHC, E2E delay, and TPD, and finally present the data in Fig. 7.10(a), Fig. 7.10(b), and Fig. 7.10(c).

All the figures show that DA-GPSR outperforms GPSR. In Fig. 7.10(a), we observe that in order to achieve an acceptable E2E delivery rate value of 90% or above, GPSR has a mHC value above 20 compared to an mHC value of around 14 for DA-GPSR. This result states that choosing routes with higher hop count in order to avoid high density areas –as done by DA-GPSR– actually pays off when we consider the number of hops a data message actually travels (i.e., mHC). Likewise, we observe in Fig. 7.10(b) that TPD is decreased by almost 50% in DA-GPSR compared to GPSR. Finally for an E2E delivery rate value of 90% or above, Fig. 7.10(c) shows that GPSR needs around 12 seconds to deliver a single data message from source to destination, while DA-GPSR can deliver the same message in almost half the time.

7.6 Related Work

Our related work is four-fold, work that is related to *Score*, topology discovery service, **RAC**, and DA-GPSR.

Unlike previous neighborhood abstraction [87, 88], in which the goal was to support a unified neighbor view for application developers, *Score* is designed to support the system developers by providing neighbor set abstraction and mechanisms for cross layer interface and coordination.

To the best of our knowledge, we are the first to define a topology discovery service in this capacity. Reference [20] used the term topology discovery in a very limited scope. The topology discovery algorithm (TopDisc) aims at finding a minimum dominating set. Nodes in the network are organized into a tree of clusters (TreC), in which nodes of the dominating set act as cluster heads. TreC is rooted at the sink and used for efficient data dissemination and aggregation. The TopDisc

algorithm is more related to topology management protocols. Our work is different from [20] in many ways. For example, it separates topology discovery from topology management, in other words, our topology discovery only provides the means (topology parameters) for a topology management protocol to select a dominating set, which is important to maintain modularity in the WSN communication stack. Also, our topology discovery service is not confined to topology management, it can also be used by other protocol layers, such as a routing protocol.

Unlike Gossip-based dissemination protocols [50], which trade off propagation time to reduce the number of transmissions needed to disseminate a data message, our RAC reduces the total number of transmissions through controlling and limiting the wireless links that should be used when flooding a message. Therefore, RAC complements their work. DA-GPSR enhances GPSR [41] E2E performance by avoiding interference. In a sense, DA-GPSR competes with MAC protocols [34, 89, 94] whose primary goal is to avoid collisions. However, DA-GPSR avoids collisions by avoiding links with high collision probability; therefore, DA-GPSR is orthogonal to MAC protocols and they can be combined to achieve even lower interference levels.

7.7 Summary

We discussed the *Score* framework including two basic network services: *neighbor* and *topology* discovery services. Together, these services provide adequate system-level support for higher level protocols such as topology management and routing. As a demonstration of the utility of our framework, we developed *RAC* and *DA-GPSR*. *RAC* reduces the total number of transmissions required to disseminate a message by up to 77% compared to *Blind*, and *DA-GPSR* reduces E2E delay, mHC, and TPD by half. In the next chapter, we present our RAT protocol, which addresses connectivity problem in environmental application.

CHAPTER 8

REDUNDANCY-AWARE TOPOLOGY MANAGEMENT IN ENVIRONMENTAL MONITORING

Extending the lifetime of wireless sensor networks remains the most challenging and demanding requirement that impedes large-scale deployments. Studies show that considerable energy savings can be achieved only by putting a node's radio into full sleep mode. In this chapter, we present RAT, which is a redundancy-aware topology management protocol. RAT selects a minimum set of active nodes that are good enough to maintain connectivity, and allows the others to sleep and save energy. RAT is designed and implemented with underlying wireless channel irregularity in mind. Scalability and low overhead are the other primary design goals of RAT as well. We implement RAT in the context of *Score*, which is a cross-layer framework that allows RAT to coordinate its `SLEEP` and `ACTIVE` state changes with the routing layer smoothly. Using TinyOS and PowerTOSSIM, we implement RAT on top of *Score*. Comparing with the all-active scenario, RAT simulation results show a total energy consumption decrease of 67% in a one-to-many routing scenario and up to 87% in a many-to-one routing scenario.

8.1 Introduction

Saving energy and extending the lifetime of unattended wireless sensor networks is still one of the most challenging design requirements of wireless sensor network applications and protocols. To save energy and so extend the lifetime of the wireless sensor network, researchers have considered power consumption at different levels, including the application level [33], routing layer [33, 75], and MAC layer [38, 94]. As previous studies showed [15], putting the node's radio into full sleep mode is the most efficient technique in saving the node's energy. Therefore, topology management protocols have the most potential in extending the network lifetime. In this chapter, we use the term topology management to describe the distributed in-network process of selecting a set of active

nodes, which together form a connected dominating set. All other redundant nodes can go to sleep and save energy.

RAT (**R**edundancy-**A**ware **T**opology Management) is a novel topology management protocol to identify node communication redundancy (the *initialization phase*), and schedule nodes for sleep and active modes (the *scheduling phase*). RAT exhibits high fidelity to dynamic and irregular underlying wireless channels, while maintaining low protocol overhead. Furthermore, RAT scales very well with high node densities as only active nodes are allowed to produce overhead traffic. To achieve high fidelity, RAT does not relate physical location to communication redundancy (e.g., fixed communication range). Instead, it uses the neighbor set as the basis to define sensor node communication redundancy and to define a node's responsibility in the multi-hop network. To maintain low overhead, RAT leverages the neighbor discovery process and the broadcast nature of wireless links to establish communication redundancy knowledge among sensor nodes instead of asking nodes to simply exchange their neighbor sets.

RAT has two variants: **B**asic RAT (B-RAT) and **E**nhanced RAT (E-RAT). In B-RAT, the scheduling phase starts at the sink by triggering a scheduling thread, which after that propagates serially until all the nodes in the network switch to sleep or active modes. We trade some uniformity of active node distribution in the sensor field in E-RAT to overcome the potential scheduling thread deadlocks and to improve B-RAT propagation time. Instead of a single thread, E-RAT triggers several scheduling threads which propagate simultaneously in the network, dispatching nodes sleep and active modes.

We implement RAT in the context of *Score*. *Score* provides mechanisms to fill the gap between topology management and the other network services and protocols. This gap refers to the lack of mechanisms to communicate the sleep mode state to other protocol layers [13]. These mechanisms are important for the routing layer, e.g., to pro-actively build alternative routes. Using nesC [26] and PowerTOSSIM [77], we designed, implemented, and evaluated RAT using two sets of performance metrics, *low level* and *high level*. We use the former to demonstrate attractive characteristics of the active nodes selected by RAT, such as the number of active nodes and how well these nodes

are distributed in the sensor field. We use the latter to exhibit the advantage of *RAT* over the all-active scenario in energy savings using the two most popular communication patterns in wireless sensor networks; *one-to-many* and *many-to-one*. RAT shows an energy savings of 67% in a *one-to-many* routing scenario and up to 87% in a *many-to-one* routing scenario for networks of high node densities.

The rest of the chapter is organized as follows, Section 8.2 presents the RAT protocol and algorithm; Section 8.3 discusses implementation details; In Section 8.4, we present our simulation setup and results. Related work and chapter summary are presented in Sections 8.5 and Section 8.6, respectively.

8.2 RAT Algorithm Design

RAT consists of two phases, the *initialization phase* and the *node scheduling phase*. In the initialization phase, each node becomes aware of its role (responsibility) in the multi-hop network, while in the scheduling phase, information from the initialization phase is used to put as many nodes as possible to sleep, while maintaining connectivity. Before delving into the protocol details, we explain key concepts, which are important for the reader to follow the discussion.

8.2.1 Basic Definitions and Notations

The neighbor set (NS_i) plays the central role in RAT and is used to define a node's responsibility in the multi-hop network and communication redundancy metric. Two nodes are considered to have a *high communication redundancy* if they share a high percentage of nodes in their neighbor sets. A formal definition of *Degree of Communication Redundancy* of two nodes denoted as ($DoCR_{i,j}$) is presented next

$\underline{DoCR_{i,j}}$ describes quantitatively how much communication redundancy node j can provide to node i. Therefore, how much responsibility can node j take away from node i in the multi-hop network.

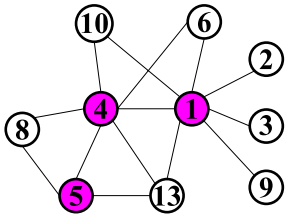
$$DoCR_{i,j} = \frac{|NS_i \cap NS_j|}{|NS_i| - 1}$$

Note that, $DoCR_{i,j}$ is not equal to $DoCR_{j,i}$ (i.e. asymmetric redundancy).

Based on the notion of the neighbor set, a node responsibility is defined as the number of nodes in its neighbor set. The more neighbors in a node's neighbor set, the more responsibility the node takes on, and so the more neighboring nodes are needed to provide the required communication redundancy if the node wants to switch to sleep. This intuition is formalized and quantified as the *neighbor set cover degree* of that node denoted as $CD_i(\alpha)$.

$\underline{CD_i(\alpha)}$ is defined as the minimum number of nodes in node's i neighbor set, which together can cover α portion of node's i neighbor set (NS_i), and is read as the α neighbor set cover degree.

From this definition we can see that α is an important parameter in calculating the *neighbor set cover degree* ($CD_i(\alpha)$), and directly affects the probability of having a disconnected network. We use α as a tuning parameter in RAT. α trades off energy saving for network connectivity. Choosing a low α value allows for higher levels of energy saving, but may result in a disconnected network. Choosing a high α value lowers the level of energy saving, but will more probably result in a connected network. We use **T**hreshold of **C**onnectivity **C**onfidence, denoted as (T_{cc}), as a representative value for α.



(a)

Node(i)	NS$_i$	d$_i$
1	{2,3,4,6,10,9,13}	7
4	{1,10,6,8,13,5}	6
5	{4,13,8}	3
SNS$_{(1,4)}$	{10,6,13}	
SNS$_{(4,5)}$	{13,8}	

(b)

Node(i)	DoCR$_{(i,j)}$=$\frac{\lvert SNS_{(i,j)}\rvert}{d_i-1}$	NSC$_i$(T$_{cc}$=80%)	CD$_i$(T$_{cc}$=80%)
1	DoCR$_{(1,4)}$=$\frac{3}{6}$	∞	-
4	DoCR$_{(4,1)}$=$\frac{3}{5}$	{1,5}	2
	DoCR$_{(4,5)}$=$\frac{2}{5}$		
5	DoCR$_{(5,4)}$=$\frac{2}{2}$	{4}	1

(c)

Figure 8.1: An example scenario with a T$_{cc}$ value of 80%: (a) communication graph, (b) corresponding NS_i and d_i, and (c) DoCRi, j, NSC$_i$(T$_{cc}$) and corresponding CD$_i$(T$_{cc}$)

8.2.2 Example Scenario

Fig. 8.1 shows an example scenario of several nodes connected by wireless links as shown on the left side, from which we can find the neighbor sets (NS_i) and the pair-wise shared neighbor sets ($SNS_{i,j}$), which is the intersection of node i and node j neighbor sets, as shown in the central table. Based on this scenario, Fig. 8.1(c) shows the pair-wise $DoCR_{i,j}$, $NSC_i(T_{cc})$, and $CD_i(T_{cc})$ for nodes (i = 1, 4, 5). The neighbor set cover of node i ($NSC_i(T_{cc})$) is simply the minimum set selected to calculate the *neighbor set cover degree* of node i ($CD_i(T_{cc})$). It is vital to note that no node maintains the entire tables, and the way this information is distributed over the nodes is presented and discussed next.

8.2.3 Redundancy-Aware Topology Management (RAT)

In the initialization phase all the nodes, which start in the `Active` mode, perform neighbor discovery (See Chapter 7), in which nodes become aware of their neighbor sets and the pair-wise shared neighbor sets with each one of their neighbors. Nodes use the neighbor sets and the shared neighbor sets together to find the $DoCR_{i,j}$ and the $CD_i(T_{cc})$. B-RAT uses $DoCR_{i,j}$ to control the propagating scheduling thread, while E-RAT uses $CD_i(T_{cc})$ to trigger several scheduling threads in the network simultaneously.

Initialization phase

The heart of the initialization phase is to obtain the pair-wise shared neighbor sets. A straightforward technique is for the neighbors to exchange their neighbor sets [15], but this can severely decrease the protocol's potential to save energy after all [8]. In RAT, we exploit the neighbor discovery procedure and the underlying wireless broadcast communication to aid each node in obtaining the shared neighbor sets with each one of its neighbors efficiently. Technically speaking, obtaining shared neighbor set does not incur any new overhead traffic. The neighbor discovery has to be performed anyway. For now, we will assume that the neighbors are aware of their pair-wise shared neighbor sets. A detailed discussion of how we do this is deferred until Subsection 8.3.1.

Once the neighbor set and the shared neighbor sets are available, each node i can locally calculate the $DoCR_{i,j}$ with each neighbor j and the $CD_i(T_{cc})$. Finding the $CD_i(T_{cc})$ is an NP-complete problem (by simple reduction from subset sum). A greedy approximation is to order the nodes in the neighbor set according to their $DoCR_{i,j}$, and start including each neighbor (highest to lowest) in the neighbor set cover until the node reaches a coverage equal to or greater than the T_{cc}. By calculating the $DoCR_{i,j}$ list, and the $CD_i(T_{cc})$, node i is done with the initialization phase and ready for the scheduling phase, which is discussed next.

Scheduling phase

In B-RAT, the sink (sender) triggers a scheduling thread by sending an active announcement. Recipient nodes initiate timers *proportional* to their $DoCR_{i,j}$ with the sender, so that nodes with the least communication redundancy with the sender switch to the `active` mode first and so minimize the total number of active nodes in the network. All other nodes hold back their timers once they receive an active announcement. During the active announcements, any node that is able to collect enough neighbor set coverage becomes eligible for the `sleep` mode and switches to sleep immediately without sending any extra overhead messages.

B-RAT suffers a deadlock problem, which is possible when all the recipient nodes become eligible for the `sleep` mode leaving no node to pick up the scheduling thread. To overcome the deadlock problem in B-RAT, E-RAT uses the neighbor set cover degree to trigger scheduling threads. In a trial to minimize the total number of active nodes, E-RAT lets nodes with higher neighbor set cover degree switch to the `active` mode first and trigger a scheduling thread, giving the opportunity for more nodes to collect enough neighbor set cover and switch to the `sleep` mode. The intuition is that nodes with higher neighbor set cover degree require more nodes to stay active and cover its neighbor set than a node with a smaller neighbor set cover degree. In E-RAT, all nodes start in "E-RAT" operation mode by starting up timers *inversely proportional* to their $CD_i(T_{cc})$. If the timer fires before any scheduling thread reaches that node, the node switches to active and triggers a new scheduling thread to break a potential deadlock. On the other hand, if the node hears an

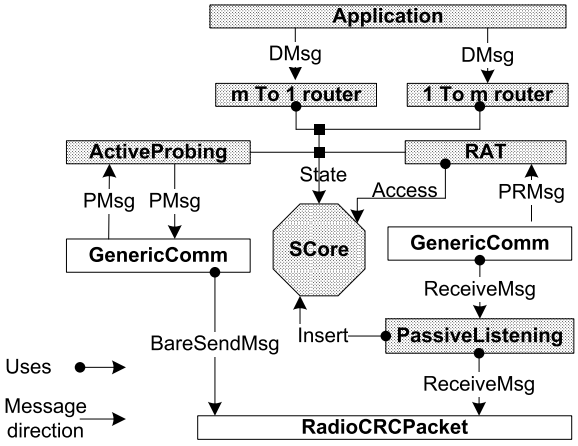


Figure 8.2: Functional components: shaded boxes represent our modules, PMsg, PRMsg, and DMsg represent Probe, Probe reply, and Data messages respectively.

active announcement before the timer fires, the node switches to normal "B-RAT" mode, in which later, the node can switch to active or sleep mode depending on how the existing scheduling threads propagate in the network, but the node will never trigger a new scheduling thread. If it happens that all B-RAT scheduling threads died, some node in the network will eventually break the deadlock by triggering a new scheduling thread as this node must have never been reached by any scheduling thread and is still running in the E-RAT mode.

RAT protocol overhead

The RAT protocol overhead messages consists of two parts, overhead messages in the *initialization phase* and in the *scheduling phase*. By careful design of the neighbor discovery process, RAT can build the shared neighbor sets without introducing any new overhead messages, See Section 8.3.1. To keep low overhead in the scheduling phase, only active nodes need to announce their states. As we present in Section 8.4, the number of active nodes stays constant even for higher density topologies. This ensures a constant message overhead during the scheduling phase.

8.3 Implementation Details

In this section we present the detailed implementation of RAT. Fig. 8.2 shows the functional decomposition including *Score*, *neighbor discovery* service, and finally the RAT module. As we discussed in Chapter 7, the former two service modules provide *integrated* neighbor set abstraction and *neighbor discovery* service. This *integrated* service is used by RAT, which performs topology management. Among other implementation details, we found it is important to discuss how RAT builds the shared neighbor sets without the need for an explicit exchange of neighbor lists.

Recall that the *neighbor discovery* service in Chapter 7 exchanges probe and probe reply messages to populate *Score* with the neighbor set. A probe message consists of the source node number (i.e., ProbMsg(i)), whereas a probe reply message consists of the original probing node number (i.e., i) and the node number of the neighbor node (i.e., j) (i.e., ProbReply(j,i)).

A ProbReply(j,i) message, on its own, tells any third party receiving node that both i and j are neighbors. RAT leverages this by overhearing these messages to build and maintain the shared neighbor sets; a detailed discussion of this process is presented next.

8.3.1 Building Shared Neighbor Sets

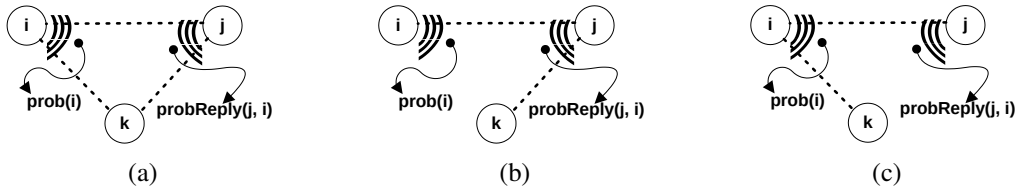


Figure 8.3: Building shared neighbor sets at node k.

As we discussed earlier, all the nodes have to have access to their neighbor sets and the shared neighbor sets with their neighbors to finish the initialization phase of RAT. The *neighbor discovery* process along with *Score* provides RAT with sequential as well as random access to the neighbor set. The shared neighbor sets are built and maintained by RAT itself. By overhearing a probe reply message sent from node j to node i, RAT at node k can add node j to the shared neighbor set of

node i, and node i to the shared neighbor set of node j only if node i is in the neighbor set of node k. This scenario is illustrated in Fig. 8.3(a). Fig. 8.3(b) and Fig. 8.3(c) show the other two possible scenarios. In Fig. 8.3(b) node j and node k are neighbors, but node k and node i are not. In this case, node k will receive the probe reply message from node j, but never add any records to the shared neighbor sets as node i is not in node k's neighbor set. In Fig. 8.3(c) node k and node i are neighbors, but node k and node j are not. In this case, node k does not receive the probe reply message from node j in the first place.

8.4 Evaluation

8.4.1 Evaluation Setup and Metrics

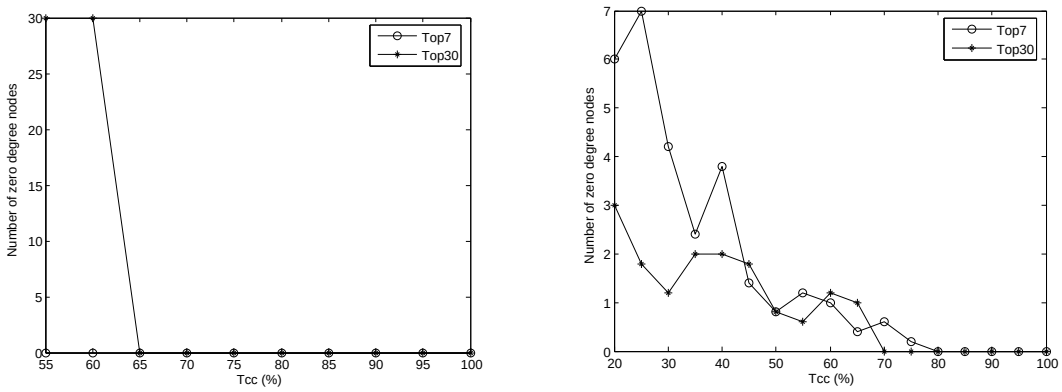
Using nesC and PowerTOSSIM we implemented B-RAT and E-RAT and compared their performance to the all-active case. In B-RAT and E-RAT cases, only active nodes participate in the multi-hop network, while in the all-active case, all the nodes actively participate in the network and forward messages.

Two sets of performance metrics are used, the first set is low level metrics to show some attractive characteristics of the protocol, while the second set is high level metrics to show the advantage of B-RAT and E-RAT over the all-active case in terms of energy savings. The first set includes four metrics. First is the *average active node degree*, which is the average number of active nodes in the neighbor sets of all the nodes in the network. Second is the *active degree distribution*, which is the number of nodes having the same active node degree. Third is the *sleeping ratio*, which is the ratio of sleeping nodes to the total number of nodes. Fourth is the *propagation time*, which is the time it takes the protocol (B-RAT and E-RAT) to finish and all the nodes switch to active or sleep.

In the second set, we use the total power consumption in comparing B-RAT and E-RAT to the all-active case. Two communication patterns are used, *one-to-many* and *many-to-one*. A detailed discussion of the experiments is presented later.

The experiments are conducted on five topologies. In all of the topologies, the nodes were randomly distributed over a fixed area of 100 by 100 units squared. In order to show the ability

of B-RAT and E-RAT in leveraging high node densities, the average node degree in the topologies varies from seven to thirty; we use (Top 7, Top 10, Top 17, Top 21, and Top 30) to refer to them. The communication ranges are adapted to get the desired node degree.



(a) (b)

Figure 8.4: Choosing appropriate T_{cc} values: (a) B-RAT, and (b) E-RAT value.

8.4.2 Simulation Results

In order to decide a good T_{cc} value, which is used to ensure a connected network, we conducted a set of experiments in which we ran B-RAT and E-RAT using T_{cc} values ranging from 20% to 100%. Each time the number of zero degree nodes was recorded and plotted as in Fig. 8.4(a) and Fig. 8.4(b). The number of zero degree nodes is used as an approximation to find out whether the network is connected. As the sleep mode eligibility is lenient for small T_{cc} values, B-RAT suffered deadlocks frequently, and in most cases the scheduling thread was not able to propagate over the entire network. Top 7 and Top 30 are used in these experiments as representatives. We can see from Fig. 8.4(b) that a T_{cc} value of 85% is good enough to ensure connectivity.

The second experiment shows the average active node degree after the application of B-RAT or E-RAT as the node density increases. A constant average node degree as the node density increases is important first to prove the correctness of the protocol and to show that the protocol actually has a constant overhead. The experiment was performed by running B-RAT and E-RAT on each one of

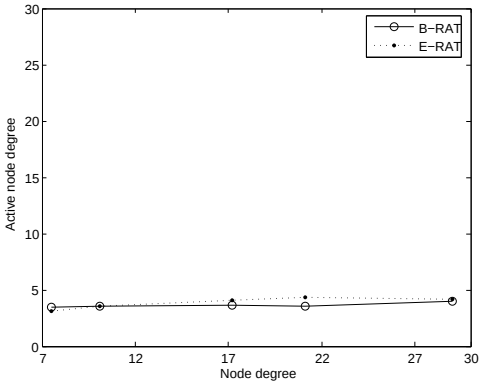
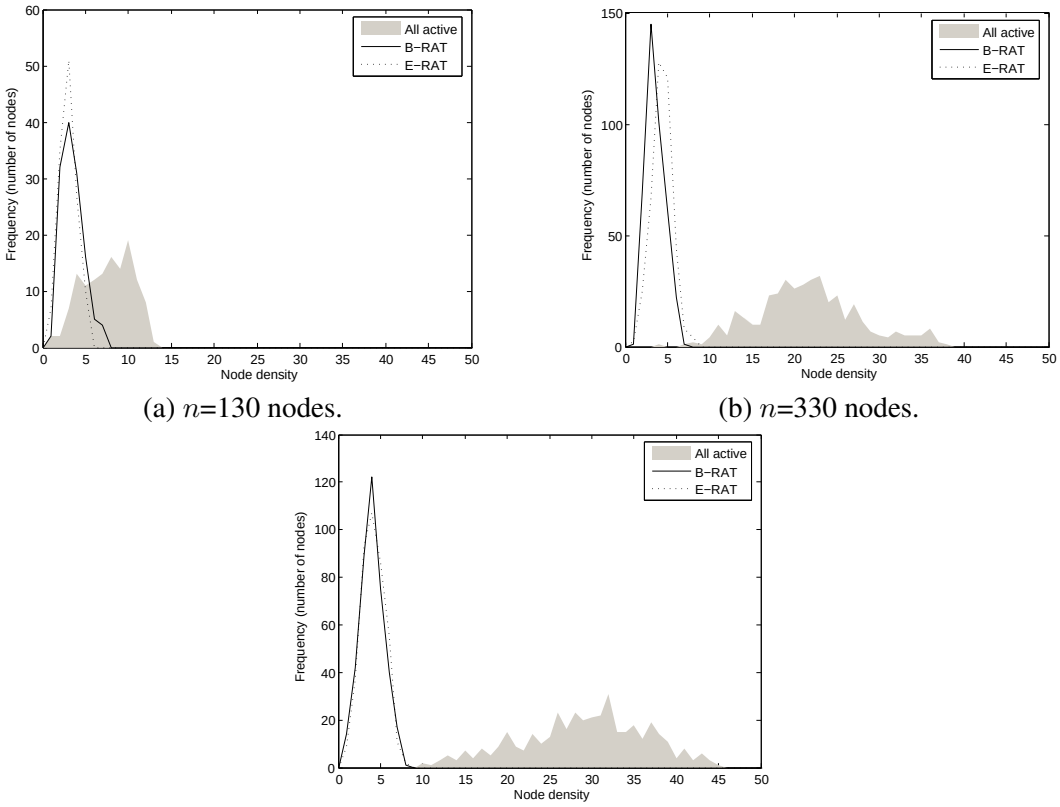


Figure 8.5: The average active node degree of different topologies in B-RAT and E-RAT.

the topologies and taking the average active node degree over all the nodes. Each value in Fig. 8.5 represents the average of 10 runs of B-RAT and E-RAT.

We can observe from Fig. 8.5 that both B-RAT and E-RAT maintain a constant active node degree as the deployment node density increases, which emphasizes two things: First, both B-RAT and E-RAT can leverage high node redundancy by selecting a constant number of nodes to be active, which is only necessary for connectivity. Second, B-RAT and E-RAT maintain a constant protocol overhead (i.e. only active nodes announce their states in the protocol). In addition to the average active node degree, the active node degree distribution is important to show how uniformly the active nodes spread over the sensor field. Having a uniform active node degree helps in avoiding a situation where some network channels have high contention (high node degrees), while others have low contention, which makes the decision of a MAC back off time, for example, difficult. An ideal distribution would be for all the nodes to have exactly the same active node degree, but this is impossible as boundary nodes by default have less node degree. Fig. 8.6 depicts the active node degree distribution for B-RAT, E-RAT, and the all-active case of three configuration, where the x-axis represents the active node degree values, while the y-axis represents the number of nodes with the corresponding active node degree. The figures show that in the all-active case, the node degree is highly variable. For example, in Fig. 8.6(c), some nodes have a degree of 40 while others have



(a) n=130 nodes. (b) n=330 nodes.

(c) n=400 nodes.

Figure 8.6: Active node degree distribution: (a) Top 7 (b) Top 21 (c) Top 30, n is the total number of nodes.

a degree of 10. In B-RAT and E-RAT, we can observe a neat distribution where most of the nodes have an active degree of 5.

An understanding of the protocol's propagation and convergence time is important. Other network services and protocols need to wait for the network to become stable before starting generating and sending data messages. Fig. 8.7 (next page) plots the cumulative distribution frequency of the decided nodes over time. By decided nodes we mean a node that either switches to `active` or `sleep` mode. Both can finish the protocol in less than 4 milliseconds. E-RAT can propagate faster in the network, which follows from the fact that E-RAT initiates multiple B-RAT threads concurrently in the network. In addition to a deadlock-free RAT, E-RAT has a faster propagation time.

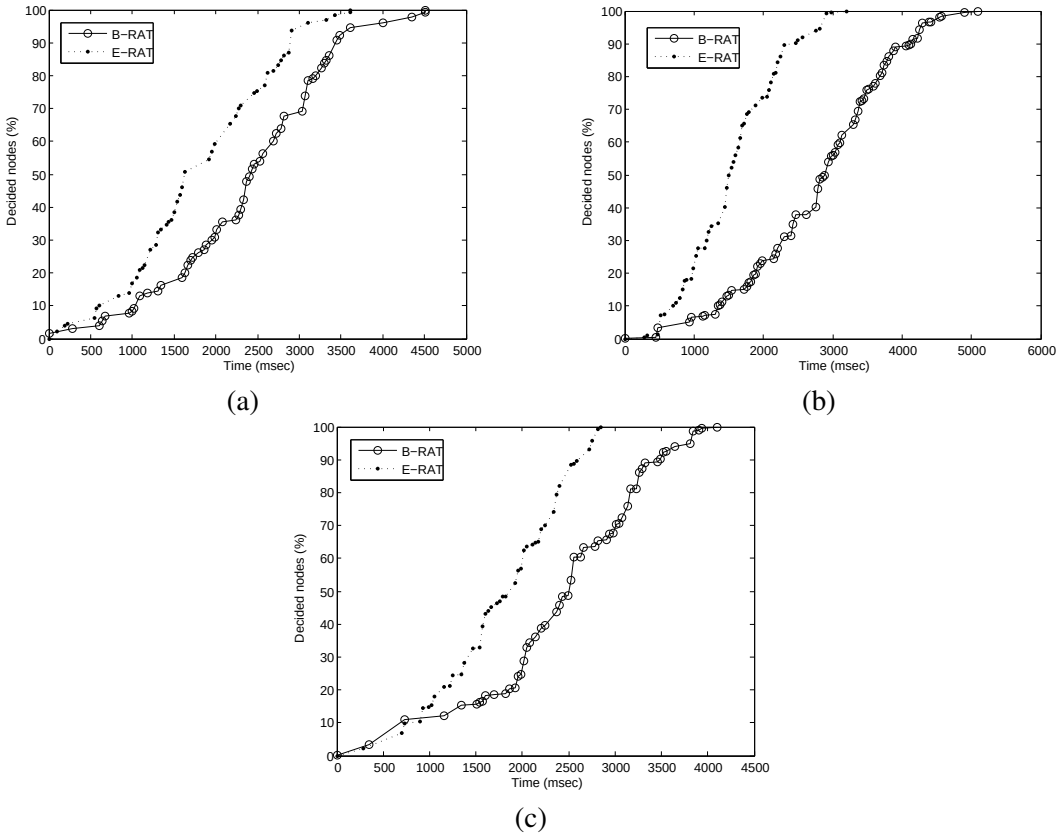


Figure 8.7: The percentage of decided nodes vs time: (a) Top 7 (b) Top 21 (c) Top 30.

The ability of B-RAT or E-RAT in saving energy can be predicted from the percentage of nodes that switch to `sleep` mode. Fig. 8.8 shows an increasing percentage of sleeping node as the node redundancy increases, a percentage of more than 85% of the nodes switched to sleeping mode in topology 30.

In the last two experiments, we used the two most common communication patterns (one-to-many and many-to-one) to compare the performance of B-RAT and E-RAT to the all-active case in terms of the total power consumption. The sink in the *one-to-many* routing scenario, periodically (every 60 seconds) broadcasts a data message, and the recipient nodes rebroadcast again. To avoid forwarding the same data message more than once, each node forwards the same data message only once. In the all-active case, all the nodes stay active all the time and participate in the forwarding

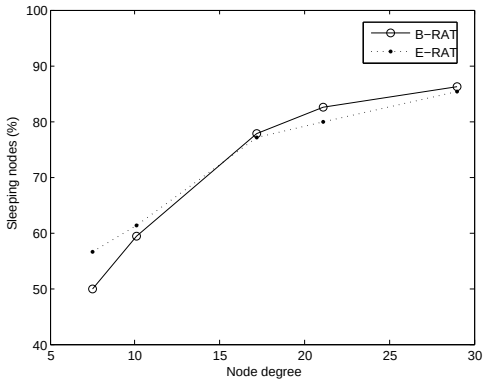


Figure 8.8: Energy saving potential.

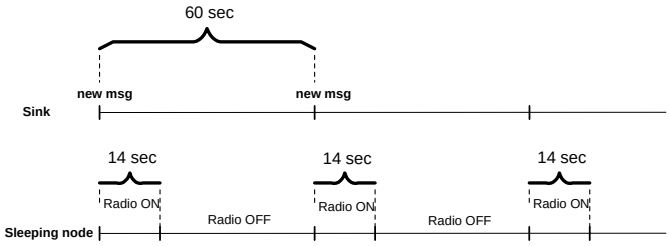


Figure 8.9: Sleeping node duty cycle.

process, while in B-RAT and E-RAT, only active nodes stay active all the time and participate in message forwarding. On the other hand, sleeping nodes do not forward data messages and turn their radios off unless they need to receive data messages. A simple time line for the sink and a sleeping node is shown in Fig. 8.9. The figure shows that a sleeping node turns its radio on for 14 seconds starting from the time a new message is generated at the sink, 14 seconds is the maximum time that a data message may take to propagate from the sink to the furthest node in the network. This time interval (i.e., 14 seconds) is a conservative value that accounts for the maximum one hop delay and the maximum number of hops in the network. A one hop delay may take up to 1 second, which is the maximum back off time a node may wait to avoid collisions, and the maximum hop in the topologies in our simulation is 14.

In the *many-to-one* routing scenario, the nodes in the network are arranged in a shortest path routing tree rooted at the sink. Each node generates a data message every 60 seconds, which is forwarded up the tree to the sink after being aggregated with other data messages. In the all-active case, all the nodes are active, join the routing tree, and participate in the data message forwarding, while in B-RAT and E-RAT, only the active nodes form the routing tree and forward data messages. Sleeping nodes turn their radios off unless they need to send a new data message generated locally at the node.

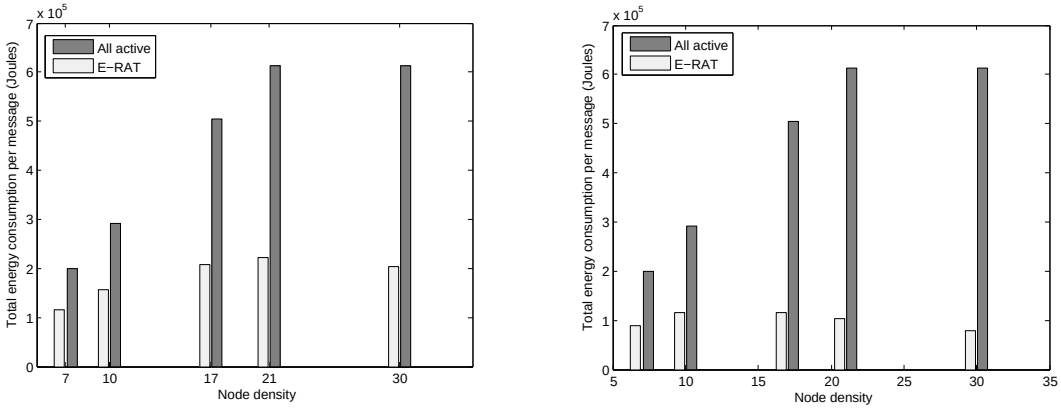


Figure 8.10: Total energy consumption per data message: (a) One-to-many routing and (b) Many-to-one routing.

Fig. 8.10 compares the total energy consumption of E-RAT and the all-active case for three different topologies in the *one-to-many* routing scenario (Fig. 8.10(a)) and the *many-to-one* routing scenario (Fig. 8.10(b)). Since B-RAT and E-RAT show similar energy saving potential in Fig. 8.8 we use E-RAT only in our comparison with the all-active case. In both Fig. 8.10(a) and Fig. 8.10(b), the y-axis represents the total energy in joules consumed by all the nodes in the network for each individual data message generated in the network, while the x-axis represents the node density of each topology. Fig. 8.10(a) shows a total energy reduction of at least half in low density topologies (Top 7) and up to 67% in total energy reduction for higher density topologies (Top 30). Fig. 8.10(b)

on the other hand, shows a total energy reduction of more than 80% for a high node density (Top 30).

In the many-to-one routing scenario, Fig. 8.10(b), E-RAT shows an even higher energy saving over the all-active case compared to the *one-to-many* routing scenario, Fig. 8.10(a). This is due to the fact that in the *many-to-one* routing scenario in Fig. 8.10(b), sleeping nodes don't need to become active to receive data messages. Instead, sleeping nodes become active only to send data messages. This allows for an even lower duty-cycle of sleeping nodes.

8.5 Related Work and Discussion

Several topology management protocols have been presented in the literature [13, 15, 18, 92], RAT complements and enhances previous work by providing a working nesC/TinyOS implementation, which accounts for more practical issues such as irregular wireless channels. We also consider new important performance metrics in addition to energy saving, such as active node density distribution and propagation time.

GAF [92] uses physical location to define communication equivalence and redundancy. The assumption that relates physical location to connectivity does not necessarily hold in real deployments [99, 101] and limits GAF applicability in real life deployments. Using the neighbor set to define communication redundancy, RAT adapts and captures harsh connectivity models. Exchanging long neighbor lists (high density deployments) among neighbors, as in SPAN and ReORG [15, 18] respectively, puts high overhead on the network. This high overhead limits the protocols' ability in saving energy. To avoid exchanging neighbor lists, RAT leverages the underlying broadcast medium of wireless channels to build communication redundancy information, which is necessary for selecting the active node set. In addition to overhead resulting from exchanging neighbor lists, ReORG requires all the nodes to announce their states (active or sleep). This results in poor scalability with the total number of nodes. RAT, on the other hand, requires only active nodes to announce their states. Since the number of active nodes is constant relative to the total number of nodes, RAT scales very well to deployments with large number of nodes.

Also, the tight coupling of the routing and topology management layers in SPAN [15] may unnecessarily limit the design space for the routing layer designers. RAT avoids any dependency on the routing layer, which makes it operational with any routing protocol. Nevertheless the *state* interface provided by *Score* allows the routing layer to get notification of any topology changes smoothly so that it can adapt its routing infrastructure appropriately. ASCENT [13] takes a different approach in selecting a set of active nodes. It uses the active node density and loss rate as driving factors in assigning nodes active and sleep states. Transient node failures and high wireless channel quality variation may compromise the integrity of a node's decision of going to sleep or active, which may lead to a disconnected network. RAT guarantees connectivity by forcing nodes to stay active unless a set of active nodes already provide enough communication redundancy.

Topology management, in which a set of nodes stay active while other nodes turn their radios off completely, is not the only mechanism that has been used to control network topology and so save energy. Several protocols and algorithms [53, 55, 90] have been proposed to control network topology by adjusting the sensor node sending power. Such mechanisms can be used side by side to complement our work and further provide higher levels of energy savings.

8.6 Summary

In this chapter, we discussed the implementation and evaluation of the RAT protocol under two routing scenarios. RAT achieves energy savings up to 87% by leveraging high node density by assigning 80% of the nodes to the sleep mode and assigning a small set of nodes, good enough to maintain connectivity, to the active mode.

CHAPTER 9
CONCLUSION AND FUTURE WORK

We dedicate the final chapter to summarize and conclude this work, we also, discuss future research directions.

We follow a top-down as well as a bottom-up approach to tackle very important aspects of WSNs including their availability, autonomy, and energy efficiency. We develop analytical systems to model WSNs availability and use these models to attack two major WSNs problems: the deployment problem and availability-aware node scheduling problem. As we shown by the analytical as well as simulation results, our solutions to these problems move WSNs toward a more dependable and autonomous as well as cost efficient systems. In an attempt to validate some of the assumptions that we have used in our models, we leverage some sensor systems failure traces to draw some conclusions about their failure patterns including effect of environmental conditions on inflicting failures. On the other hand, we approach dependable and autonomous WSNs from a more systems perspective by proposing a new and more flexible structuring of the WSN communication stack (*Score*). In the context of *Score* the network protocols collaborate and coordinate in arbitrary manner without the need for tight coupling. On top of *Score*, we develop several network services and protocols that enable autonomous WSNs including neighbor the discovery and topology discovery services. Finally, we propose, implement, and evaluate the *RAT* protocol, which leverages high node density to schedule nodes `ON` and `OFF` to save energy.

My future work is twofold: I plan to continue on improving availability and reliability modeling by incorporating more aspects of the WSN system. These models help in addressing similar pre-deployment stage problems. On the other hand, I plan to continue on developing more network services that benefit from and runs on top of the *Score* framework and improve on the existing ones to support operational stage autonomous WSNs. Furthermore, I plan to utilize simulation techniques

that permit capturing WSNs failure scenarios that are difficult to capture in the analytical models. In attempt to bridge the gap between operational stage protocols and decrease the pressure on pre-deployment stage models, I plan to investigate a new self-learning deployment strategy that does not assume prior knowledge of the sensor node failure rate (i.e., λ). The new strategy employs life-testing techniques to estimate λ in the early phases of the WSN lifetime. During this early phase, the deployment strategy performs deployment visits on an on-demand basis, and once a λ estimate with certain confidence is reached, the deployment strategy adopts the **pro-active** approach.

Appendix A

PUBLICATION LIST

A.1 Published

- **Safwan Al-Omari** and Weisong Shi, Toward Low Cost and Highly Reliable Sensor Networks Deployment [extended abstract], ACM CoNEXT Student Workshop 2008, Spain, Madrid, December 9, 2008.

- Kewei Sha, Guoxing Zhan, **Safwan Al-Omari**, Tim Calappi, Weisong Shi and Carol Miller, Data Quality and Failures Characterization of Sensing Data in Environmental Applications, in Proceedings of the 4th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom'08), Orlando, November 13-16.

- **Safwan Al-Omari** and Weisong Shi, Availability Modeling and Analysis of Autonomous In-Door WSNs, in Proceedings of the 4th IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS'07), Pisa, Italy, October 8-11, 2007.

- **Safwan Al-Omari** and Weisong Shi, A Novel Topology Discovery Service for Self-Organized WSNs, in Proceedings of the 1st International Conference on Wireless Algorithms, Systems and Applications (WASA'07), Chicago, August 1-3, 2007.

- **Safwan Al-Omari** and Weisong Shi, Towards Highly-Available WSNs for Assisted Living, in Proceedings of the 1st International Workshop on Systems and Networking Support for Healthcare and Assisted Living Environments (HealthNet'07), in conjunction with USENIX/ACM MobiSys'07, San Juan, June 11-14, 2007.

- **Safwan Al-Omari** and Weisong Shi. Redundancy-aware topology control in wireless sensor networks. In Proceedings of the 2nd International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom06), Atlanta, Georgia, USA, November 17th - 20th, 2006.

- **Safwan Al-Omari**, Junzhao Du, and Weisong Shi. Score: A sensor core framework for cross-layer design [extended abstract]. In Proceedings of The Third International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks (QShine06), Waterloo, Ontario, Canada, August 7 9, 2006.

A.2 Under Submission

- **Safwan Al-Omari** and Weisong Shi, A Novel Topology Discovery Service for Self-Organized WSNs. Submitted to IJDSN (major review).

- **Safwan Al-Omari** and Weisong Shi, Incremental Sensor Node Deployment for Low Cost and Highly Available WSN, MIST-TR-2008-002. Submitted to IEEE TVT.

- **Safwan Al-Omari**, Weisong Shi, and Carol J. Miller. Sesame: A sensor system accessing and monitoring environment. Technical Report MIST-TR-2004-018, Wayne State University, November 2004.

REFERENCES

[1] S. Al-Omari and W. Shi. Redundancy-aware topology control in wireless sensor networks. In *Proc. of CollaborateCom'06*, 2006.

[2] S. Al-Omari and W. Shi. Availability modeling and analysis of autonomous in-door wsns. In *Proceedings of IEEE MASS'07*, September 2007.

[3] S. Al-Omari, W. Shi, and C. J. Miller. Sesame: A sensor system accessing and monitoring environment. Technical Report MIST-TR-2004-018, Wayne State University, November 2004.

[4] Lichun Bao and J. J. Garcia-Luna-Aceves. Topology management in ad hoc networks. In *MobiHoc '03: Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 129–140, New York, NY, USA, 2003. ACM Press.

[5] M. Bebbington, C. Lai, and R. Zitikis. Useful periods for lifetime distributions with bathtub shaped hazard rate functions. *IEEE Tran. Reliability*, 55(2):245–251, 2006.

[6] Christian Bettstetter. On the minimum node degree and connectivity of a wireless multihop network. In *Proceedings of the MobiHoc'02*, June 2002.

[7] R. Bhatia and M. Kodialam. On power efficient communication over multi-hop wireless networks: joint routing, scheduling, and power control. March 2004.

[8] D. M. Blough and P. Santi. Investigating upper bounds on network lifetime extension for cell-based energy conservation techniques in stationary ad hoc networks. In *Proceedings of the 8th Annual ACM/IEEE International Conference on Mobile Computing and Networking(MobiCom'02)*, 2002.

[9] P. Brass. Bounds on coverage and target detection capabilities for models of networks of mobile sensors. *ACM Transactions on Sensor Networks*, 3(2), 2007.

[10] N. Bulusu, J. Heidemann, and D. Estrin. Adaptive beacon placement. In *Proceedings of the 21st International Conference on Distributed Computing Systems*, April 2001.

[11] Martin Burkhart, Pascal von Rickenbach, Roger Wattenhofer, and Aaron Zollinger. Does topology control reduce interference? In *MobiHoc '04: Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*, pages 9–19, New York, NY, USA, 2004. ACM Press.

[12] A. Cerpa and D. Estrin. ASCENT: Adaptive self-configuring sensor network topologies. In *Proceedings of the IEEE Infocom'02*, June 2002.

[13] Alberto Cerpa and Deborah Estrin. Ascent: Adaptive self-configuring sensor networks topologies. *IEEE Transactions on Mobile Computing Special Issue on Mission-Oriented Sensor Networks*, 3(3), July-September 2004.

[14] K. Chakrabarty, S. Iyengar, H. Qi, and E. Cho. Grid coverage of surveillance and target location in distributed sensor networks. *IEEE Transaction on Computers*, 51(12):1448 – 1453, 2002.

[15] 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 7th Annual ACM/IEEE International Conference on Mobile Computing and Networking(MobiCom'01)*, July 2001.

[16] M. Chiang. To layer or not to layer: Balancing transport and physical layers in wireless multihop networks. March 2004.

[17] T. Clouqueur, V. Phipatanasuphorn, P. Ramanathan, and K. Saluja. Sensor deployment strategy for target detection. In *Proceedings of WSNA'02*, September 2002.

[18] W. Steven Conner, Jasmeet Chhabra, Mark Yarvis, and Lakshman Krishnamurthy. Experimental evaluation of synchronization and topology control for in-building sensor network applications. In *WSNA '03: Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, pages 38–49, New York, NY, USA, 2003. ACM Press.

[19] K. Dasgupta et al. An efficient clustering-based heuristic for data gathering and aggregation in sensor networks. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC'03)*, March 2003.

[20] B. Deb, S. Bhatnagar, and B. Nath. A topology discovery algorithm for sensor networks with applications to network management. In *In IEEE CAS workshop(short paper)*, September 2002.

[21] Budhaditya Deb and Badri Nath. On the node-scheduling approach to topology control in ad hoc networks. In *MobiHoc '05: Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, pages 14–26, New York, NY, USA, 2005. ACM Press.

[22] S. Dhillon and K. Chakrabarty. Sensor placement for effective coverage and surveillance in distributed sensor networks. In *Proceedings of WCNC'03*, March 2003.

[23] S. Dhillon, K. Chakrabarty, and S. Iyengar. Sensor placement for grid coverage under imprecise detections. In *Proceedings of the 5th International Conference on Information Fusion (fusion'02)*, July 2002.

[24] Devdatt Dubhashi, Alessandro Mei, Alessandro Panconesi, Jaikumar Radhakrishnan, and Arvind Srinivasan. Fast distributed algorithms for (weakly) connected dominating sets and

linear-size skeletons. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 717–724, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[25] Yong Gao, Kui Wu, and Fulu Li. Analysis on the redundancy of wireless sensor networks. In *WSNA '03: Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, pages 108–114, New York, NY, USA, 2003. ACM Press.

[26] D. Gay, P. Levis, R. Behren, M. Welsh, E. Brewer, and D. Culler. The nesc language: A holistic approach to networked embedded systems. In *Proc. of PLDI'03*, June 2003.

[27] O. Gnawali, K. Jang, J. Paek, M. Vieira, R. Govindan, B. Greenstein, A. Joki, D. Estrin, and E. Kohler. The tenet architecture for tiered sensor networks. In *Proc. of SenSys '06*, 2006.

[28] B. Gnedenko, Y. Belyayev, and A. Solovyev. *Mathematical Methods of Reliability Theory*. Acadamic Press, 1969.

[29] B. Greenstein, E. Kohler, and D. Estrin. A sensor network application construction kit (snack). In *Proc. of ACM SenSys 2004*, November 2004.

[30] Himanshu Gupta, Samir R. Das, and Quinyi Gu. Connected sensor cover: self-organization of sensor networks for efficient query execution. In *MobiHoc '03: Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 189–200, New York, NY, USA, 2003. ACM Press.

[31] P. Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 19(2):73–85, February 2000.

[32] MohammadTaghi Hajiaghayi, Nicole Immorlica, and Vahab S. Mirrokni. Power optimization in fault-tolerant topology control algorithms for wireless multi-hop networks. In *MobiCom*

'03: Proceedings of the 9th annual international conference on Mobile computing and networking, pages 300–312, New York, NY, USA, 2003. ACM Press.

[33] B. Hamdaoui and P. Ramanathan. *Energy-Efficient and MAC-Aware Routing for Data Aggregation in Sensor Networks*. IEEE Press, October 2004.

[34] Balakrishnan H Heinzelman WR, Chandrakasan A. Energy-efficient communication protocol for wireless microsensor networks. In *Proc. of Hawaii Internaltional Conference On System Sciences (HICSS'00)*, January 2000.

[35] A. Howard, M. Matadd, and G. Sukhatme. An incremental self-deployment algorithm for mobile sensor networks. *Autonomous Robots, Special Issue on Intelligent Embedded Systems*, 13(2):113–126, 2001.

[36] A. Howard, M. Mataric, and G. Sukhatme. Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In *Proceedings of DARS'02*, 2002.

[37] V. Isler, K. Daniilidis, and S. Kannan. Sampling based sensor-network deployment. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, September 2004.

[38] S. Jayashree, B. S. Manoj, and C. Siva Ram Murthy. On using battery state for medium access control in ad hoc wireless networks. In *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 360–373, New York, NY, USA, 2004. ACM Press.

[39] W. Jiang, C. Hu, Y. Zhou, and A. Kanevsky. Are disks the dominant contributor for storage failures? a comprehensive study of storage subsystem failure characteristics. In *Proceedings of USENIX FAST'08*, January 2008.

[40] Aditya Karnik and Anurag Kumar. Distributed optimal self-organisation in a class of wireless sensor networks. In *Proc. of IEEE Conference on Computer Communications (INFOCOM'04)*, March 2004.

[41] Brad Karp and H. T. Kung. Gpsr: greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking(MobiCom'00)*, August 2000.

[42] R. H. Katz, J. M. Kahn, and K. J. Pister. Mobile networking for smart dust. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'99)*, Seattle, WA, August 1999.

[43] U. Kozat, I. Koutsopoulos, and L. Tassiulas. A framework for cross-layer design of energy-efficient communication with qos provisioning in multi-hop wireless networks. March 2004.

[44] B. Krishnamachari. *Networking Wireless Sensors*. Cambridge University Press, 2006.

[45] S. Kumar, T. Lai, and J. Balogh. On k-coverage in a mostly sleeping sensor network. In *Proceedings of the 10th Annual ACM/IEEE International Conference on Mobile Computing and Networking(MobiCom'04)*, September 2004.

[46] P.F. Lagasse and E.V. Richardson. Asce compendium of stream stability and bridge scour papers. 127(7):531, 2001.

[47] P.F. Lagasse, J.D. Schall, and E.V. Richardson, 2001. Stream stability at highway structures (3^{rd} edition): Federal Highway Administration Hydraulic Engineering Circular No. 20.

[48] P.F. Lagasse, L.W. Zevenbergen, J.D. Schall, and P.E. Clopper, 2001. Bridge scour and stream instability countermeasures: Experience, selection, and design guidelines (2^{nd} edition): Federal Highway Administration Hydraulic Engineering Circular No. 23.

[49] M. Leoncini, G. Resta, and P. Santi. Analysis of a wireless sensor dropping problem in wide-area environmental monitoring. In *Proceedings of IPSN'05*, April 2005.

[50] P. Levis et al. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proceedings of the First USENIX/ACM Networked System Design and Implementation*, March 2004.

[51] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: Accurate and scalable simulation of entire tinyos applications. In *Proceedings of the First ACM SenSys'03*, November 2003.

[52] Li Li, Joseph Y. Halpern, Paramvir Bahl, Yi-Min Wang, and Roger Wattenhofer. A cone-based distributed topology-control algorithm for wireless multi-hop networks. *IEEE/ACM Trans. Netw.*, 13(1):147–159, 2005.

[53] N. Li, C. Hou, and L. Sha. Design and analysis of an mst-based topology control algorithm. In *Proc. of INFOCOM'03*, March 2003.

[54] N. Li and J. Hou. Topology control in hetergeous wireless networks: Problems and solutions. In *Proc. of IEEE Conference on Computer Communications (INFOCOM'04)*, March 2004.

[55] N. Li and J. C. Hou. Localized topology control algorithms for heterogeneous wireless networks. *IEEE/ACM Trans. on Networking*, December 2005.

[56] Errol L. Lloyd, Rui Liu, Madhav V. Marathe, Ram Ramanathan, and S. S. Ravi. Algorithmic aspects of topology control problems for ad hoc networks. *Mob. Netw. Appl.*, 10(1-2):19–34, 2005.

[57] Errol L. Lloyd, Rui Liu, Madhav V. Marathe, Ram Ramanathan, and S. S. Ravi. Algorithmic aspects of topology control problems for ad hoc networks. *Mob. Netw. Appl.*, 10(1-2):19–34, 2005.

[58] X. Luo, M. Dong, and Y. Huang. On distributed fault-tolerant detection in wireless sensor networks. *IEEE Trans. Computers*, 55(1):58–70, 2006.

[59] S. Madden, M. J. Franklin, J. Hellerstein, and W. Hong. Tag: A tiny aggregation service for ad-hoc sensor network. In *Proc. of the Fifth USENIX Symposium on Operating Systems Design and Implementation*, December 2002.

[60] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor network for habitat monitoring. In *Proceedings of the First ACM Workshop on Wireless Sensor Networks and Applications (WSNA)*, September 2002.

[61] D. Marsh, R. Tynan, G O'Hare, and A. Ruzzelli. The effects of deployment irregularity on coverage in wireless sensor networks. In *Proceedings of ISSNIP'05*, December 2005.

[62] D.S. Mueller and C.R. Wagner, 2002. Field Observations and Evaluations of Streambed Scour at Bridges, Federal Highway Administration Report.

[63] S. Nath et al. Synopsis diffusion for robust aggregration in sensor networks. In *Proc. of ACM SenSys 2004*, November 2004.

[64] National Transportation Safety Board (NTSB), 1988. Collapse of the New York Thruway (I-90) Bridge of Schoharie Creek, near Amsterdam, New York, April 5^{th}, 1987.

[65] U.S. Department of Transportation (USDOT), 1991. Evaluating scour at bridges, Technical Advisory.

[66] Jianping Pan, Y. Thomas Hou, Lin Cai, Yi Shi, and Sherman X. Shen. Topology control for wireless sensor networks. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 286–299, New York, NY, USA, 2003. ACM Press.

[67] E.V. Richardson and S.R. Davis, 2001. Evaluating scour at bridge (4^{th} edition): Federal Highway Administration Hydraulic Engineering Circular No. 18.

[68] P. Santi and D. Blough. The critical transmitting range for connectivity in sparse wireless ad hoc networks. *IEEE Transactions on Mobile Computing*, 2(1):25–39, 2003.

[69] Paolo Santi, Douglas M. Blough, and Feodor Vainstein. A probabilistic analysis for the range assignment problem in ad hoc networks. In *MobiHoc '01: Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, pages 212–220, New York, NY, USA, 2001. ACM Press.

[70] B. Schroeder and G. Gibson. Disk failures in the real world: What does an mttf of 1,000,000 hours mean to you? In *Proceedings of USENIX FAST'07*, February 2007.

[71] C. Schurgers et al. Topology management for sensor networks: exploiting latency and density. In *Proceedings of the MobiHoc'02*, June 2002.

[72] C. Schurgers and M.B. Srivastava. Energy efficient routing in wireless sensor networks. In *MILCOM Proceedings on Communications for Network-Centric Operations: Creating the Information Force*, 2001.

[73] Loren Schwiebert, Sandeep K.S. Gupta, and Jennifer Weinmann. Research challenges in wireless networks of biomedical sensors. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 151–165, New York, NY, USA, 2001. ACM Press.

[74] K. Seada, M. Zuniga, A. Helmy, and B. Krishnamachari. Energy-efficient forwarding strategies for geographic routing in lossy wireless sensor networks. In *Proc. of ACM SenSys 2004*, November 2004.

[75] R. Shah and J. Rabaey. Energy aware routing for low energy ad hoc sensor networks. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC'02)*, March 2002.

[76] W. Shi and C. Miller. Waste containment system monitoring using wireless sensor networks. Technical Report MIST-TR-2004-009, Wayne State University, March 2004.

[77] V. Shnayder et al. Simulating the power consumption of large-scale sensor network applications. In *Proc. of ACM SenSys 2004*, November 2004.

[78] K. Sohrabi, J. Gao, V. Ailawadhi, and G. J. Pottie. Protocols for self-organization of a wireless sensor network. *IEEE Personal Communications*, 7(5):16–27, October 2000.

[79] R. Szewczyk, A. Mainwaring, J. Polastre, and D. Culler. An analysis of a large scale habitat monitoring application. In *Proc. of ACM SenSys 2004*, November 2004.

[80] Di Tian and Nicolas D. Georganas. A coverage-preserving node scheduling scheme for large wireless sensor networks. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 32–41, New York, NY, USA, 2002. ACM Press.

[81] S. Tilak, N. Abu-Ghazaleh, and W. Heinzelman. Infrastructure tradeoffs for sensor networks. In *Proceedings of WSNA'02*, 2002.

[82] N. Vlajic and D. Xia. Wireless sensor networks: To cluster or not to cluster? In *Proceedings WoWMoM'06*, June 2006.

[83] Peng-Jun Wan, Khaled M. Alzoubi, and Ophir Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. *Mob. Netw. Appl.*, 9(2):141–149, 2004.

[84] G. Wang, G. Cao, and T. La Porta. Movement-assisted sensor deployment. *IEEE Transactions on Mobile Computing*, 5(6):640–652, 2006.

[85] Xiaorui Wang, Guoliang Xing, Yuanfang Zhang, Chenyang Lu, Robert Pless, and Christopher Gill. Integrated coverage and connectivity configuration in wireless sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 28–39, New York, NY, USA, 2003. ACM Press.

[86] Yu Wang, WeiZhao Wang, and Xiang-Yang Li. Distributed low-cost backbone formation for wireless ad hoc networks. In *MobiHoc '05: Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, pages 2–13, New York, NY, USA, 2005. ACM Press.

[87] M. Welsh and G. Mainland. Programming sensor network using abstract regions. In *Proceedings of the First USENIX/ACM Networked System Design and Implementation*, March 2004.

[88] K. Whitehouse, C. Sharp, D. Culler, and E. Brewer. Hood: A neighborhood abstraction for sensor networks. In *Proc. of ACM MobiSys'04*, June 2004.

[89] A. Woo and D. Culler. A transmission control scheme for media access in sensor networks. In *Proceedings of the 7th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'01)*, Rome, Italy, July 2001.

[90] J. Wu and F. Dai. Mobility-sensitive topology control in mobile ad hoc networks. *accepted to apear in IEEE Tran. on Parallel and Distributed Systems*, June 2006.

[91] Guoliang Xing, Xiaorui Wang, Yuanfang Zhang, Chenyang Lu, Robert Pless, and Christopher Gill. Integrated coverage and connectivity configuration for energy conservation in sensor networks. *ACM Trans. Sen. Netw.*, 1(1):36–72, 2005.

[92] Y. Xu, J. Heidemann, and D. Estrin. Geography-informed energy conservation for ad hoc routing. In *Proceedings of the 7th Annual ACM/IEEE International Conference on Mobile Computing and Networking(MobiCom'01)*, July 2001.

[93] F. Ye, G. Zhong, S. Lu, and L. Zhang. PEAS: A robust energy conserving protocol for long-lived sensor networks. In *Proc. of ICDCS '03*, 2003.

[94] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In *Proceedings of IEEE Infocom'02*, New York, NY, June 2002.

[95] M. Younis, M. Youssef, and K. Arisha. Energy-aware routing in cluster-based sensor network. In *Proceedings of ACM/IEEE MASCOTS'2002*, October 2002.

[96] S. Yu, A. Yang, and Y. Zhang. Dada: A 2-dimensional adaptive node schedule to provide smooth sensor network services against random failures. In *Workshop on Information Fusion and Dissemination in Wireless Sensor Networks*, 2005.

[97] H. Zhang and A. Arora. GS^3: Scalable self-configuration and self-healing in wireless sensor networks. *Computer Networks (Elsevier)*, 2003.

[98] Pei Zhang, Christopher M. Sadler, Stephen A. Lyon, and Margaret Martonosi. Hardware design experiences in zebranet. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 227–238, New York, NY, USA, 2004. ACM Press.

[99] Jerry Zhao and Ramesh Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 1–13, New York, NY, USA, 2003. ACM Press.

[100] Rong Zheng, Jennifer C. Hou, and Lui Sha. Asynchronous wakeup for ad hoc networks. In *MobiHoc '03: Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 35–45, New York, NY, USA, 2003. ACM Press.

[101] G. Zhou, T. He, S. Krishnamurthy, and J. A. Stankovic. Impact of radio irregularity on wireless sensor networks. In *Proc. of ACM MobiSys'04*, June 2004.

[102] Y. Zou and K. Chakrabarty. Sensor deployment and target localization based on virtual forces. In *Proceedings of IEEE INFOCOM 2003*, March 2003.

[103] Y. Zou and K. Chakrabarty. Uncertainty-aware and coverage-oriented deployment for sensor networks. *Journal of Parallel and Distributed Computing*, 64(7):788–798, 2004.

[104] Yi Zou and K. Chakrabarty. A distributed coverage- and connectivity-centric technique for selecting active nodes in wireless sensor networks. *IEEE Trans. Computers*, 2005.

ABSTRACT

PETRA: TOWARD DEPENDABLE AND AUTONOMIC NETWORKED SENSOR SYSTEMS

by

SAFWAN AL-OMARI

DECEMBER 2008

Advisor: Dr. Weisong Shi

Major: Computer Science

Degree: Doctor of Philosophy

The recent development of small wireless sensing devices has opened the door for a plethora of new applications. In a typical application, these battery-powered and failure-prone sensor nodes are required to operated autonomously for extended period of time. In this work, we propose analytical models, system-level support, and network protocols that move WSNs forward toward autonomous, dependable, energy efficient, and cost effective WSNs. Our work falls into two major categories. First, we use analytical models to formalize and quantify WSN dependability characteristics, these models allow for the systematic integration of WSN dependability as a primary dimension in the design space of the communication stack protocols and algorithms as well as the deployment problem. Second, we propose a novel structure of the WSN communication stack based on our Sensor Core (Score). In the context of Score, we propose and develop fundamental network services that perform automatic low-level discovery services such as neighbor and topology discovery. Score along with discovery services provide a framework that hosts our protocols and provide them with system-level support to collaborate and coordinate with each other in a cross-layer approach.

AUTOBIOGRAPHICAL STATEMENT

Safwan Al-Omari is a PhD candidate at the Department of Computer of Wayne State University. He received his BSc degree in computer science from the University of Jordan in 1999, and his MSc in computer and Information science from the University of Michigan-Dearborn in 2003. He started his PhD in the winter of 2004, since then, he has been working on Wireless Sensor Networks in the Mobile and Internet SysTems lab. His research focuses on developing analytical models as well as systems support that help in building more reliable and highly available Wireless Sensor Networks. He has several publications in well-known International conferences and workshops including MASS'07, HealthNet'07, WASA'07, CollaborateCom'06, and Qshine'06.